Introducción a la Arquitectura de Sistemas
Apuntes de Cátedra

Hugo J. Curti

16 de marzo de 2011

Resumen

Este apunte presenta un resumen de los temas estudiados en la materia Introducción a la Arquitectura de Sistemas y tiene como objetivo introducir al alumno a cada uno de los conceptos fundamentales. En su carácter de resumen *no debe tomarse* como única fuente de estudio, sino como una lectura previa a abordar la vasta bibliografía disponible sobre el tema.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Representación de números y cantidades

Este capítulo fue escrito a partir del apunte [Sut98], y cubre la primera unidad de la materia sobre representación de números y cantidades.

[Sut98, Des02, Tan99, PLT99]

1.1. Introducción

La habilidad para contar, cuantificar y representar números y cantidades es uno de los pilares de la inteligencia humana. Tal es así que a lo largo de la historia civilizaciones enteras han dificultado o detenido su desarrollo científico/tecnológico debido a que no contaban con un buen sistema de representación de cantidades. En este capítulo se estudiarán diferentes formas de representar números según sus características de una manera adecuada para las máquinas digitales..

Antes de comenzar el estudio es menester hacer una distinción de conceptos. Una cosa es la cantidad representada y otra diferente es la simbología utilizada para la representación de la misma. La dificultad en comprender esta diferencia radica en que para entender una cantidad la mente humana utiliza una forma de representación intrínsecamente, que en nuestro caso particular es el *sistema posicional en base diez*, o *decimal*. Cuando se dice "en esa mesa hay *tres* manzanas" se está *representando* la cantidad, un concepto abstracto, con una palabra o símbolo previamente convenido (en este caso la palabra "tres" o el símbolo "3")

La razón por la cual se utiliza la base diez parece ser arbitraria: se cree que es porque el ser humano posee diez dedos en sus manos, y los antiguos utilizaban los dedos para contar. De la misma forma, el desarrollo de la electrónica digital impuso una nueva arbitrariedad. El componente fundamental con el cual se construye una máquina digital moderna se denomina *transistor*, y funciona "permitiendo" o "no permitiendo" el paso de la corriente[1], dando lugar a dos (en lugar de diez) estados o símbolos con los cuales dichas máquinas pueden contar. Esto trae grandes consecuencias, algunas de las cuales se enumeran a continuación:

- Se adopta la base dos o *binaria* para representar números y cantidades, utilizando la ausencia de corriente para representar el *cero* y la presencia de la misma para representar el *uno*, y bases potencia de dos, como la base ocho u *octal* y la base dieciséis o *hexadecimal*, para facilitar

[1] Esto es en realidad una simplificación. El funcionamiento del transistor está fuera del alcance de este apunte y de esta materia, y se estudiará en detalle en una materia posterior.

la comprensión humana de los números binarios.

- Se posee una cantidad limitada de unidades de almacenamiento, con lo cual las cantidades que se pueden representar también están limitadas.
- Solamente se pueden representar *ceros* y *unos*, con lo cual cualquier otro símbolo (como la coma raíz o el signo negativo) debe ser codificado de manera representable con estos símbolos.

Para poder representar los números se requiere por lo tanto crear un *Sistema de Representación*, que puede definirse como un conjunto de símbolos y reglas que se utilizan para la representación de cantidades y un conjunto de operaciones asociadas a los mismos.

1.2. Sistemas posicionales de representación

Los sistemas posicionales permiten representar números *Reales*. Poseen tantos símbolos como indique su base para representar las *cifras*, un *signo*, y una *coma raíz*[2] para separar la parte *entera* de la parte *fraccionaria*. Se denominan posicionales porque una misma cifra tiene distinto valor de acuerdo a su *posición relativa* dentro del número.

La ausencia del signo en un número implica que su valor es positivo, y la ausencia de coma indica un valor entero[3]. El valor representado puede calcularse utilizando la ecuación 1.1, donde N es la cantidad representada, n es la cantidad de cifras a la izquierda de la coma raíz (parte entera), d es la cantidad de cifras a la derecha de la coma raíz (parte fraccionaria), b es la base de representación, y $C = \{c_i \quad -d \leq i < n\}$ es el conjunto de las cifras que conforman el número ordenadas por su *peso* (o posición relativa a la coma, positiva empezando de cero a izquierda y negativa a derecha).

$$N = \sum_{i=-d}^{n-1} c_i b^i \tag{1.1}$$

1.2.1. Sistema decimal

Como ya se ha mencionado, el sistema decimal es el más utilizado, y debido a esto algunas cosas adquieren nombres que se utilizan exclusivamente para este sistema, y no deben generalizarse para todos los sistemas posicionales. Algunas de estas cosas son las siguientes:

- A la *coma raíz* se le llama *coma decimal*.
- A la parte fraccionaria se le llama *parte decimal* y a sus componentes *decimales.*
- A la cifra de peso 0 se le llama *unidad*, a la de peso 1 se le llama *decena*, a la de peso 2 se le llama *centena*, a la de peso 3 se le llama *unidad de mil*, y así sucesivamente.

Las ecuaciones 1.2 muestran ejemplos de representaciones decimales aplicando la ecuación 1.1.

$$\begin{aligned} 1234 &= 1\times10^3+2\times10^2+ \\ &\quad 3\times10^1+4\times10^0 \\ 31,25 &= 3\times10^1+1\times10^0+ \\ &\quad 2\times10^{-1}+5\times10^{-2} \end{aligned} \tag{1.2}$$

[2] En otros países (fundamentalmente de origen Sajón como Estados Unidos o Inglaterra) se utiliza el *punto*. En Argentina se utiliza la coma.

[3] Se puede asumir que la coma esta a la derecha del número, y su parte fraccionaria vale cero.

d	bin	d	bin
0	0_b	8	1000_b
1	1_b	9	1001_b
2	10_b	10	1010_b
3	11_b	11	1011_b
4	100_b	12	1100_b
5	101_b	13	1101_b
6	110_b	14	1110_b
7	111_b	15	1111_b

Cuadro 1.1: Representaciones binarias de los números enteros del 0 al 15.

1.2.2. Sistema binario

El sistema binario o de base dos es el adoptado naturalmente para representar números en las máquinas digitales debido a las razones explicadas en la sección 1.1. Debido a que la base es pequeña muchas cifras son necesarias para representar cantidades relativamente pequeñas. La tabla 1.1 muestra algunas representaciones binarias. Puede notarse en la tabla el sufijo $_b$ en todos los números binarios. Esta convención será utilizada en lo sucesivo en este apunte para evitar ambigüedades en las representaciones. La ausencia de sufijo siempre indica que el número está representado en decimal.

El bit

A la cifra binaria se la denomina comúnmente *bit*, que es la contracción de la frase inglesa "*binary digit*". El bit también se utiliza como unidad para medir el espacio ocupado por la información. En este contexto, una celda de memoria de un bit es aquella que es capaz de almacenar dos estados. Un ejemplo común de tal celda es un interruptor de luz.

El cuarteto

Dado que el bit es una unidad muy pequeña habitualmente no se utiliza por sí misma, sino agrupada. De esta forma, se le llama *cuarteto* o *nibble* a un conjunto de cuatro bits. Una celda de memoria de un cuarteto puede almacenar $2^4 = 16$ estados, y por lo tanto contar desde 0 hasta 15.

El byte

El byte es la unidad binaria más utilizada y conocida. Equivale a un conjunto de ocho bits. Por lo tanto una celda de memoria de un byte permite almacenar $2^8 = 256$ estados y contar desde 0 hasta 255.

Los multiplicadores de las unidades binarias

Para las unidades binarias se pueden utilizar los mismos multiplicadores que se utilizan para las unidades físicas, aunque clásicamente se utilizó siempre un factor de multiplicación de 1024 en lugar de 1000. La razón de este cambio es que $2^{10} = 1024$, o sea que 1024 se representa con un uno seguido de diez ceros, y por lo tanto forma un número "redondo" en binario, de la misma forma en que 1000 es un número "redondo" en decimal. Esta forma de tomar las unidades fue desde siempre un gran motivo de ambigüedad y confusión. En algunas áreas de la disciplina, donde no existe una necesidad fuerte de mantener un número "redondo" a nivel binario, se optó por mantener el factor de multiplicación de 1000[4]. Para resolver esta ambigüedad la Comisión Electro-

[4]Notablemente en las especificaciones de las capacidades de los discos rígidos, o en la especificación de ancho de banda de las conexiones de red.

Unidad Decimal	Equiv.	Bytes	Unidad Binaria	Equiv.	Bytes
kilobyte (kB)	1000 B	10^3	kibibyte (KiB)	1024 B	$2^{10} = 1{,}024$
megabyte (MB)	1000 kB	10^6	mebibyte (MiB)	1024 KiB	$2^{20} = 1{,}048{,}576$
gigabyte (GB)	1000 MB	10^9	gibibyte (GiB)	1024 MiB	$2^{30} = 1{,}073{,}741{,}824$
terabyte (TB)	1000 GB	10^{12}	tebibyte (TiB)	1024 GiB	$2^{40} = 1{,}099{,}511{,}627{,}776$
petabyte (PB)	1000 TB	10^{15}	petibyte(PiB)	1024 TiB	$2^{50} = 1{,}125{,}899{,}906{,}842{,}624$
exabyte (EB)	1000 PB	10^{18}	exbibyte (EiB)	1024 PiB	$2^{60} = 1{,}152{,}921{,}504{,}606{,}846{,}976$
zetabyte (ZB)	1000 EB	10^{21}	zebibyte (ZiB)	1024 EiB	$2^{70} = 1{,}180{,}591{,}620{,}717{,}411{,}303{,}424$
yottabyte (YB)	1000 ZB	10^{24}	yobibyte (YiB)	1024 ZiB	$2^{80} = 1{,}208{,}925{,}819{,}614{,}629{,}174{,}706{,}176$

Cuadro 1.2: Multiplicadores decimales y binarios aplicados al byte.

técnica Internacional (IEC) estandarizó en al año 1999 una serie de prefijos especiales para las unidades binarias. Para elaborar estos prefijos se toma la contracción del prefijo decimal con la palabra inglesa *binary*. Por ejemplo el prefijo binario kilo (*kilo binary*) se contrae como kibi. La tabla 1.2 muestra los multiplicadores decimales y binarios aplicados al byte. Debe prestarse atención a que el prefijo 'kilo' se identifica con la letra k minúscula, mientras que el resto de los prefijos se identifican con letras mayúsculas. Esto no se mantiene para las unidades binarias, donde todos los prefijos, incluyendo el prefijo Kibi, se identifican con la letra mayúscula del prefijo decimal seguida de la letra i minúscula.

Conversión de binario a decimal

La conversión de binario a decimal puede hacerse directamente aplicando la ecuación 1.1. En la ecuación 1.3 pueden verse un par de ejemplos.

$$\begin{aligned} 1101_b &= 1\times 2^3 + 1\times 2^2 \\ &= +0\times 2^1 + 1\times 2^0 = 13 \\ 10,01_b &= 1\times 2^1 + 0\times 2^0 \\ &= +0\times 2^{-1} + 1\times 2^{-2} = 2,25 \end{aligned} \tag{1.3}$$

Conversión de decimal a binario

Para convertir de decimal a binario, o en general de decimal a cualquier base, se debe factorizar el número según los diferentes pesos. Una forma sencilla de lograr esto es por el método de las divisiones/multiplicaciones sucesivas. La figura 1.1 muestra un ejemplo de como aplicar este método. Debe observarse que la parte entera puede leerse de los restos de las

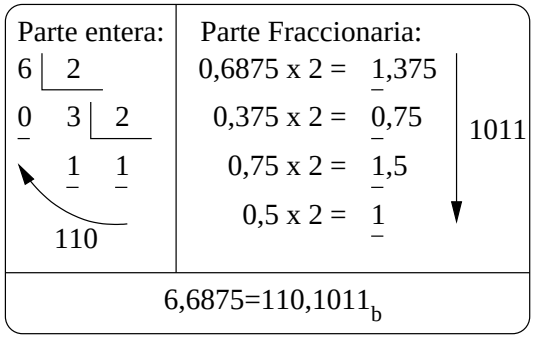


Figura 1.1: Aplicación del método de las divisiones/multiplicaciones sucesivas.

sucesivas divisiones[5] en sentido inverso. En la parte fraccionaria se deben hacer las sucesivas multiplicaciones siempre con la parte fraccionaria del resultado de la operación anterior, y se debe detener el proceso cuando no haya más parte fraccionaria, cuando se detecte una periodicidad[6] o cuando se alcance la precisión deseada, descartando el resto.

Operaciones aritméticas en binario

Para realizar sumas o restas en binario se puede utilizar el mismo método utilizado en decimal. La figura 1.2 muestra un ejemplo de suma y un ejemplo de resta de dos números binarios. Puede observarse en la suma que la última cifra queda superada, debiéndose por lo tanto agregar una cifra más al número. A esta cifra se la denomina *acarreo* en las sumas. En las restas, un fenómeno parecido se produce cuando el sustraendo es mayor que el minuendo, indicando que el resultado es negativo. A esta cifra se la denomina *arrastre* en las restas.

La figura 1.3 muestra un ejemplo de multi-

[5] La ultima división, que debería dar 0 como cociente, fue suprimida.

[6] Debe tenerse presente que un número que en decimal es periódico podría no serlo en binario, y viceversa.

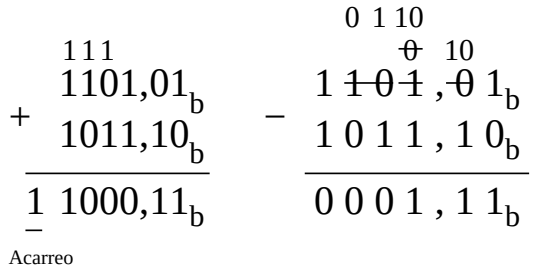


Figura 1.2: Ejemplo de suma y resta en binario.

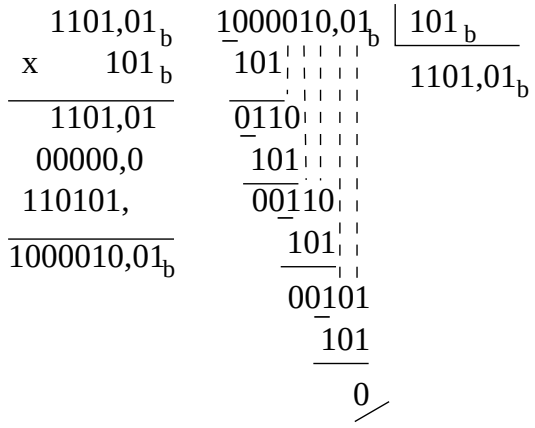


Figura 1.3: Ejemplo de multiplicación y división en binario.

plicación y otro de división donde puede apreciarse que el método a seguir es similar al utilizado para realizar estas mismas operaciones manualmente en decimal.

1.2.3. Sistemas Octal y Hexadecimal

La legibilidad de los números en binario es muy dificultosa para el ser humano debido a la cantidad de cifras que se necesitan para representar una cantidad. Debido a esto se suelen utilizar los sistemas octal (base ocho) y hexadecimal (base dieciséis) para representar números binarios. Para el sistema octal se utilizan los primeros ocho símbolos del sistema decimal, del 0 al 7. Para el sistema hexadecimal, donde se necesitan dieciséis símbolos, se utilizan los diez símbolos del sistema decimal y se agregan las letras de la A hasta la F a modo de seis símbolos más, de modo que $0_h = 0$, $1_h = 1$, $\ldots$, $9_h = 9$, $A_h = 10$, $B_h = 11$, $\ldots$, $F_h = 15$.

Debido a que ocho y dieciséis son potencias de dos, una cifra octal representa *exactamente* tres cifras binarias, y una cifra hexadecimal *exactamente* cuatro cifras binarias. Esto facilita por un lado la lectura de los números debido a que hay menos cifras para representar la misma cantidad, y a su vez la conversión desde y hacia binario se hace linealmente y por agrupamiento, sin necesidad de operaciones de multiplicación o división. La figura 1.4 muestra ejemplos de conversión por agrupamiento. Obsérvese especialmente la forma en que se completa con ceros el número sin alterar su valor para conseguir las cifras hexadecimales u octales enteras. La figura 1.5 muestra la demostración matemática de este método para convertir nueve cifras binarias a tres

$$
\begin{aligned}
N &= b_8 2^8 + b_7 2^7 + b_6 2^6 + b_5 2^5 + b_4 2^4 + b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0 2^0 \\
N &= \left(b_8 2^2 + b_7 2^1 + b_6 2^0\right) 2^6 + \left(b_5 2^2 + b_4 2^1 + b_3 2^0\right) 2^3 + \left(b_2 2^2 + b_1 2^1 + b_0 2^0\right) 2^0 \\
N &= \left(b_8 2^2 + b_7 2^1 + b_6 2^0\right) 8^2 + \left(b_5 2^2 + b_4 2^1 + b_3 2^0\right) 8^1 + \left(b_2 2^2 + b_1 2^1 + b_0 2^0\right) 8^0 \\
N &= o_2 8^2 + o_1 8^1 + o_0 8^0 \\
o_i &= b_{(3i+2)} 2^2 + b_{(3i+1)} 2^1 + b_{(3i)} 2^0 \qquad 0 \leq i < n_o
\end{aligned}
$$

Figura 1.5: Demostración de la conversión por agrupamiento.

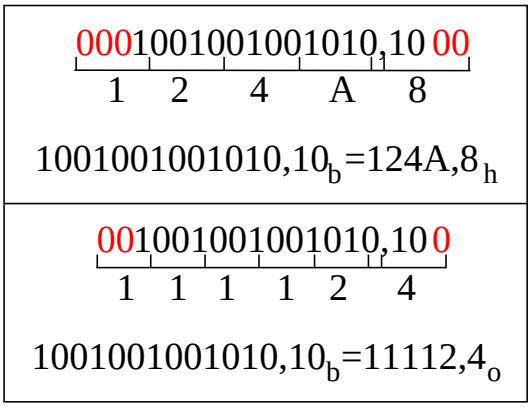


Figura 1.4: Ejemplo de conversión por agrupamiento de binario a hexadecimal/octal.

cifras octales, que puede ser extendida por inducción a cualquier cantidad de cifras. Obsérvese que los coeficientes o_i del número octal se calculan directamente a partir los tres coeficientes binarios correspondientes, interpretándolos como un número aislado.

La conversión desde y hacia decimal, así como las operaciones aritméticas pueden hacerse de la misma forma que las binarias explicadas en las sección 1.2.2, página 10. Para las multiplicaciones de las cifras en hexadecimal es conveniente convertirlas a decimal, hacer la multiplicación y volver a convertir a hexadecimal, dado que el aprendizaje de las tablas de multiplicar en hexadecimal es muy costoso.

1.2.4. Ejercicios

1. Comprobar que las operaciones realizadas en las figuras 1.2 y 1.3 están correctamente realizadas pasando los valores de los operandos y el resultado de cada una a decimal.

2. Calcule la representación binaria del año corriente utilizando el método de las divisiones sucesivas. ¿Cuántas cifras serán necesarias? Utilice el mismo método para calcular la representación hexadecimal del mismo año. ¿Cuántas cifras serán necesarias?

3. Realizar las operaciones de las figuras 1.2 y 1.3 en hexadecimal, pasando primero los operandos de cada una a dicha base, realizando la operación y pasando el resultado de vuelta a binario para comprobar su correctitud.

4. Realizar una demostración similar a la de la figura 1.5 convirtiendo doce cifras binarias (ocho enteras y cuatro fraccionarias) en tres cifras hexadecimales.

1.3. Representación de números naturales en máquinas digitales

Los sistemas posicionales hasta ahora estudiados permiten representar números positivos y negativos, tanto enteros como fraccionarios y con un rango y precisión ilimitada. Para aumentar el rango basta con agregar más cifras a la parte entera, y para aumentar la precisión basta con agregar más cifras a la parte fraccionaria. Para representar números negativos se puede agregar el signo adelante.

Las máquinas digitales modernas se basan, como ya se mencionó más arriba, en un componente denominado *transistor* que permite o no permite el paso de la corriente. Utilizando este componente se construye la denominada *unidad básica de memoria*. Dicha unidad puede ser programada o *escrita* con un valor de tensión alto, o con un valor de tensión bajo. Posteriormente la misma celda puede ser consultada o *leída* para obtener el mismo valor de tensión previamente escrito. Si se utiliza, por ejemplo, el nivel de tensión alto para representar el 1 y el nivel bajo para representar el 0, entonces esta unidad de memoria puede almacenar un bit. Agrupando ocho unidades básicas de memoria se puede hacer lo que a partir de ahora se denominará una *celda de memoria* de un byte, dado que permite almacenar ocho bits.

Se puede concluir que una celda de memoria puede utilizarse para almacenar la representación binaria de un número, sin embargo se deben analizar algunas diferencias entre una celda de memoria y el sistema binario puro:

1. En los sistemas binarios la cantidad de cifras es *variable* según el valor del número, mientras que la celda de memoria tiene una cantidad *fija* de cifras.

   Este problema se soluciona fácilmente cuando la cantidad de cifras de la representación es menor que el tamaño de la celda de memoria, dado que las cifras no usadas se dejan a la izquierda y con valor cero. Sin embargo, cuando la cantidad de cifras de la representación es mayor que el tamaño de la celda, el número no se puede representar en esa celda. Cuando se intenta almacenar un número en una celda que no lo puede contener se produce un fenómeno conocido como *rebalse* (u *overflow* en inglés). El rebalse será estudiado en detalle más adelante.

2. Dado que solamente se poseen dos símbolos, no existe manera natural de representar el signo de un número, así como tampoco existe manera natural de representar la coma raíz.

Como corolario se puede decir que la representación binaria de un número se puede almacenar en una celda de memoria siempre y cuando el número sea natural[7] y la cantidad de cifras que se necesiten para representarlo sea menor o igual a la cantidad de unidades que posea la celda de memoria. El rango que se puede representar es $R = [0\,,\, 2^n - 1]$ dando un total de 2^n representaciones, donde n es el tamaño de la celda de memoria. Por ejemplo, una celda de memoria de 16 bits puede almacenar $2^{16} = 65536$ representaciones diferentes, permitiendo un rango $R = [0\,,\, 65535]$.

[7] Esto es así porque se asume signo positivo y la coma raíz a la derecha de la representación.

1.4. Representación de números enteros

Es evidente que si las máquinas digitales pudieran representar únicamente números naturales su capacidad sería extremadamente limitada. En esta sección se explicarán sistemas de representación alternativos que permiten representar números positivos y negativos utilizando únicamente unos y ceros, y de esta forma hacer posible su almacenamiento en una celda de memoria.

Existen dos criterios de naturaleza diferente para solucionar este problema. El primero es agregar información extra a la representación. El segundo busca sustituir o *mapear* los números enteros con representaciones naturales, *repartiendo* el espacio de representación entre los números positivos y los negativos.

Los sistemas de representación pueden utilizarse para cualquier base, por lo cual se estudiarán de manera general y se aplicarán particularmente a la base dos.

1.4.1. Signo valor absoluto

El sistema de representación de enteros más intuitivo y simple es el denominado *Signo Valor Absoluto* o SVA, en el cual se representa el signo agregando un bit a la representación, por lo cual soluciona el problema utilizando el primero de los criterios mencionados al comienzo de la sección 1.4. Normalmente este bit valdrá cero cuando se represente un número positivo, y uno cuando se represente un número negativo.

Para identificar un sistema de Signo Valor Absoluto se utiliza la dupla $\mathrm{SVA}(b,d)$, donde b es la base de la representación y d es la cantidad de cifras que soporta. Las ecuaciones 1.4 muestran algunos ejemplos de este sistema.

$$\begin{array}{rcll} 18 & \equiv & \underline{0}\,012_h & \mathrm{SVA}(16,3) \\ -4 & \equiv & \underline{1}\,0100_b & \mathrm{SVA}(2,4) \\ -234 & \equiv & \underline{1}\,00234 & \mathrm{SVA}(10,5) \end{array} \tag{1.4}$$

El rango que se puede representar es $R = \left[-b^d+1\,,\,b^d-1\right]$ con un total de $2b^d$ representaciones. Puede observarse que el cero tiene una doble representación, dado que se puede representar el cero positivo y el cero negativo, que matemáticamente no tienen diferencia.

Para guardar una representación SVA en una celda de memoria se debe usar la base dos, y se debe tener presente que se hay que destinar un bit al signo, por lo cual $b = 2$ y $d = n-1$, siendo n el tamaño de la celda de memoria. Normalmente se utiliza el bit de más alto peso[8] para representar el signo. Por ejemplo, en una celda de memoria de 16 bits $b = 2$ y $d = 15$, por lo que el rango disponible es $R = [-32767\,,\,32767]$.

Operaciones aritméticas en SVA

Para realizar las operaciones aritméticas en SVA se pueden utilizar los mismos circuitos que se utilizan para los binarios puros, y luego se debe aplicar una conversión de acuerdo al signo de los operandos. En las sumas y restas se utiliza el signo para decidir si se suma o se resta, y en las multiplicaciones y divisiones se utiliza la regla de los signos para calcular el signo del resultado. En todos los casos se requiere lógica adicional que hace más complejos los circuitos.

[8] El bit más a la izquierda de la representación

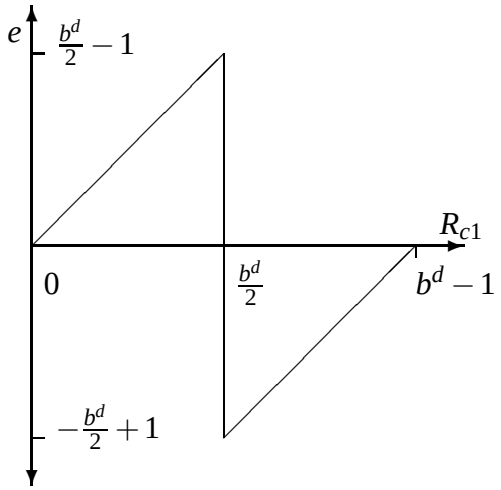


Figura 1.6: Distribución del espacio de representación en complemento a uno.

1.4.2. Complemento a uno

Este sistema de representación es un poco más complejo que signo valor absoluto y divide el espacio de representación entre los números positivos y los negativos, por lo tanto utiliza el segundo de los criterios mencionados al comienzo de la sección 1.4. Para calcular la representación en complemento a uno r_{c1} a partir del entero e que se desea representar se utiliza la ecuación 1.5. La figura 1.6 muestra la distribución de los enteros en el espacio de representación. El eje X muestra el avance de la representación como número natural y el eje Y el valor del entero representado.

$$r_{c1} = \begin{cases} e & 0 \leq e < \frac{b^d}{2} \\ b^d - 1 + e & -\frac{b^d}{2} < e \leq 0 \end{cases} \tag{1.5}$$

Para identificar un sistema complemento a uno se utiliza la dupla $\mathrm{C1}(b,d)$, donde b es la base de la representación y d es la cantidad de cifras que soporta. Las ecuaciones 1.6 muestran algunos ejemplos de este sistema que pueden calcularse utilizando la ecuación 1.5.

$$\begin{array}{rcll} 18 & \equiv & 012_h & \mathrm{C1}(16,3) \\ -4 & \equiv & 1111_b - 0100_b & \\ & = & 1011_b & \mathrm{C1}(2,4) \\ -234 & \equiv & 99999 - 00234 & \\ & = & 99765 & \mathrm{C1}(10,5) \end{array} \tag{1.6}$$

El rango que se puede representar es $R = \left[-\frac{b^d}{2}+1\,,\,\frac{b^d}{2}-1\right]$ con un total de b^d representaciones. Puede observarse que, al igual que en SVA, el cero tiene una doble representación[9].

Para guardar una representación C1 en una celda de memoria se debe usar la base dos, haciendo $b = 2$ y $d = n$, siendo n el tamaño de la celda de memoria. Por ejemplo, en una celda de memoria de 16 bits $b = 2$ y $d = 16$, por lo que el rango disponible es $R = [-32767\,,\,32767]$.

Operaciones aritméticas en C1

Las operaciones aritméticas se pueden aplicar directamente sobre las representaciones, y luego aplicar un factor de corrección para obtener la representación del resultado. Las ecuación 1.7 en la figura 1.7 muestra el factor de corrección a aplicar sobre las representaciones para las sumas, donde r_1 representa al entero e_1, r_2 representa al entero e_2 y r_{1+2} representa la suma de los enteros e_1 y e_2. Naturalmente, esta expresión será válida cuando la operación no produce rebalse. Debe observarse especialmente la utilización del operador módulo m, tal que $|nm+i|_m = i$, cuya semántica es "*cuando se supera el número máximo, se vuelve a cero*". Esta semántica es natural cuando la cantidad de cifras es limitada,

[9] $0 \equiv b^d - 1 \quad \mathrm{C1}(b,d)$ y $0 \equiv 0 \quad \mathrm{C1}(b,d)$

$$r_{1+2} = \begin{cases} |r_1 + r_2|_{b^d} & e_1 \geq 0 \wedge e_2 \geq 0 \\ & e_1 \geq 0 \wedge e_1 + e_2 < 0 \\ |r_1 + r_2 + 1|_{b^d} & e_1 < 0 \wedge e_2 < 0 \\ & e_1 < 0 \wedge e_1 + e_2 \geq 0 \end{cases} \quad (1.7)$$

$$r_{12} = \begin{cases} |r_1 r_2|_{b^d} & e_1 \geq 0 \wedge e_2 \geq 0 \\ |r_1 r_2 + r_1 - 1|_{b^d} & e_1 \geq 0 \wedge e_2 < 0 \\ |r_1 r_2 + r_1 + r_2 + 1|_{b^d} & e_1 < 0 \wedge e_2 < 0 \end{cases} \quad (1.8)$$

Figura 1.7: Cálculos de corrección de operaciones aritméticas en complemento a uno.

como por ejemplo en el cuentakilómetros de un auto o en una máquina digital.

La ecuación 1.8 en la figura 1.7 muestra los factores de conversión a aplicar en el caso de los productos, con las mismas aclaraciones que para la ecuación 1.7.

Comparación entre complemento a uno y signo valor absoluto

Una de las características distintivas del sistema complemento a uno en base dos es el hecho de que para cambiar el signo de un número basta con invertir la representación bit a bit. En signo valor absoluto lo mismo se puede lograr invirtiendo nada más que el bit de signo.

La lógica extra necesaria para trabajar en complemento a uno en máquinas digitales es bastante menor que en signo valor absoluto para el caso de las sumas. De hecho la solución es trivial en base dos, dado que las condiciones para corregir indicadas en la ecuación 1.7 se dan sí y solo sí la suma produce acarreo, con lo cual basta con sumar las representaciones y luego sumar el acarreo producido por esta última operación.

Para las multiplicaciones la lógica necesaria es mayor que en signo valor absoluto, dado que en este se debe tomar una decisión lógica a partir de los signos, mientras que en complemento a uno se debe tomar esa misma decisión y luego hacer hasta dos sumas.

1.4.3. Complemento a la base

Este sistema de representación es muy similar a complemento a uno. Distribuye el espacio de representación de una forma muy similar. La diferencia más importante entre complemento a la base y complemento a uno es que el primero no tiene doble representación del cero, aprovechando de esta forma una combinación más. Este cambio tiene consecuencias en las operaciones aritméticas que se estudiarán a continuación. Para calcular la representación en complemento a la base r_{cb} a partir del entero e que se desea representar se utiliza la ecuación 1.9. La figura 1.8 muestra la distribución de los enteros en el espacio de representación.

$$r_{cb} = \begin{cases} e & 0 \leq e < \frac{b^d}{2} \\ b^d + e & -\frac{b^d}{2} \leq e < 0 \end{cases} \quad (1.9)$$

Para identificar un sistema complemento a la base se utiliza la dupla $\mathrm{CB}(b,d)$, donde b es la base de la representación y d es la cantidad de cifras que soporta. Las ecua-

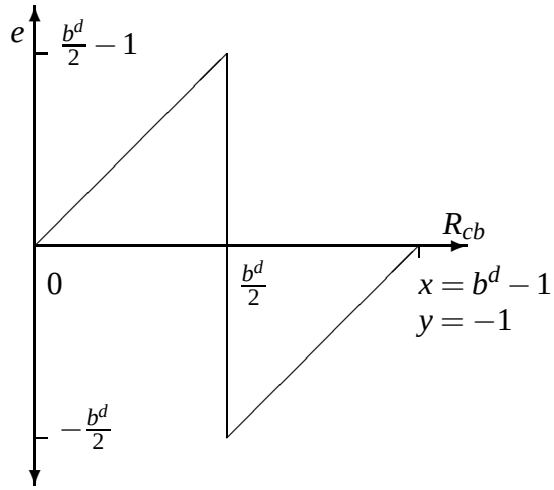


Figura 1.8: Distribución del espacio de representación en complemento a la base.

ciones 1.10 muestran algunos ejemplos de este sistema que pueden calcularse utilizando la ecuación 1.9.

$$\begin{aligned} 18 &\equiv 012_h && \text{CB}(16,3) \\ -4 &\equiv 10000_b - 0100_b \\ &= 1100_b && \text{CB}(2,4) \\ -234 &\equiv 100000 - 00234 \\ &= 99766 && \text{CB}(10,5) \end{aligned} \tag{1.10}$$

El rango que se puede representar es $R = \left[-\frac{b^d}{2}, \frac{b^d}{2}-1\right]$ con un total de b^d representaciones. Puede observarse que, a diferencia de C1, el cero tiene una única representación, y el rango negativo se extiende un número más lejos.

Para guardar una representación CB en una celda de memoria se debe usar la base dos, haciendo $b = 2$ y $d = n$, siendo n el tamaño de la celda de memoria. Por ejemplo, en una celda de memoria de 16 bits $b = 2$ y $d = 16$, por lo que el rango disponible es $R = [-32768, 32767]$.

Operaciones aritméticas en CB

Una de las características más destacadas del sistema de complemento a la base es que *no requiere conversiones ni para las sumas ni para las multiplicaciones*. Eso quiere decir que sumando las representaciones se obtiene la representación de la suma de los enteros, y multiplicando las representaciones se obtiene la representación de la multiplicación de los enteros.

Demostración matemática para la suma

Las representaciones se operan módulo b^d, que como ya se dijo, es natural cuando la cantidad de cifras es limitada. Además se asume que las operaciones no producirán rebalse. La demostración se hará estudiando cuatro casos que se distinguen por el signo de los operandos y el signo del resultado. Sean e_1 y e_2 dos números enteros, y sean r_1, r_2 y r_{1+2} las representaciones CB(b,d), de e_1, e_2 y e_1+e_2 respectivamente.

Primer caso Operandos enteros positivos, resultado positivo y sin rebalse: $e_1 \geq 0$, $e_2 \geq 0$, $0 \leq e_1 + e_2 < \frac{b^d}{2}$.

Por 1.9 $r_1 = e_1$, $r_2 = e_2$ y $r_{1+2} = e_1 + e_2$, por lo tanto

$$|r_1 + r_2|_{b^d} = |e_1 + e_2|_{b^d} \tag{1.11}$$
$$|r_1 + r_2|_{b^d} = e_1 + e_2 \tag{1.12}$$
$$|r_1 + r_2|_{b^d} = r_{1+2} \tag{1.13}$$

En 1.12 puede eliminarse el módulo debido a que en este caso la suma de los enteros e_1 y e_2 es mayor que cero y menor que b^d.

Segundo caso Un operando positivo y otro negativo, resultado positivo y sin rebalse: $e_1 \geq 0$, $e_2 < 0$, $0 \leq e_1 + e_2 < \frac{b^d}{2}$

Por 1.9 $r_1 = e_1$, $r_2 = b^d + e_2$ y $r_{1+2} = e_1 + e_2$, por lo tanto

$$|r_1 + r_2|_{b^d} = \left|e_1 + b^d + e_2\right|_{b^d} \quad (1.14)$$
$$|r_1 + r_2|_{b^d} = e_1 + e_2 \quad (1.15)$$
$$|r_1 + r_2|_{b^d} = r_{1+2} \quad (1.16)$$

En 1.15 puede eliminarse el módulo eliminado también el término b^d debido a que en este caso también la suma de los enteros e_1 y e_2 es mayor que cero y menor que b^d.

Tercer caso Un operando positivo y otro negativo, resultado negativo y sin rebalse: $e_1 \geq 0$, $e_2 < 0$, $-\frac{b^d}{2} \leq e_1 + e_2 < 0$

Por 1.9 $r_1 = e_1$, $r_2 = b^d + e_2$ y $r_{1+2} = b^d + e_1 + e_2$, por lo tanto

$$|r_1 + r_2|_{b^d} = \left|e_1 + b^d + e_2\right|_{b^d} \quad (1.17)$$
$$|r_1 + r_2|_{b^d} = b^d + e_1 + e_2 \quad (1.18)$$
$$|r_1 + r_2|_{b^d} = r_{1+2} \quad (1.19)$$

En 1.18 puede eliminarse el módulo debido a que en este caso la suma de los enteros e_1 y e_2 es negativa y mayor que $-b^d$, por lo tanto al sumar b^d quedará mayor que cero y menor que b^d.

Cuarto caso Dos operandos negativos, resultado negativo y sin rebalse: $e_1 < 0$, $e_2 < 0$, $-\frac{b^d}{2} \leq e_1 + e_2 < 0$

Por 1.9 $r_1 = b^d + e_1$, $r_2 = b^d + e_2$ y $r_{1+2} = b^d + e_1 + e_2$, por lo tanto

$$|r_1 + r_2|_{b^d} = \left|2b^d + e_1 + e_2\right|_{b^d} \quad (1.20)$$
$$|r_1 + r_2|_{b^d} = b^d + e_1 + e_2 \quad (1.21)$$
$$|r_1 + r_2|_{b^d} = r_{1+2} \quad (1.22)$$

En 1.21 puede eliminarse el módulo y uno de los dos términos b^d, debido a que en este caso la suma de los enteros e_1 y e_2 es negativa y mayor que $-b^d$, por lo tanto al sumar b^d (una vez) quedará mayor que cero y menor que b^d.

Demostración matemática para el producto

Las representaciones se operan módulo b^d y se asume que las operaciones no producirán rebalse. La demostración se hará estudiando tres casos que se distinguen por el signo de los operandos y el signo del resultado. Sean e_1 y e_2 dos números enteros, y sean r_1, r_2 y r_{12} las representaciones $\mathrm{CB}(b,d)$, de e_1, e_2 y $e_1 e_2$ respectivamente.

Primer caso Operandos enteros positivos, resultado positivo y sin rebalse: $e_1 \geq 0$, $e_2 \geq 0$, $0 \leq e_1 e_2 < \frac{b^d}{2}$.

Por 1.9 $r_1 = e_1$, $r_2 = e_2$ y $r_{12} = e_1 e_2$, por lo tanto

$$|r_1 r_2|_{b^d} = |e_1 e_2|_{b^d} \quad (1.23)$$
$$|r_1 r_2|_{b^d} = e_1 e_2 \quad (1.24)$$
$$|r_1 r_2|_{b^d} = r_{12} \quad (1.25)$$

En 1.24 puede eliminarse el módulo debido a que en este caso el producto de los enteros e_1 y e_2 es mayor que cero y menor que b^d.

Segundo caso Un operando positivo y otro negativo, resultado negativo y sin rebalse: $e_1 \geq 0$, $e_2 < 0$, $-\frac{b^d}{2} \leq e_1 e_2 < 0$

Por 1.9 $r_1 = e_1$, $r_2 = b^d + e_2$ y $r_{12} = b^d + e_1 + e_2$, por lo tanto

$$|r_1 r_2|_{b^d} = \left| e_1 \left(b^d + e_2 \right) \right|_{b^d} \quad (1.26)$$
$$|r_1 r_2|_{b^d} = \left| b^d e_1 + e_1 e_2 \right|_{b^d} \quad (1.27)$$
$$|r_1 r_2|_{b^d} = b^d + e_1 e_2 \quad (1.28)$$
$$|r_1 r_2|_{b^d} = r_{12} \quad (1.29)$$

En 1.28 puede eliminarse el módulo eliminando todos los b^d menos uno, debido a que en este caso el producto de los enteros e_1 y e_2 es negativo y mayor que $-b^d$, por lo tanto al sumar b^d una vez quedará mayor que cero y menor que b^d.

Tercer caso Dos operandos negativos, resultado positivo y sin rebalse: $e_1 < 0$, $e_2 < 0$, $0 \leq e_1 e_2 < \frac{b^d}{2}$

Por 1.9 $r_1 = b^d + e_1$, $r_2 = b^d + e_2$ y $r_{12} = e_1 e_2$, por lo tanto

$$|r_1 r_2|_{b^d} = \left| \left(b^d + e_1 \right) \left(b^d + e_2 \right) \right|_{b^d} \quad (1.30)$$
$$|r_1 r_2|_{b^d} = \left| b^{2d} + b^d e_1 + b^d e_2 + e_1 e_2 \right|_{b^d}$$
$$|r_1 r_2|_{b^d} = e_1 e_2 \quad (1.31)$$
$$|r_1 r_2|_{b^d} = r_{12} \quad (1.32)$$

En 1.31 puede eliminarse el módulo y todos los b^d, debido a que en este caso el producto de los enteros e_1 y e_2 es positivo y menor que b^d.

Comparación entre complemento a la base y complemento a uno

La inversión de signo en complemento a la base es un poco más compleja que en complemento a uno. Se debe invertir la la representación bit a bit y luego sumar uno.

La ventaja fundamental que posee complemento a la base es la posibilidad de operar sin ningún tipo de corrección, tanto para las multiplicaciones como para la suma. Esta ventaja es muy importante y posiblemente es la causa principal por la cual sea el sistema de representación más utilizado para números enteros.

1.4.4. Cero desplazado

El sistema de representación conocido como cero desplazado o notación en exceso también divide el espacio de representación entre positivos y negativos, de manera similar a complemento a la base. La diferencia más importante es que en cero desplazado los números negativos están primero, y los positivos después. Para calcular la representación en cero desplazado r_{cd} a partir del entero e que se desea representar se utiliza la ecuación 1.33. En esta ecuación f representa una constante característica del sistema de representación. Se denomina *frontera* y es el valor que representa al cero. Los valores inferiores a f representan enteros negativos, y los valores superiores representan enteros positivos. El caso particular de cero desplazado donde $f = \frac{b^d}{2}$ se denomina de *frontera equilibrada*. Los sistemas de frontera equilibrada utilizan la mitad del espacio de representación para enteros positivos y la otra mitad para enteros negativos, siendo por esta razón los más utilizados. En este apunte se llamará *cero desplazado* a secas a un sistema de frontera equilibrada, mientras que se denominará *cero desplazado con exceso a n* a un sistema de cero desplazado que utiliza *n* como frontera. La figura 1.9 muestra la distribución de los enteros en el espacio de representación con frontera equilibrada.

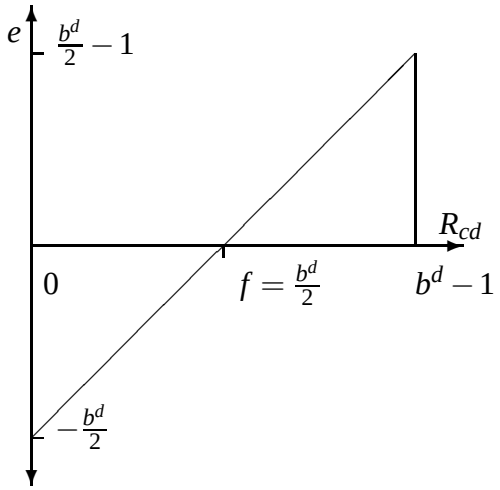


Figura 1.9: Distribución del espacio de representación en cero desplazado con frontera equilibrada.

$$r_{cd} = f + e \tag{1.33}$$

Para identificar un sistema de cero desplazado se utiliza la dupla $\mathrm{CD}(b,d)$, donde b es la base de la representación y d es la cantidad de cifras que soporta. Las ecuaciones 1.34 muestran algunos ejemplos de este sistema que pueden calcularse utilizando la ecuación 1.33.

$$\begin{aligned} 18 &\equiv 800_h + 12_h \\ &= 812_h && \mathrm{CD}(16,3) \\ -4 &\equiv 1000_b - 0100_b \\ &= 0100_b && \mathrm{CD}(2,4) \\ -234 &\equiv 50000 - 00234 \\ &= 49766 && \mathrm{CD}(10,5) \end{aligned} \tag{1.34}$$

El rango que se puede representar es $R = \left[-f\,,\, b^d - f - 1\right]$ con un total de b^d representaciones.

Para guardar una representación CD en una celda de memoria se debe usar la base dos, haciendo $b = 2$ y $d = n$, siendo n el tamaño de la celda de memoria. Por ejemplo, en una celda de memoria de 16 bits $b = 2$ y $d = 16$, por lo que el rango disponible es $R = [-32768\,,\, 32767]$ con frontera equilibrada.

Operaciones aritméticas en CD

Las operaciones aritméticas se pueden aplicar directamente sobre las representaciones, y luego aplicar un factor de corrección para obtener la representación del resultado. Las ecuación 1.35 muestra el factor de corrección a aplicar sobre las representaciones para las sumas, donde r_1 representa al entero e_1, r_2 representa al entero e_2 y r_{1+2} representa la suma de los enteros e_1 y e_2.

$$r_{1+2} = |r_1 + r_2 - f|_{b^d} \tag{1.35}$$

La ecuación 1.36 muestra el factor de conversión a aplicar en el caso de los productos. Esta corrección compleja se simplifica bastante si se utiliza frontera equilibrada y base dos.

$$r_{12} = \left|r_1 r_2 - (r_1 + r_2 - 1)\, f + f^2\right|_{b^d} \tag{1.36}$$

Comparación entre cero desplazado y complemento a la base

Una característica que distingue a los sistemas de cero desplazado de los otros sistemas en general es la capacidad de los primeros por distribuir asimétricamente el espacio de representación. Es decir, no necesariamente se debe dar la mitad del espacio para los números positivos y la otra mitad para los negativos. Por ejemplo, un sistema $\mathrm{CD}(10,2)$ con exceso a 10 posee un rango de $[-10\,,\, 89]$.

Otra característica de los sistemas de cero desplazado es que las representaciones y los

enteros representados se encuentran en el mismo orden lógico, por lo cual una comparación de las representaciones da el mismo resultado que la comparación de los enteros representados, cosa que no se da en los otros sistemas, donde siempre hay que utilizar lógica adicional para comparar.

Como corolario se puede decir que en aquellas circunstancias en que se desea distribuir asimétricamente el espacio de representación, o donde se desean favorecer las comparaciones los sistemas de cero desplazado son más convenientes.

1.4.5. Ejercicios

1. Extienda la demostración matemática de complemento a la base para las restas y las divisiones. ¿Qué ocurre con las divisiones?

2. Utilizar una demostración matemática similar a la utilizada para complemento a la base para demostrar la correctitud de las ecuaciones 1.7 y 1.8 para complemento a uno y las ecuaciones 1.35 y 1.36 para cero desplazado.

3. Calcular la corrección que debe utilizarse en el producto de cero desplazado con frontera equilibrada y base dos. ¿Es fácil de implementar esta corrección?

4. Encontrar una función que permita convertir una representación en cero desplazado con frontera equilibrada a la representación del mismo entero en complemento a la base. Considere ambos sistemas con la misma base y cantidad de cifras.

5. Buscar una ecuación que permita cambiar el tamaño de una representación complemento a la base, o sea pasar de una cantidad de cifras a otra mayor, o viceversa. Normalmente a este tipo de operaciones se las denomina *extensión de signo*.

1.5. Representación de números reales

Para representar números reales en máquinas digitales debe implementarse una manera de codificar la coma raíz, y de esta forma poder representar tanto la parte entera como la parte fraccionaria. Se deben tener en cuenta las siguientes consideraciones:

- Al haber una cantidad limitada de cifras destinadas a la parte fraccionaria, la precisión que se puede obtener es limitada, y por lo tanto *existirá un error máximo en la representación de un número real* denominado *error de precisión* o *error de redondeo*. Este error puede ser menor o incluso nulo para los números racionales[10], pero siempre existirá en los números irracionales.

- Para poder representar la parte fraccionaria es necesario sacrificar el rango de alcance de la representación. Es decir, cuanto más cifras fraccionarias posea una representación, mayor precisión y menor

[10] El lector debe recordar que los números racionales son aquellos que se pueden obtener del cociente $\frac{a}{b}$, siendo a y b dos números enteros, y se caracterizan por tener una cantidad finita de cifras o una periodicidad en su parte fraccionaria. Por otro lado los números irracionales tienen una cantidad infinita de cifras en su parte fraccionaria y no se detecta una periodicidad en la misma, por ejemplo, el número π.

rango poseerá para una misma cantidad total de cifras.

1.5.1. Precisión, avance, error absoluto y error relativo

Se denomina *avance* a la menor[11] diferencia que puede existir entre dos valores representados. El avance a puede calcularse con la ecuación 1.37 en función de la base de la representación b y la cantidad de cifras fraccionarias f. Cuanto menor es el avance más precisa es la representación. Otra forma de medir la precisión es mediante el *error absoluto máximo* que la representación puede cometer. Se define como error absoluto a la diferencia existente entre el número que se desea representar y el número representado. La ecuación 1.38 muestra esta definición, donde E es el error absoluto, v_r es el número representado y v el número que se desea representar. Como a priori no se define una política de aproximación[12] se puede asumir que el error absoluto máximo que se va a cometer en la representación es a lo sumo un avance. Por último, también se puede utilizar el *error relativo*, cuyo cálculo se muestra en la ecuación 1.39, en el cual se pondera el valor del error con el valor del número a representar en particular, o con el mayor número representable v_m en general[13], o inclusive se puede utilizar b^e. Este error es una razón (o un porcentaje) y por lo tanto refleja mejor el peso que el error posee sobre los números a representar.

$$a = b^{-f} \tag{1.37}$$

$$E = v_r - v \tag{1.38}$$

$$e = \frac{v_r - v}{v_m} \tag{1.39}$$

1.5.2. Representaciones de coma fija

La forma más simple de representar números con parte entera y parte fraccionaria es utilizar los sistemas denominados de *coma fija*. Se implementan conviniendo la posición de la coma raíz en cualquier cifra de la representación, desde la extrema izquierda, donde todas las cifras son fraccionarias, hasta la extrema derecha, donde todas las cifras son enteras[14]. Por lo tanto, todos los sistemas estudiados en la sección 1.4 pueden ser utilizados para representar números con parte fraccionaria. Las operaciones aritméticas por lo tanto también son las mismas[15].

Por ejemplo, para un sistema de representación $\mathrm{CB}(b,d)$, donde e cifras se destinan a la parte entera y f cifras a la parte fraccionaria por lo que $e+f=d$, la ecuación 1.40 muestra el cálculo del error absoluto máximo, la ecuación 1.41 muestra el cálculo del error relativo máximo, considerando que el mayor valor representable es $\frac{b^e}{2}$ y la ecuación 1.42 muestra el cálculo del rango.

$$E = b^{-f} \tag{1.40}$$

[11] Se refiere al valor positivo más pequeño distinto de cero.

[12] La política de aproximación o de redondeo define la acción a tomar cuando existe más información fraccionaria de la que se puede representar. La solución más simple es suprimir la información sobrante, o sea "*cortar los decimales*". Si bien es la solución más simple, también es una de las que más error introduce.

[13] Se adoptará esta forma de calcular el error relativo para el resto de este apunte.

[14] De hecho, los sistemas de representación de enteros son un caso particular de los sistemas de coma fija.

[15] Con la excepción de las multiplicaciones, que en algunos casos requieren extensiones de se signo.

$$e = \frac{2b^{-f}}{b^e} \quad (1.41)$$

$$R = \left[-\frac{b^e}{2}, \frac{b^e}{2} - b^{-f}\right] \quad (1.42)$$

1.5.3. Representaciones de coma flotante

Uno de los mayores inconvenientes que presentan las representaciones de coma fija es la imposibilidad de representar cantidades con diferencias de varios órdenes de magnitud, debido a que el error absoluto está fijo.

Por ejemplo, si se desea medir la distancia de la tierra a la luna se puede tolerar un error de algunas decenas de kilómetros. Si se mide la distancia entre dos ciudades ese mismo error no sería aceptable. Si se mide la distancia entre dos lugares dentro de una ciudad pequeña, el error invalidaría la medida. Sería una gran ventaja si con un mismo sistema se pudieran representar números muy grandes y números muy pequeños, que el error relativo se mantenga constante y varíe el error absoluto en función del orden de magnitud elegido. En este caso los sistemas de coma flotante, al cumplir con esta característica, se desempeñan de mejor forma.

La base de los sistemas de coma flotante es codificar por un lado un valor relativo, denominado *mantisa* y por el otro la posición de la coma raíz, denominada *exponente* dentro de ese valor.

Notación Científica

El sistema denominado de *notación científica* es un claro ejemplo de sistema de coma flotante. La ecuación 1.43 muestra algunos ejemplos de este sistema. En general el exponente siempre se aplica a la base de la mantisa, mostrando de esta forma el *corrimiento* de la coma. También puede observarse que hay muchas formas de representar el mismo número.

$$\begin{aligned} 145 &= 0,145 \times 10^3 \\ 145 &= 14,5 \times 10^1 \\ 0,0001234 &= 1,234 \times 10^{-4} \\ -0,273 &= -0,273 \times 10^0 \end{aligned} \quad (1.43)$$

Sistemas de punto flotante en máquinas digitales

Para definir un sistema de representación en punto flotante se deben especificar los siguientes parámetros:

Sistema de representación de la mantisa Define el sistema de representación, la cantidad de cifras y la base en la que se representará la mantisa. Obsérvese también que *esta es la misma base en la que se efectuarán los corrimientos.*

Sistema de representación del exponente Define el sistema de representación, la cantidad de cifras, y la base en la que se representará el exponente.

Normalización de la mantisa Define en qué cifra de la mantisa se considerará la coma raíz cuando el exponente vale cero, además de lo que se asume antes y después de las cifras de la mantisa, para el caso en que existan cifras implícitas.

Sistema de representación de coma flotante de la IBM360

A modo de ejemplo se mostrará el sistema de representación utilizado por la IBM360,

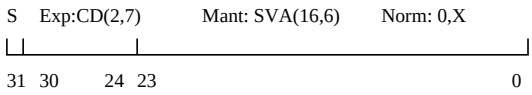


Figura 1.10: Representación de coma flotante utilizada por el sistema IBM 360

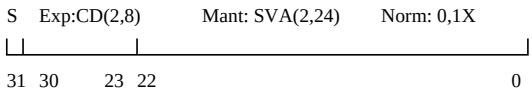


Figura 1.11: Representación de coma flotante utilizada por el sistema PDP/11

que posee un ancho de palabra[16] de 32 bits. La figura 1.10 muestra la distribución de los bits de la palabra entre la codificación de la mantisa y del exponente. Puede apreciarse que el bit de más alto peso almacena el signo de la mantisa, los siguientes siete bits almacenan el valor del exponente y los primeros veinticuatro bits almacenan la mantisa. Además puede verse la normalización, representada como "$0,X''$, que quiere decir "$0,$ *seguido del valor indicado por la mantisa*", es decir, que la coma raíz se encuentra a extrema izquierda. La ecuación 1.44 muestra cómo se calcula el valor real representado r en función del valor de la mantisa m y del exponente e.

$$r = 0,m \times 10_h^e \tag{1.44}$$

Las ecuaciones 1.45 muestran un ejemplo de conversión de un número a la representación utilizada por la IBM360.

$$\begin{array}{rcl} -18,5 & = & -12,8_h \\ -12,8_h & = & -0,128_h \times 10_h^2 \\ m & = & \underline{1}128_h \ \mathrm{SVA}(16,6) \\ e=2 & = & 1000010_b \ \mathrm{CD}(2,7) \\ r & = & 1\ 1000010_b\ 128_h \\ r & = & C2128000_h \end{array} \tag{1.45}$$

[16] Se define como ancho de palabra de un procesador a la cantidad de bits que el mismo puede manejar de una vez.

Sistema de representación de coma flotante de la PDP/11

El sistema de representación utilizado por la PDP/11, ofrece diferencias significativas con respecto al de la IBM360, y también posee un ancho de palabra de 32 bits. La figura 1.11 muestra la distribución de los bits de la palabra entre la codificación de la mantisa y del exponente. Puede apreciarse que el bit de más alto peso almacena el signo de la mantisa, los siguientes ocho bits almacenan el valor del exponente y los primeros veintitrés bits almacenan la mantisa. Además puede verse la normalización, representada como "$0,1X''$, que quiere decir "$0,1$ *seguido del valor indicado por la mantisa*", es decir, que la coma raíz se encuentra a extrema izquierda, y además existe un *uno implícito u oculto*, que permite una ganancia de precisión. La ecuación 1.46 muestra cómo se calcula el valor real representado r en función del valor de la mantisa m y del exponente e.

$$r = 0,1m \times 10_b^e \tag{1.46}$$

Las ecuaciones 1.47 muestran un ejemplo de conversión de un número a la representación utilizada por la IBM360

$$\begin{aligned} -18,5 &= -10010,1_b \\ -10010,1_b &= -0,100101_b \times 10_b^5 \\ m &= \underline{1}\,1\underbrace{00101_b}_{\text{explícito}}\ \mathrm{SVA}(2,24) \\ e = 5 &= 10000101_b\ \mathrm{CD}(2,8) \\ r &= 1\ 10000101_b\ 00101_b \\ r &= C2940000_h \end{aligned} \tag{1.47}$$

1.5.4. Ejercicios

1. ¿En qué casos utilizaría un sistema de coma fija antes que uno de coma flotante?

2. Calcule los límites (valores máximos y mínimos distintos de cero) para el sistema de representación de coma flotante de la IBM360, considerando siempre la mantisa normalizada.

3. Calcule los límites (valores máximos y mínimos distintos de cero) para el sistema de representación de coma flotante de la PDP/11.

4. ¿Por qué el uno oculto en el sistema de la PDP/11 ofrece una ganancia de precisión?

5. ¿Qué ventajas y desventajas ofrece el sistema de la PDP/11 sobre el de la IBM360?

Capítulo 2

Representación de Información

En el capítulo anterior se estudiaron diferentes formas de representar números y cantidades de manera soportada por la tecnología actual en máquinas digitales. En este capítulo se estudiaran formas para representar caracteres, imagenes y sonido de manera tal que las máquinas los puedan manejar.

2.1. Representación de caracteres

Es fundamental poder representar caracteres en una máquina digital. De esta forma se hace posible almacenar y procesar no solamente números y cantidades, sino también texto en general. La solución utilizada es la de asociar cada caracter que se desea representar a un número natural, de manera convencional o estandarizada. De esta forma se construyen las *paginas de códigos*, que son tablas que asocian en forma unívoca caracteres con números. Se estudiarán a continuación diferentes estándares.

2.1.1. Código ISO 646 o ASCII

Uno de los códigos estandarizados más utilizado para representar caracteres es el denominado ASCII, que es la contracción de *American Standard Code for Information Interchange* (Código estándar americano para intercambio de información), posteriormente internacionalizado como ISO 646. Se codifica con 7 bits, permitiendo por lo tanto 128 combinaciones. Incluye las letras del alfabeto latino inglés (tanto mayúsculas como minúsculas), los números del 0 al 9[1], los signos de puntuación, el espacio y en general los símbolos que pueden encontrarse en el teclado de una máquina de escribir. También incluye algunos caracteres no imprimibles que se utilizan para controlar terminales e impresoras de línea. La tabla 2.1 muestra cada caracter ASCII con su código asociado [Man96]. Los primeros 32 caracteres corresponden a códigos de control y no son imprimibles. Por eso se muestra su nombre estándar y, para los que existe una, la representación que se utiliza en el lenguaje C.

2.1.2. Códigos ISO 8859

Uno de los problemas fundamentales del ASCII es que solamente codifica los símbolos

[1] No se debe confundir un número con la representación ASCII de una cifra. Por ejemplo, el dígito 0 se representa con el código ASCII 48 (30_h), teniendo en este contexto valor como caracter, y no como cantidad.

0	nul	‘\0’	16	dle	32	spc	48	0	64	@	80	P	96	‘	112	p
1	soh		17	dc1	33	!	49	1	65	A	81	Q	97	a	113	q
2	stx		18	dc2	34	"	50	2	66	B	82	R	98	b	114	r
3	etx		19	dc3	35	#	51	3	67	C	83	S	99	c	115	s
4	eot		20	dc4	36	$	52	4	68	D	84	T	100	d	116	t
5	enq		21	nak	37	%	53	5	69	E	85	U	101	e	117	u
6	ack		22	syn	38	&	54	6	70	F	86	V	102	f	118	v
7	bel	‘\a’	23	etb	39	’	55	7	71	G	87	W	103	g	119	w
8	bs	‘\b’	24	can	40	(	56	8	72	H	88	X	104	h	120	x
9	ht	‘\t’	25	em	41	)	57	9	73	I	89	Y	105	i	121	y
10	lf	‘\n’	26	sub	42	*	58	:	74	J	90	Z	106	j	122	z
11	vt	‘\v’	27	esc	43	+	59	;	75	K	91	[	107	k	123	{
12	ff	‘\f’	28	fs	44	,	60	<	76	L	92	\	108	l	124	|
13	cr	‘\r’	29	gs	45	-	61	=	77	M	93	]	109	m	125	}
14	so		30	rs	46	.	62	>	78	N	94	^	110	n	126	~
15	si		31	us	47	/	63	?	79	O	95	_	111	o	127	del

Cuadro 2.1: Representación ASCII de caracteres.

128	pad	144	dcs	160		176	°	192	À	208	Đ	224	à	240	ð
129	hop	145	pu1	161	¡	177	±	193	Á	209	Ñ	225	á	241	ñ
130	bph	146	pu2	162	ç	178	²	194	Â	210	Ò	226	â	242	ò
131	nbh	147	sts	163	£	179	³	195	Ã	211	Ó	227	ã	243	ó
132	ind	148	cch	164	°	180	´	196	Ä	212	Ô	228	ä	244	ô
133	nel	149	mw	165	Y	181	µ	197	Å	213	Õ	229	å	245	õ
134	ssa	150	spa	166	|	182	¶	198	Æ	214	Ö	230	æ	246	ö
135	esa	151	epa	167	§	183	·	199	Ç	215	×	231	ç	247	÷
136	hts	152	sos	168	¨	184	¸	200	È	216	Ø	232	è	248	ø
137	htj	153	sgci	169	©	185	¹	201	É	217	Ù	233	é	249	ù
138	vts	154	sci	170	ª	186	º	202	Ê	218	Ú	234	ê	250	ú
139	pld	155	csi	171	«	187	»	203	Ë	219	Û	235	ë	251	û
140	plu	156	st	172	¬	188	¼	204	Ì	220	Ü	236	ì	252	ü
141	ri	157	osc	173	-	189	½	205	Í	221	Ý	237	í	253	ý
142	ss2	158	pm	174	®	190	¾	206	Î	222	Þ	238	î	254	þ
143	ss3	159	apc	175	¯	191	¿	207	Ï	223	ß	239	ï	255	ÿ

Cuadro 2.2: Representación ISO 8859-1 de caracteres (parte alta).

Variante	Alfabetos	Alias
ISO 8859-1	Lenguas de Europa occidental	Latin-1
ISO 8859-2	Lenguas de Europa oriental	Latin-2
ISO 8859-3	Lenguas del sudeste de Europa, y otras	Latin-3
ISO 8859-4	Lenguas escandinavas/balcánicas	Latin-4
ISO 8859-5	Latín/cirílico	
ISO 8859-6	Latín/árabe	
ISO 8859-7	Latín/griego	
ISO 8859-8	Latín/hebreo	
ISO 8859-9	Modificación de Latin-1 para el turco	Latin-5
ISO 8859-10	Lenguas lapona/nórdica/esquimal	Latin-6
ISO 8859-15	Soporte para el Euro (€) y el centimo (¢)	

Cuadro 2.3: Lista de variantes del ISO 8859

utilizados por el alfabeto inglés. El estándar ISO 8859 incluye varias extensiones de 8 bits al conjunto de caracteres ASCII. El objetivo de estos códigos de página es incluir soporte para otros alfabetos y otros idiomas. Existen diferentes variantes, de las cuales la más conocida es la ISO 8859-1, el *Alfabeto Latino Nro. 1* o *latin1*, el cual ha sido implementado tan extensamente que se puede considerar de hecho el sustituto éstandar del ASCII. La tabla 2.2 muestra la disposición de los caracteres entre los códigos desde el 128 hasta el 255 [Man99]. Los primeros 32 caracteres corresponden a caracteres de control. La primera parte de la codificación corresponde exactamente a la ASCII mostrada en la tabla 2.1. La tabla 2.3 muestra las distintas variantes del ISO 8859 y los alfabetos/idiomas que incluye cada uno.

2.1.3. Códigos ISO 10646, UCS y UNICODE

Estos estándares forman la evolución de los anteriores y buscan universalizar y hacer compatibles todos los sistemas del mundo bajo una misma norma, además de proveer mecanismos de conversión bidireccionales con todos los estándares existentes [Kuh95a].

El estándar internacional ISO 10646 define el conjunto de caracteres UCS (Conjunto Universal de Caracteres). UCS contiene todos los caracteres de todos los demás estándares de conjuntos de caracteres. También garantiza una compatibilidad de ida y vuelta, es decir, se pueden construir tablas de conversión de tal forma que no se pierda ninguna información cuando una cadena se convierta desde cualquier otra codificación a UCS y viceversa.

UCS contiene el conjunto de los caracteres necesarios para representar casi todos los lenguajes conocidos. Esto incluye, además de los numerosos lenguajes que usan una extensión de la escritura Latina, los siguientes alfabetos y lenguajes: Griego, Cirílico, Hebreo, Árabe, Armenio, Gregoriano, Japonés, Chino, Hiragana, Katakana, Coreano, Hanguliano, Devangari, Bengalí, Gurmuki, Gujarati, Oriya, Tamil, Telugu, Kannada, Malayam, Tai, Lao, Bopomofo, y algunos otros. Hay trabajo en curso

para la inclusión de alfabetos tales como Tibetano, Jemer, Rúnico, Etíope, Jeroglíficos, varios lenguajes indo-europeos, y muchos otros. Para muchos de estos últimos alfabetos, en el momento de la publicación del estándar en 1993, aún no resultaba claro cómo codificarlos de la mejor forma.

Además de los caracteres necesarios para estas escrituras, también se han incluido un enorme número de símbolos gráficos, tipográficos, matemáticos y científicos, como los proporcionados por TEX, PostScript, MS-DOS, Macintosh, Videotext, OCR, y muchos sistemas de procesamiento de textos, además de códigos especiales que garanticen la compatibilidad de ida y vuelta con todos los demás estándares existentes de juegos de caracteres.

El estándar UCS (ISO 10646) describe una arquitectura del conjunto de caracteres de 31 bits. No obstante, hoy en día sólo a los primeros 65534 códigos (desde 0x0000 a 0xfffd), que se denominan BMP (Plano Multilingüe Básico), se les han asignado caracteres, y se espera que sólo caracteres muy exóticos (como los jeroglíficos) de uso científico especial obtengan alguna vez un lugar fuera de este BMP de 16 bits.

Los caracteres UCS 0x0000 a 0x007f son idénticos a los del conjunto de caracteres ASCII clásico y los caracteres en el rango de 0x0000 a 0x00ff son idénticos a los del conjunto de caracteres ISO 8859-1.

Caracteres de combinación

Algunos códigos en UCS han sido asignados como caracteres de combinación. Éstos son similares a las teclas de acento en una máquina de escribir. Un carácter de combinación sólo añade un acento al carácter previo. Los caracteres acentuados más importantes tienen códigos propios en UCS, sin embargo, el mecanismo de combinación de caracteres permite añadir acentos y otras marcas diacríticas a cualquier carácter. Los caracteres de combinación siempre siguen al carácter al cual modifican. Por ejemplo, en alemán el carácter Umlaut-A (“A mayúscula con diéresis” o “Ä”) puede representarse por el ya compuesto código UCS 0x00c4, o alternativamente como la combinación de una “A mayúscula” normal seguida por una “diéresis de combinación”: 0x0041 0x0308.

Niveles de implementación

Puesto que no se espera que todos los sistemas soporten mecanismos avanzados tales como los caracteres de combinación, el ISO 10646 especifica los siguientes tres grados de implementación del UCS

Nivel 1 Los caracteres de combinación y caracteres Hangul Jamo (una codificación especial del coreano, más compleja, en la que las sílabas Hangul se codifican como 2 ó 3 subcaracteres) no están implementados.

Nivel 2 Igual que el nivel 1, sin embargo en algunos alfabetos se permiten ciertos caracteres de combinación (como Hebreo, Árabe, Devangari, Bengalí, Gurmukhi, Oriya, Tamil, Telugo, Kannada, Malayalam, Tai, y Lao).

Nivel 3 Todos los caracteres de UCS están soportados.

El estándar Unicode 1.1 publicado por Unicode Consortium contiene exactamente el UCS BMP implementado al nivel 3, según se describe en ISO 10646. Unicode 1.1 también aña-

de algunas definiciones semánticas para ciertos caracteres a las definiciones de ISO 10646.

Area Privada

En el BMP, en el rango de 0xe000 a 0xf8ff ningún carácter será nunca asignado por el estándar y dicha zona se reserva para uso privado.

2.1.4. Código UTF-8

La codificación UTF-8 de Unicode y UCS es una forma intermedia que se utiliza para usar el conjunto de caracteres Unicode bajo sistemas operativos al estilo UNIX [Kuh95b].

Propiedades

- Los caracteres UCS 0x00000000 a 0x0000007f se codifican simplemente como los bytes 0x00 a 0x7f (compatibilidad con ASCII). Esto significa que los ficheros y cadenas que contengan solamente caracteres ASCII de 7 bits tienen la misma codificación en ASCII y en UTF-8.

- Todos los caracteres UCS desde el 0x80 hacia arriba se codifican como una secuencia multibyte formada solamente por bytes en el rango 0x80 a 0xfd, por lo que ningún byte ASCII puede aparecer como parte de otro carácter y no hay problemas con, por ejemplo, '\0' o '/'.

- Se preserva la enumeración lexicográfica de las cadenas UCS-4[2].

- La totalidad de los códigos posibles en UCS pueden codificarse con UTF-8.

- Los bytes 0xfe y 0xff no se usan nunca en la codificación UTF-8.

- El primer byte de una secuencia multibyte que represente un carácter no ASCII UCS siempre se halla en el rango 0xc0 a 0xfd, e indica la longitud de la secuencia. El resto de los bytes de la secuencia se hallan en el rango 0x80 a 0xbf. Esto permite una fácil resincronización y resulta en una codificación sin estado y robusta frente a la pérdida de bytes.

- Los caracteres UCS codificados en UTF-8 pueden llegar a ser de 6 bytes, no obstante los caracteres del BMP sólo pueden ser de 3 bytes a lo sumo.

2.1.5. Ejercicios

1. Calcular la cantidad de espacio que se necesita para almacenar el texto de 200 libros de 500 páginas en promedio cada uno, con 72 líneas de 80 letras cada uno en promedio. Utilice la codificación ISO-8859-1.

2. Calcule la cantidad de espacio que se necesitaría en el mismo caso que en el ejercicio anterior, pero utilizando la codificación UTF-8. Considere que el 10 % de los caracteres del texto en promedio se codifican con la parte alta del ISO-8859-1, y por lo tanto ocupan 2 bytes en UTF-8.

[2] Se denomina UCS-4 al UCS de 31 bits, y UCS-2 al suconjunto que codifica el BMP.

2.2. Representación de imágenes

Para conseguir crear, almacenar y/o procesar imágenes utilizando máquinas digitales es necesario codificar la información contenida en estas imágenes en formatos basados en el sistema binario. Se estudiarán a continuación dos formas de representar imágenes en binario.

2.2.1. *Raster* o mapa de bits

La forma más sencilla de construir una imagen es como la unión de muchos puntos o *pixeles* de diferentes colores. Si los puntos son lo suficientemente pequeños y se los observa de suficiente distancia el ojo humano los compone para dar la sensación de una imagen continua. Si se codifica el color y la posición de cada uno de los pixeles se obtiene un formato *raster* o de mapa de bits. La figura 2.1 muestra un ejemplo de este tipo de codificación en binario y hexadecimal. Se puede apreciar que la imagen posee un total de 64 pixeles formando una matriz de 8 x 8. Como se codifican 4 colores diferentes es necesario utilizar 2 bits por pixel. La ecuación 2.1 permite calcular el tamaño en bytes b de la imagen en función de la cantidad de colores c, la cantidad de líneas y y la cantidad de columnas x. A b debe sumarse un encabezado que especifique el tamaño de la imagen, la forma en que se distribuyen las líneas y las columnas, la cantidad de colores y la forma en que estos se codifican.

$$b = \frac{xy\lceil \log_2 c \rceil}{8} \tag{2.1}$$

Entre los formatos de mapa de bits estándar más comunes pueden mencionarse el "*Graphics Internet Format*" (GIF), el "*Joint Photographic Expert Group*" (JPEG), el "*Portable Pixmap Format*" (PPM), el "*Portable Network Graphic*" (PNG) y el "*Tag Image File Format*" (TIFF). Otro formato propietario pero muy utilizado es el "*Windows Bitmap*" (BMP).

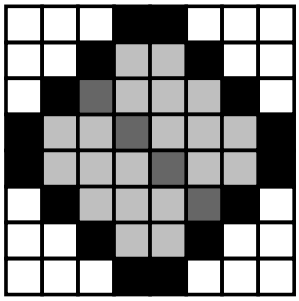


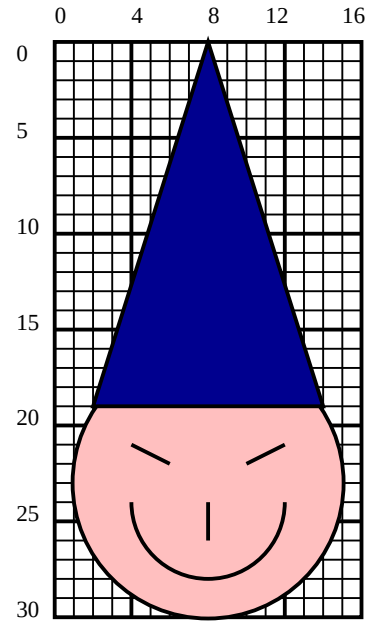
Figura 2.1: Mapa de bits junto con un ejemplo de codificación binaria y hexadecimal.

Factor de compresión y factor de calidad

Algunos de los formatos citados utilizan algoritmos de compresión para disminuir el tamaño final de los archivos. Se denomina *factor de compresión* a la relación $\frac{U}{C}$ donde U es el tamaño del archivo sin comprimir y C es el tamaño del archivo comprimido.

A fin de conseguir factores de compresión más elevados algunos de estos algoritmos de compresión introducen *pérdida* en la imagen, lo cual quiere decir que no es posible recuperar la información original al comprimir y posteriormente descomprimir la imagen.

El *factor de calidad* es una razón que mide cuánta información de la imagen se conserva. Normalmente es inversamente proporcional al factor de compresión en los algoritmos con pérdida.



triangulo({8 , 0} , { 2 , 19 } , { 14 , 19 } , fondo azul)
circulo({ 8 , 23 } , radio 7 , fondo rosa)
semicirculo({ 4 , 24 } , { 8 , 28 } , { 12 , 24 })
linea({ 8 , 26 } , { 8 , 24 })
linea({ 4 , 21 } , { 6 , 22 })
linea({ 10 , 22 } , { 12 , 21 })

Figura 2.2: Ejemplo de representación de una imagen en formato vectorizado.

2.2.2. Formatos Vectorizados

Otra forma de representar una imagen es codificar el proceso de construcción de la misma en lugar de su descripción. La figura 2.2 muestra un ejemplo de este tipo de representación. Cada una de las partes de la figura se arma mediante un *vector*, que indica un tipo de forma y una serie de parámetros como la posición de algunos puntos, el tamaño, los colores que la conforman, etc.

Entre los formatos vectorizados estándar más comunes puede citarse el "*Encapsulated Postscript*" (EPS) y el "*Computer Graphics Metafile*" (CGM). Otro formato propietario pero muy utilizado es el "*Windows Metafile*" (WMF).

Comparación entre mapas de bits y vectorizados

Se enumeran a continuación las diferencias más importantes entre ambos sistemas de representación de imagenes.

- La cantidad de información que se debe guardar depende del *tamaño* en el mapa de bits, mientras que en los vectores depende de la complejidad de la imagen. La complejidad está dada por la cantidad y el tamaño de los vectores.

- Cuando se escala un mapa de bits (tanto para aumentar como para reducir) se produce pérdida de información, mientras que los vectorizados se pueden escalar fácilmente sin que se produzca pérdida alguna.

- Para procesar un vectorizado se necesita mayor capacidad de procesador, dado que su construcción suele requerir gran cantidad de operaciones de punto flotante, mientras que los mapas de bits necesitan mayor cantidad de memoria.

- Los vectorizados son más aptos para gráficos generados, mientras que los mapas de bits son más aptos para imágenes naturales, como las fotografías.

- La conversión de un vectorizado a un mapa de bits es relativamente simple y muy eficiente, mientras que el camino inverso es extremadamente complejo y muchas

veces imposible de conseguir de manera eficiente[3].

2.2.3. Ejercicios

1. Calcular el tamaño en bytes de una imagen de mapa de bits de 640×480 y 2^{32} colores.

2. Calcular el espacio necesario para esa misma imagen si se guarda con un formato que posee un factor de compresión de 500 % en promedio.

2.3. Representación de audio

El sonido es un efecto físico que corresponde a una variación periódica de presión que viaja por un medio material, como el aire, en forma de ondas. Esas variaciones son detectadas por el oído humano y procesadas en el cerebro.

Mediante transductores se pueden convertir las variaciones de presión en variaciones de tensión o de corriente eléctrica y por lo tanto se puede transportar la información de audio por un cable, amplificarla, procesarla y luego volver a convertirla mediante un altavoz en variaciones de presión para que lleguen al oído.

Para poder guardar o procesar audio en una máquina digital es necesario convertirlo en información digital. La forma más utilizada para la digitalización de señales de audio es la denominada *Pulse Coded Modulation* (PCM). Esta forma consiste en tomar muestras de la señal a una frecuencia fija y conocida, y generar un número binario para cada una de ellas proporcional a su valor de tensión. La figura 2.3 muestra la toma de muestras de una señal de audio, y la figura 2.4 muestra el proceso de reconstrucción de la señal de audio a partir de las muestras. Puede apreciarse que el resultado de la unificación de las muestras forma una señal "cuadrada" que se caracteriza por tener una gran cantidad de armónicos. Mediante filtros analógicos pasabanda y pasabajos se consigue la interpolación de los puntos para recrear una señal muy similar a la original.

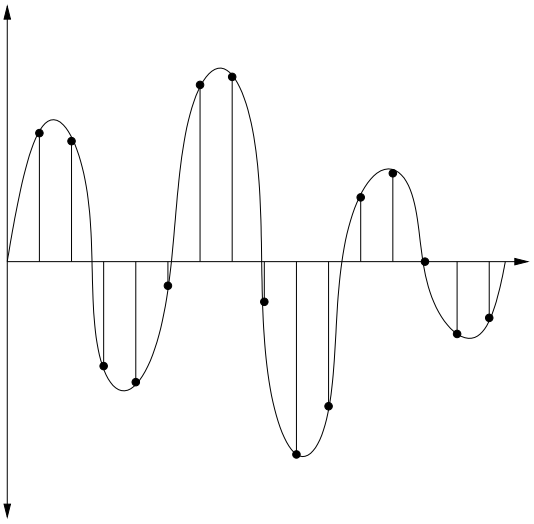


Figura 2.3: Digitalización de una señal de sonido mediante la toma de muestras.

Según el teorema de NYQUIST, una señal puede reconstruirse a partir de muestras de la mismas si la frecuencia de las muestras es por lo menos el doble de la frecuencia de la señal. Las frecuencias de audio que el oído humano escucha van desde los 20 Hz[4] y pueden llegar hasta los 20 kHz para un oído entrenado, por

[3] Siempre se puede crear un vector para cada punto del mapa de bits, pero esto es extremadamente ineficiente.

[4] Hercios, *Hertz* o Hz es la unidad de medida de frecuencia. Significa *ciclos por segundo*. 1 kilo Hercio o kHz es igual a 1000 Hz.

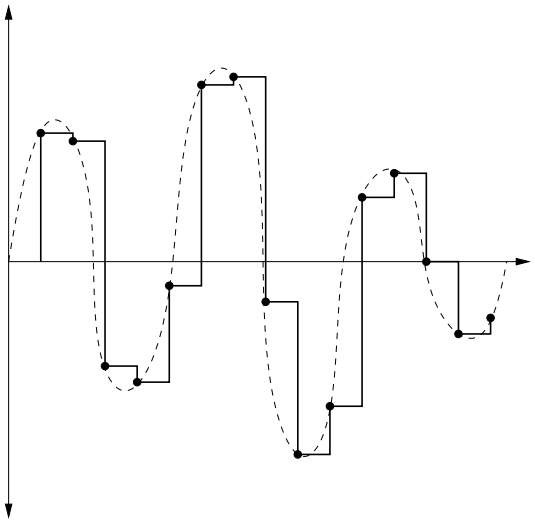


Figura 2.4: Reconstrucción de una señal a partir de sus muestras.

lo que si se desea muestrear audio de alta calidad se debe utilizar una frecuencia de muestreo mínima de 40 kHz. Sin embargo, está demostrado también que se puede entender y reconocer una voz humana con las frecuencias comprendidas entre los 20 Hz y los 4 kHz, por lo que una frecuencia de muestreo de 8 kHz puede alcanzar para almacenar y transmitir en forma digital una voz humana.

La ecuación 2.2 muestra la cantidad de espacio en bytes necesario para almacenar una señal de audio, donde b es la cantidad de espacio requerido en bytes, c es la cantidad de canales[5], B_m es la cantidad de bits por muestra, F_m es la frecuencia de muestreo en Hz y T es el tiempo de duración de la señal.

$$b = \frac{c \times B_m \times F_m \times T}{8} \qquad (2.2)$$

Puede verse que el espacio ocupado es propocional a la cantidad de canales, a la cantidad de bits de la muestra, a la frecuencia de muestreo y al tiempo de duración de la señal. Una mayor cantidad de bits por muestra devuelve una mayor precisión en la reconstrucción de la señal, dando por resultado una señal de más calidad, mientras que una mayor frecuencia de muestreo permite reconstruir señales de frecuencias más elevadas.

Los formatos de audio en general utilizan un encabezado donde se guardan los parámetros de la digitalización seguido de la sucesión de muestras. Para reducir la cantidad de espacio que ocupa la información de audio se suelen utilizar algoritmos de compresión. Como formatos conocidos de archivos de audio puede mencionarse el WAV, el AU y el MPEG 3 (o MP3). Este último utiliza un algoritmo de compresión que puede alcanzar factores de compresión de 1000 % introduciendo pérdida.

2.3.1. Ejercicios

1. Calcular la cantidad de espacio necesaria para almacenar 5 minutos de música estereofónica con muestras de 16 bits y una frecuencia de muestreo de 44 kHz.

2. Calcular la cantidad de espacio necesaria para almacenar un mensaje de 5 minutos monofónico con muestras de 8 bits y una frecuencia de muestreo de 8 kHz.

3. Calcular el tiempo de música codificada con los mismos parámetros del ejercicio 1 que puede guardarse en un CDROM de 650 MB.

4. Repetir el cálculo del ejercicio anterior, pero utilizando la compresión MP3.

[5] El sonido monofónico lleva un canal, el estereofónico lleva dos canales.

Bibliografía

[Des02] **J. P. Deschamps**. *Síntesis de circuitos digitales*. Thomson, 2002.

[Kuh95a] **M. Kuhn**. "Páginas de manual en línea de los sistemas GNU.", Dic 1995. unicode(7).

[Kuh95b] **M. Kuhn**. "Páginas de manual en línea de los sistemas GNU.", Nov 1995. utf-8(7).

[Man96] "Páginas de manual en línea de los sistemas GNU.", Dic 1996. ascii(7).

[Man99] "Páginas de manual en línea de los sistemas GNU.", May 1999. iso-8859-1(7).

[PLT99] **A. Prieto, A. Lloris y J. C. Torres**. *Introducción a la Informática*. McGraw–Hill interamericana de España, 1999. ISBN: 84-481-1627-5.

[Sut98] **G. Sutter**. "Sistemas de numeración en máquinas digitales." Apuntes de cátedra de Introducción a la Arquitectura de Sistemas.

[Tan99] **A. Tanembaum**. *Structured Computer Organization*. Prentice Hall, 1999. ISBN: 0-13-095990-1.