DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores

Representaciones Numéricas Parte 1

1. Motivación

Todo tipo de información puede ser representada mediante números. Una palabra, por ejemplo, puede ser representada mediante números reemplazando cada letra por un número específico (como el código ASCII). Una imagen puede representarse como un conjunto de posiciones (números) y el valor de color de la imagen en esa posición (también números). En definitiva, podemos reducir el problema de representar información al problema de representar números, por lo que es fundamental conocer las representaciones numéricas existentes y como se utilizan.

2. Representaciones de números naturales

La representación numérica más simple corresponde a usar un símbolo por cada incremento en uno de un número. Por ejemplo, si ocupamos el símbolo «*» para representar el número cuatro, dibujamos cuatro símbolos: «****». El problema de esta representación es que no escala: para representar el número un millón, necesitamos dibujar un millón de veces el símbolo «*» lo que sería un costo tanto en espacio (por ejemplo papel si estuviésemos escribiendo el número en un cuaderno) y de tiempo (escribir un millón de veces el símbolo tomará al menos un millón de segundos, aproximadamente 11 días sin parar).

Para solucionar los problemas que presenta esta representación, se inventaron representaciones más avanzadas, que permiten de alguna forma representar números grandes con pocos símbolos. Una de estas representaciones fueron los números romanos, que ocupaban símbolos específicos para acumulaciones de números: V para cinco, X para diez, L para cincuenta, etc. Esta representación, a pesar de ser una mejora sobre la anterior, tiene serias limitaciones prácticas, ya que no contempla un símbolo para el cero ni permite hacer operaciones algebraicas.

Otra representación ideada fueron los números indo-arábigos, que es la que ocupamos hoy en día, basada en diez símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9. Esta representación se caracteriza por pertenecer a un grupo de representaciones denominadas «representaciones posicionales».

2.1. Representaciones posicionales

Las representaciones posicionales, como la indo-arábiga, se basan en dos elementos para determinar el valor de un número: la posición de los símbolos en la secuencia de estos y la cantidad de símbolos posible, lo que se denomina la base. En el caso de los números indo-arábigos la base es diez (i.e. hay diez símbolos posibles), y por esto esta representación numérica también se denomina «representación decimal».

Para entender como afecta la posición en el valor numerico, es necesario realizar un ejemplo. Como estamos tan acostumbrados a asociar un número a su representación en el sistema decimal, para este

ejemplo, vamos a reemplazar los símbolos asociados al número uno y dos por «?» y «&» de manera de poder abstraernos del número y fijarnos en la representación.

Supongamos se tiene la secuencia de numérica «??&». La expresión para poder interpretar este número en el sistema decimal es la siguiente:

$$? \times diez^{dos} + ? \times diez^{uno} + \& \times diez^{cero} \tag{1}$$

Al evaluar esta expresión obtendremos el valor de la secuencia numérica, pero debemos primero elegir la representación en la que queremos hacer el cálculo. Si utilizamos la representación decimal, y reemplazamos los símbolos «?» y «&» por sus valores 1 y 2 obtenemos:

$$1 \times 10^2 + 1 \times 10^1 + 2 \times 10^0 = 112 \tag{2}$$

A partir de esto podemos obtener la regla general para obtener el valor de una secuencia de símbolos en representación decimal:

$$\sum_{k=0}^{n-1} s_k \times 10^k \tag{3}$$

donde s_k corresponde al símbolo en la posición k, n corresponde al número de símbolos en la secuencia y k toma valores desde 0 hasta $n-1$, es decir, las posiciones parten desde 0, tomando como inicio el símbolo de más a la derecha.

De la expresión anterior se puede inferir que podemos crear representaciones posicionales con otras bases, para lo cual bastaría reemplazar el número diez por la base deseada. De esta manera, la expresión general para obtener el valor numérico de una determinada representación posicional es:

$$\sum_{k=0}^{n-1} s_k \times b^k \tag{4}$$

donde b es la base de la representación elegida, la cual podría ser cualquier valor numérico. De hecho, el uso de la base diez que consideramos habitual se debe solamente a que los seres humanos tenemos diez dedos, y por tanto atribuimos a ese número una cierta característica especial. Si tuviéramos ocho dedos, seguramente ocuparíamos una sistema posicional con base ocho. En definitiva, no existe un sistema que de por si sea mejor que otro, todo depende de las circunstancias y de su uso.

2.2. Representaciones binaria, octal y hexadecimal

Existen tres representaciones habitualmente usadas en el contexto de la computación: binaria, octal y hexadecimal, siendo la primera la de mayor importancia. Tal como sus nombres lo señalan, la representación binaria tiene base dos, la octal base ocho y la hexadecimal base dieciseis. Vamos a comenzar analizando la base octal, para luego pasar a las otras dos.

Dado que un número octal tiene base ocho, necesitamos ocho símbolos distintos para su representación. Por comodidad utilizaremos los primeros ocho números indo-arábigos: 0, 1, 2, 3, 4, 5, 6 y 7. Es importante

destacar que aunque podríamos ocupar cualquier otro grupo de ocho símbolos, sólo ocuparemos estos por conveniencia, dado que estamos acostumbrados a trabajar matemáticamente con éstos.

Tomemos como ejemplo de número octal el 112. El primer elemento importante a destacar es que esta secuencia de símbolos no representa al número «ciento doce» que habitualmente asociaríamos a esos símbolos si estuviesen en representación decimal. Es decir, la secuencia de símbolos es la misma, pero la representación distinta. Para evitar confusiones, para todas las bases no decimales se usa especificar la base junto a la secuencia de símbolos. Entonces, en nuestro caso, el número sería $(112)_8$, es decir la secuencia 112 en representación octal.

Si utilizamos la expresión (4) podemos obtener el valor numérico de esta secuencia octal. Es importante resaltar que, al ocupar esta expresión, debemos decidir en que representación queremos el resultado. Lo habitual es que éste quede en representación decimal, pero podría quedar también en otra si lo quisiéramos. Más adelante veremos ejemplos de este tipo, pero por ahora haremos el cálculo de manera de obtener la el valor en representación decimal:

$$1 \times 8^2 + 1 \times 8^1 + 2 \times 8^0 = 74 \tag{5}$$

Es decir, la secuencia 112 en octal, representa el número decimal 74.

Revisemos ahora la representación binaria, la cual tiene como base el número dos, y por tanto requiere sólo dos símbolos. Ocuparemos, nuevamente y por comodidad, los dos primeros números indo-arábigos, 0 y 1. Tomemos como ejemplo el número $(1011)_2$. Nuevamente es importante recordar que esta secuencia no tienen ninguna relación con el número «mil once» en representación decimal, solamente tienen la misma secuencia de símbolos. Podemos nuevamente aplicar la expresión (4) para obtener el valor numerico en representación decimal:

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 1^0 = 11 \tag{6}$$

La ventaja de esta representación es que nos basta con dos símbolos para representar todos los posibles números. La desventaja es que necesitamos más posiciones para representar números. Esta desventaja la hace poco práctica para ser usada por humanos, pero la ventaja de tener sólo dos estados la hace ideal para realizar cálculos automáticos, por ejemplo en un computador.

Podemos también definir una representación cuya base sea mayor que diez. Una representación importante que cumple con esto es la representación hexadecimal, la cual tiene como base el número dieciséis. Más adelante veremos que la utilidad de esta representación está en su relación con la representación binaria, y lo fácil que es convertir de una a otra. Para esta representación necesitamos dieciseis símbolos, por lo que a diferencia de las otras representaciones estudiadas no nos basta con los números indo-arábigos, necesitamos seis símbolos más. Para no tener que inventar nuevos símbolos específicos de esta representación, habitualmente se ocupan los diez símbolos indo-arábigos para representar los diez primeros dígitos y las primeras seis letras del abecedario (A=diez, B=once, C=doce, D=trece, E=catorce, F=quince) para representar los seis restantes.

Por ejemplo, la secuencia $(A1F)_{16}$ puede ser interpretada como un número hexadecimal. Para obtener el valor numérico en representación decimal, nuevamente ocupamos la expresión (4):

$$A \times 16^2 + 1 \times 16^1 + F \times 16^0 = ? \tag{7}$$

En este caso tenemos un problema al evaluar directamente la expresión: aparecen símbolos (A y F) que no son válidos en la representación decimal. Para solucionar esto, podemos convertir estos símbolos directamente a representación decimal: A=10 y F=15, y con estos valores reescribir la expresión para obtener el resultado:

$$10 \times 16^2 + 1 \times 16^1 + 15 \times 16^0 = 2560 + 16 + 15 = 2591 \tag{8}$$

Este ejemplo demuestra una regla importante: si ocupamos la expresión (4) para convertir un número de una representación A a otra representación B, en el caso de que la base de A sea mayor que B (por ejemplo dieciséis >diez) necesariamente necesitamos un paso previo antes de realizar el cálculo: convertir todos los símbolos no válidos de la representación A a la representación B (en el ejemplo $A = 10$ y $F = 15$).

A modo de ejemplo de la regla anterior, veamos el caso de convertir un número decimal a representación binaria, por ejemplo el número 123. Si aplicamos directamente la ecuación (4) tenemos:

$$1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 = 1 \times 100 + 2 \times 10 + 3 \times 1 \tag{9}$$

El primer problema que se nos presenta es el explicado en la regla anterior: los símbolos 2 y 3 no existen en la representación binaria, por lo cual debemos en primer lugar reemplazarlos por su representación: $2 = (10)_2$ y $3 = (11)_2$:

$$(1)_2 \times 100 + (10)_2 \times 10 + (11)_2 \times 1 \tag{10}$$

Ahora tenemos un segundo problema: los números 100 y 10 están representados en decimal, por tanto debemos también reemplazarlos por su valor en esta representación: $100 = (1100100)_2$ y $10 = (1010)_2$.

$$(1)_2 \times (1100100)_2 + (10)_2 \times (1010)_2 + (11)_2 \times (1)_2 = (1100100)_2 + (10100)_2 + (11)_2 = (1111011)_2 \tag{11}$$

Este ejemplo nos muestra una clara desventaja de ocupar la expresión (4) para transformar desde representación decimal a una de menor base: necesitamos saber a priori la representación de la base y todos las potencias de la base (en este caso 100 y 10) en la representación no decimal para realizar el cálculo. Más adelante estudiaremos otros mecanismos que permiten realizar esta conversión sin necesidad de este conocimiento.

Nota: En general, cuando todos los números que se están usando sean de la misma representación, se puede obviar la notación $(num)_{base}$. Para el caso de número binarios y hexadecimales, existen otras notaciones habituales usadas cuando se quiere diferenciar la representación:

- Un número binario $(num)_2$ se suele escribir también como $numb$, por ejemplo $(1011)_2$ se puedes escribir como $1011b$
- Un número hexadecimal $(num)_{16}$ se suele escribir también como $numh$, por ejemplo $(A1)_{16}$ se puedes escribir como $A1h$

- Un número hexadecimal $(num)_{16}$ se suele escribir también como $0xnum$, por ejemplo $(A1)_{16}$ se puedes escribir como $0xA1$

2.3. Aritmética en distintas representaciones

La gran ventaja de las representaciones posicionales es que los procedimientos aritméticos como suma y multiplicación son equivalentes para toda representación. Estudiaremos primero la aritmética de la representación decimal, a la cual estamos habituados y a partir de ésta generalizaremos las reglas de la suma que son válidas para cualquiera de estas representaciones.

Como ejemplo realizaremos paso a paso la suma entre los números 112 y 93. El algoritmo tradicional para realizar la suma es el siguiente:

1. Escribir uno de los números debajo del otro, alineados por la derecha:

$$\begin{array}{r} 132 \\ 93 \\ \hline \end{array}$$

2. Sumar los dos dígitos de más a la derecha. Si la suma es menor que diez (5 en este caso), continuar con el siguiente dígito hacia la izquierda:

$$\begin{array}{r} 132 \\ 93 \\ \hline 5 \end{array}$$

3. Sumar los siguientes dos dígitos. Si la suma es menor que diez, continuar con el siguiente dígito hacia la izquierda. Si la suma es es mayor o igual que diez (12 en este caso), restarle 10 a la suma y colocar como resultado la resta (2 en este caso). Convertir los 10 en 1 y agregarlo como sumando a los siguientes dígitos (este valor se denomina **acarreo** o **carry** en inglés) :

$$\begin{array}{r} 1 \\ 132 \\ 93 \\ \hline 25 \end{array}$$

4. Sumar los siguientes dos dígitos. En caso de haber acarreo sumarlo también. Revisar los mismos casos que antes :

$$\begin{array}{r} 1 \\ 132 \\ 93 \\ \hline 225 \end{array}$$

El algoritmo general para la suma en representación decimal de dos números $num1$ y $num2$ sería:

1. Comenzar desde la derecha en la posición 0 de ambos números.

2. Mientras no lleguemos al fin de ambos números por la izquierda ni queden acarreos sin sumar:

   a) Sumar los dígitos de la posición actual más algún posible acarreo previo.

   b) Si la suma es menor que 10, colocar en la posición actual del resultado el valor de la suma.

   c) Si la suma es mayor igual que 10, restarle 10 a la suma, colocar lo que resulta de la resta en la posición actual del resultado y agregar un acarreo para la siguiente posición.

En el algoritmo anterior usamos solamente dos veces el número 10: al comparar que la suma de los dígitos sea mayor que 10 y luego en caso de que se cumpla esto, al restarle 10 al valor obtenido. En base a esto podemos generalizar este algoritmo para sumar cualquier par de números en representación posicional de base b reemplazando el número 10 por la variable b:

1. Comenzar desde la derecha en la posición 0 de ambos números.

2. Mientras no lleguemos al fin de ambos números por la izquierda ni queden acarreos sin sumar:

   a) Sumar los dígitos de la posición actual más algún posible acarreo previo.

   b) Si la suma es menor que b, colocar en la posición actual del resultado el valor de la suma.

   c) Si la suma es mayor o igual que b, restarle b a la suma, colocar lo que resulta de la resta en la posición actual del resultado y agregar un acarreo para la siguiente posición.

Usemos este algoritmo con un ejemplo de dos números binarios $(110)_2$ y $(11)_2$, en el cual la base $b = 2$:

1. Comenzar desde la derecha en la posición 0 de ambos números:

$$\begin{array}{r} 110 \\ 11 \\ \hline \end{array}$$

2. Mientras no lleguemos al fin de ambos números por la izquierda ni queden acarreos sin sumar

   a) Sumar los dígitos de la posición actual más algún posible acarreo previo. Si la suma **(en este caso 1)** es menor que 2 , colocar en la posición actual del resultado el valor de la suma.

$$\begin{array}{r} 110 \\ 11 \\ \hline \mathbf{1} \end{array}$$

   b) Sumar los dígitos de la posición actual más algún posible acarreo previo. Si la suma **(en este caso 2)** es mayor igual que 2 , restarle 2 a la suma, colocar lo que resulta de la resta en la posición actual del resultado **(en este caso 0)** y agregar un acarreo para la siguiente posición.

$$\begin{array}{r} \mathbf{1} \\ 110 \\ 11 \\ \hline \mathbf{0}1 \end{array}$$

   c) Sumar los dígitos de la posición actual más algún posible acarreo previo. Si la suma **(en este caso 2)** es mayor igual que 2 , restarle 2 a la suma, colocar lo que resulta de la resta en la posición actual del resultado **(en este caso 0)** y agregar un acarreo para la siguiente posición.

$$\begin{array}{r} \mathbf{1}1 \\ 110 \\ 11 \\ \hline \mathbf{0}01 \end{array}$$

d) Sumar los dígitos de la posición actual más algún posible acarreo previo. Si la suma **(en este caso 1)** es menor que 2 , colocar en la posición actual del resultado el valor de la suma.

$$\begin{array}{r} 11 \\ 110 \\ 11 \\ \hline \mathbf{1}001 \end{array}$$

Ejercicio: Multiplique los números binarios $(101)_2$ y $(10)_2$. Para hacerlo, primero analice el algoritmo de multiplicación de números decimales y luego aplíquelo a la multiplicación de números binarios.

2.4. Algoritmos de conversión entre representaciones

Como se analizó previamente la ecuación (4) representa un método general de conversión entre bases. Esta ecuación funciona bien para transformar un número en representación no decimal a la decimal (en la cual podemos realizar operaciones aritméticas rápidamente), sin embargo, como también se vio, para el caso inverso no era tan útil, y por tanto es necesario explorar otros algoritmos. Adicionalmente, para el caso de los números binarios existen otros algoritmos o heurísticas que pueden resultar más simples.

En esta sección se revisarán primero algoritmos de conversión binario-decimal y decimal-binario, y luego algoritmos de conversión entre representaciones hexadecimal, octal y binaria.

2.4.1. Algoritmos de conversión binario-decimal

La ecuación (4) representa un método útil para convertir entre números binarios y decimales, sin embargo, requiere que seamos capaces de recordar rápidamente las potencias del número dos, lo cual puede ser fácil para potencias bajas, pero para potencias altas puede volverse complejo y por tanto ser necesario tener que calcular estas potencias previo a la conversión. A continuación se presentan dos algoritmos que permiten convertir un número binario a decimal sin conocer las potencias de dos.

Algoritmo de acarreo inverso

La idea de este algoritmo es comenzar desde la izquierda e ir invirtiendo el proceso de acarreo, es decir por cada vez que vemos un número 1 lo devolvemos a la derecha como un número 2. Veamos el algoritmo a través de un ejemplo: vamos a convertir el número $(101101)_2$ a representación decimal.

1. Tomamos el primer 1, **1**01101, lo convertimos en 2 y lo sumamos al dígito de la derecha: 0**2**1101
2. Repetimos el paso para el siguiente dígito. Esta vez tenemos un 2 lo que equivale a dos 1 y por tanto debemos sumar dos veces 2 = 4 al siguiente dígito de la derecha: 00**5**101
3. Repetimos el paso para el siguiente dígito. Esta vez tenemos un 5 lo que equivale a cinco 1 y por tanto debemos sumar cinco veces 2 = 10 al siguiente dígito de la derecha: 000**11**01
4. Repetimos y ahora sumamos 22 al siguiente dígito de la derecha: 0000**22**1
5. Repetimos y ahora sumamos 44 al siguiente dígito de la derecha: 0000**45**. Llegamos al fin del número, y por tanto tenemos que el resultado de la conversión es 45.

Algoritmo de multiplicar e incrementar

Este algoritmo es equivalente al anterior, pero presenta reglas más simples y no requiere pensar el proceso de acarreo inverso. Las reglas de este algoritmo son:

- Tomar el 1 de más a la izquierda del número binario, e ir avanzando de izquierda a derecha. Comenzar con el resultado en 1.
- Por cada número 0 que se encuentra, multiplicar el resultado actual por 2.
- Por cada número 1 que se encuentra, multiplicar el resultado actual por 2 y sumar 1.

Veamos el algoritmo a través de un ejemplo: vamos a convertir el número $(101101)_2$ a representación decimal.

1. Tomamos el 1 de más a la izquierda, **1**01101. Comenzamos con el resultado = 1.
2. Tomamos el 0 que sigue, 1**0**1101: multiplicamos por 2 el resultado: resultado = 2
3. Tomamos el 1 que sigue, 10**1**101: multiplicamos por 2 el resultado y sumamos 1: resultado = 5
4. Tomamos el 1 que sigue, 101**1**01: multiplicamos por 2 el resultado y sumamos 1: resultado = 11
5. Tomamos el 0 que sigue, 1011**0**1: multiplicamos por 2 el resultado: resultado = 22
6. Tomamos el 1 que sigue, 10110**1**: multiplicamos por 2 el resultado y sumamos 1: resultado = 45

2.4.2. Algoritmos de conversión decimal-binario

Como mencionamos anteriormente, la ecuación (4) no es un método práctico para realizar esta conversión. A continuación presentaremos dos algoritmos para realizar la conversión:

Algoritmo de acarreos sucesivos

La idea de este algoritmo es comenzar con todos los número acumulados a la derecha (se puede pensar como si se tuvieran fichas acumuladas como una torre) e ir acarreando grupos de a dos hacia la izquierda, convirtiéndolos en 1 a medida que se acarrean.

Veamos el algoritmo a través de un ejemplo: vamos a convertir el número 45 a representación binaria.

1. Comenzamos con 45 «fichas» a la derecha.
2. Acarreamos todos los pares de fichas que hayan a la izquierda, en este caso 22 y por cada par, sumamos un 1 a la izquierda. En la primera posición nos sobra una ficha, lo que representa un número 1. En este momento llevamos el número 22 **1**
3. Acarreamos todos los pares de fichas que hayan en la próxima posición, en este caso 11 y por cada par, sumamos un 1 a la izquierda. Esta vez no nos sobra una ficha, lo que representa un número 0. En este momento llevamos el número 11 **0**1
4. Acarreamos todos los pares de fichas que hayan en la próxima posición, en este caso 5 y por cada par, sumamos un 1 a la izquierda. Esta vez sobra una ficha, lo que representa un número 1. En este momento llevamos el número 5 **1**01

5. Acarreamos todos los pares de fichas que hayan en la próxima posición, en este caso 2 y por cada par, sumamos un 1 a la izquierda. Esta vez sobra una ficha, lo que representa un número 1. En este momento llevamos el número 2 **1**101

6. Acarreamos todos los pares de fichas que hayan en la próxima posición, en este caso 1 y por cada par, sumamos un 1 a la izquierda. Esta vez no nos sobra una ficha, lo que representa un número 0. En este momento llevamos el número 1 **0**1101.

7. Cómo el número de más a la izquierda es un 1 no tenemos nada más que acarrear y el resultado es 101101

Algoritmo de divisiones sucesivas

Este el algoritmo es equivalente al anterior, pero permite mecanizar de mejor forma la conversión, sin tener que pensar en el acarreo. Consiste en los siguientes pasos:

- Dividir el número actual por 2: si la división es exacta (es decir, no hay resto), agregar un 0 a la izquierda del resultado. Actualizar el número actual como el resultado de la división.
- Dividir el número actual por 2: si la división no es exacta (es decir, hay resto), agregar un 1 a la izquierda del resultado. Actualizar el número actual como el resultado de la división.
- Detenerse si es que el número actual es 0.

Veamos el algoritmo a través de un ejemplo: vamos a convertir el número 45 a representación binaria.

1. Dividimos el número actual 45 en dos = 22, resto = 1. Como hay resto, agregamos un 1 a la izquierda del resultado=**1**.

2. Dividimos el número actual 22 en dos = 11, resto = 0. Como no hay resto, agregamos un 0 a la izquierda del resultado=**0**1.

3. Dividimos el número actual 11 en dos = 5, resto = 1. Como hay resto, agregamos un 1 a la izquierda del resultado=**1**01.

4. Dividimos el número actual 5 en dos = 2, resto = 1. Como hay resto, agregamos un 1 a la izquierda del resultado=**1**101.

5. Dividimos el número actual 2 en dos = 1, resto = 0. Como no hay resto, agregamos un 0 a la izquierda del resultado=**0**1101.

6. Dividimos el número actual 1 en dos = 0, resto = 1. Como hay resto, agregamos un 1 a la izquierda del resultado=**1**01101.

7. Nos detemos por que el resultado actual es 0.

2.4.3. Algoritmos de conversión entre representaciones hexadecimal, octal y binaria

La conversión entre representaciones binaria y octal y binaria y hexadecimal es mucho más simple que la conversión entre binarios y decimales. Esto se debe a la relación de sus bases: ocho y dieciséis son potencias de dos. De esta forma, para convertir un número binario a octal, basta ir agrupando de a **tres** ($8 = 2^{\mathbf{3}}$) dígitos binarios e ir reemplazando su valor por el número octal correspondiente. A modo de ejemplo, realicemos la conversión del número $(10110)_2$ a octal:

binario:	010 110
octal:	2 6

Se observa además que el método funciona en ambas direcciones: para convertir el número octal $(26)_8$ en binario basta tomar cada uno de sus dígitos, representarlo como número binario y luego ubicarlos sucesivamente: $(2)_8 = (010)_2$ y $(6)_8 = (110)_2$ por lo tanto $(26)_8 = (10110)_2$

De manera equivalente, la conversión binaria-hexadecimal consiste en ir agrupando de a **cuatro** $(16 = 2^{\mathbf{4}})$ dígitos binarios e ir reemplazando su valor por el número hexadecimal correspondiente. A modo de ejemplo, realicemos la conversión del número $(10110)_2$ a hexadecimal:

binario:	0001 0110
hexadecimal:	1 6

Al igual que con los octales, la conversión es bidireccional: para convertir el número hexadecimal $(16)_{16}$ en binario basta tomar cada uno de sus dígitos, representarlo como número binario y luego ubicarlos sucesivamente: $(1)_{16} = (0001)_2$ y $(6)_{16} = (0110)_2$ por lo tanto $(16)_{16} = (10110)_2$.

Debido a esta fácil conversión entre representaciones, es habitual que se ocupen tanto la representación octal como la hexadecimal para referirse a números binarios, ya que requieren menos símbolos.

3. Representación de números negativos

Hasta ahora hemos visto representaciones sólo de números enteros y positivos. La primera pregunta que surge es ¿cómo representamos números negativos? Nuestra experiencia nos dice que la respuesta es simple: agregamos un «-» a la izquierda del número, por ejemplo -12 sería un número decimal negativo; $(-1100)_2$ sería un número binario negativo.

El problema de esto es que necesitamos agregar un nuevo símbolo a nuestro sistema numérico para poder representar números negativos. En el caso de los números binarios, en vez de tener sólo dos símbolos, necesitaríamos tener uno adicional sólo para indicar que un número es negativo. Nos gustaría buscar un mecanismo que aproveche los símbolos que ya tenemos en nuestra representación, y así evitar tener que agregar un símbolo nuevo.

A continuación se revisarán distintos métodos para representar números negativos binarios ocupando sólo los símbolos 0 y 1.

Dígito de signo

Un primer método que se puede ocupar es agregar un dígito extra a la izquierda del número que en caso de ser 0 indica que el número es positivo, y en caso de ser 1, que el número es negativo. Por ejemplo si tenemos el número 10011 y queremos representar el número -10011, reemplazamos el «-» por un 1, obteniendo: **1**10011.

Importante: al ocupar este tipo de representaciones ya no basta sólo con ver la secuencia de símbolos para saber su valor, debemos saber además que **tipo** de información se está guardando, por ejemplo en este caso un número entero, tal que el primer dígito indica el signo.

El problema de esta representación es que no nos sirve para realizar operaciones aritméticas. Para que una representación de un número negativo sirva debe cumplir la ecuación: $A + (negativo(A)) = 0$ es decir, debe ser un inverso aditivo válido.

En nuestro caso, si al número positivo **0**10011 le sumamos el número negativo **1**10011 obtenemos 1000110. Debemos entonces buscar otras representaciones para poder realizar operaciones aritméticas con números negativos.

Complemento a 1

Una posible mejora corresponde la representación denominada «complemento a 1». Esta consiste en agregar un dígito de signo, al igual que en el caso anterior, y luego, para obtener el negativo de un número, reemplazar todos los 0s por 1s y los 1s por 0s. Siguiendo nuestro ejemplo con el número 10011, primero se le agrega un cero a la izquierda: **0**10011, lo que se interpreta como «agregar el bit de signo al número positivo». Luego se hace la inversión para todos los dígitos **1**01100, lo que nos deja un número «negativo» dado que su bit de signo es 1.

Ahora si realizamos la suma entre **0**10011 y **1**01100 obtenemos **1**11111. Si hacemos lo mismo con el número 101 por ejemplo, obtendremos que el número negativo sería **1**010 y si los sumamos, obtenemos **1**111.

Podemos observar que aparece un patrón común al ocupar complemento a 1: la suma entre un número y su inverso aditivo resultará en una secuencia de números uno. Esto nos indica que estamos por buen camino, y nos falta sólo un detalle para llegar a una representación más adecuada.

Complemento a 2

La representación que soluciona el problema del complemento a 1 se denomina «complemento a 2». Consiste en ejecutar los mismos pasos que en complemento a 1, y adicionalmente sumarle 1 al número obtenido. Veamos paso a paso con un ejemplo, el número 10011:

- El primer paso consiste en agregar el cero a la izquierda: **0**10011
- Luego se realiza el reemplazo de 0s y 1s: **1**01100
- Por último, le sumamos un 1 al número: **1**01101

Ahora probaremos si efectivamente esta representación nos entrega un inverso aditivo válido. Si sumamos **0**10011 con **1**01101 obtenemos: **1**000000, lo que podemos interpretar como «0», ya que el 1 de más a la izquierda lo podemos interpretar como el signo, y sabemos que $0 = -0$ por lo tanto se cumple que la representación en complemento a 2 si funciona como inverso aditivo.

De ahora en adelante, cuando hablemos de un número binario negativo, asumiremos que está representado en complemento a 2, a menos que se diga lo contrario.

Referencias

- Morris Mano, M.; Computer System Architecture, 3 Ed., Prentice Hall, 1992. Capítulo 3: Representación de datos.