Introducción a la Computación

Contenidos

Introducción a la Computación....1
Programa de la materia....3
Unidad I: La información....3
Unidad II: Arquitectura y organización de computadoras....3
Unidad III: Sistemas operativos....3
Unidad IV: Transmisión de datos....4
La información....5
Datos e información....5
Procesamiento de la información....6
Tratamiento automático de la información....6
Perspectiva histórica....7
Representación analógica y digital....8
La información como objeto matemático....9
....11
Cantidad de información....11
Información y computadoras....12
Sistemas de numeración....15
Sistemas posicionales....15
Sistema binario....19
Sistema hexadecimal....19
Conversión de sistema decimal a otras bases....19
Conversión de decimal a binario....20
Conversión de decimal a hexadecimal....21
Trucos para cálculo rápido....22
Conversión de otras bases a sistema decimal....23
Conversión entre sistemas binario y hexadecimal....24
Sistema octal....24
Codificación de datos....25
Codificación de texto....25
Codificación de multimedia....27
Imagen....27
Sonido....30
Archivos....31
Múltiplos del bit y del byte....32
Sistema Internacional (SI)....32
Prefijos Binarios....32
Compresión....33
Compresión sin pérdida....34
Compresión con pérdida....35
Representación de datos numéricos....37
Datos y tipos de datos....37
Rango de representación....37
Representación de enteros con signo....39
Signo-magnitud....39
Complemento a 2....39
Aritmética en complemento a 2....45
Overflow....45
Representación de números reales....47
Notación de punto fijo....47
Notación científica....48
Notación en punto flotante....49
Representación de un decimal en punto flotante....50
Conversión de punto flotante a expresión decimal....51
Modelo Computacional Binario Elemental....53

Unidades funcionales del MCBE....54
Descripción detallada del MCBE....55
Ciclo de instrucción....58
Detalles operativos del MCBE....59
Diagrama estructural del MCBE....60
Conjunto de instrucciones....61
Ejemplos de programas MCBE....61

Programa de la materia

Unidad I: La información

1. Concepto de Información
2. Procesamiento de la información: evolución histórica
3. Sistemas de numeración:
   3.1. Sistemas posicionales y no posicionales
   3.2. Sistema decimal
   3.3. Sistema binario
   3.4. Sistema hexadecimal
4. Unidades de información
   4.1. bits, bytes, nibbles, palabras
   4.2. Múltiplos y submúltiplos

Unidad II: Arquitectura y organización de computadoras

1. Modelo computacional binario elemental
   1.1. Memoria
   1.2. CPU
   1.3. Conjunto de instrucciones
   1.4. Ciclo de instrucción
2. Representación e interpretación de la información
   2.1. Instrucciones, datos, números, caracteres, ASCII, multimedia, etc.
3. Programa almacenado
4. Lenguajes
   4.1. Lenguaje de máquina
   4.2. Lenguaje de alto nivel
   4.3. Intérpretación
   4.4. Compilación

Unidad III: Sistemas operativos

1. Modelo de Sistema de Computación
   1.1. Usuarios
   1.2. Programas de aplicación
   1.3. Sistema operativo
   1.4. Hardware
2. Definición de sistema operativo
   2.1. Por lo que es
   2.2. Por lo que hace
3. Componentes del SO
   3.1. Kernel
   3.2. Shell
   3.3. Otros componentes
   3.4. Procesos y recursos
4. Funciones de un SO
   4.1. Planificación de procesos
   4.2. Planificacion de recursos (memoria, sistema de archivos, almacenamiento secundario)
   4.3. Protección y control
   4.4. Contabilidad

5. Evolución histórica de los SO
   5.1. Desde máquina pelada hasta sistema embebido
   5.2. Monoprogramación y multiprogramación
   5.3. Batch e interactivo
6. Servicios de un SO
   6.1. Ejecución de programas
   6.2. Operaciones de E/S
   6.3. Manejo del file system
   6.4. Detección de errores
   6.5. Comunicaciones
   6.6. System calls, traps

Unidad IV: Transmisión de datos

La información

No es ningún secreto que la vida moderna nos da muchas herramientas para hacer las cosas más rápidas, o más fáciles; o para hacer cosas que hace tiempo no se hubieran creído posibles. A través de Internet podemos manejar nuestras cuentas bancarias, consultar los diarios, mirar videos, estudiar, comunicarnos con nuestros amigos y miles de otras cosas. Sacamos fotos con celulares o con cámaras digitales, nos comunicamos con teléfonos móviles, leemos libros electrónicos, jugamos o escuchamos música con dispositivos portátiles. Hasta los autos y los lavarropas tienen una determinada inteligencia para hacer las cosas.

En todos estos casos, hay algo en común: en todas estas situaciones, existe esencialmente algún **dispositivo** que **procesa información**.

Diariamente utilizamos información en casi todo lo que hacemos. La necesitamos para poder desarrollar nuestra vida diaria. En esta materia nos vamos a ocupar de conocer, entre otras cosas, qué es la información, cómo es que se procesa la información por medios automáticos y cómo están construidos los dispositivos capaces de hacerlo.

Datos e información

Cuando Alicia dice “19/3/95”, nos está comunicando un **dato**. Cuando además nos dice que ese dato es su fecha de nacimiento, nos está dando **información**. La información se compone de datos, pero puestos en contexto de manera que sean relevantes, es decir, importantes, y modifiquen de alguna manera nuestra visión del mundo.

Esta información puede ser producida, almacenada, recuperada, comunicada, procesada, de varias maneras.

5. La información se **produce** a raíz de algún **evento** importante, a partir del cual podemos **tomar alguna decisión**.
    1. Evento: las tostadas saltan de la tostadora. Información: las tostadas están listas. Decisión: sacarlas de la tostadora y comerlas.
    2. Evento: el semáforo para peatones cambia a “caminar”. Información: me toca el paso. Decisión: avanzar y cruzar la calle.
    3. Evento: Alicia nos dice "hoy cumplo 18 años". Información: es el cumple de Alicia. Decisión: felicitarla, y quizás hacerle un regalo.
6. Cuando la información se **almacena**, es porque queda escrita o registrada de alguna forma en un **soporte** o medio de almacenamiento. Puede tratarse de un papel escrito con la lista de las compras, una pantalla que muestra horarios de llegada de aviones, o un pendrive con canciones.
7. La información puede ser **comunicada**. Comunicamos información al recitarle la lista de las compras al almacenero; al decirle por teléfono a alguien a qué hora vamos a llegar mientras miramos la pantalla de los horarios; al escuchar las canciones almacenadas en el pendrive.
8. A veces, para poder comunicar información hace falta **copiarla** de un soporte a otro. Ocurre esto al pasar en limpio la lista de compras en otro papel; o al ingresarla en la computadora, o al sacarle una foto.

¿En qué otras situaciones se produce información? Describa situaciones de la vida diaria donde la información se almacene. ¿Qué ejemplos de soportes de información conoce? ¿Se puede perder la información? ¿Se puede destruir un soporte de información y sin embargo conservarse la información? ¿Puede existir información sin un soporte? ¿Qué otras situaciones de comunicación de información puede describir?

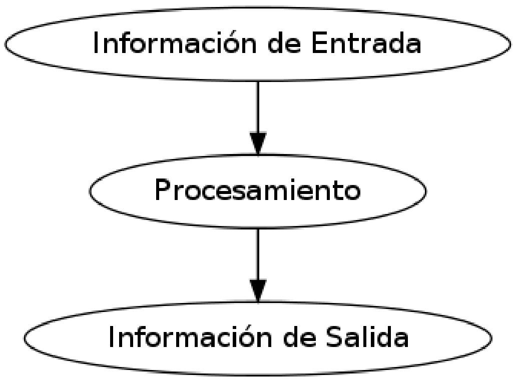
Procesamiento de la información

Cuando decimos que la información es **procesada**, queremos decir que hay alguien o algo que modifica esa información, presentándola de otra manera o combinándola con alguna otra pieza de información. Ese alguien o algo es un **agente** (agente quiere decir "que realiza una acción") de procesamiento de información. Todos procesamos información continuamente, durante todo el día, para varios fines.

El procedimiento exacto que se necesita para modificar la información depende de cada problema, de la forma como está representada la información, y de la forma como se desea obtener al final. Para lograr esa transformación se requiere utilizar la información tal como está representada, y operar con ella para crear una representación diferente.

Ese procedimiento necesario, en teoría, puede ser llevado a cabo por una o varias personas. Es lo que hacen, por ejemplo, los docentes en los colegios cuando toman la planilla de notas y calculan la nota final para cada alumno; los empleados contables, cuando toman el registro de asistencias e inasistencias de un empleado junto con su legajo y calculan el monto de lo que debe cobrar; el médico, cuando recibe el análisis de sangre de un paciente y combina esos datos con sus propias observaciones del paciente para formular un diagnóstico; el sistema de navegación automática de un avión, cuando analiza los datos de su posición y velocidad, y calcula la corrección necesaria para mantener el rumbo.

Un agente de procesamiento de información, primero que nada, debe **recuperar** esta información, es decir, leerla del soporte donde está almacenada. La información de la cual se parte para resolver un problema se llama información de entrada, o simplemente **entrada**; y la que produce el agente de procesamiento, información de salida, o simplemente **salida**.



Tratamiento automático de la información

La información es tan importante para nuestras actividades, que resulta interesante comprender y aplicar las formas de procesarla en forma **automática**, es decir, usando máquinas. Las ventajas son, por supuesto, que es posible construir máquinas que ejecuten este procesamiento sin cansarse, sin equivocarse, y en muchísimo menos tiempo. Y como consecuencia de todo esto, a mucho menor costo.

En muchos casos, el procesamiento de información podría hacerse a mano. Lamentablemente, algunos casos de procesamiento de información son tan complicados, o llevan tanto trabajo, que a mano resultan impracticables. En caso de realizarlos a mano, con lápiz y papel, el resultado tardaría tanto en estar disponible que directamente no serviría para nada.

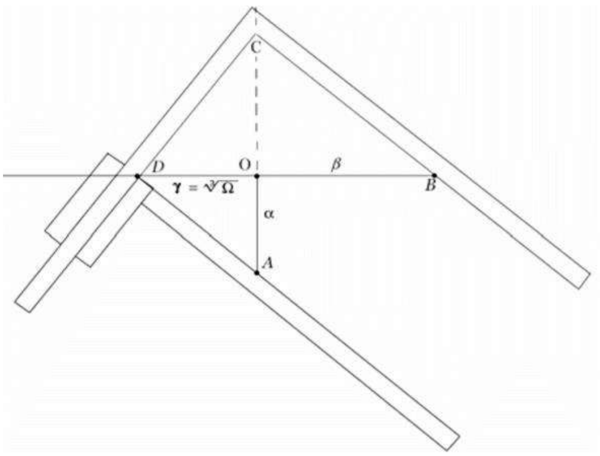
Ejemplo

¿De dónde sale la información que nos da la señorita del pronóstico del tiempo por la televisión? ¿Cómo se computa esta información? ¿Cuánto tiempo le llevaría a una persona realizar esos cálculos? ¿Y a más de una persona?

Perspectiva histórica

En realidad, el procesamiento de información viene de muy antiguo. Todos los pueblos que han desarrollado sistemas de escritura, con alfabetos u otros sistemas de signos, han sido capaces de almacenar información. Inclusive podemos decir que es antiguo el tratamiento de la información por medios más o menos **automáticos**, y más o menos mecánicos. El *ábaco*[1] es un mecanismo de cómputo antiquísimo, extremadamente simple y maravillosamente efectivo, que aún se enseña en las escuelas de algunos países. Los *quipus*[2] de los incas fueron una forma de almacenamiento de información. Los pueblos antiguos diseñaron máquinas, a veces sumamente complicadas, para ayudarse en el diseño de armas[3], o para completar cálculos astronómicos[4].

Extractor mecánico de raíces cúbicas, inventado por un geómetra griego anónimo de los siglos III o IV. Se utilizaba para calcular el diámetro de las cuerdas de una catapulta.



1 http://es.wikipedia.org/wiki/Ábaco
2 http://es.wikipedia.org/wiki/Quipus
3 http://www.mlahanas.de/Greeks/war/Catapults.htm
4 http://es.wikipedia.org/wiki/Mecanismo_de_Anticitera

—Los alejandrinos han obtenido una fórmula —dijo Arquímedes, satisfecho—. Es probable que tú no la conozcas porque todavía es nueva, pero se han hecho muchas pruebas con ella, y funciona. Se toma el peso que debe ser lanzado y se multiplica por cien, luego se calcula la raíz cúbica, se le suma un décimo, y de ese modo se obtiene el diámetro del calibre en ancho de dedos.
Eudaimon se burló.
—¿Y qué es una raíz cúbica, en nombre de todos los dioses? —preguntó.
Arquímedes lo observó, demasiado asombrado para poder hablar. «La solución al problema délico —pensó—, la piedra angular de la arquitectura, el secreto de la dimensión, la diversión de los dioses.» ¿Cómo era posible que alguien que fabricaba catapultas no supiese lo que era una raíz cúbica?
Eudaimon lo miró con desagrado. Luego arrugó el papiro con furia, simuló limpiarse el trasero con él y lo arrojó al suelo.

Fragmento de *El Contador de Arena*, de Gillian Bradshaw.

Lo que nos diferencia de estos pueblos antiguos es que con la tecnología moderna podemos tratar cantidades muchísimo más grandes de información, en tiempos muchísimo más reducidos, y con formas de procesamiento muchísimo más complicadas, usando, claro está, computadoras.

Representación analógica y digital

Esos dispositivos de la antigüedad tienen además una diferencia importante con las computadoras modernas. Las computadoras que usamos hoy sólo pueden tratar la información cuando está expresada, o representada, mediante **valores discretos**, es decir, **numerables**[5]**.** Por esto se dice que las computadoras modernas son **sistemas digitales** (es decir, basados en dígitos, o números).

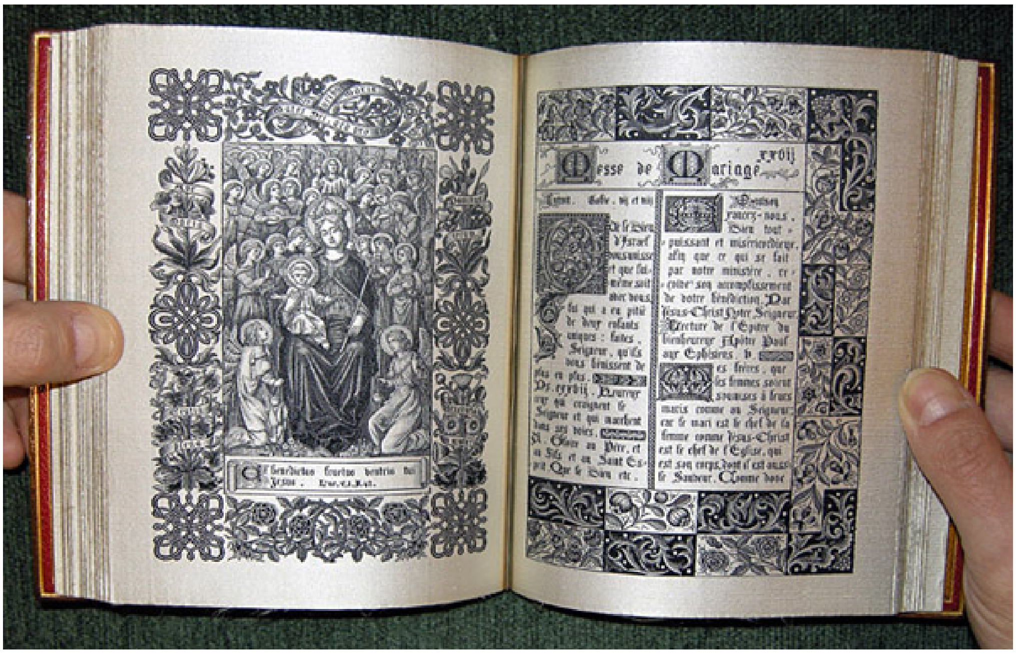
Contrariamente, algunos de los dispositivos antiguos que hemos visto, como el extractor de raíces cúbicas, son una forma de **computadoras analógicas**. Los sistemas analógicos tratan con la información representada como un conjunto de **valores continuos**. Las variables físicas (peso, temperaturas, distancias, tiempos) suelen tomar valores continuos: un objeto puede pesar 2Kg, 3 Kg, o cualquier valor intermedio. En cambio, la cantidad de objetos es un valor digital.

En una computadora digital no pueden representarse todas las cantidades que pueden tomar las variables analógicas. Si bien en Matemática tenemos perfectamente definido el valor de "raíz de 2" (que es *el número que, elevado al cuadrado, da 2*), en computación no podemos representarlo con toda precisión. En un sistema digital sólo podemos trabajar con **aproximaciones** a $\sqrt{2}$, porque al ser un irracional **tiene infinitos decimales**. Lo mismo pasa con cualquier otro número irracional (como **Pi**), y con muchos números racionales. Y además, tenemos limitaciones para trabajar con enteros demasiado grandes, o demasiado pequeños. Todo esto se debe a que los **recursos físicos** de una computadora digital para almacenar dígitos de información siempre son limitados.

La idea de representar **digitalmente** la información, de todos modos, no es algo exclusivo de nuestros días. A principios del siglo XIX ya existía el telar de Jacquard[6], ingenioso dispositivo para tejer telas usando tarjetas perforadas. Estas tarjetas eran una forma **digital** de

5 http://es.wikipedia.org/wiki/Variable_discreta_y_variable_continua
6 http://es.wikipedia.org/wiki/Telar_de_Jacquard

representación de información; en este caso, de patrones de diseño bidimensionales. Como suele ocurrir en ciencia y tecnología, este aparato inspiró a otros, entre ellos a Charles Babbage[7], quien ideó lo que se considera la primera computadora, y a Herman Hollerith[8], quien desarrolló una computadora digital electromecánica que sirvió para automatizar los primeros censos de la inmigración europea hacia América del Norte. Estas máquinas son una forma de primitivas computadoras digitales de **programa almacenado**.



"Libro de Plegarias" impreso en 1886 sobre seda, utilizando el telar de Jacquard con tarjetas perforadas. ¡Un antecesor del libro electrónico!

La información como objeto matemático

No hay una única definición de información, ya que cada ciencia o disciplina utiliza el concepto en forma diferente. Un epistemólogo define la información como "una diferencia que importa[9]". Wikipedia dice que la información es "una secuencia de símbolos que puede ser interpretada como un mensaje[10]". Si preguntamos a una persona cualquiera qué significa para ella información, es posible que lo primero que se le venga a la cabeza sean las noticias del diario o de la TV.

Aunque todo lo anterior resulta correcto en su contexto, nosotros hablaremos de información en un sentido más preciso. Esto implica considerar la información desde un punto de vista matemático.

Siempre que los matemáticos consideran cosas del mundo real para trabajar, tratan de quedarse con lo esencial de esas cosas, aquello que puede ser reducido a cantidades y propiedades esenciales. Este procedimiento mental se llama **abstracción**. Cuando pensamos en la información en forma abstracta, no nos importa cómo está construido el mensaje (la **sintaxis**) ni lo que quiere decir (la **semántica**). Al abstraer datos, símbolos y mensajes, dejamos afuera su forma y su significado, sacando todo lo que sobra, y nos queda... *la información*. Ahora esta información puede ser medida y reducida a unidades.

Para ejemplificar, volvamos al episodio de la tostadora que eyecta sus tostadas listas. Ésta era

7 http://es.wikipedia.org/wiki/Charles_Babbage
8 http://es.wikipedia.org/wiki/Herman_Hollerith
9 http://es.wikipedia.org/wiki/Gregory_Bateson
10 http://es.wikipedia.org/wiki/Información

una situación de producción de información. Si comenzamos a abstraer propiedades no esenciales de esa situación, ¿qué nos queda? Para el matemático no es importante que se trate de una tostadora, ni de tostadas, ni que el evento sea exactamente la eyección de una tostada. Lo importante es que *antes no había ocurrido el evento, y ahora sí*. En el caso del semáforo, no es importante que se trate de un semáforo, ni la altura a la cual está puesto, ni siquiera el color. Lo importante es que se ha producido el evento. Antes no había ocurrido el evento, y ahora sí. Antes, el símbolo era uno, y ahora es otro. Esta diferencia en el tiempo es lo que genera la información, no la tostadora, ni la tostada, ni el semáforo.

Si alguien no pudiera ver la tostadora, o el semáforo, y necesitara tomar una decisión, podría preguntar a alguien más si el evento ya ha ocurrido, y ese alguien más podría responderle **"sí" o "no"**. Esa persona le estaría comunicando la información necesaria para tomar la decisión. Ésta es la "diferencia que importa" y es la mínima unidad de información que se puede comunicar.

Las preguntas "de sí o no" se llaman preguntas **binarias**, porque su respuesta tiene dos posibilidades. Esa respuesta permite tomar una decisión entre dos posibles acciones (retirar la tostada/no retirar la tostada; avanzar/no avanzar). No hay una decisión más simple, ¡porque no se puede tomar una decisión entre **menos** posibilidades!

La cantidad de información que es aportada por la respuesta a una pregunta binaria recibe el nombre de **bit**. Un bit es, en definitiva, una pieza de información que puede tomar uno de dos valores. Esos dos valores suelen representarse con **0 y 1**, que como veremos más adelante, son los dos dígitos del **sistema de numeración binario**. El dígito binario, o, abreviadamente, "bit", es la **mínima unidad de información** en Informática. Al haber identificado esa unidad mínima de información, ahora podemos medir la "cantidad de información" de cualquier pieza de información.

Si quisiéramos diseñar un artefacto de comunicación de información muy simple, podríamos hacerlo con una luz eléctrica provista de un interruptor. Cuando la persona interesada en comer la tostada viera la luz encendida, recibiría la información de que la tostada está lista. Nuestro dispositivo de comunicación habría acabado de comunicar **un bit de información**.

En una escena de una película muy popular, de no hace mucho tiempo, hay un dispositivo similar para comunicar el inicio de una guerra... ¿Recuerdan la escena? ¿De cuál película estamos hablando?

¿Cómo se analiza esta escena en términos de información? ¿Qué operaciones tienen lugar sobre la información durante toda la secuencia? ¿Cuánta información se transmite? ¿Quiénes son los agentes de procesamiento?

Cantidad de información

El semáforo para peatones del ejemplo comunica un bit de información, porque nos permite decidir entre avanzar y no avanzar. Uno compuesto, como los semáforos para automovilistas que indican permiso de avanzar y permiso de giro, nos permite decidir entre más posibilidades. Representemos esas posibilidades:

Semáforo de avanzar	Semáforo de giro a izquierda	Interpretación
Verde	Verde	Puedo avanzar o girar
Verde	Rojo	Puedo avanzar pero no girar
Rojo	Verde	No puedo avanzar pero sí girar
Rojo	Rojo	No puedo ni avanzar ni girar

Los dos elementos del semáforo compuesto, independientes pero combinados, suman **dos bits** y nos dan la decisión entre **cuatro** posibilidades. Sin embargo, tomados individualmente, cada elemento del semáforo sigue comunicando un bit de información. Cada vez que agrego un bit de información, **multiplico por dos** las posibles decisiones.

¿Cuántas situaciones posibles nos da un semáforo que tiene **tres elementos**: luz de avanzar, luz de giro a izquierda y luz de giro a derecha? Digamos que la luz de giro a la izquierda puede estar encendida o apagada, y que es el primer elemento del semáforo. Lo mismo la luz de avanzar, y supongamos que es el segundo elemento. Lo mismo para el tercer elemento, que es la luz de giro a la derecha. Utilicemos una notación muy resumida para especificar este semáforo de tres elementos. Escribamos una luz encendida como “1” y una luz apagada como “0”. No es coincidencia que elijamos precisamente esos símbolos, que son los dígitos del **sistema binario** de numeración.

Entonces, un estado posible del semáforo sería **011**, que quiere decir: **no puedo girar a la izquierda, pero sí puedo avanzar, o girar a la derecha**. Si escribimos todas las combinaciones posibles de tres símbolos 0 y 1 tendremos las ocho posibilidades:

000

001

010

011

100

101

110

111

Como puede verse, hay una relación muy directa entre la cantidad de posibilidades para un evento, la cantidad de información que necesitamos para describir completamente ese evento (es decir, para representar todas sus posibilidades), y los números binarios de una cierta cantidad de dígitos.

5. ¿Cuántos bits necesitamos para representar un evento que tiene **16** posibilidades?
6. ¿Cuál es la fórmula general para conocer la cantidad de bits (o de dígitos binarios) necesarios para describir un evento de **n** posibilidades? Si numeramos las posibilidades de ese evento con números decimales, comenzando desde 0, ¿qué número le corresponde a la última posibilidad?
7. ¿Cuántas posibilidades puedo representar con un conjunto de **ocho bits**?
8. Ana piensa un número entre **0** y un número máximo conocido, **M**, y Belén tiene que adivinarlo. A Belén se le permite hacer la pregunta "**¿Es mayor que X?**" para cualquier X que Belén elija, y Ana contesta correctamente. ¿Cuáles deben ser esas preguntas? ¿Cuánta información le da a Belén cada respuesta? Si se sabe que el número pensado por Ana está entre 0 y 7 inclusive, ¿en cuántas preguntas puede Belén adivinar el número pensado? Dicho en otras palabras, ¿cuántos bits de información necesita Belén? ¿Y si el número pensado por Ana está entre 0 y 15? ¿Y entre 0 y 255? ¿Y entre 0 y 1023?

Información y computadoras

Si bien el tratamiento de la información tiene una existencia independiente de la computación, hoy prácticamente toda la información se trata mediante **computadoras**. El Diccionario de la Lengua de la Real Academia Española dice que el término **Informática** viene del francés y designa el "conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento automático de la información por medio de ordenadores"[11].

Sin embargo, hay que tener presente que la información y su tratamiento presentan muchos problemas interesantes para la ciencia aun sin necesidad de pensar en utilizar computadoras, o

11 http://lema.rae.es/drae/?val=inform%C3%A1tica

aunque las computadoras nunca se hubieran inventado. Es decir: **Informática** y **Computación** son dos disciplinas diferentes, aunque estrechamente relacionadas.

Los circuitos de las computadoras electrónicas son más fáciles de diseñar y construir, más económicos, y más útiles, cuando funcionan en base a sólo dos clases de señales o **estados**. Los microscópicos circuitos de las computadoras digitales modernas están en uno de dos posibles estados: activo o inactivo. Por este motivo se llaman **biestables**. Un elemento biestable de una computadora está **activo** cuando por él circula una determinada corriente. En esta condición, o estado, el biestable representa un 1. Cuando el mismo elemento está inactivo, representa un 0. De esta forma, las computadoras son dispositivos construidos de modo de poder manipular bits.

¿Por qué las computadoras son capaces de procesar tantas clases diferentes de información? Para poder ser almacenada, recuperada, procesada o comunicada por una computadora, la información debe estar representada por bits. Al ser capaces de tratar con bits y bytes, las computadoras pueden procesar cualquier clase de información, porque cualquier pieza de información puede descomponerse en partes y ser representada por bits. A la hora de representar datos de la vida real, para tratarlos por medios automáticos (es decir, al momento de procesar información con computadoras), esos datos pueden ser traducidos a bits de muchas maneras. Una forma natural de representar los números enteros, por ejemplo, es utilizar el sistema binario de numeración. Esta representación se traduce a bits de manera inmediata: un 0 o 1 binario será un bit con estado respectivamente inactivo, o activo, en la circuitería de la computadora.

Sistemas de numeración

Para representar números, habitualmente utilizamos el sistema de numeración decimal, que tiene diez **dígitos** (llamados así porque representan a los diez dedos con los que contamos). Como todos sabemos, éstos son el 0, el 1, el 2, el 3, el 4, el 5, el 6, el 7, el 8 y el 9.

¿Cómo podemos contar objetos en cantidades mayores que diez, con solamente diez dígitos? El truco es crear un sistema **posicional**, donde los dígitos tienen diferente valor según en qué lugar del número se encuentran.

Contando con el cero y los naturales

Asignamos dígitos a las cosas que queremos contar, y cuando se agota la secuencia de los dígitos, agregamos un nuevo dígito **1** a la izquierda para representar el hecho de que se nos acabó **una vez** la secuencia, y volvemos a empezar:

	0...	Cero...
	1...	uno...
	...	(varios dígitos...)
	7...	siete...
	8...	ocho...
	9...	nueve... ¡se acabaron los dígitos!
No importa, tomemos un nuevo dígito para la posición siguiente a la izquierda...	**1 0...**	y repetimos la misma secuencia de diez dígitos a la derecha volviendo a empezar... diez...
	1 1...	once...
	1 2...	doce...
	...	y seguimos...
	1 8...	dieciocho...
	1 9...	diecinueve... ¡se acabaron 2 veces los dígitos!
Tomemos el siguiente dígito en la secuencia para la posición de la izquierda...	**2 0...**	y repetimos la secuencia a la derecha de nuevo...
y sigamos contando...	**2 1...**	veintiuno...
	2 2...	veintidós...

¿Qué pasa cuando se agota la secuencia en la posición de la izquierda? Es decir, ¿qué pasa cuando llegamos al **99**? ¿Y al **999**? Tenemos que tomar todavía una posición más a la izquierda y volver a empezar la secuencia con todas las posiciones de la derecha (luego del **99** escribimos **100**). Esto ocurre en una posición diferente, cada vez más a la izquierda, cada vez que agotamos la secuencia.

Como vemos, este procedimiento puede seguir indefinidamente, permitiendo escribir números de cualquier cantidad de dígitos.

Sistemas posicionales

Nuestro sistema es posicional: cuando escribimos un número, el **valor absoluto** de cada dígito será siempre el mismo, pero su significado o **valor relativo** depende de la posición donde se encuentra. El **2** es un dígito cuyo valor absoluto es, claro, **2**. Pero el dígito **2** de la línea donde contamos **veintiuno (21)** no tiene el mismo valor relativo que el **2** de la línea **doce (12)**. Los símbolos que se encuentran más a la izquierda tienen mayor valor relativo: el **2** de **21** vale

veinte, o sea **diez veces más** que el **2** de **12**, porque representa el hecho de que la secuencia de **diez** dígitos se agotó dos veces.

En el número **21**, el **2** está desplazado a la izquierda una posición; por lo tanto, su valor se multiplica por **10**, valiendo **20**. En el número **215**, el **2** está desplazado a la izquierda dos posiciones; por lo tanto su valor se multiplica **dos veces** por **10** (o sea, $\mathbf{10 * 10 = 10^2 = 100}$), dando **200**.

La cantidad de dígitos de un sistema numérico se llama la **base** del sistema. En cualquier sistema posicional, cada vez que un dígito se desplaza a la izquierda una posición, para obtener su valor relativo hay que multiplicarlo por una potencia de la base del sistema (cualquiera sea dicha base). En este caso, la base es **10**, porque estamos utilizando el sistema decimal. Cuando escribimos **215**, en realidad estamos expresando su desarrollo como suma de potencias de la base:

$$\mathbf{2 * 10^2 + 1 * 10^1 + 5 * 10^0}.$$

Cuando escribimos el “número” **215**, lo que estamos escribiendo en realidad son los coeficientes de las potencias de la base en el desarrollo del número **215**.

Para que este desarrollo sea el correcto, las potencias de la base deben estar **ordenadas descendentemente** (es decir, de mayor exponente a menor exponente) y **completas** (sin que falte **ningún** exponente, **incluyendo el exponente 0**).

Preguntas

9. ¿Cuál es el desarrollo en potencias de 10 del número 1322? ¿Y del 10502?
10. ¿Qué potencias de la base utilizamos cuando escribimos números con coma (o punto) y parte fraccionaria? ¿Cómo son los exponentes de esas potencias?
11. ¿Cuál es el desarrollo en potencias de 10 del número 125.61?

Símbolos y significado de los dígitos

Ya que estamos, tratemos de aclarar lo siguiente, que es un poco difícil de explicar: los símbolos que escribimos, y que llamamos “números”, no son más que una representación de los verdaderos **números**. ¿Se entiende esto? A ver con ejemplos:

1. La figura siguiente no es música.

   Es solamente una notación para los sonidos musicales. La verdadera música aparece cuando alguien toca esta notación en un instrumento y nosotros la escuchamos, o la imaginamos en nuestra cabeza.

2. Éstas no son palabras: *CASA, SOL, PERRO*. Son una notación para las palabras. Las verdaderas palabras aparecen cuando alguien las dice, o cuando su significado está en nuestra cabeza.

3. ¡Ésta no es una pipa!

   Es una pintura que representa una pipa.

4. De la misma manera, **12, 21 y 3.1416**, ¡no son los verdaderos números, sino una notación para los números! Y esta notación se hace eligiendo una base, y esta base puede ser cualquiera. Dependiendo de la base elegida, los **símbolos** (los dígitos) para **representar** el mismo número serán unos u otros.

Así que podríamos elegir cualquier otra representación que quisiéramos para los números (igual que para las palabras, o para la música). Y como veremos, el mismo **número** podrá tener dos, o varias, **representaciones**.

Un sistema “frutal”

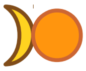
¿Cómo sería nuestro sistema de numeración si tuviéramos otra cantidad de dedos en nuestras manos? Depende de cuál fuera esa otra cantidad de dedos, pero muy probablemente seguiría siendo un sistema posicional. Y los dígitos, ¿necesariamente tienen que seguir siendo como los conocemos?

Imaginemos un sistema numérico con sólo dos símbolos, **naranja** y **banana**, sabiendo que **naranja** equivale a nuestro **0** y **banana** equivale a nuestro **1**. ¿Podremos usar estos símbolos para contar objetos?

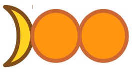
Escribamos una secuencia de números para contar en nuestro sistema *frutal* y anotemos la correspondencia con nuestro sistema habitual.

 **0**

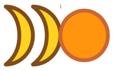
 **1** Se acabó la secuencia...

 **2** Entonces agregamos un 1 a la izquierda y volvemos a iniciar la secuencia.

 **3** ¡Se volvió a agotar la secuencia!

 **4** Tendríamos que cambiar el dígito de la izquierda por el siguiente en la secuencia... pero no hay más dígitos. Entonces tomamos una posición más a la izquierda. Agregamos un dígito **1** y volvemos a iniciar la secuencia en todas las demás posiciones.

 **5** Se agotó la secuencia en la posición de más a la derecha, paso al siguiente dígito en la posición siguiente.

 **6** Y vuelvo a empezar la secuencia.

Nuestra secuencia de símbolos ahora tiene sólo dos “dígitos”, en lugar de los diez del sistema habitual. Para contar hasta 6, los números con la secuencia de naranjas y bananas se forman exactamente como hicimos antes: mientras contamos objetos, escribimos los símbolos de la secuencia (pasos **0** y **1**). Cuando se agota la secuencia, tomamos el siguiente símbolo de la secuencia y lo colocamos a la izquierda, porque nuestro sistema es posicional. Siempre con ese símbolo a la izquierda, volvemos a repetir la secuencia de dos símbolos mientras seguimos contando (**2** y **3**). Se acabó de nuevo la secuencia, entonces tenemos que agregar un nuevo “dígito” a la izquierda (la banana que aparece a la izquierda, en el **4**).

En realidad, el sistema *frutal* no es otra cosa que el **sistema binario**, de dos dígitos, donde en lugar de **0 y 1** quisimos escribir **naranja y banana** para mostrar que lo importante de los dígitos no son los símbolos, sino su significado.

Preguntas

1. ¿Cómo se escriben los números hasta el 10 con este sistema?
2. Así como el **2** de **12** vale **2** pero el **2** de **21** vale **20**, las bananas y las naranjas tienen diferente valor relativo según en qué posición se encuentran. ¿Cuánto valen la banana de 1, la de 2 y la de 4? ¿Y cada una de las dos bananas de 5?
3. Un náufrago anota con marcas en la palmera de su isla los días que van pasando. Cada día hace una nueva rayita en la palmera. ¿Cuántos dígitos tiene el sistema numérico que utiliza? ¿Éste es un sistema posicional?
4. ¿Qué valores tienen, en el sistema decimal, los dígitos **naranja**, **banana** y **uva**, de un

Sistema binario

El sistema de numeración **binario** es un sistema posicional de base 2, donde sólo tenemos dos dígitos: 0 y 1. Escribamos algunos pocos números en el sistema binario.

Decimal	0	1	2	3	4	5	6	7	8	9	10
Binario	0	1	10	11	100	101	110	111	1000	1001	1010

Como vemos, los dos símbolos **0** y **1** del sistema **binario** son los mismos que el **0** y el **1**, símbolos que ya conocemos, del sistema **decimal**. Esta reutilización de los símbolos puede llevar a confusión: cuando escribimos **101**, ¿de qué número estamos hablando exctamente, del **101** o del **5**? En otras palabras, ¿en qué base está expresado el número?

Para evitar este problema, conviene aclarar en qué base estamos escribiendo los números, indicándola con un número subscripto. Por ejemplo, $\mathbf{101_{(2}}$ quiere decir que nos referimos al número 101 en base 2, que equivale al 5 en base 10 (o sea $\mathbf{101_{(2} = 5_{(10}}$). Si quisiéramos escribir, sin ambigüedades, el título de cierta película, escribiríamos “Los $\mathbf{101_{(10}}$ Dálmatas”.

Como ocurre con cualquier sistema posicional, cada número expresado en el sistema binario es en realidad el resultado de un desarrollo, o cuenta, donde utilizamos las potencias descendentes, ordenadas y completas, de la base para calcular el valor relativo de cada dígito. Así, por ejemplo,

$$\mathbf{101_{(2} = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 4 + 0 + 1 = 5_{(10}.}$$

Sistema hexadecimal

Así como hemos desarrollado un sistema numérico posicional con sólo dos dígitos, también es posible crear uno con **más dígitos** que en el sistema decimal. En el sistema **hexadecimal** tenemos **16** símbolos. Los primeros 10 símbolos se copian de los del sistema decimal (y valen lo mismo). La base del sistema es **16**, ¡así que nos faltan 6 símbolos! Pero, como hemos visto, los símbolos pueden elegirse a gusto, así que para el sistema hexadecimal se toman las letras **A** a la **F** como “dígitos” que toman los valores entre 10 y 15.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

El sistema hexadecimal es muy usado en computación porque aporta importantes ventajas: además de que la expresión de los números será en general más corta, resulta bastante más fácil convertir entre los sistemas binario y hexadecimal que entre binario y decimal.

Conversión de sistema decimal a otras bases

El procedimiento general para convertir un número expresado en sistema **decimal** a otra base, es **dividir** sucesivamente el número a convertir, y los sucesivos **cocientes**, por la base deseada. La expresión final se forma tomando **el último cociente y la sucesión de los restos en orden inverso**.

Un esquema útil

Para exponer nuestros cálculos en el presente material, nos resultará práctico construir un esquema como el siguiente:

Número	Cociente (base)	Resto

Los pasos para utilizar el esquema son:

1. En el primer renglón de la columna *Número* escribimos el número a convertir.
2. Dividimos *Número* por la base, llenando *Cociente* y *Resto*.
3. Copiamos el *Cociente* en la columna *Número*. Éste es nuestro nuevo *Número.*
4. Repetimos los pasos 2 y 3 hasta que ya no se pueda seguir dividiendo. Notemos tres cosas importantes:
    - Los *Restos*, por el algoritmo de división, deben necesariamente ser menores que el divisor (la base). De lo contrario, se podría seguir dividiendo.
    - Lo mismo ocurre con el último *Cociente.* De lo contrario, se podría seguir dividiendo.
    - Por lo dicho anteriormente, tanto los *Restos* como el último *Cociente*, al ser menores que la base, se pueden escribir con **un único dígito** en el sistema deseado.
5. Finalmente escribimos el último *Cociente* y los *Restos*, de abajo hacia arriba, en la base deseada, siguiendo las flechas. Ésta es la expresión que buscamos.

Ejemplo: Convertir **66** a base **3**.

Número	**Cociente (3)**	**Resto**
66	22	0
22	7	1
7	2	1

Finalmente, el número **66** expresado en base **3** se escribe **2110**$_{(3}$.

Conversión de decimal a binario

Como se explicó, el procedimiento es dividir sucesivamente el número a convertir por la base, que ahora será **2**. La expresión en el sistema binario se forma tomando el último *Cociente* y la sucesión de los *Restos* en orden inverso. Todos estos números serán menores que la base y por lo tanto son dígitos 0 o 1. Convirtamos, por ejemplo, **531**$_{(10}$ a sistema binario:

Número	Cociente (2)	Resto
531	265	1
265	132	1
132	66	0
66	33	0
33	16	1
16	8	0
8	4	0
4	2	0
2	1	0

Es decir, **531**$_{(10}$ **= 1000010011**$_{(2}$.

Truco rápido para la conversión (o la comprobación) en el caso particular en que la base es 2: notemos que el *Resto* es 1 cuando el *Número* es impar, y 0 cuando es par.

Conversión de decimal a hexadecimal

Nuevamente, el procedimiento es dividir sucesivamente el número a convertir por la base, que ahora es **16**. La expresión en hexadecimal se forma, como antes, tomando el último *Cociente* y la sucesión de los *Restos* en orden inverso[12]. Pero ahora tenemos que tener cuidado con cómo escribimos los sucesivos *Restos*: tienen que estar en el sistema **hexadecimal**.

Ejemplo

Convertir $\mathbf{531_{(10}}$ a hexadecimal:

Número	Cociente (16)	Resto
531	33	3
33	2	1

O sea, $\mathbf{531_{(10} = 213_{(16}}$.

Ahora veamos el caso donde algún resultado intermedio es mayor que 10.

Convertir $\mathbf{7158}_{(10}$ a hexadecimal:

Número	Cociente (16)	Resto
7158	447	6
447	27	15
27	1	11

Aquí hay que tener cuidado de expresar todos los restos en dígitos pertenecientes al sistema. Al construir **la representación en hexadecimal** de 7158, no puedo escribir los restos **15** u **11** (que están escritos en decimal) si no los expreso en hexadecimal. Se tiene que $\mathbf{11_{(10}} = \mathbf{B_{(16}}$ y que $\mathbf{15_{(10}} = \mathbf{F_{(16}}$. O sea, $\mathbf{7158_{(10} = 1BF6_{(16}}$.

Hay más detalles sobre conversiones numéricas en varias páginas online[13].

12 http://es.wikipedia.org/wiki/Sistema_hexadecimal
13 http://es.wikipedia.org/wiki/Sistema_binario#Conversi.C3.B3n_entre_binario_y_decimal

Trucos para cálculo rápido

1. Usando potencias

Un truco útil para convertir rápidamente números decimales (pequeños) al sistema binario es memorizar los valores de algunas potencias de 2 y utilizarlos en las cuentas. Por ejemplo:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	**64**	**32**	**16**	**8**	**4**	**2**	**1**

Entonces, si nos preguntan cómo se escribe en el sistema binario, por ejemplo, el número **$78_{(10}$**, sólo necesitamos ver de qué manera se descompone 78 como suma de estos valores. Hay una sola manera:

$78_{(10} = 64 + 8 + 4 + 2 = 2^6 + 2^3 + 2^2 + 2^1$

Si completamos las potencias:

$78_{(10} = 64 + 8 + 4 + 2 = 2^6 + 2^3 + 2^2 + 2^1 = 1 * 2^6 + 0 * 2^5 + 0 * 2^4 + 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0$.

Por lo tanto, cuando queramos escribir **$78_{(10}$** en binario, utilizaremos los coeficientes de la expresión que acabamos de escribir: **$78_{(10} = 1001110_{(2}$**.

Comprobemos nuestro resultado con el esquema:

Número	Cociente (2)	Resto
78	39	0
39	19	1
19	9	1
9	4	1
4	2	0
2	1	0

O sea, **$78_{(10} = 1001110_{(2}$**.

2. Usando desplazamientos y sumas

Sumar un número a sí mismo es equivalente a multiplicarlo por dos. Esto nos da una forma fácil de multiplicar por dos un número binario sin necesidad de utilizar la operación de multiplicación.

Otra forma de multiplicar un número binario por dos es desplazar los bits a la izquierda, completando con ceros. Para multiplicar por cuatro, desplazamos dos lugares; para multiplicar por ocho, desplazamos tres lugares, y así sucesivamente. Inversamente, si desplazamos a la derecha una posición (dos, o tres) esto equivale a dividir por dos (por cuatro, por ocho, respectivamente).

Ejemplo
$30_{(10} = 15 * 2 = 1111_{(2} * 2 = 11110_{(2}$
$60_{(10} = 15 * 4 = 1111_{(2} * 4 = 111100_{(2}$
$80_{(10} = 10 * 8 = 1010_{(2} * 4 = 1010000_{(2}$

Memorizando algunas pocas combinaciones de 0 y 1 y aplicando lo anterior, más algunas operaciones simples de suma, podemos escribir rápidamente números binarios. Por ejemplo:

- **14** = 7 * 2 = 7 desplazado a la izquierda un lugar = **1110**
- **28** = 7 * 4 = 7 desplazado a la izquierda dos lugares = **11100**
- **30** = 7 * 4 + 2 = 7 desplazado a la izquierda dos lugares, más dos = 11100 + 10 = **11110**
- **45** = 10 * 4 + 5 = 1010 * 4 + 101 = 101000 + 101 = **101101**

Cuando utilizamos la forma rápida de dividir por 2 un número binario, con desplazamientos a la derecha, ¿qué pasa con el dígito binario menos significativo? ¿Qué pasa con el resultado cuando el número es par y cuando es impar?

Conversión de otras bases a sistema decimal

Para convertir de otras bases al sistema decimal usamos lo que se explicó al hablar de la **base** de los sistemas numéricos posicionales y en el ejemplo del truco anterior de cálculo rápido con potencias. Todo lo que hay que hacer es **multiplicar los dígitos de la expresión en la base actual por potencias ordenadas y completas de la base**.

En el truco vimos el desarrollo de un número decimal en base **2**; leyendo el ejemplo al revés, tenemos el mecanismo para volver a convertir a sistema decimal. Si quisiéramos convertir un número en sistema hexadecimal a decimal, consideraremos las potencias de **16**.

Ejemplo

Convertir **$10101110_{(2}$** a decimal.

$10101110_{(2} =$
$1 * 2^7 + 0 * 2^6 + 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0 =$
$1 * 128 + 0 * 64 + 1 * 32 + 0 * 16 + 1 * 8 + 1 * 4 + 1 * 2 + 0 * 1 =$
$128 + 0 + 32 + 0 + 8 + 4 + 2 + 0 =$
$174_{(10}$

Ejemplo

Convertir **$1C8A09_{(16}$** a decimal.

$1C8A09_{(16} =$
$1 * 16^5 + 12 * 16^4 + 8 * 16^3 + 10 * 16^2 + 0 * 16^1 + 9 * 16^0 =$
$1048576 + 12 * 65536 + 8 * 4096 + 10 * 256 + 0 + 9 * 1 =$
$1048576 + 786432 + 32768 + 2560 + 9 = 1870345_{(10}$

Una fórmula alternativa, más rápida, es, procediendo desde el dígito **más** significativo, **multiplicar iterativamente por la base y sumar el dígito siguiente**. En nuestro ejemplo de recién, haríamos:

$1C8A09_{(16} =$

$(((((((((1 * 16) + 12) * 16) + 8) * 16) + 10) * 16) + 0) * 16) + 9 = 1870345_{(10}$.

Conversión entre sistemas binario y hexadecimal

El truco consiste en tener en cuenta que cada **dígito hexadecimal** se representa por **cuatro dígitos binarios**. La tabla de equivalencias es como sigue:

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

Es muy útil memorizar esta tabla. Nos permite deducir fácilmente que, por ejemplo, $AB4C_{(16} = 1010101101001100_{(2}$.

Sistema octal

Otra base muy interesante en computación, pero que por el momento no usaremos, es la base **8** (base del **sistema octal**). Tanto 16 como 8 son potencias de 2; de ahí que es fácil hacer las conversiones entre esas bases y el sistema binario.

Codificación de datos

Hemos visto que la memoria de las modernas computadoras digitales está diseñada para almacenar información en forma de bits, agrupados en conjuntos de a ocho, y que cada una de estas unidades de ocho bits se llama un byte. Usando la analogía natural entre valores de bits (inactivo/activo) y dígitos binarios (0 o 1), hemos visto cómo la memoria de una computadora digital moderna puede almacenar números no negativos entre 0 y 255.

Sin embargo, los problemas de la vida real involucran otras clases de datos. Muchos problemas requerirán manipular números mayores que 255, o números negativos, o con decimales; o aun, datos que no sean numéricos, como texto, imágenes, sonido, o video. Si la única manera posible de guardar contenidos en la memoria de la computadora es en forma de bits, ¿cómo podremos manipular estas otras clases de datos?

Codificación de texto

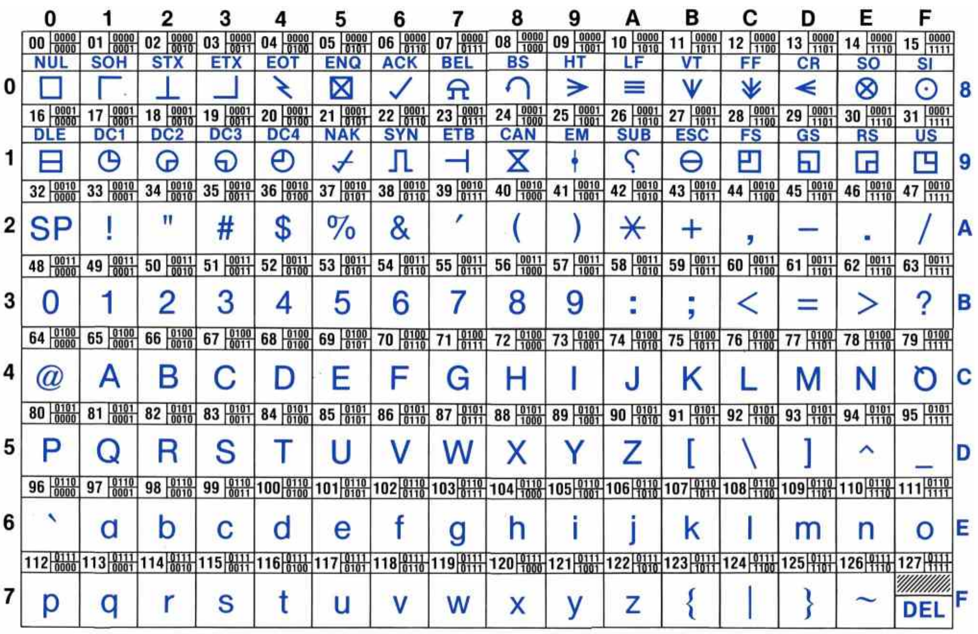
Cuando escribimos texto en nuestra computadora, estamos almacenando temporariamente en la memoria una cierta secuencia de caracteres, que son los símbolos que tipeamos en nuestro teclado. Estos caracteres tienen una **representación gráfica** en nuestro teclado, en la pantalla o en la impresora, pero mientras están en la memoria **no pueden ser otra cosa que bytes**, es decir, conjuntos de ocho dígitos binarios. Para lograr almacenar caracteres de texto necesitamos adoptar una **codificación**, es decir, una tabla que asigne a cada carácter un patrón de bits fijo.

Además, esta codificación debe ser **universal**: para poder compartir información entre usuarios, o entre diferentes aplicaciones, se requiere algún estándar que sea respetado por todos los usuarios y las aplicaciones. Inicialmente se estableció con este fin el **código ASCII**[14], que durante algún tiempo fue una buena solución. El código ASCII asigna patrones de siete bits a un conjunto de caracteres que incluye:

12. Las 25 letras del alfabeto inglés, mayúsculas y minúsculas;
13. Los dígitos del 0 al 9,
14. Varios símbolos matemáticos, de puntuación, etc.,
15. El espacio en blanco,
16. Y 32 caracteres no imprimibles. Estos caracteres no imprimibles son combinaciones de bits que no tienen una representación gráfica, sino que sirven para diversas funciones de comunicación de las computadoras con otros dispositivos.

En general, prácticamente todos los símbolos que figuran en nuestro teclado tienen un código ASCII asignado. Como sólo se usan siete bits, el bit de mayor orden (el de más a la izquierda) de cada byte siempre es cero, y por lo tanto los códigos ASCII toman valores de 0 a 127.

14 http://es.wikipedia.org/wiki/ASCII



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	00 0000 0000 NUL	01 0000 0001 SOH	02 0000 0010 STX	03 0000 0011 ETX	04 0000 0100 EOT	05 0000 0101 ENQ	06 0000 0110 ACK	07 0000 0111 BEL	08 0000 1000 BS	09 0000 1001 HT	10 0000 1010 LF	11 0000 1011 VT	12 0000 1100 FF	13 0000 1101 CR	14 0000 1110 SO	15 0000 1111 SI	8
1	16 0001 0000 DLE	17 0001 0001 DC1	18 0001 0010 DC2	19 0001 0011 DC3	20 0001 0100 DC4	21 0001 0101 NAK	22 0001 0110 SYN	23 0001 0111 ETB	24 0001 1000 CAN	25 0001 1001 EM	26 0001 1010 SUB	27 0001 1011 ESC	28 0001 1100 FS	29 0001 1101 GS	30 0001 1110 RS	31 0001 1111 US	9
2	32 0010 0000 SP	33 0010 0001 !	34 0010 0010 "	35 0010 0011 #	36 0010 0100 $	37 0010 0101 %	38 0010 0110 &	39 0010 0111 '	40 0010 1000 (	41 0010 1001)	42 0010 1010 *	43 0010 1011 +	44 0010 1100 ,	45 0010 1101 –	46 0010 1110 .	47 0010 1111 /	A
3	48 0011 0000 0	49 0011 0001 1	50 0011 0010 2	51 0011 0011 3	52 0011 0100 4	53 0011 0101 5	54 0011 0110 6	55 0011 0111 7	56 0011 1000 8	57 0011 1001 9	58 0011 1010 :	59 0011 1011 ;	60 0011 1100 <	61 0011 1101 =	62 0011 1110 >	63 0011 1111 ?	B
4	64 0100 0000 @	65 0100 0001 A	66 0100 0010 B	67 0100 0011 C	68 0100 0100 D	69 0100 0101 E	70 0100 0110 F	71 0100 0111 G	72 0100 1000 H	73 0100 1001 I	74 0100 1010 J	75 0100 1011 K	76 0100 1100 L	77 0100 1101 M	78 0100 1110 N	79 0100 1111 O	C
5	80 0101 0000 P	81 0101 0001 Q	82 0101 0010 R	83 0101 0011 S	84 0101 0100 T	85 0101 0101 U	86 0101 0110 V	87 0101 0111 W	88 0101 1000 X	89 0101 1001 Y	90 0101 1010 Z	91 0101 1011 [	92 0101 1100 \	93 0101 1101]	94 0101 1110 ^	95 0101 1111 _	D
6	96 0110 0000 `	97 0110 0001 a	98 0110 0010 b	99 0110 0011 c	100 0110 0100 d	101 0110 0101 e	102 0110 0110 f	103 0110 0111 g	104 0110 1000 h	105 0110 1001 i	106 0110 1010 j	107 0110 1011 k	108 0110 1100 l	109 0110 1101 m	110 0110 1110 n	111 0110 1111 o	E
7	112 0111 0000 p	113 0111 0001 q	114 0111 0010 r	115 0111 0011 s	116 0111 0100 t	117 0111 0101 u	118 0111 0110 v	119 0111 0111 w	120 0111 1000 x	121 0111 1001 y	122 0111 1010 z	123 0111 1011 {	124 0111 1100 |	125 0111 1101 }	126 0111 1110 ~	127 0111 1111 DEL	F

Tabla ASCII, que codifica los caracteres más importantes utilizando siete bits (el bit más significativo de cada ocho es siempre cero).

Sin embargo, el código ASCII es insuficiente para muchas aplicaciones: no contempla las necesidades de diversos idiomas. Por ejemplo, nuestra letra **Ñ** no figura en la tabla ASCII. Tampoco las vocales acentuadas, ni con diéresis, como tampoco decenas de otros caracteres de varios idiomas europeos. Peor aún, con solamente 256 posibles patrones de bits, es imposible representar algunos idiomas orientales como el chino, que utilizan miles de ideogramas.

Por este motivo se estableció más tarde una familia de nuevos estándares, llamada **Unicode**[15]. Uno de los estándares de codificación definidos por Unicode, el más utilizado actualmente, se llama **UTF-8**[16]. Este estándar mantiene la codificación que ya empleaba el código ASCII para ese conjunto de caracteres, pero agrega códigos de dos, tres y cuatro bytes para otros símbolos. El resultado es que hoy, con UTF-8, se pueden representar todos los caracteres de cualquier idioma conocido.

Codificación de multimedia

Otras clases de datos, diferentes del texto, también requieren codificación (porque siempre deben ser almacenados en la memoria en forma de bits y bytes), pero su tratamiento es diferente. Introducir en la computadora, por ejemplo, una **imagen analógica** (tal como un dibujo o una pintura hecha a mano), o un fragmento de **sonido** tomado del ambiente, requiere un proceso previo de digitalización. **Digitalizar** es **convertir en digital** la información que es **analógica**, es decir, convertir un rango continuo de valores (lo que está en la naturaleza) a un conjunto discreto de valores (que puede ser ingresado en la computadora).

15 http://es.wikipedia.org/wiki/Unicode
16 http://es.wikipedia.org/wiki/Utf-8

Imagen

En el caso de una imagen analógica, el proceso de digitalización involucra la división de la imagen en una fina cuadrícula, donde cada elemento de la cuadrícula abarca un pequeño sector cuadrangular de la imagen. A cada pequeño sector, o elemento cuadrangular, se le asignan valores discretos que codifican el color de la imagen en ese lugar. Por ejemplo, se pueden asignar tres valores, codificando la cantidad de rojo, de verde y de azul que contiene cada lugar de la imagen. Luego cada uno de estos valores se puede expresar como un entero. Si fuéramos a guardar cada uno de estos enteros en un byte, quedaríamos limitados a valores entre 0 y 255.

9. Un elemento que fuera completamente rojo tendría valores (255, 0, 0). Un elemento de color verde puro tendría valores (0, 255, 0).
10. Un elemento blanco tendrá los máximos valores para todos los colores, que mezclados dan el color blanco: (255, 255, 255).
11. Un elemento que es gris tendrá los tres valores aproximadamente iguales pero menores que 255.

En general, mientras más elementos podamos codificar, mejor será la aproximación a nuestra pieza de información original. Mientras más fina la cuadrícula (es decir, mientras mayor sea la **resolución**[17] de la imagen digitalizada), y mientras más valores discretos usemos para representar los colores, más se parecerá nuestra versión digital al original analógico.

Notemos que la digitalización de una imagen implica la discretización de dos **variables analógicas**, o sea, discretización en dos sentidos: por un lado, los infinitos puntos de la imagen analógica, bidimensional, deben reducirse a unos pocos rectángulos discretos. Por otro lado, los infinitos valores de color deben reducirse a sólo tres coordenadas, con finitos valores en el rango de nuestro esquema de codificación. Son dos ejemplos de conversión de variables analógicas a digitales. Este proceso de digitalización es el que hacen automáticamente una cámara de fotos digital o un celular, almacenando luego los bytes que representan la imagen tomada. Sin embargo, las modernas cámaras utilizan un esquema de codificación con mucha mayor **profundidad de color**[18] (es decir, más bytes por cada coordenada de color) que en el ejemplo anterior.

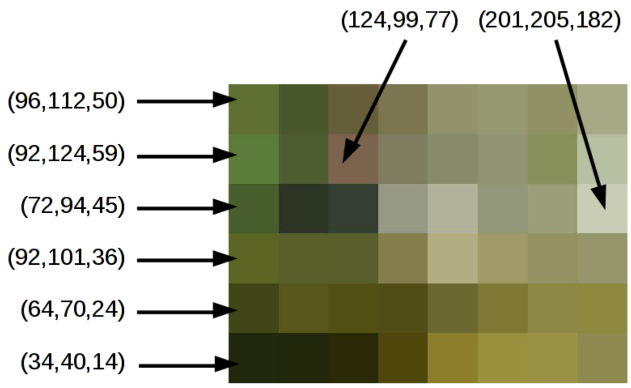


Imagen en baja resolución con 24 bits de profundidad de color.

7. *La primera columna tiene predominancia de verde (la segunda componente es la mayor).*
8. *Colores más oscuros tienen valores menores.*
9. *El elemento que más tiende al rojo tiene la primera componente mayor.*
10. *El elemento gris claro tiene las tres componentes aproximadamente iguales, y de valor alto.*

17 http://es.wikipedia.org/wiki/Resoluci%C3%B3n_de_im%C3%A1genes
18 http://es.wikipedia.org/wiki/Profundidad_de_color

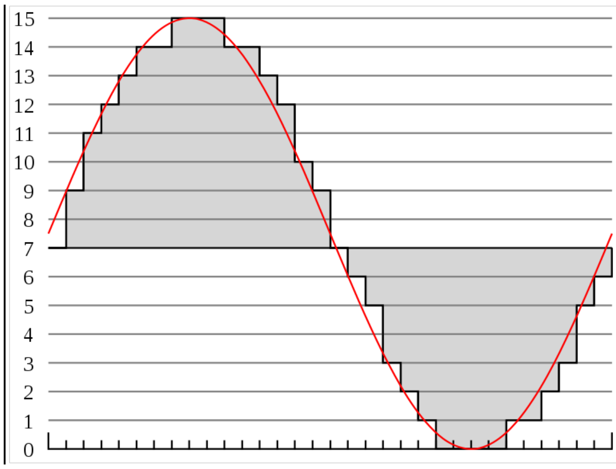
La misma imagen digital, en varias resoluciones

Sonido

En el caso del **sonido** (o **audio**), también tenemos dos variables analógicas para digitalizar. Un micrófono actúa como un **transductor**: traduce los impulsos físicos del sonido, que llegan por el aire, a impulsos eléctricos. Necesitamos discretizar esos impulsos eléctricos, y como en el caso anterior, esto ocurrirá en dos sentidos. Por un lado, los impulsos eléctricos, provocados en el micrófono por el sonido, ocupan un intervalo continuo de tiempo, pero nos quedaremos sólo con unos cuantos valores por segundo. Por otro lado, en cada intervalo de tiempo, los impulsos eléctricos entregados por el micrófono ocupan un rango continuo de valores, entre ciertos valores eléctricos extremos; pero necesitamos quedarnos sólo con algunos valores enteros en ese rango.

Nuevamente, mientras más muestras por segundo y más valores diferentes reconozcamos en el rango de entrada, mejor se aproximará nuestra digitalización del sonido al original (ver figura). Aproximadamente así funcionan al captar nuestra voz los celulares, una cámara filmadora digital, o un programa de grabación de sonido que usamos en nuestra computadora. El resultado es una secuencia de bytes que codifican el sonido que ingresó por el micrófono.

Digitalización de un ciclo de una onda sonora en 16 niveles.



Archivos

Una vez que se ha obtenido la secuencia de bytes que representa la información, ya se trate de texto, de multimedia o de otros tipos de datos, si no queremos perder esta información necesitamos guardarla en algún medio de almacenamiento permanente, como un disco rígido, o un pendrive, creando un **archivo.** Este archivo **es precisamente esa secuencia de bytes**, almacenados en algún medio de almacenamiento.

El archivo, que contiene y transporta nuestra información, ocupará una cierta cantidad de bytes en ese medio de almacenamiento, y, dependiendo de la información que guarda, puede tratarse de grandes cantidades de bytes. Si, por ejemplo, se trata de una imagen de muy alta **resolución** (donde la cuadrícula de digitalización es sumamente fina) y con gran **profundidad de color** (donde los colores son representados con muchos valores discretos, necesitando muchos bytes), entonces la imagen digitalizada será más grande. Es típico que las cámaras digitales modernas entreguen imágenes digitales con una resolución de decenas de millones de puntos o **pixels**[19], y estas imágenes suelen ser de muy gran tamaño[20].

Múltiplos del bit y del byte

Para manejar con comodidad esas grandes cantidades de bits y de bytes recurrimos a múltiplos, tal como se hace habitualmente con los metros para grandes distancias (en cuyo caso hablamos de kilómetros, u otras unidades), o con los gramos para grandes pesos (en cuyo caso hablamos de kilogramos, u otras unidades).

19 http://es.wikipedia.org/wiki/P%C3%ADxel
20 http://www.360cities.net/london-photo-es.html: ¡Una fotografía panorámica de miles de millones de pixels!

Hay dos sistemas de múltiplos de bits y bytes en uso, y a veces esto causa confusión: el Sistema Internacional (SI) y el sistema de Prefijos Binarios.

Sistema Internacional (SI)

El **Sistema Internacional (SI)**[21] define múltiplos para todas las unidades de medida. Estos múltiplos se denominan mediante prefijos que acompañan a la unidad fundamental de cada magnitud (como, por ejemplo, **kilo** acompaña a **gramo** para crear el múltiplo **kilogramo**). En el SI, estos prefijos corresponden a múltiplos elegidos como potencias de 10. Los múltiplos que suelen ser más interesantes para las ciencias y las ingenierías corresponden a aquellas potencias de 10 cuyo exponente es múltiplo de 3 (como 10^3, 10^6, 10^9...).

Por ejemplo, 10^3 (que es igual a 1000) se asocia con el prefijo **kilo** y se simboliza **k**, de modo que kilogramo significa mil gramos y se escribe 1 kg. Del mismo modo, 10^6 recibe el prefijo **mega** (simbolizado por **M**), 10^9 recibe el prefijo **giga** (simbolizado por **G**), y 10^{12} recibe el prefijo **tera** (simbolizado por **T**). De esta manera, **un millón de bytes** se indica como **1 megabyte** y se escribe **1MB**.

Prefijos Binarios

Sin embargo, en computación es habitual medir la información con otros múltiplos que no son potencias de 10 sino potencias de 2. Por ejemplo, para ciertos usos es conveniente considerar los bytes en conjuntos de 1024 (que es 2^{10}) y no en conjuntos de 1000 (que es 10^3). Por este motivo una organización de estándares, la **IEC**, creó un conjunto de múltiplos basados en **Prefijos Binarios**[22]. Estos múltiplos son cantidades que tienen tamaños similares a los múltiplos del SI y son simbolizados con las mismas letras, pero son potencias de 2 cuyo exponente es múltiplo de 10 (como 2^{10}, 2^{20}, 2^{30}...). Para construir los prefijos del sistema de Prefijos Binarios, se agrega la sílaba **bi** luego de la primera sílaba del prefijo SI que los representa.

Así, la unidad binaria más cercana al kilobyte (kB) es el **Kibibyte** (**KiB**), que vale 2^{10} bytes (o sea, 1024 bytes, y no 1000). La unidad binaria más cercana al megabyte (MB) es el **Mebibyte** (**MiB**) que vale 2^{20} bytes (o sea, 1048576 bytes, y no exactamente un millón). La unidad binaria más cercana al gigabyte (GB) es el **Gibibyte** (**GiB**) que vale 2^{30} bytes (1073741824 bytes, y no exactamente mil millones). Existen otros múltiplos mayores, que por el momento no tienen uso diario, pero que de acuerdo con las tendencias tecnológicas, iremos encontrando con mayor frecuencia en el futuro cercano (**Tera, Peta, Exa, Zetta, Yotta...**).

En computación se utilizan, en diferentes situaciones, ambos sistemas de unidades, porque es costumbre usar el SI para hablar de **velocidades de transmisión de datos**, pero usar Prefijos Binarios al hablar de **almacenamiento**. Así, cuando un proveedor de servicios de Internet ofrece un enlace de **1Mbps**, nos está diciendo que por ese enlace podremos transferir **exactamente 1 millón de bits por segundo**. Por el contrario, los fabricantes de **medios de almacenamiento** (como memorias, discos rígidos o pendrives) acostumbran hablar en términos de múltiplos binarios, y por lo tanto deberían (aunque normalmente no lo hacen) utilizar **Prefijos Binarios** para expresar las capacidades de almacenamiento de esos medios. Así, un “pendrive de cuatro gigabytes”, que normalmente tiene una capacidad de $4 * 2^{30}$ bytes, debería publicitarse en realidad como “pendrive de cuatro Gibibytes”.

Compresión

Muchas veces es interesante reducir el tamaño de un archivo, para que ocupe menos espacio

21 http://es.wikipedia.org/wiki/Sistema_Internacional_de_Unidades#Tabla_de_m.C3.BAltiplos_y_subm.C3.BAltiplos
22 http://es.wikipedia.org/wiki/Prefijo_binario

de almacenamiento o para que su transferencia a través de una red sea más rápida. Al ser todo archivo una secuencia de bytes, y por lo tanto de números, disponemos de métodos y herramientas matemáticas que permiten, en ciertas condiciones, reducir ese tamaño. La manipulación de los bytes de un archivo con este fin se conoce como **compresión.**

La compresión de un archivo se ejecuta mediante un programa que utiliza un algoritmo especial de compresión. Este algoritmo puede ser de **compresión sin pérdida, o con pérdida**[23].

Compresión sin pérdida

Decimos que la compresión ha sido sin pérdida cuando, del archivo comprimido, puede extraerse exactamente la misma información que antes de la compresión, utilizando otro algoritmo que ejecuta el trabajo inverso al de compresión. En otras palabras, la compresión sin pérdida es **reversible**: siempre puede volverse a la información de partida. Esto es un requisito indispensable cuando necesitamos recuperar exactamente la secuencia de bytes original, como en el caso de un archivo de texto. Como usuarios de computadoras, es muy probable que hayamos utilizado más de una vez la compresión sin pérdida, al tener que comprimir un documento de texto que nosotros hicimos utilizando un programa utilitario como ZIP[24], RAR u otros. Si la compresión no fuera reversible, no podríamos recuperar el archivo de texto tal cual lo escribimos.

Compresión con pérdida

En algunos casos, el resultado de la compresión de un archivo es otro archivo del cual **ya no puede recuperarse** la misma información original, pero que de alguna manera sigue sirviendo a los fines del usuario. Es el caso de la compresión de imágenes, donde se reduce la calidad de la imagen, ya sea utilizando menos colores, o disminuyendo la resolución. También es el caso de la compresión de audio, al descartar componentes del sonido con frecuencias muy bajas o muy altas, inaudibles para los humanos (como en la tecnología de grabación de CDs), con lo cual la diferencia entre el original digital y el comprimido no es perceptible al oído. También es útil, para algunos fines, reducir la calidad del audio quitando componentes audibles (lo que hacen, por ejemplo, algunos grabadores “de periodista” para lograr archivos más pequeños, con audio de menor fidelidad, pero donde el diálogo sigue siendo comprensible).

23 http://es.wikipedia.org/wiki/Digitalizaci%C3%B3n#Compresi.C3.B3n
24 http://es.wikipedia.org/wiki/Formato_de_compresi%C3%B3n_ZIP

Representación de datos numéricos

Hasta el momento hemos visto los sistemas numéricos, especialmente el **sistema binario**, desde el punto de vista puramente matemático, y hemos considerado únicamente la representación y la aritmética del conjunto de los naturales y el cero, es decir, de **números enteros no negativos**.

Sin embargo, en el dominio de la computación, tarde o temprano nos enfrentaremos con algunas dificultades:

17. Existen otros conjuntos numéricos, como los **enteros (positivos, cero y negativos)**, que aparecen frecuentemente en los problemas de la vida real.
18. Otros conjuntos muy importantes que aún no podemos manejar son los **racionales**, los **reales** y los **complejos**.
19. Además, hemos visto anteriormente que las computadoras disponen de una **cantidad limitada de dígitos** binarios para la representación numérica.

Estudiaremos entonces cómo resolver el problema de representar diferentes conjuntos numéricos con una **cantidad finita** de dígitos binarios.

Datos y tipos de datos

Al procesar los datos, las computadoras los mantienen en contenedores, o unidades de almacenamiento, de un tamaño determinado. Los tamaños habituales para estas unidades son, por ejemplo, **8, 16, 32 o 64 bits**. Por otro lado, los lenguajes de programación proveen una cantidad limitada de **tipos de datos**, con los que pueden operar los programas escritos en esos lenguajes. Los datos de cada tipo de datos se guardan en unidades de almacenamiento de un tamaño conveniente.

Además de usar unidades de almacenamiento de diferentes tamaños, diferentes tipos de datos tienen determinadas características. Cuando un programador necesita manejar un cierto dato dentro de un programa, elige cuidadosamente un tipo de dato provisto por el lenguaje, que le ofrezca las características que necesita para su problema. Por ejemplo, un tipo de datos **entero** puede ser **con signo** (en cuyo caso puede representar datos positivos o negativos) **o sin signo** (en cuyo caso representará únicamente positivos y el cero). O bien, un tipo de datos **real** puede tener **una determinada precisión**.

Dejemos claro que cuando hablamos de **enteros con o sin signo**, nos referimos a los **tipos de datos**, es decir, a la forma convencional como los representamos, y no a los datos mismos. Un **dato** puede ser un número **positivo** (y no llevar un signo “menos”) pero estar representado con un tipo de dato **con signo**. Como la información de si un número es o no negativo corresponde a una **pregunta binaria**, un tipo de dato entero **con signo** necesita destinar un bit para representar esta información, sea o no negativo el dato almacenado.

Rango de representación

Al representar números enteros, con o sin signo, con una cantidad finita **n** de dígitos, en un sistema cualquiera de base **b**, tenemos una primera limitación. No todos los números pueden ser representados. ¿Cuántos números podremos representar en el sistema decimal, **con n dígitos**? El intervalo de números representables en el sistema se llama su **rango de representación**. El tamaño del rango de representación es una importante característica de un tipo de datos.

12. En tipos de datos **sin signo**, el **menor número** que podemos representar es el 0; pero el mayor número **sin signo** representable en un sistema de base **b** resulta ser b^n-1. El intervalo **$[0, b^n-1]$ es el llamado rango de representación** del sistema. En el sistema binario los números **sin signo** abarcan el rango **de 0 a 2^n-1**, siendo **n** la cantidad de dígitos.

13. Cuando consideramos números **con signo**, este rango de representación cambia **según la técnica de representación** que usemos.

Representación de enteros con signo

Veremos con detalle dos métodos de **representación con signo** de enteros[25]. El primero se llama **signo-magnitud**, y el segundo, **complemento a 2**.

Signo-magnitud

La notación **signo-magnitud** representa el **signo** de los datos con su bit de orden más alto, o **bit más significativo** (el bit más a la izquierda), y el **valor absoluto** del dato con los restantes bits. Por ejemplo, si la unidad de almacenamiento del dato mide 8 bits, **-1** se representará como **10000001**, y **+1** como **00000001** (igual que si se tratara de un entero sin signo). El número **-3** se representará como **10000011** y el número **-64** como **11000000**. El método es fácil de ejecutar a mano, si sabemos representar los enteros sin signo.

Notemos que al usar un bit para signo, automáticamente perdemos ese bit para la representación de la magnitud o valor absoluto, disminuyendo así el rango de representación, el cual se divide por 2. Con la misma unidad de almacenamiento de 8 bits, el entero **sin signo** más grande representable era **2^8-1 = 255** (que se escribe **11111111**). Ahora, con signo-magnitud, el entero con signo más grande representable será **2^7-1 = 127** (que se escribe **01111111**).

El rango de representación en **signo-magnitud** con **n dígitos** es **$[-(2^{n-1} - 1), 2^{n-1} - 1]$**.

Para los humanos, el método de signo-magnitud resulta sumamente cómodo, pero presenta algunos inconvenientes para su uso dentro de la computadora.

7. En primer lugar, el 0 tiene dos representaciones, lo que complica las cosas. Cuando la computadora necesita saber si un número es 0, debe hacer dos comparaciones distintas, una por cada valor posible del cero.
8. En segundo lugar, la aritmética en signo-magnitud es bastante dificultosa, tanto para los humanos como para la computadora. Para ver la dificultad de operar en este sistema, basta con tratar de hacer a mano algunas operaciones sencillas de suma o resta en aritmética binaria en signo-magnitud. Primero hay que decidir si los signos son iguales; en caso contrario, identificar el de mayor magnitud para determinar el signo del resultado; luego hacer la operación con los valores absolutos, restituir el signo, etc. La operación se vuelve bastante compleja.

Otros métodos de representación (los sistemas de complemento) resuelven el problema con mayor limpieza.

Complemento a 2

El segundo método es el de **complemento a 2**, que resuelve el problema de tener dos representaciones para el 0 y simplifica la aritmética binaria, tanto para los humanos como para las computadoras. El método de complemento a 2 **también emplea el bit de orden más alto de acuerdo al signo**, pero la forma de representación es diferente:

6. Los positivos se representan igual que si se tratara de enteros sin signo (igual que en el caso de signo-magnitud).
7. Para los negativos, se usa el procedimiento de **complementar a 2,** que puede

25 http://es.wikipedia.org/wiki/Representación_de_números_con_signo

resumirse en tres pasos:

- expresamos en binario el **valor absoluto** del número, en la cantidad de bits del sistema;
- **invertimos los bits**
- y **sumamos 1**.

Ejemplo

Expresar los números **-23, -9 y 8** en sistema binario en complemento a 2 con 8 bits.

- El valor absoluto de -23 es **23**. Expresado en binario en 8 bits es **00010111**. Invertido bit a bit es **11101000**. Sumando 1, queda **11101001**, luego **$-23_{(10}$ = $11101001_{(2}$**.
- El valor absoluto de -9 es **9**. Expresado en binario en 8 bits, es **00001001**. Invertido bit a bit es **11110110**. Sumando 1, queda **11110111**, luego **$-9_{(10}$ = $11110111_{(2}$**.
- El número **8** es positivo. Su expresión en el sistema de complemento a 2 es la misma que si fuera un número sin signo. Queda **$8_{(10}$= $00001000_{(2}$**.

El rango de representación para el sistema de **complemento a 2 con n dígitos** es **[-2^{n-1} , 2^{n-1} - 1]**.

Es decir, el rango de representación de complemento a 2 **contiene un valor más** que **signo-magnitud**.

Otra propiedad interesante del método es que, si sabemos que un número binario es **negativo** y está escrito en complemento a 2, podemos recuperar su valor en decimal usando casi exactamente el mismo procedimiento: **invertimos los bits** y **sumamos 1**. Convertimos a decimal como si fuera un entero sin signo y luego agregamos el signo **menos** habitual de los decimales.

Ejemplo

Convertir a decimal los números, **en complemento a 2**, $11101001_{(2}$, $11110111_{(2}$ y $00000111_{(2}$.

- **11101001** es negativo, ya que su bit de orden más alto es **1**; invertido bit a bit es 00010110; sumando 1 queda 00010111 que es 23. Agregando el signo **menos** queda **$11101001_{(2}$ = $-23_{(10}$**.
- **11110111** es negativo; invertido bit a bit es 00001000; sumando 1 queda 00001001 que es 9. Agregando el signo **menos** queda **$11110111_{(2}$ = $-9_{(10}$**.
- **00000111** es positivo, porque su bit de orden más alto es 0. Lo interpretamos igual que si fuera un entero sin signo y entonces **$00000111_{(2}$ = $7_{(10}$**.

Ejercicios

- Utilizando sólo dos bits, representar todos los números permitidos por el rango de representación en complemento a 2 e indicar su valor decimal.
- Idem con tres bits.
- ¿Cuál es el rango de representación en complemento a 2 para **n = 8**?

Nota

Es interesante ver cómo los diferentes sistemas de representación asignan diferentes valores a cada combinación de símbolos. Veámoslo para unos pocos bits.

Decimal	Sin signo	Signo-magnitud	Complemento a 2
-8			1000
-7		1111	1001
-6		1110	1010
-5		1101	1011
-4		1100	1100
-3		1011	1101
-2		1010	1110
-1		1001	1111
0	0000	0000 y 1000	0000
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0011
4	0100	0100	0100
5	0101	0101	0101
6	0110	0110	0110
7	0111	0111	0111
8	1000		
9	1001		
10	1010		
11	1011		
12	1100		
13	1101		
14	1110		
15	1111		

La tabla muestra que la representación en complemento a 2 es más conveniente para las operaciones aritméticas:

- La operación de complementar a 2 da un número negativo que permite resolver las restas. Las computadoras **no restan: suman complementos**.
- El cero tiene una única representación.
- En complemento a 2 se ha recuperado la representación del -8, que estaba desperdiciada en signo-magnitud.
- El rango de representación tiene un valor más en complemento a 2 que en

signo-magnitud.

- Los números en complemento a 2 se disponen como en una rueda. Si tomamos cada número en la columna de complemento a 2 (salvo el último) y le sumamos 1, obtenemos en forma directa el número aritméticamente correcto. Similarmente si restamos 1 (salvo al primero). Al sumar 1 al **7**, obtenemos el menor número negativo posible. Al restar 1 a **-8**, obtenemos el mayor positivo (“entramos a la tabla por el extremo opuesto”). Esta propiedad, que no se cumple para signo-magnitud, hace más fácil la implementación de las operaciones aritméticas en los circuitos de la computadora.

Por estos motivos, la representación en complemento a 2 es más natural y consistente. Se la elige normalmente para las computadoras, en lugar de otros métodos de representación.

Aritmética en complemento a 2

Una suma en complemento a 2 se ejecuta sencillamente bit a bit, sin las consideraciones necesarias para signo-magnitud. Una resta **A – B** se interpreta como la suma de **A** y el complemento de **B**.

- La suma se realiza bit a bit desde la posición menos significativa, con **acarreo** (nos “llevamos un bit” cuando la suma de dos bits da $10_{(2}$).
- Cuando uno de los sumandos es negativo, lo complementamos y entonces hacemos la suma.

Ejemplos

- Restar $\mathbf{2_{(10}}$ **de** $\mathbf{23_{(10}}$. Complementamos 2 para convertirlo en negativo: $\mathbf{-2_{(10} = 11111110_{(2}}$**,** y lo sumamos a $\mathbf{23_{(10}}$. La cuenta es $\mathbf{23_{(10} + (-2_{(10})=}$ $\mathbf{00010111_{(2} + 11111110_{(2} = 00010101_{(2}}$**,** que es positivo y vale **21**.
- Sumar $\mathbf{23_{(10}}$ **y** $\mathbf{-9_{(10}}$ con 8 bits. Se tiene que $\mathbf{23_{(10} = 00010111_{(2}}$ y $\mathbf{-9_{(10}}$ = $\mathbf{11110111_{(2}}$. La suma da $\mathbf{00001110_{(2}}$, que es positivo y equivale a $\mathbf{14_{(10}}$.
- Sumar $\mathbf{-23_{(10}}$ **y** $\mathbf{-9_{(10}}$ con 8 bits. Se tiene que $\mathbf{23_{(10} = 00010111_{(2}}$ y por lo tanto $\mathbf{-23_{(10} = 11101000 + 1 = 11101001_{(2}}$. Por otro lado $\mathbf{-9_{(10}}$ = $\mathbf{11110111_{(2}}$ como calculamos antes. Al sumarlos bit a bit resulta $\mathbf{11101001_{(2} + 11110111_{(2}}$ = $\mathbf{11100000_{(2}}$. Descartamos el acarreo que se “cae” de los 8 bits. El resultado es negativo porque su bit más significativo es 1, y para calcular su valor decimal repetimos el procedimiento de complemento a 2: invertimos y sumamos 1. El número $\mathbf{11100000_{(2}}$ invertido es $\mathbf{00011111_{(2}}$. Sumando 1 obtenemos $\mathbf{00100000_{(2}}$ que es $\mathbf{32_{(10}}$. Como sabíamos que este número era negativo, agregamos el signo **menos** y obtenemos $\mathbf{-32_{(10}}$.

Overflow

Al disponer de finitos dígitos binarios, las operaciones aritméticas pueden exigir más bits de los que tenemos disponibles, originándose errores de cómputo. Cuando ocurren estos errores, la computadora debe detectarlos y declarar la operación no válida, para que estos errores no afecten otras operaciones.

Notemos que en los casos de sumar 23 y -9, o -23 y -9, el acarreo (o "**carry**") puede llegar hasta el bit de signo (**carry-in**), y también pasar más allá, siendo descartado (**carry-out**). El acarreo a veces denuncia un problema con las magnitudes permitidas por el rango de representación del sistema, como se ve en el siguiente ejemplo.

Ejemplo

Sumar **123$_{(10}$ y 9$_{(10}$** con 8 bits. Se tiene que **123$_{(10}$ = 01111011 $_{(2}$** y **9$_{(10}$** = **00001001$_{(2}$**. Notemos que la suma provoca el acarreo de un **1** que llega al bit de signo ("**carry-in**"). El resultado da **10000100$_{(2}$**. Claramente **tenemos un problema**, porque la suma de 123 + 9 debería dar positiva, pero el resultado que obtuvimos tiene el bit de orden más alto en **1**; por lo cual es negativo. Este es un caso patológico de la suma que se llama **overflow**.

Los números de nuestro ejemplo, **123$_{(10}$** y **9$_{(10}$**, pueden expresarse individualmente como enteros con signo en 8 bits, pero suman **132$_{(10}$**, número positivo que excede el rango de representación. El resultado simplemente **no se puede** expresar en 8 bits con signo en complemento a 2. Es una consecuencia de tener una cantidad limitada de bits en nuestro tipo de datos.

Cuando pasa esto se dice que ha ocurrido una condición de **overflow** o **desbordamiento aritmético**[26]. Ocurre **overflow** cuando se suman dos números positivos y se obtiene un resultado en bits que sería negativo en el rango de representación del sistema, o cuando se suman dos negativos y el resultado es positivo.

Sin embargo, en complemento a 2, nunca puede ocurrir **overflow** cuando se suman un positivo y un negativo.

Las computadoras detectan la condición de overflow porque **el bit de acarreo al llegar a la posición del signo** (**carry-in**) y el que se descarta (**carry-out**) son de diferente valor. Al sumar **23 más -9**, el carry-in y el carry-out eran **iguales**, por lo cual la suma no sufrió overflow. Al sumar **123 más 9**, el carry-in es 1 pero el carry-out es 0, lo que nos advierte que ha ocurrido overflow.

	1	**1**	1	1		1	1	1	
23		0	0	0	1	0	1	1	1
+									
-9		1	1	1	1	0	1	1	1
		0	0	0	0	1	1	1	0

No hay overflow

	0	**1**	1	1	1		1	1	
123		0	1	1	1	1	0	1	1
+									
9		0	0	0	0	1	0	0	1
		1	0	0	0	0	1	0	0

Hay overflow

Representación de números reales

Notación de punto fijo

Al representar enteros, con o sin signo, y en cualquier base, hemos supuesto que al final de los dígitos estaba el **punto fraccionario (o coma fraccionaria)** que separa la **parte entera** de

26 http://es.wikipedia.org/wiki/Desbordamiento_aritmético

la **parte fraccionaria**. No se trata de otra cosa que lo que llamamos, en el sistema numérico decimal habitual, **punto o coma decimal**. En el sistema decimal, cuando trabajamos con parte fraccionaria, el desarrollo en suma de potencias se extiende a los exponentes negativos:

$$\mathbf{3.1416 = 3 * 10^{0} + 1 * 10^{-1} + 4 * 10^{-2} + 1 * 10^{-3} + 6 * 10^{-4}}$$

Los sistemas numéricos que hemos usado hasta el momento, sin dígitos fraccionarios, son sistemas **de punto fijo con cero dígitos fraccionarios**. Si hacemos la convención de que utilizamos un sistema numérico **de punto fijo con n dígitos fraccionarios**, sigue siendo innecesario escribir el punto (porque sobreentendemos que el punto está **n lugares contando desde la derecha** del número escrito), pero ahora podemos representar números con parte fraccionaria. Por ejemplo, si declaramos que usamos el **sistema binario de punto fijo a 8 dígitos con 3 dígitos fraccionarios**, entonces el número **01001101** debe ser interpretado como:

$$\mathbf{01001.101 = 0 * 2^{4} + 1 * 2^{3} + 0 * 2^{2} + 0 * 2^{1} + 1 * 2^{0} + 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} =}$$
$$\mathbf{= 8 + 1 + 0.5 + 0.125 = 25.625}$$

Sin embargo, esta representación no es la más adecuada para todas las cantidades. En ciencias e ingenierías es común tener que manejar números muy grandes o muy pequeños, que resultan de mediciones físicas con una determinada precisión (es decir, con una cierta cantidad de decimales). Por ejemplo, la masa de la Tierra, la velocidad de la luz, o las distancias entre los planetas, son números “grandes”; la masa del electrón, o el diámetro de una célula, son números sumamente “pequeños”. En el caso de las cantidades “grandes”, lo más importante es el **orden de magnitud** o cantidad de cifras enteras, y no tiene mayor sentido consignar los decimales, ya que raramente se conocen con precisión. Por el contrario, en el caso de las cantidades “pequeñas”, normalmente la parte de mayor interés es la **precisión**, dada por los decimales.

En notación de punto fijo, una vez elegida la cantidad de dígitos enteros y fraccionarios, nos encontraremos que, o bien los dígitos que reservamos para la parte entera son insuficientes para una clase de números, o los dígitos fraccionarios son escasos para la otra clase. Como veremos, hay soluciones mejores que la representación de punto fijo.

Notación científica

Las matemáticas aplicadas permiten manejar estos números con una notación especial, la llamada **notación científica**[27], que expresa de una manera uniforme estos números tan diferentes. Si tuviéramos que hacer intervenir, en un mismo cómputo manual, cantidades muy grandes y muy pequeñas, la notación científica sería la forma más cómoda para operar.

En notación científica, los números se expresan como el producto de un coeficiente (**mayor o igual que 1 y menor que la base**) y una potencia entera de la base. Por ejemplo, la velocidad de la luz, 300000000 m/s, se expresa como $\mathbf{3x10^{8}}$ m/s, y la masa de un electrón es de alrededor de 0.00000000000000000000000000000091 kg, o sea $\mathbf{9.1x10^{-31}}$ kg. El signo del exponente nos indica si estamos tratando con números “grandes” o “pequeños”.

Notación en punto flotante

Al momento de almacenar y manejar estas diferentes clases de números con computadoras, tiene sentido tomar la idea de la notación científica, porque así podemos representar un gran rango de números en una cantidad fija de bits. La forma de representación de números reales más utilizada en las computadoras es la de **punto flotante**[28], que se inspira en la notación científica (pero **de base 2**), y está estandarizada por la organización de ingeniería IEEE[29].

27 http://es.wikipedia.org/wiki/Notación_científica
28 http://es.wikipedia.org/wiki/Coma_flotante
29 http://es.wikipedia.org/wiki/IEEE_coma_flotante

Un número en punto flotante tiene una representación bastante compleja, con tres componentes: un bit de **signo** (**s**); 8 bits para el **exponente** (**e**), y 23 bits para los dígitos fraccionarios, llamados **la mantisa** (**m**). De esta manera, un número en punto flotante se almacenará en **32 bits** (cuatro bytes). Se han estandarizado otros formatos con diferentes cantidades de bits, pero esta forma (llamada de **simple precisión**) es la más sencilla de estudiar.

1	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
s	e								m																						

Estos tres componentes **s, e y m** tienen las siguientes particularidades.

- El signo, almacenado en el campo **s**, puede ser 0 (positivo) o 1 (negativo).
- El exponente de la notación científica puede ser negativo o positivo, pero se almacena en el campo **e** como un número sin signo (por lo tanto, entre 0 y 255), sumándole el valor 127 antes de almacenarlo.
- El coeficiente de la notación científica siempre comienza por un dígito 1. Como este dígito se conoce, no se almacena en la mantisa, para ahorrar espacio.
- Finalmente, un número **n** expresado en la notación en punto flotante se escribirá como
  $\mathbf{n = (-1)^s * 2^{(e - 127)} * (1 + m)}$.

Representación de un decimal en punto flotante

Representemos en punto flotante el número **3.14**. El procedimiento manual lleva varios pasos:

1. Para pasar el número a sistema binario lo separamos en parte entera y en parte fraccionaria.
   - ¡La parte entera es fácil!: $\mathbf{3_{(10} = 11_{(2}}$.
   - Para escribir la parte fraccionaria en sistema binario, multiplicamos sucesivamente por 2 la parte fraccionaria decimal y tomamos la parte entera del resultado como dígito binario, repitiendo hasta llegar a 0, o hasta lograr la precisión que buscamos[30].

Fracción	Fracción * 2	Parte entera
0.14	0.28	0
0.28	0.56	0
0.56	1.12	1
0.12	0.24	0
0.24	0.48	0
0.48	0.96 ≈ **1.00**	**1 (¡aproximando!)**
0.00	**0.00**	**0**
0.00	**0.00**	**0**

Como **0.96** está suficientemente cercano a **1.0**, podemos aproximarlo a 1 y dar por terminado el desarrollo (ya que todos los dígitos siguientes serán 0). Los dígitos binarios del resultado se leen de arriba hacia abajo. Finalmente obtuvimos que $\mathbf{3.14_{(10} = 11.001001_{(2}}$.

2. Para **normalizar** esta expresión hay que desplazar el punto fraccionario de **11.001001** a la izquierda **de modo de dejar un único dígito 1** en la parte entera. Al correr el punto **un lugar a la izquierda** hemos dividido por 2. Compensamos la división por 2

30 http://es.wikipedia.org/wiki/Sistema_de_numeración_binario#Decimal_.28con_decimales.29_a_binario

que hicimos, multiplicando todo el número por 2, para que no cambie su valor:

$$\mathbf{11.001001 = 1.1001001 * 2^{1}}.$$

Hemos obtenido el coeficiente de la notación científica, que es **1.1001001**, y el exponente, que en este caso es **1**. Si hubiéramos desplazado el punto fraccionario más lugares hacia la izquierda, el exponente habría sido mayor. Si hubiéramos desplazado el punto fraccionario hacia la derecha, el exponente habría sido negativo.

3. Para almacenar el exponente 1 en los 8 bits del campo **e** de punto flotante le sumamos 127: $\mathbf{00000001_{(2} + 01111111_{(2} = 10000000_{(2}}$. El exponente **e** del punto flotante será $\mathbf{10000000_{(2}}$.
4. La mantisa **m** es la parte fraccionaria del coeficiente (con parte entera 1) que hemos calculado: $\mathbf{1001001_{(2}}$, con tantos ceros agregados a la derecha como haga falta para llenar los 23 bits.
5. El signo **s** es **0** porque 3.14 es positivo. Finalmente, tenemos que **3.14** se representará en punto flotante como la siguiente sucesión de bits en la memoria:

1	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s	e								m																						

Conversión de punto flotante a expresión decimal

Si quisiéramos volver a obtener el número decimal original, a partir de una representación en punto flotante, podemos llegar a encontrarnos con algo diferente al valor de partida. Reconstruyamos el número decimal que hemos representado en punto flotante:

$$\mathbf{n = (-1)^{s} * 2^{(e-127)} * (1 + m) = 1 * 2^{(128-127)} * 1.1001001_{(2} = 2^{1} * 1.1001001_{(2} =}$$

$$\mathbf{= 11.001001_{(2} = 2^{1} + 2^{0} + 2^{-3} + 2^{-6} = 3 + 1/8 + 1/64 = 3 + 0.125 + 0.015625 = 3.140625}$$

Notemos que no podremos representar números de cualquier tamaño (al ser finita la cantidad de bits de **e**), ni almacenar todos los dígitos fraccionarios (al ser finita la cantidad de bits de **m**) de cualquier número, sino que en la mayoría de los casos utilizaremos **aproximaciones.** De hecho, el número que hemos representado no es **3.14** sino **3.140625**. En este caso, esto ocurre porque anteriormente hemos **truncado** el desarrollo fraccionario, al aproximar el séptimo dígito. De cualquier manera, el desarrollo fraccionario no podría haber continuado más allá del dígito 23, porque sólo hay esa cantidad de dígitos fraccionarios en la representación en punto flotante en 32 bits.

Al definirse un formato de punto flotante como el que hemos descripto, la cantidad de bits asignada al exponente determina el **rango** de los números que se podrán manejar en esa representación. Con más bits en el exponente, hubiéramos podido representar números **más grandes y más pequeños**. Con más bits en la mantisa, hubiéramos podido almacenar más dígitos fraccionarios, es decir, **con mayor precisión**. Las cantidades de bits para los campos **e** y **m** se han escogido de modo de satisfacer la mayoría de los problemas de cómputo y a la vez mantener un costo aceptable en trabajo de máquina y en espacio de almacenamiento.

Modelo Computacional Binario Elemental

En esta unidad describiremos la arquitectura de una computadora hipotética. Aunque es sumamente elemental, comparte las características esenciales de casi todas las computadoras digitales, y está inspirada en máquinas que han existido realmente.

Repitamos que esta computadora **no tiene existencia real**: es tan poco potente que hoy ya no sería razonable implementarla, salvo por motivos de enseñanza. Pero, aun tan simple como es, puede ejecutar tareas de complejidad bastante interesante y nos servirá para mostrar muchos de los problemas relacionados con la arquitectura y la organización de las computadoras reales.

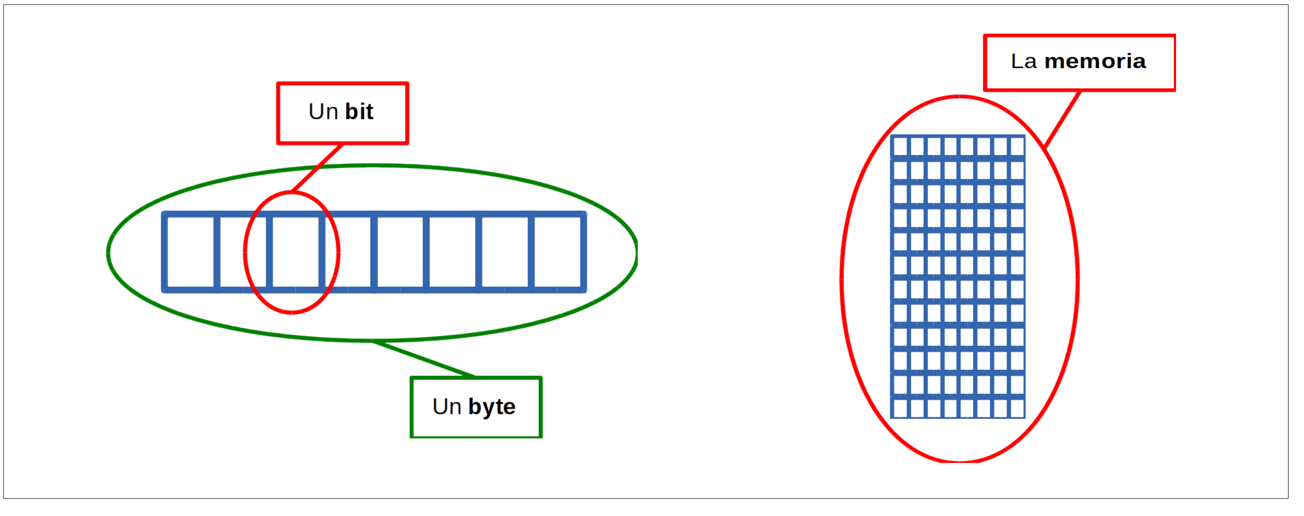
No se trata, entonces, más que de un **modelo**, y como utiliza aritmética binaria, lo hemos bautizado **MCBE (Modelo Computacional Binario Elemental)**. El MCBE es un ejemplo muy sencillo de **computador de programa almacenado**[31].

Unidades funcionales del MCBE

Esta computadora elemental se compone de dos unidades funcionales principales: **memoria y CPU**.

Memoria

La **memoria** de la computadora es un conjunto de circuitos **biestables**, cada uno de los cuales puede almacenar **un bit** de información. Esos circuitos de la memoria están dispuestos en celdas de **ocho** biestables. Cada una de estas celdas ocupa una **posición** de memoria, que puede almacenar un **byte** de información.



Las posiciones de la memoria se encuentran numeradas consecutivamente **a partir de 0**, por lo cual podemos imaginarnos que la memoria es algo así como una alta estantería vertical, de muchos estantes numerados. Cada uno de esos estantes, de ocho casilleros, será capaz de guardar un determinado contenido. Como cada biestable representa un bit y cada posición de memoria representa un byte, a veces esos circuitos y celdas de circuitos se llaman directamente **bits y bytes de la memoria**. La posición relativa de cada byte se llama su **dirección**. Al acceder a un dato contenido en una posición de memoria, ya sea para leerlo o para modificarlo, necesariamente tenemos que mencionar su dirección; cuando hacemos esto decimos que **direccionamos** esa posición de la memoria.

Es costumbre representar las direcciones de memoria con la posición inicial (la dirección 0) en la base del diagrama.

31 http://es.wikipedia.org/wiki/Computador_de_programa_almacenado

CPU

Las siglas **CPU** se refieren a *Central Processing Unit*, **Unidad Central de Procesamiento**. La CPU es un dispositivo complejo, formado por varios componentes, que al activarse es capaz de ejecutar **instrucciones** que transformarán la información almacenada en la memoria.

La CPU, a su vez, contiene tres unidades funcionales propias: la **Unidad de Control (UC)**, la **Unidad Lógico-aritmética (ULA)**, y la **Unidad de Entrada/Salida (UE/S)**. Las unidades cuentan con **registros** especiales, que son espacios de almacenamiento, similares a los de la memoria, pero situados en otro lugar de la circuitería.

- Unidad de Control
  - Su función es gobernar la actividad de la CPU, indicando cuál es la próxima instrucción a ejecutar y de qué modo debe cumplirse.
- Unidad Lógico-aritmética
  - Contiene la circuitería necesaria para ejecutar operaciones matemáticas y lógicas.
- Unidad de Entrada/Salida
  - La UE/S conecta a la MCBE con dispositivos como teclados, pantallas o impresoras. La Unidad de Entrada/Salida se requiere para poder comunicar la máquina con el resto del mundo. Si no existiera la UE/S, la máquina no podría recibir los datos con los que tiene que trabajar, ni podría hacer saber al usuario de la máquina los resultados de sus cálculos.

En las computadoras, las unidades funcionales comparten datos, y para eso están relacionadas mediante **buses**, que son canales de comunicación que permiten transferir datos entre las unidades.

Descripción detallada del MCBE

Recordemos que las computadoras **no toman decisiones por sí solas**. Todo lo que hacen **está determinado por el programa almacenado**, cuya escritura es responsabilidad del usuario. Especificaremos entonces con todo detalle, en el diseño de esta máquina elemental, cuál será la respuesta de la computadora a cada instrucción de un programa.

Instrucciones

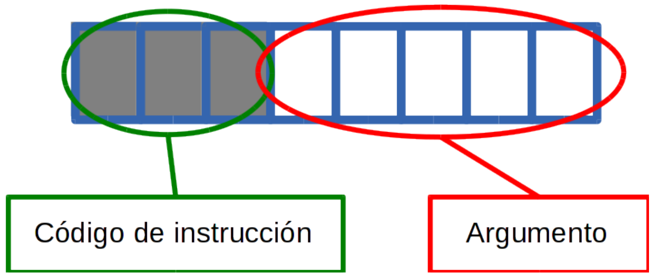
Hay tan sólo **ocho diferentes instrucciones** que puede seguir esta máquina. Algunas sirven para realizar cálculos; otras, para mover datos de un lugar a otro; otras, para modificar el curso de las acciones a seguir por el programa. En cuanto a operaciones aritméticas, el MCBE sólo sabe **sumar y restar** datos. Sin embargo, basándose en esas únicas dos operaciones, puede seguir un **programa** que implemente otras operaciones más complejas.

Para ayudar a los programadores, las instrucciones reciben **nombres mnemotécnicos**, derivados del inglés (**LD, ST, ADD, SUB, JMP, JZ, HLT, NOP**). Se acostumbra usar estos nombres, u otros muy similares, en la programación de máquinas parecidas de la realidad. Sin embargo, estos nombres únicamente sirven para que los humanos comprendan mejor el modelo y su programación. El MCBE los ignora completamente y **sólo utiliza la expresión binaria** de esas instrucciones, residente en la memoria. La Unidad de Control de la CPU es quien interpretará cada una de las posiciones de memoria, ya sea como un dato numérico, o como una instrucción.

Interpretación de instrucciones

Cuando el byte contenido en una posición de memoria represente una instrucción, los **tres bits de orden más alto** (los tres bits situados **más a la izquierda**) indicarán el **código** de la operación. En el caso de ciertas instrucciones, los restantes bits en el byte (los **cinco bits de**

orden más bajo) representarán un **argumento** para la instrucción, es decir, un dato para que esa instrucción trabaje.



Argumentos

Cuando la instrucción utilice argumentos, éstos pueden ser de una de dos clases: **direcciones y desplazamientos**.

2. Cuando la instrucción sea de transferencia entre el acumulador y la memoria (LD, ST, ADD, SUB) el argumento será una **dirección**, y los cinco bits de orden bajo codificarán esa dirección. La dirección servirá para ir a buscar un dato a la memoria, o para acceder a una posición y dejar allí el resultado de un cálculo.
3. Normalmente, luego de cumplir una instrucción, el MCBE continúa con la que se encuentre en la posición siguiente en la memoria. Sin embargo, ciertas instrucciones pueden alterar esa rutina. Las **instrucciones de salto** (JMP, JZ) sirven para desviar el curso de la ejecución. En estos casos el argumento representará un **desplazamiento**, y será interpretado como un entero con signo, en representación **complemento a dos**. Un desplazamiento es una cantidad de bytes que deben sumarse o restarse al PC, para **transferir el control** a una posición diferente a la siguiente.

Ciclo de instrucción

El **ciclo de instrucción** es la rutina que continuamente ejecuta el MCBE, leyendo y ejecutando las instrucciones del programa almacenado.

Al inicio de la operación, la máquina comenzará leyendo la posición 0, interpretándola como una instrucción y ejecutándola, según la especificación del ciclo de instrucción. El resto del comportamiento de la máquina depende de qué secuencia particular de instrucciones y datos (es decir, qué **programa**) haya preparado el usuario en la memoria.

El ciclo de instrucción se realiza continuamente hasta encontrar una instrucción **HLT**, y siempre de la misma manera:

- Se carga en el registro IR la instrucción cuya dirección está en el registro PC.
- Se decodifica la instrucción.
    3. La máquina examina los tres primeros bits del IR, identificando de **qué instrucción** del conjunto de instrucciones se trata.
    4. El resto de los bits, cuando corresponda, se utilizan como argumento de la instrucción, representando **una dirección o un desplazamiento** según se trate.
- Se **ejecuta** la instrucción. Cada instrucción tiene un efecto determinado sobre los registros o la memoria, que se detalla en la tabla adjunta.

Luego de la ejecución de la instrucción, y según cuál haya sido esa instrucción, los registros tienen posiblemente otros valores y ha ocurrido, posiblemente, algún efecto sobre la memoria.

Con ese nuevo estado de la máquina, el MCBE se dirige a la siguiente instrucción a ejecutar.

Detalles operativos del MCBE

6. La Unidad de Control de la máquina MCBE posee dos registros especiales, llamados **PC** (por *Program Counter*, Contador de Programa[32]) e **IR** (por *Instruction Register*, Registro de Instrucciones[33]).
   1. La función del PC es contener la dirección de la próxima instrucción a ejecutar.
   2. El IR contiene el valor de la última instrucción que se ha leído de la memoria.
7. La Unidad Lógico-Aritmética de la máquina posee un registro especial llamado **A** (por *Acumulador*).
   1. El acumulador es un lugar de trabajo para efectuar aritmética binaria, y sirve de zona de comunicación entre los registros y la memoria.
8. La máquina tiene **32 posiciones** de memoria. Cada posición aloja un byte de información.
9. En la memoria se distinguen dos posiciones especiales, con direcciones 30 y 31. Estas posiciones sirven para Entrada/Salida, es decir, para comunicación de la máquina con otros dispositivos.
   1. La posición 30 es de sólo lectura, y sirve para ingresar datos (Entrada) a los programas. Cuando un programa ejecuta una instrucción de lectura de la dirección 30, el programa se detiene hasta que el usuario de la máquina ingrese un dato.
   2. Inversamente, la posición 31 es de sólo escritura. Cuando se escribe un dato en la posición 31, el programa se detiene hasta que el dato sea recogido por un dispositivo de visualización. Ese dispositivo se encargará de emitir el dato (Salida) para que pueda verlo el usuario.
10. Cada vez que un valor se copia del acumulador A a una posición de memoria B, el valor de A no cambia. El valor anterior de B se pierde y la posición B pasa a contener un valor igual al de A. Inversamente cuando se copia un valor desde una posición de memoria al acumulador.
11. La máquina puede cargarse con un programa escrito por el usuario, y a continuación este programa se ejecuta. Al momento previo a la ejecución de un programa, todos los registros están inicialmente en 0.

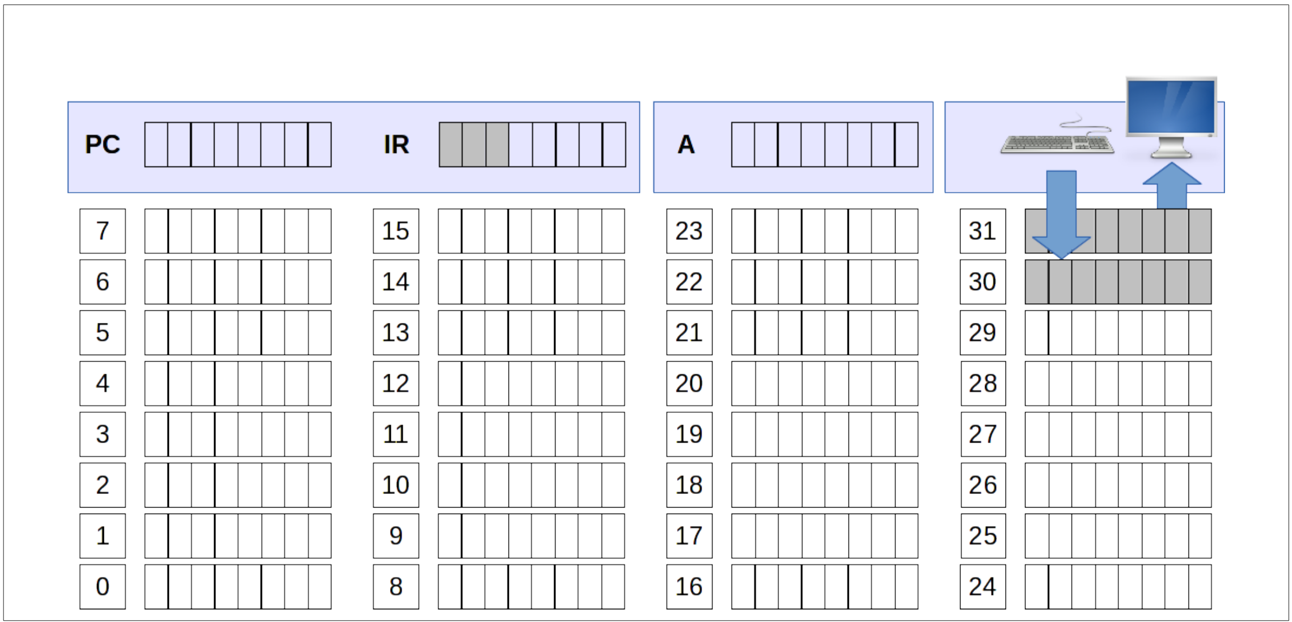
Preguntas

- ¿Cuál es la dirección de la primera instrucción que ejecutará la máquina?
- El MCBE, ¿puede encontrar una instrucción que no sea capaz de decodificar?
- Supongamos que hemos almacenado en la posición 14 un dato numérico que representa la edad de una persona. ¿Qué pasa si en algún momento de la ejecución el PC contiene el número 14? ¿Qué pasará si esa persona tiene 33 años? ¿Qué pasará si tiene 65? ¿Y si tiene menos de 20?
- ¿Qué pasa si el programa no contiene una instrucción HLT?
- ¿Podría aumentarse la capacidad de memoria del MCBE? ¿Esto requeriría algún cambio adicional a la máquina?

32 http://es.wikipedia.org/wiki/Contador_de_programa
33 http://es.wikipedia.org/wiki/Registro_de_instrucci%C3%B3n

- ¿Cómo se podría aumentar la cantidad de instrucciones diferentes del MCBE? ¿Esto tendría algún efecto sobre la longitud de los programas que puede correr la máquina?

Diagrama estructural del MCBE



Conjunto de instrucciones

Instrucción	Cód.	Efecto sobre memoria y registros	Efecto sobre el PC
LD <dirección>	010	El argumento se trata como una dirección. El contenido de esa dirección se copia en el acumulador.	Se incrementa en 1.
ST <dirección>	011	El argumento se trata como una dirección. El contenido del acumulador se copia en esa dirección.	Se incrementa en 1.
ADD <dirección>	100	El argumento se trata como la dirección de un dato, que será sumado al acumulador.	Se incrementa en 1.
SUB <dirección>	101	El argumento se trata como la dirección de un dato, que será restado al acumulador.	Se incrementa en 1.
JMP <desplazamiento>	110	Salta <desplazamiento> bytes. El argumento se trata como un desplazamiento, es decir, un entero con signo.	El desplazamiento será sumado al PC.
JZ <desplazamiento>	111	Salta <desplazamiento> bytes en forma condicional. El argumento se trata como un desplazamiento, es decir, un entero con signo.	El desplazamiento será sumado al PC únicamente en caso de que el acumulador contenga un valor igual a 0.
HLT	001	Detiene la máquina. Los registros y la memoria quedan con el último valor que recibieron.	
NOP	000	No ejecuta ninguna acción. La instrucción no tiene ningún efecto sobre el acumulador ni sobre la memoria.	Se incrementa en 1.

Ejemplos de programas MCBE

Los ejemplos siguientes se dan en la notación **posición / mnemotécnico o dato / argumento / contenido binario.**

Ejemplo 1. Leer un dato en la posición 4, sumarle el contenido de la posición 5 y escribirlo en la celda 6.

0	LD	4	01000100
1	ADD	5	10000101
2	ST	6	01100110
3	HLT		00100000
4	99		01100011
5	2		00000010

El efecto sobre la memoria de este programa será:

6	101		01100101

¿Qué pasaría si no estuviera la instrucción HLT de línea 3?

Ejemplo 2. Leer un dato del teclado, sumarle el contenido de la posición 5, restarle el contenido de la posición 6 y escribir el resultado por pantalla.

0	LD	30	01011110
1	ADD	5	10000101
2	SUB	6	10100110
3	ST	31	01111111
4	HLT		00100000
5	18		00010010
6	3		00000011

Ejemplo 3. Leer dos datos del teclado y escribir su suma por pantalla.

0	LD	30	01011110
1	ST	6	11100110
2	LD	30	01011110
3	ADD	6	10000110
4	ST	31	01111111
5	HLT		00100000
6	0		00000000

Ejemplo 4. Implementar la función y = 3x-2.

0	LD	30	01011110
1	ST	7	10000111
2	ADD	7	10000111
3	ADD	7	10000111
4	SUB	8	10101000
5	ST	31	01111111
6	HLT		00100000
7	0		00000000
8	2		00000010

Ejemplo 5. Leer un dato del teclado, restarle cuatro veces el contenido de la posición 7, y escribir el resultado por pantalla.

0	LD	30	01011110
1	SUB	7	10000111
2	SUB	7	10000111
3	SUB	7	10000111
4	SUB	7	10000111
5	ST	31	01111111
6	HLT		00100000
7	18		00010010

Ejemplo 5. Imprimir 10 veces el dato situado en la posición 9.

0	LD	11	01001011

1	JZ	7	11100111
2	LD	9	01001001
3	ST	31	01111111
4	LD	11	01001011
5	SUB	10	10101010
6	ST	11	01101011
7	JMP	-7	11010111
8	HLT		00100000
9	2		00000010
10	1		00000001
11	10		00001010

Ejemplo 6. Leer un dato del teclado, restarle seis veces el contenido de la posición 16, y escribir el resultado por pantalla. Objetivo similar a un ejemplo anterior, pero diferente programa. La ventaja de este programa es que la operación de resta se puede hacer una cantidad cualquiera de veces, con sólo modificar el valor de la posición 14.

0	LD	30	01011110
1	ST	13	01101101
2	LD	14	01001110
3	JZ	7	11100111
4	SUB	15	10101111
5	ST	14	01101110
6	LD	13	01001101
7	SUB	16	10110000
8	ST	13	01101101
9	JMP	2	11000010
10	LD	13	01001101
11	ST	31	01111111
12	HLT		00100000
13	0		00000000
14	6		00000110
15	1		00000001
16	7		00000111

Ejemplo 7. ¿Qué hace este programa? ¿Qué problema presenta?

0	LD	30	01011110
2	ADD	6	10000110
3	ST	31	01111111
4	JMP	-2	11010010
5	HLT		00100000
6	1		00000001