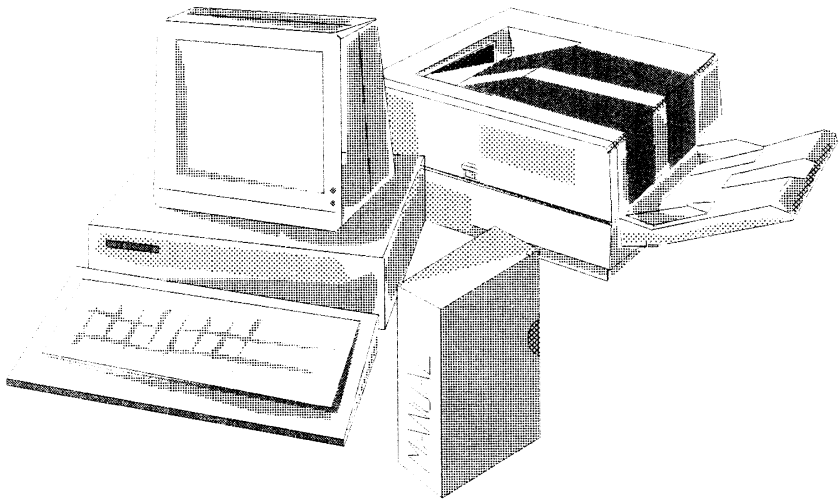
DATA BUS

PUBLICACION EDITADA POR LA ASOCIACION JUVENIL PARA LA INFORMATICA VIGUESA

NUM. 1 ABRIL AÑO 1990

Dep. Legal VG-87-90



DATA BUS
Núm. 1
Abril
Año 1990

DIRECTOR
Simón Gómez Cabanelas

REDACTORES
Jesús Cea
Nacho Agulló
Eduardo Cunha

COLABORADORES
Telmo Lago
Pablo Lobariñas

DISEÑO
Eduardo Cunha

EDITA
Asociación Juvenil para la Informática Viguesa
c/ Estrada, 36- 4 D
Tel: 20 67 18

Depósito Legal: VG-87-90
VIGO, Marzo de 1990

Publicación íntegramente realizada con equipo de autoedición ATARI

INDICE

EDITORIAL

Amigo lector:

Tienes en tus manos el primer ejemplar de una publicación que tratará sobre las múltiples facetas del mundo de la Informática. Intentaremos abordarlas con la mayor profundidad posible, procurando, al mismo tiempo, hacerlo de una forma amena.

Esta publicación pretende responder al gran interés que la Informática despierta entre los jóvenes en estos tiempos. Por supuesto, siempre estaremos abiertos a cualquier tipo se sugerencia o colaboración, sobre cualquier tema relacionado con la Informática que contribuya a mejorar esta publicación número a número.

En lo que respecta al presente, daremos comienzo a una serie de secciones que tendrán su continuidad en posteriores ediciones. Estas secciones serán

- ALGORITMOS, que presentará varias formas de realizar tareas concretas, como, por ejemplo, ordenación de palabras;

- GUIA DEL PROFANO, que desde un punto de vista muy básico intentará introducir al lector en este mundillo;

- MUSICA POR ORDENADOR, con los avances en una aplicación poco conocida por los usuarios;

- CURIOSIDADES INFORMATICAS, presentando programas útiles, anécdotas,...

- GLOSARIO, que incluirá los términos de significado más oscuro para los lectores que menos conozcan el tema;

- Y una HISTORIA DE LA INFORMATICA, con los hechos más significativos desde su existencia.

Además de estas secciones fijas, en todos los números se podrán encontrar artículos sobre las más diversas temáticas. Y más adelante se incluirá otra sección, donde tendrá cabida cualquier colaboración que se nos envíe, y trataremos de contestar las posibles consultas que lleguen a nuestras manos sobre el tema.

Esperamos que le guste, y hasta el próximo número.

LA REDACCION

GUIA DEL PROFANO

Por E. Cunha

Esta serie de artículos nace con la idea de que todos sepamos algo más sobre el mundo de la Informática, aunque ya tengamos nuestra experiencia y conocimientos en el tema. Siempre se pueden encontrar datos o anécdotas interesantes sobre asuntos que creemos dominados.

El artículo de este número hablará de los fundamentos y funcionamiento de un ordenador en general. En sucesivos números hablaremos de lo que rodea a la maquina, como son los periféricos, la programación, lenguajes, etc, y el uso que se le puede a un ordenador por pequeño que sea.

LOS FUNDAMENTOS DE UN ORDENADOR

Sentados ante un ordenador, no solemos preguntamos cómo funciona. Trabajamos con él y nos basta. Y verdaderamente es poca la gente que lo conoce por dentro; no es sólo el saber que esté repleto de integrados y resistencias, sino cómo es que esos componentes hacen que podamos pilotar un F-18, o crear imágenes en movimiento. Para ello hay que profundizar hasta olvidarnos del ordenador.

Está claro que el ordenador trabaja mediante electricidad, y es esa electricidad en forma de impulsos la que recorre las múltiples pistas del circuito impreso hacia los circuitos integrados o chips. De esta forma, surge el primer problema. ¿ Cómo podríamos representar un número mediante esos impulsos eléctricos ? La respuesta es simple. Nosotros trabajamos normalmente en base 10, necesitamos diez cifras (del 0 al 9) para crear números. Si reducimos la base a la mínima, la base 2, sólo necesitamos dos cifras: el 1 y el 0, el SI o el NO, el impulso o la tensión nula.

A partir de ahora, distinguiremos dos formas de tratar una cifra: Por un lado, la representación de esa cifra como caracteres o símbolos, y por otro la cifra en sí. Esta sería la forma que tiene el ordenador de representar símbolos: las combinaciones de 1 y 0 que hagamos determinarán las letras y el resto de caracteres. Como ejemplo, dispongamos de 16 huecos para rellenar con 1 ó 0, y así definir las letras. Podríamos poner un 1 en las posiciones 16 y 1 para tener la A, o en las 16 y 2 para la B, etc. Al final habría letras y signos representadas por cadenas diferentes de dieciséis 1,0. En grandes rasgos, sería este el sistema para caracteres. Para las cifras, en cambio, se usa el sistema de numeración en base 2 (binario) tal y como se entiende, con equivalencias en la base 10 segun la tabla:

DECIMAL - BINARIO

1	1	7	111
2	10	8	1000
3	11	9	1001
4	100	10	1010
5	101	11	1011
6	110	12	1100

La unidad mínima de información es, segun vimos, un 1 o un 0, cifra a la que llamaremos BIT. Para simplificar las operaciones, se definen grupos de bits: BYTE, de 8 bits; PALABRA, de 16 bits; PALABRA LARGA, de 32 bits.

Las formas reales de codificación de los caracteres o símbolos no difieren en gran medida de lo expuesto en el ejemplo. Hay algunos estándars, como el ASCII, que almacena los códigos en un byte, (es el más usado), aunque muchos fabricantes trabajen con su propio sis-

tema. En realidad, esta cuestión no tiene mayor importancia, pues sólo indica que, cuando escribimos la letra S, por ejemplo, el ordenador entienda un número u otro. Es cuestión sólo de su funcionamiento interno. Al usuario, generalmente, le da igual.

Teniendo ya la representación binaria, tendremos la solución del problema. Los bits abrirán o cerrarán circuitos, cambiarán estados de polarización, entrarán en puertas lógicas... (una puerta lógica es un pequeño circuito que necesita la entrada de dos impulsos, y devuelve un bit que es resultado de ambos).

En lo que se refiere a la cuestión del almacenamiento en memoria de esos números (voy a explicar un tipo muy básico de memorias), hay que recurrir al electromagnetismo. Decir, a modo de breve explicación, que la memoria es una inmensa malla con múltiples núcleos, de un material que se magnetiza con muchísima facilidad en corto tiempo, y que conserva ese magnetismo. Estos núcleos estan recorridos por conductores, y es el impulso eléctrico, al pasar por los mismos, el que puede cambiar la polarización del núcleo, determinando si hay un 1 ó un 0 almacenado.

La memoria, al estar dispuesta como una malla, facilita la grabación o almacenamiento de datos en núcleos determinados, ya que cada hilo de la malla atraviesa una fila de núcleos, (en vertical y horizontal), pero dos hilos coinciden sólo en un núcleo determinado. Con mandar impulsos por esos dos hilos se logra la selección, y, por tanto, la nueva polarización (que es independiente de la que había antes en el núcleo).

COMPONENTES ESENCIALES

El primer componente, sin el que, obviamente, el ordenador no existiría, es la Unidad Central de Proceso o CPU. Es el circuito encargado de las entradas y salidas, de la manipulación de datos, y de ejecutar las instrucciones que se le asignen. Como controladora de las entradas y salidas, recibe señales, rige las transferencias entre sí misma u otros circuitos con la memoria,etc. Como manipuladora de datos, toma grupos de bits de longitud fija: bytes, palabras o palabras largas, y con ellos realiza los cálculos aritméticos; esencialmente, sumas, restas, movimientos, rotaciones y comparaciones. (De estas operaciones que puede hacer el procesador hablaremos en próximos números). Decir del procesador que, dependiendo de la longitud de cadena de bits con que trabaje, se puede clasificar como 8 bits, 16 bits o 32 bits (hasta ahora). Decir también que el manejar números de, por ejemplo, 8 bits (1 byte), no significa que no pueda usar números mayores. Se pueden definir los números como varios bytes, dos por ejemplo, y tener como cifra máximo 65535 (en binario 1111111111111111), en vez de 255 (en binario 11111111).

Otro componente del ordenador es la memoria. Fundamentalmente, es de dos tipos: memoria de solo lectura, ROM, y de lectura-escritura, o RAM. Estas últimas no resisten la falta de electricidad, y su contenido de borra al apagar el ordenador. Normalmente, mientras el ordenador funciona, la CPU se encarga de "refrescarla" cada poco tiempo (milisegundos). La otra memoria, ROM, resiste este inconveniente, y además no se puede modificar por el ordenador. Sólo se puede acceder a ella para leer lo que contiene. Es de esta memoria de la que la CPU toma el programa que ha de seguir al inicializarse el ordenador, pues la otra esta totalmente vacía en ese momento (obviamente). Ese programa, normalmente, comprueba los circuitos y pasa el control al sistema operativo, que es realmente quien nos da el control de la máquina. Este sistema es un juego de instrucciones que podemos usar para cargar programas en memoria, ejecutarlos, etc. En algunos ordenadores, sobre todo de 8 bits, el sistema es en realidad un intérprete de BASIC u otro lenguaje desde el que también se puede controlar la máquina.

Los dispositivos de almacenamiento externo de datos nos son prácticamente imprescindibles (si cada vez que se fuera a usar un programa largo lo tuviésemos que teclear, el uso de un ordenador llegaría a, en vez de facilitar las cosas, empeorarlas mucho). Estos dispositivos se controlan desde la CPU. Nos permiten la grabación en un sistema externo de parte de la memoria del ordenador, para más tarde poder introducirla (o cargarla) otra vez en la memoria, para trabajar con ella. Para dar una idea general del proceso y lo que ha de hacer el ordenador, pondremos un pequeño ejemplo. Para ello, introduciríamos una orden en el sistema operativo comunicando a la CPU que deseamos cargar en memoria un determinado fichero de datos, que podemos tener en un disco. El sistema operativo traduce esa orden a un código numérico, que el ordenador toma en binario. La CPU, más tarde, interpreta la orden (cargar de disco). Es de notar que la máquina lo toma todo como códigos numéricos en binario, todo unos y ceros: El mismo sistema operativo, para el ordenador, es una larga lista de instrucciones de unos y ceros que ocupan sitio en memoria y que, como parte de su misión, ha de ejecutar por orden.

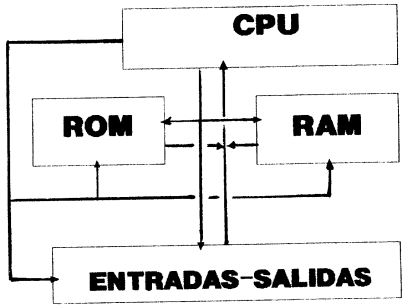
Existen muchas maneras de almacenar información en el exterior del aparato, pero el proceso es el mismo. Volvemos a lo de siempre al decir que se graba el contenido de la memoria, unos y ceros. El que estemos grabando una pantalla gráfica, un dibujo, no quiere decir que almacenemos una foto. Se almacenan todos los unos y ceros que la representan. Y es algo que los dispositivos de almacenaje agradecen. No habrá cosa más sencilla para ellos que guardar solo dos tipos de datos posibles. Y eso se nota en su variedad, casi tantas como formas que tenemos de hacerlo: Las cintas de casette (un agudo es un 1, un grave un 0), las antiguas tarjetas perforadas (una perforación un 1, el contrario un 0),las unidades de disco en sus múltiples tipos...

Otras partes del ordenador, ya tomadas como periféricos (lo que se conecta al ordenador, ya que podemos tomar el mismo como una CPU y su memoria), son la pantalla de visualización, salida de datos, y el teclado, la entrada. En ambos casos, su funcionamiento se puede resumir brevemente.

El teclado es, normalmente, independiente de la CPU. Su misión es enviarle los códigos de las letras que se pulsan. Como decíamos antes, no envía caracteres, sino el número que los representa, de acuerdo al sistema de codificación (¿ se acuerda del ASCII ?).

La pantalla de visualización es algo más complicado de explicar: Un integrado especialmente dedicado a ello lee, cada muy poco tiempo (50, 60 ó incluso 70 veces por segundo, dependiendo de la frecuencia de barrido vertical del monitor) el contenido de una porción de la memoria reservada para ese cometido ("memoria de pantalla"). El mismo circuito traduce esos datos a "puntos" de la pantalla y los representa como tales. En otras palabras, que si vemos una línea en pantalla es porque hay una fila de unos en la "memoria de pantalla" que,interpretados como puntos, son impresos en pantalla. Este proceso se complica a veces por diversos motivos, pero en esencia es siempre lo mismo.

Y en la próxima página comenzamos la segunda parte de esta serie de artículos, los sistemas de numeración y su aritmética.



SISTEMAS DE NUMERACION

Como se ha comentado, el ordenador no trabaja en base 10, sino en base 2. En la tabla anterior se pueden ver las primeras equivalencias. Podemos observar que, mientras en base 10 añadimos una nueva cifra a la izquierda cada diez unidades, y otra más a la izquierda cada cien, en binario ocurre algo parecido. Los unos se desplazan, pero no en potencias de diez, sino, obviamente, de dos. Tendremos valores, de derecha a izquierda, de 1, 2, 4, 8, 16, 32, etc. según las posiciones que ocupe el 1 (1, 10, 100, 1000,...). O sea, todas las potencias posibles de dos (por algo se le llama base 2).

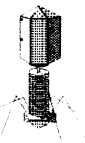
Si ha leído el artículo desde el comienzo, puede que ahora se esté preguntando qué pasa con el signo del número. Tomemos como ejemplo un número de 8 bits, un byte. Ese byte contendrá, como máximo, el número 255 (11111111= 2^8 =256, pero el primer número es el cero). Entonces podríamos designar uno de los bits para el signo, por ejemplo el último. Si ese bit es 0, el número formado por los 7 bits restantes (1111111 = 2^7 =127, si de nuevo incluimos el cero) será positivo, y si es 1 será negativo. Aquí, entonces, aparece un nuevo concepto: El complemento a dos, que será el equivalente negativo de un número positivo. Se calcula, en el caso de un byte, restando 256 menos el número positivo. Nótese que el numero 11111111 sería el numero -1 (los negativos cuentan hacia abajo) y 00000001 sería el número 1. El cambio de signo lo situaríamos en el 01111111 que sería positivo (127) y el siguiente, negativo, 10000000 (-128, porque el cero se toma positivo; se va de 0 a 127 y de -1 a -128).

La otra cuestión pendiente, además de la aritmética binaria, (de la que se hablará en el próximo número), es la del número máximo. Hemos dicho que los procesadores o CPU tienen, entre sus características, la de que trabajen con bytes, palabras o palabras largas. Entonces, si operan con, por ejemplo, bytes (es el más sencillo), su número máximo sería el 255. Pero sabemos que no es así, pues hay programas de contabilidad que elevan su máximo a muchos millones. La respuesta a esto es el trabajar con varios bytes. Pongamos que lo hacemos con dos, como anteriormente. Si vamos sumando al primero de ellos una unidad, partiendo de cero, sucesivamente, llegaremos a que el numero indicará 255. El siguiente número sería sumar la unidad al segundo byte y comenzar de nuevo, poniendo a cero el primer byte. Tendremos un contador que adelanta una unidad cada vez que el otro llega a 255. Como ese contador también es un byte, se pueden acumular numeros de 256 x 256 = 65536. Si usásemos otro byte más, volveríamos a multiplicar por 256. El máximo sería, con tres bytes, 24 bits, 16777216, más que suficiente para muchas cosas. Por supuesto, cuantos más bytes resta algo de velocidad al proceso de datos, pero al ser milisegundos no se aprecia lo más mínimo. Decir que, en máquinas de 8 bits, es usado el número de dos bytes; al que cuenta las unidades se le llama byte bajo, y al que acumula las veces que se llega a 255, byte alto.

Existen otras bases usadas al tratar con ordenadores. La base 16, por ejemplo, es muy útil porque cada una de las cifras que forman un número se mete, exactamente, en 4 bits. De esta forma un número en hexadecimal (base 16) coincide con su representación binaria. (Si no lo ve claro, intente tomar dos contadores de 4 bits como si fueran byte alto y byte bajo, con el bajo representando la cifra de las unidades).

Para terminar, a partir de ahora representaremos los números binarios precedidos del símbolo "%", y los hexadecimales con "#", para distinguirlos de los decimales (10, %10, #10).

El próximo artículo hablará sobre la aritmética binaria y comenzará la descripción de los periféricos, en especial las unidades de disco y las impresoras.



VIRUS EN TU ORDENADOR

Por S. Gómez

Si buscamos en el diccionario la palabra VIRUS, nos revela que un virus es un microorganismo no celular microscópico capaz de multiplicarse Unicamente en el interior de una célula viva. Para los virus informáticos o de ordenador, esta definición no es acertada, aunque su nombre quizás lo sea, ya que aunque no sean virus biológicos tienen sorprendentes semejanzas con estos. Mientras que un virus biológico infecta una célula, el virus de ordenador lo hace con un programa, un diskette, un disco duro, etc. Un virus biológico posee un código genético (una cadena de ADN); en el virus informático el código es un conjunto de instrucciones, es decir, un programa generalmente de poca longitud para poder pasar mejor desapercibido, y que el usuario no se dé cuenta que está contaminado hasta que el virus se dé a conocer, ya que cuanto más grande sea el virus, mayor longitud de memoria necesita o mayor espacio en disco para estar alojado,con el consiguiente riesgo de que sea detectado con mayor facilidad, o lo que es lo mismo; es muy distinto que un virus ocupe unos pocos bytes que unos cuantos Kbytes.

Cuando se ejecuta el virus, lo que puede pasar después es tan inesperado y sorprendente como decirle "algo maravilloso está pasando en tu ordenador", pasando por inutilizar los discos; aunque puede ser también que se trate de un virus de broma y que te diga que están inutilizados y no sea cierto; después el usuario los formatea y borra toda la información que había almacenado por una simple broma. También puede ser que esté tranquilamente echando una partida a un juego cualquiera y, mientras, el virus se esté encargando de formatear el disco duro sin que se entere, hasta que quiera acceder a él y se dé cuenta de que no hay nada en el disco.

Se puede pensar que los virus son algo reciente, pero no es así. Ya en la década de los cincuenta, los científicos estudiaban unos programas a los que denominaban "Autómatas Autoalterables". Unos años más tarde, concretamente en la década de los sesenta, los investigadores de la casa BELL disputaban batallas entre virus en un juego llamado "Guerra Total". Dicho juego consistía en fabricar un virus lo más pequeño posible para infiltrarse en los virus enemigos y fuera capaz de destruirlos sin que se dieran cuenta.

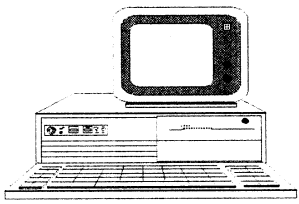
El término Virus, tal como lo conocemos, ya se utilizaba en los setenta en novelas de ciencia ficción. También son de esta misma época las investigaciones de la casa XEROX que enseñaron un pequeño programa que tenía la capacidad de autoduplicarse. Éstos y otros investigadores tenían la confianza de usarlos como marcadores y como ahorradores de tiempo en sus programas.

Pero es en la década de los ochenta donde hacen su boom. Han caído en manos de una serie de hábiles programadores que los han hecho pasar de ser una herramienta de ayuda en un instrumento maléfico para encauzar sus perversiones. Estos programadores los distribuían a través de revistas por todo el país. Algunos de ellos se dedicaban a crear nuevas variantes más peligrosas y maléficas; así se cree que en los Estados Unidos debe haber un centenar de ellos circulando por ahí. Los virus incluso han llegado a aparecer en tiras de comics y en la televisión, más concretamente en la serie "Star Trek".

También a la Unión Soviética han llegado los virus; investigadores del Instituto de Sistemas Programados diseñaron un virus denominado "Lágrimas

Verticales" y que provoca que todos los caracteres caigan hasta el extremo inferior de la pantalla formando un montón. Otro virus soviético es el "Agujero Negro" que obliga a la CPU a realizar operaciones sin sentido, lo que hace perder mucho tiempo de trabajo. También produçe un punto negro en un extremo de la pantalla, de ahí su nombre.

El famoso virus israelí "Viernes 13" saltó a la luz cuando los relojes de los ordenadores marcaban la fecha del 13 de Octubre del 89. Se estimó que unos 100.000 ordenadores estaban contagiados con este virus. Se temía que el fatídico día llegara; incluso por esas fechas salieron en los Telediarios noticias acerca de este virus que mantuvo en vilo a muchas empresas. Se descubrió una vacuna, el "Sábado 14", que destruía el virus, pero por desgracia sólo llegó a un 10% de los ordenadores infectados. Todos temían perder la información, en muchas ocasiones, vital. Y llegó el tan anhelado día, con el mundo pendiente de sus ordenadores, sin quitar ojo a la pantalla y comprobando que hasta el último bit de información residía en su sitio. Pasó el día y no sucedió nada. De repente todo el mundo se olvidó de él. Todos pensaron que tanto revuelo para después nada. Unos días después se supo que varias empresas habían perdido información vital, pero eran mínimas. Todo aquel revuelo de lo que se había dado en llamar SIDA informático aparentemente no sirvió para nada, solamente para que todas las empresas y usuarios extremaran las medidas de protección de sus datos y programas contra una cosa que empezó siendo una broma y acabó siendo una plaga.



No todos los virus son malignos, como pudiera parecer en un principio, también los hay benignos, aunque son los menos. Sirva de ejemplo un virus del Apple Macintosh que imprimía en la pantalla un mensaje de paz a todos los usuarios de Mac del mundo.

CLASES DE VIRUS

1. LOS QUE SE ESTABLECEN EN EL BOOT DEL DISKETTE:

El Boot del diskette ó sector de arranque es el primer sector que lee el ordenador cuando se introduce un disco en la unidad. Es el sector que contiene el conjunto de intrucciones iniciales que recibe el ordenador y cuya misión es preparar el camino para poder instalar el sistema operativo que controla todas las operaciones del ordenador y que sirve para que el usuario se pueda entender con la máquina; Aparte también controla los periféricos.

Una parte de un virus puede situarse en ese boot sector. La otra parte del virus se sitúa en cualquier otro sitio del disco, ded forma que, normalmente, cuando el usuario intente grabar algo en ese sector, el ordenador dirá que el sector está dañado. Cuando se inserta un disco "contagiado" en el ordenador, lo primero que hace es ejecutar el boot sector; en este caso, cargar el virus en la memoria y ejecutarlo, pudiendo después infectar otros discos, programas, borrar o alterar datos,... Aunque también puede asegurarse su descendencia grabando una copia de sí mismo en los discos que podamos introducir más tarde y, entonces y sólo entonces, autodestruirse.

2. LOS QUE SE ESTABLECEN EN FICHEROS EJECUTABLES:

Esta clase de virus se mezcla con un

fichero de programa, y al ejecutar dicho fichero también se ejecuta el virus, estableciéndose en una zona de la memoria lo más oculta posible, para más tarde cumplir su "misión". Cualquier cosa es posible, todo depende de la imaginación de sus programadores, llamados "hackers", "worms", punkies cibernéticos, etc.

3. CABALLO DE TROYA:

Consiste en un virus que está camuflado por algo que estás viendo en la pantalla, el nombre de un programa que debería estar en ese disco, con su correcta longitud, pero que no es tal. Es una simple tapadera de lo que realmente se ha cargado en la memoria. Se distingue este virus del que se establece en ficheros ejecutables en que el "caballo de Troya" es un programa diseñado para contener un virus dentro, está relacionado con el virus; en cambio, el otro es un virus cualquiera que se mezcla con un programa, y no tienen nada que ver entre ellos. El "caballo de Troya" no es un virus autocopiable. Sólo se reproduce al hacer una copia de él por error, creyendo que es el verdadero programa.

COMO LUCHAR CONTRA LOS VIRUS

1. Intentar conseguir una vacuna, es decir; un antivirus lo suficientemente bueno y eficaz para que acabe con la mayoría. De todas formas, hay que tener mucho cuidado porque podría ser el propio antivirus un "caballo de Troya", y que en vez de aniquilar al virus esté inoculando otro. Si puede ser, intentar conseguir versiones actualizadas de ese antivirus, ya que los virus son cada vez más "inteligentes", y despistan o pasan desapercibidos por muchos antivirus.

2. Cada cierto periodo de tiempo, pasar por el "médico" (antivirus) los discos, pues pudiera ser que algún disco que le fuese prestado por alguien estuviere contaminado, y el virus se esté extendiendo por todos ellos. También se deduce de esto que hay que controlar todos los discos que llegan del exterior. Cuesta 5 minutos, pero podría ahorrar horas de examinar todos los discos por si se hubiera filtrado alguno.

3. De detectar algún virus en la memoria hay que apagar el ordenador durante unos segundos, ya que el botón de "Reset" no es suficiente para ellos.

Existen 3 clases de antivirus:

FILTRO, que previene las infecciones. Siempre está residente en la memoria y está atento a cualquier intento de infiltración vírica.

DETECCION DE INFECCION, que periódicamente echa un vistazo al sistema y va comprobando los datos con los obtenidos durante la supervisión. Si algo cambiase daría la alarma.

DESTRUCTORES, que detectan y eliminan los virus más comunes. Son útiles cuando se trata de buscar protección de virus ya conocidos, pero fracasan ante las nuevas mutaciones y variaciones de los virus con capacidad de automodificarse, o, simplemente, nuevas versiones de los mismos.

CURIOSIDADES SOBRE LOS VIRUS

-Existe una tarjeta interna para los PC's, denominada "INMUNETEC PC' que comprueba la presencia de un virus en el sistema.

-Existe una empresa que alardea haber construido un ordenador a prueba de virus. A este ordenador lo han denominado el "Sistema Inmune" (¡Recuerden el "Titanic"!).

-Existe una empresa americana que sólo se dedica a la comercialización de antivirus. Esta empresa se denomina "Centro de Control de las Enfermedades Informáticas".

ALGORITMOS

TECNICAS DE COMPRESION

Por J. Cea

INTRODUCCION

El estudio y desarrollo de algoritmos de compresión de algo fascinante. Intentar reducir el tamaño de un programa, buscando la forma más ventajosa de hacerlo es una tarea que depara muchas satisfacciones. Por mi parte, he programado algunos algoritmos muy conocidos y he desarrollado algunos que considero novedosos. Dado el interés que parece despertar el tema, me he decidido a escribir este resumen de mi trabajo en este campo. Como puede verse, he empezado por los métodos más sencillos, complicándolos cada vez más hasta llegar a sistemas bastante extraños pero eficientes.

Para leer este artículo no se precisa una preparación especial aunque recomiendo tener algunas nociones léxicas básicas: bit, byte, buffer, flag, etc.

1. ¿ COMO SABER DONDE SE HA COMPRIMIDO Y DONDE NO?

No basta con comprimir un programa, sino que es preciso que el descompresor sea capaz de tratarlo para devolvernos luego el programa original. El programa descompresor debe saber qué bytes debe dejar inalterados y cuáles debe descomprimir. Esto se puede lograr de varias formas:

1. FLAG: Al principio de una secuencia comprimida ponemos un valor (flag) determinado, pero variable en cada caso, conocido por el compresor y el descompresor. El problema es que ese flag no debe coincidir con otro valor de la memoria, para evitar confusiones.

En un ordenador de 64 Kbytes hay 2^{16} valores posibles, si cada célula de memoria tuviera un valor distinto de todas las demás. Esto, en la práctica, es imposible. Siempre hay algún valor que se repite y, por lo tanto, algún valor que no está en ninguna célula de memoria. Este valor no utilizado puede ser usado como flag debido a que podemos estar seguros de que siempre que aparezca estará seguido por bytes comprimidos. Con este método precisamos 2 bytes para la bandera.

Sabemos que cuanto mayor sea la cabecera de compresión, más bytes deben ser comprimidos para que el método se rentable. Por ello, sería más ventajoso que el flag fuera de ún solo byte. Veamos esto:

Cuanto más largo sea el programa, menos posibilidades de encontrar un valor de 1 byte no utilizado pero, ¿qué pasaría si buscamos el byte menos utilizado y ponemos otro flag para indicar cuándo debe dejarse como está y cuándo marca una compresión? El sistema funciona según el método de compresión utilizado. Da unos resultados bastante aceptables con el algoritmo de series, donde una longitud de 0 puede marcar que el byte en cuestión debe dejarse como está. También depende de la longitud del fichero a comprimir. Veremos esto luego con más detalle.

2. PUNTERO: Este sistema es el que utilizo con el algoritmo de compresión por bloques. En vez de poner una secuencia conocida al principio, se usa una tabla de punteros con las direcciones de los bloques comprimidos. En su momento estudiaremos esto con más detalle.

3. CONTADOR: Este es un método muy utilizado para comprimir pantallas

gráficas. Tiene la ventaja de su sencillez y su extrema velocidad a la hora de descomprimir. El algoritmo de compresión es un tanto complicado pero la descompresión es sencillísima:

Cogemos el primer byte. Si es menor que 128 (positivo), copia los próximos n+1 bytes bytes sin modificar y vuelve a empezar. En el caso de que sea mayor que 127 (negativo), copia 2-n veces el byte que sigue. Una vez hecho esto, repite de nuevo todo el proceso.

Como puede verse, es un método sencillísimo.

1. ALGORITMOS DE COMPRESION DE DATOS

En esta parte estudiaremos algunas formas de compresión de datos (incluidas algunas variantes, claro). Algo que quiero hacer notar es que esto es sólo un somero resumen de los métodos que he utilizado y quiero destacar que, a veces, se desarrolla un algoritmo distinto para un caso particular dadas sus características propias. Con esto no solo quiero decir que a cada tipo de fichero le convenga un algoritmo distinto (no es igual comprimir una pantalla gráfica que un fichero de direcciones) sino que, a veces, se crea un algoritmo propio para un programa que, por sus características, así lo precise por la razón que sea. Normalmente esto se hace adaptando un sistema ya desarrollado al caso particular que interese. Esta forma de trabajar no es, en absoluto, imprescindible pero sí ventajoso en determinadas situaciones.

Así mismo, otro hecho destacable es que muchas veces se comprime un programa sucesivamente por distintos métodos, consiguiendo de esta forma las ventajas que cada uno tiene por separado. Esto no es siempre factible porque, normalmente, el super-descompresor es más grande de lo que se ha conseguido reducir el fichero en cuestión, creando así un programa más largo que el propio original.

Algo que hay que recordar es que no da igual el orden en que se aplican las sucesivas compresiones sino que es variable con cada programa . Sólo cabe probar en cada caso.

Por último, un punto importante es si el descompresor descompactará el programa sobre el original o si utilizará un área aparte. Esto, que puede parecer trivial, es un detalle absolutamente vital para la organización interna del fichero comprimido y, sobre todo, para la velocidad.

Pasemos ahora (por fin !!) a los algoritmos propiamente dichos.

1. ALGORITMO DE COMPRESION POR BYTE REPETIDO

Consiste en buscar un byte determinado, normalmente el 0, y comprimirlo si el número de veces que se repite es mayor de 2 ó 3, según se use el método de flag de 1 ó 2 bytes. El formato es algo así:

1. Flag de un byte: FLAG, LONG (en el caso de que long sea igual a 0, este byte debe saltarse)

2. Flag de dos bytes: FLAG LO, FLAG HI, LONG.

Como se ve, este método es uno de los más sencillos y rápidos, pero tiene el problema de que sólo comprime un byte determinado, por lo que se suele usar con pantallas gráficas, donde suelen abundar los ceros, y similares. Naturalmente, podemos escoger comprimir otro byte distinto de cero, dependiendo esto del programa.

2. ALGORITMO DE COMPRESION DE VALORES REPETIDOS

Es, básicamente, igual al anterior, pero soluciona el problema de éste dado que indica a continuación del flag qué byte se ha comprimido.La desventaja de este método es que sólo puede comprimir si se repite más de 3 ó 4 bytes, según el esquema siguiente:

1. Flag de un byte: FLAG, BYTE, LONG (si long es 0, el flag no marca compresión)

2. Flag de dos bytes: FLAG LO, FLAG HI, BYTE, LONG

Una importante variación de este sistema es tener DOS flags distintos. Uno de ellos marca la compresión de los ceros y el otro la compresión de los demás valores. Este es el método que más había utilizado hasta que descubrí otros mejores...

3. ALGORITMO DE COMPRESION DE SERIES

Los métodos vistos hasta ahora son los más conocidos pero no los mejores. Personalmente, consideraba bastante decepcionante que series de bytes como: 0, 1, 0, 1, 0, 1 ... no fueran comprimidas. Así que decidí buscar sistemas alternativos... Y los encontré.

El primero que desarrollé fue éste: Se seleccionan tres flags distintos: uno marca compresión de ceros (necesita más de 3 repetidos), otro marca la compresión de un byte cualquiera distinto de cero (necesita más de cuatro repetidos) y el tercer flag marca compresión de serie. El esquema es el siguiente: FLAG LO, FLAG HI, LONG (de la serie), REPETICIONES (cuántas veces se repite dicha serie), SERIE...

Por ejemplo: Supongamos el flag $2835 y la serie 0, 1 repetida 8 veces: $35, $28, $02, $08, $00, $01. Esta posibilidad de comprimir series enteras de bytes abre grandes posibilidades, pero no era suficiente.

¿Que pasaría si las series se repitieran, por ejemplo, en posiciones discontinuas y aleatorias de memoria?

¿Podríamos comprimir cosas como 1, 2, 3, 5, 1, 2, 3, 4?

Este afán de superación me llevó al siguiente algoritmo, el de compresión de bloques.

4. ALGORITMO DE COMPRESION DE BLOQUES

Este es el método que estoy utilizando actualmente aunque quizás lo cambie por el de compresión por bits. Es un algoritmo bastante complicado, así que sólo daré una somera descripción de su funcionamiento.

Para empezar, diré que éste método no utiliza la marcación por flags, sino que usa punteros para referenciar las secuencias comprimidas (ver al principio). Empieza cogiendo una pequeña parte del programa a comprimir (256 bytes) y comprueba si se repite en CUALQUIER otra parte de programa (ver figura 1). Si es el caso, borra los bloques repetidos pero guarda sus direcciones en forma de puntero. Naturalmente, se conserva una copia del bloque junto con los punteros a donde debe ser colocado e indicadores como longitudes, etc.

Una vez hecho esto, o cuando no encuentra repeticiones, coge otros 256 bytes de programa, los mete en el buffer de trabajo y repite todo el proceso. Todas estas operaciones se repiten hasta que se llega al final del programa. Cuando esto ocurre, decrementa la longitud del bloque (255, 254...) y vuelve a empezar hasta que el tamaño del bloque es demasiado pequeño para una compresión eficiente (unos 3 ó 4 bytes, según el caso).

La organización interna del programa una vez comprimido es algo así:

Nº total de compresiones, Longitud, Nº de compresiones de esa longitud, Bloque 1, Nº de compresiones de ese bloque, Punteros, Bloque 2..., Longitud, Nº de →

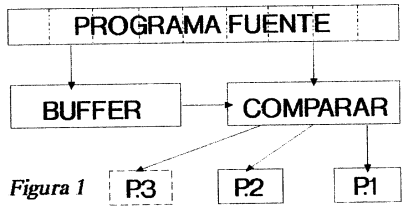


Figura 1

compresiones de esa longitud, Bloque X...

Como puede verse, es una técnica muy sofisticada que da unos resultados realmente impresionantes. Este algoritmo comprime absolutamente todo lo que se le ponga por delante, incluso el programa más desordenado e inconexo que podamos imaginar.

Seguramente ha llegado a la conclusión, querido lector, de que este método es lento. Y tiene razón. La compresión es extremadamente lenta y cuanto más largo es el programa, peor (un programa de 38Kb tarda unos 32 segundos), pero el resultado vale la pena (el programa anterior quedó reducido a 8'5 Kbytes). De todos modos, la descompresión de ese mismo programa, que es la tarea más común con creces (un programa se comprime una sola vez, pero se descomprime cada vez que lo queramos usar), apenas tardó 3 segundos. Claro que para llegar a estos tiempos tuve que recurrir a infinidad de trucos para optimizar el descompresor.

Como es obvio, este método puede UNIRSE con el algoritmo de series, dando unos resultados aún más increíbles si cabe (8'3 Kbytes).

Con estas cifras quizás se crea que se ha llegado ya al límite de lo posible pero, como se verá, parece que aún está lejos:

¿Cómo poder comprimir secuencias como: 0, 1, 1, 6, 1, 3, 1, 5, 3... y similares?

¿ Cómo comprimir código ASCII donde todos los caracteres sólo ocupan 6 ó 7 bits en el peor de los casos? (Un byte tiene 8 bits)

5. ALGORITMO DE COMPRESION DE BITS

Al contrario que los anteriores, este método aún lo estoy desarrollando. De todos modos, aunque sólo he hecho algunos prototipos poco útiles, presenta unos resultados prometedores. Dado que ando un poco corto de páginas haré sólo un breve esbozo. Este algoritmo es, más o menos, como sigue:

Al igual que la rutina anterior, copia una sección del programa a comprimir en un buffer para evitar destruir el fichero a compactar durante el proceso de optimización. Sí, sí; ha leído bien. Este algoritmo no sólo comprime el texto, sino que opera activamente con él para intentar reducirlo aún más. Esto se hace buscando el valor mínimo de esa área y restándoselo luego a todos los valores del buffer de trabajo. De esta forma, los bytes que quedan ya no representan sus valores propios, sino el offset (desplazamiento) con respecto al valor mínimo. Con este sistema conseguimos reducir el intervalo de valores absolutos facilitando con ello la compresión.

El siguiente paso es buscar el valor máximo, calculando el número de bits que ocupa. Una vez hecho esto, la compresión se realiza SOLAPANDO los bits no usados de un byte con los sí usados del siguiente (la parte alta no utilizada de un byte se sustituye por la parte baja del siguiente; ver figura 2). Por último, se coloca el flag y los indicadores apropiados para que luego el descompresor sepa como tratar el segmento y ya está.

Hay que tener en cuenta que las operaciones que he citado se repiten varias veces con buffers de distintos tamaños y con secciones diferentes de memoria, evaluando cada vez el número de bytes que se ahorran y, de esta forma, poder escoger la mejor al final. El esquema se compresión es algo similar a ésto: FLAG LO, FLAG HI, LONG, BITS, OFFSET, VALORES.

Como ejemplo, veamos como se comprimiría la secuencia 1, 1, 2, 2, 3, 3, 2, 2, 1, 1 con un supuesto flag $FFFE. Se busca el valor mínimo (1), que se toma como offset, y se resta de la secuencia. Queda algo así: 0, 0, 1, 1, 2, 2, 1, 1, 0, 0. Ahora se busca el valor máximo (2) y se evalúan sus bits. Como 2 es %00000010 (se puede representar como %10, ocupando sólo 2 bits de los 8 disponibles) en binario y como es el va-

lor máximo de la secuencia, se deduce que TODOS los valores de dicha secuencia se pueden representar con un máximo de 2 bits, ahorrando 6 bits por cada byte comprimido. Por lo tanto, se pueden comprimir 4 bytes en sólo uno.

Ahora se calcula cuántos bytes ocupa la nueva secuencia multiplicando su longitud inicial por el número máximo de bits que ocupará cada uno de los bytes iniciales, dividiendo por ocho y redondeando al entero superior. Así, en el caso que nos ocupa, la secuencia queda reducida a sólo 2*10/8 = 3 bytes, ahorrando por ello un total de 10-3 = 7 bytes. Como se precisan 5 bytes para indicadores, el ahorro neto es de 2 bytes. El resultado final es algo así: $FE, $FF, $03, $02, $01, %00000101, %10100101, %00000000

Algunas variantes muy interesantes de este algoritmo son utilizar flags de un sólo byte, o asimilar juntos la longitud y el número de bits (suponiendo que la longitud siempre sea menor que 32). También puede suprimirse el offset cuando no sea necesario (algo que ocurre bastante a menudo, por cierto).

El algoritmo que he descrito funciona con lo que yo llamo "cota superior". Como es normal, también puede diseñarse para funcionar con cotas inferiores o con ambas. Yo todavía estoy experimentando.

Así mismo, fijémonos en lo siguiente: el indicador de bits vale, como mucho, 7 y el indicador de longitud, 224 (es 7*256/8). Los valores fuera de los respectivos rangos pueden servir como banderas indicadoras de simetrías, potencias, corrimientos, series, y de todo lo que podamos imaginar, consiguiendo reducir aún más el tamaño del código resultante.

Bueno, hasta aquí hemos llegado. Espero que este trabajo haya servido para divulgar, aunque sea mínimamente, una faceta oscura y poco conocida de nuestra afición. Aún me quedan en el tintero varios algoritmos más, pero están más bien diseñados para aplicaciones específicas, tales como la compresión de gráficos o de largos textos. Los iremos viendo en números posteriores.

Por último, indicar que los algoritmos de compresión de series, de bloques y de bits han sido diseñados por mí y están registrados. Por lo tanto, queda prohibida la difusión comercial salvo autorización por escrito del autor.

Para cualquier comentario o colaboración, por favor, no dudéis en poneros es contacto conmigo a través de la asociación (podéis ver las señas y el teléfono en la portada). Prometo contestar.

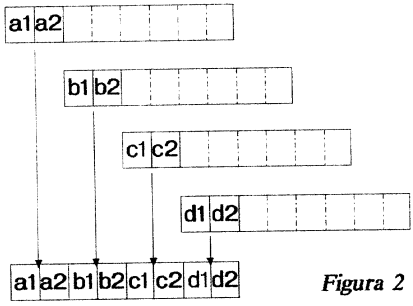
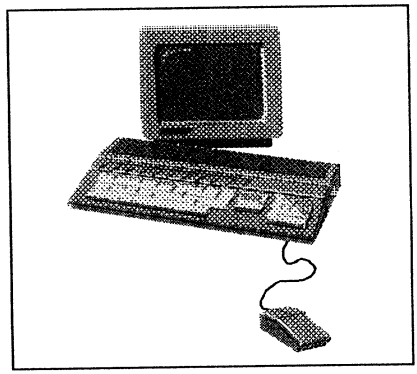


Figura 2



MUSICA POR ORDENADOR

EL PROTOCOLO MIDI

Por E. Cunha

Este artículo pretende dar una descripción general del protocolo MIDI, que permite controlar enormemente la creación musical, hasta el punto de que un ordenador podría elaborar él sólo una composición musical, sin más ayuda que la de un sintetizador y una programación adecuada (por supuesto, el ordenador no es inteligente). Permite, a la vez, facilitar el trabajo al artista, que puede modificar su obra mas cómodamente, reduciendo considerablemente el tiempo empleado para realizarla, y permitiendo futuras mejoras en la melodía con sólo grabarla en un disco de ordenador.

ORIGENES DEL ESTANDAR

El MIDI es producto de la unión, en 1983, de Commodore y Apple, fabricantes de ordenadores, con Yamaha y Roland, de instrumentos musicales. Estos fabricantes, vistas las posibilidades que los ordenadores tenían en ese campo, decidieron crear un protocolo común para la emisión y recepción de datos por ambos, ordenadores y sintetizadores. De esta forma, se puede conectar un sintetizador con un ordenador, con otro sintetizador, con cajas de ritmos,... Ese protocolo llegó a ser lo que hoy conocemos; un estándar mundial asumido por la inmensa mayoría de empresas dedicadas a estos campos.

Pero quizá las comunicaciones entre tan diversos aparatos no hubieran llegado tan lejos de no producirse el avance tan rápido de la microelectrónica, tanto para los ordenadores como para los propios sintetizadores. En un principio, un sintetizador era algo parecido a un monstruo, como los primeros ordenadores. Ocupaban una habitación entera y se basaban en uno o más osciladores, cuyo sonido se sumaba o restaba a otras frecuencias, o se filtraba mediante otros procesos. Su manejo era muy lioso: todo eran potenciómetros, controles manuales, cables que se enchufaban en infinidad de sitios,... Y lo más importante: si se conseguía el sonido deseado había que apuntar la posición de TODOS los controles para reproducirlo otra vez.

Estos procesos evolucionaron, al tiempo que la microelectrónica, hasta que los microprocesadores entraron en escena. A partir de entonces, se fabricaron mejores sintetizadores; más baratos, más pequeños, y que realizaban muchas más tareas. El mercado fue saturándose de marcas diferentes con una sola cosa en común: todas tenían mejores instrumentos y más baratos que las demás. Y no solo eso. Ninguno era compatible con ningún otro. Ante este caos, las compañías ya mencionadas inventaron el protocolo MIDI, en un intento de eliminar este caos que se cernía sobre un recién abierto mundo, el de la música electrónica.

DESCRIPCION TECNICA

Un sistema MIDI es, pues, únicamente para comunicaciones entre aparatos, de la misma forma que el módem o el RS232 comunican ordenadores. Es importante esto, pues no es sonido lo que se envía a través del interface MIDI, sino sólo cadenas de números en binario. Tampoco tiene nada que ver el chip de sonido del ordenador con él, aunque en algún caso y a nivel de hardware sea ese chip de sonido el que controle la interface. Es este motivo, que el mensaje sean códigos, lo que da la potencia al MIDI.Cualquier señal que llega al ordenador (o a otro sintetizador) puede ser tratada y modificada por el receptor, para corregirla o simplemente tocarla.

Un interface MIDI consta de tres conectores, que normalmente son clavijas

pentapolares, a las que se conectan los cables que, en la definición del estándar, tienen su longitud predefinida (que no es muy necesario para el buen funcionamiento). Estos tres conectores son MIDI-OUT, que es por donde sale la señal, MIDI-IN, por donde se recibe, y MIDI-THRU, que reproduce lo que entra por MIDI-IN para poder continuar la cadena, dado que no está restringida a un emisor y un receptor, sino que puede haber más enlaces en la misma. El conector MIDI-THRU no se suele instalar en ordenadores, ya que si bien el ordenador es un elemento más en la cadena, ésta alcanza su máxima potencia cuando la máquina emite datos o los recibe para modificar.

Los datos se envían por el circuito a una velocidad cercana a los 31,25 Kbits/seg, y pueden ser tres los tipos de información lanzada: códigos estándar (pulsación de una tecla, fuerza de pulsación,...), códigos opcionales (vibratos, cambios en el tono, distorsiones,...) y funciones exclusivas, que dependen de cada máquina para su control. Todo esto a través de 16 canales independientes (en cada circuito MIDI), con los que se controlan desde 16 voces en un sintetizador a, por ejemplo, 16 sintetizadores con una voz en cada uno de ellos.

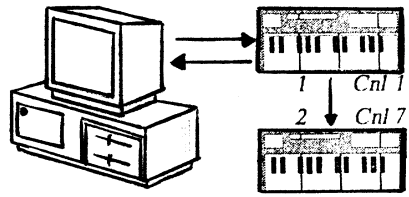
EL PAPEL DEL ORDENADOR

El protocolo MIDI alcanza toda su potencia cuando es el ordenador el que controla todos los instrumentos musicales (más bien, el operador mediante el ordenador). Como se habló más arriba, la secuencia con que se trabaja y emite por MIDI es una cadena de códigos. Entonces qué mejor que el ordenador para tratarla.

Imaginemos un músico con su sintetizador y su ordenador. Primero toca un canal o pista en el sintetizador y la recibe el ordenador. Luego la retoca, la afina, cambia tres o cuatro notas, unas longitudes, y la vuelve a oír las veces que haga falta con precisión absoluta, pues es el ordenador el que ahora manda la señal para ser tocada. Más tarde, incluye alguna otra pista con acompañamientos más complejos, mientras el ordenador toca el resto. Y al final, el músico coge una guitarra, pone el ordenador a tocar el sintetizador y actúa como solista en directo. El resultado es francamente bueno. Fue necesario un músico y un ordenador para tocar una melodía, al margen de un importante ahorro de tiempo.

Este ejemplo puede dar muestra de lo que es el protocolo MIDI y lo que se puede hacer con él. Luego, con una programación más adecuada (y quizás más complicada), se podrían sincronizar luces u otros sonidos a los datos enviados, y quién sabe cuántas cosas más. El límite está aún por alcanzar.

Y una anécdota. Uno de los primeros sintetizadores que salió con MIDI fue el DX-7, de Yamaha, una de las empresas creadoras. De los 16 canales del MIDI sólo funcionaba bien el 1.



El ejemplo muestra un ordenador que manda datos al sintetizador 1. Este sintetizador, asignado al canal 1, toca su melodía. Así mismo, el otro sintetizador recibe los mismos datos que el 1 gracias al conector THRU (es como si fueran directos a él) y, asignado al canal 7, toca su código, seleccionándolo de los otros canales. En este mismo ejemplo, el ordenador sólo puede recibir datos del número 1, pues es al que esta conectado para la recepción.

PROCESAMIENTO EN PARALELO

Por T. Lago

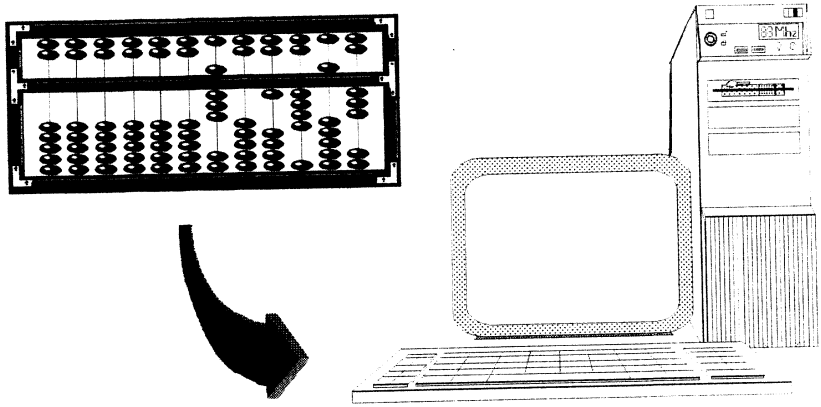
Hasta hace pocos años era impensable que un ordenador pudiera tener más de una CPU, y mucho menos que trabajaran todas a la vez. Una vez investigada y estudiada esta nueva posibilidad, se convirtió en la piedra de toque en lo que concierne a la filosofía de arquitectura de ordenador, siendo además un concepto fundamental en cuanto al diseño de medianos y grandes sistemas informáticos.

Básicamente, el procesamiento en paralelo es un tipo de asociación de procesadores similar al que se podría hacer con resistencias o condensadores, por poner un ejemplo. Es decir; entran todos los datos juntos y después se reparten en distintas líneas que van a cada uno de los procesadores, de forma que cada procesador hace parte del trabajo. A la salida, los datos, ya elaborados, se vuelven a juntar en líneas de datos para ir a los dspositivos de entrada y salida.

En los ambientes de un nivel medio-alto de informatización, el dispositivo empleado para esta técnica es el denominado "transputer". Es un poderoso microprocesador creado por la firma británica Inmos Technologies en 1985. El impacto producido por este componente en cuanto al diseño de hardware es similar al producido por la implantación del transistor en circuitos de bajo coste allá por los años 60-70.

Y por ahí puede preguntarse alguien que por qué tanto afán por fabricar sofisticados dispositivos con alto un coste (aunque dentro de algunos años ya veremos) si sólo pueden disponer de ellos máquinas muy especializadas. El constante vicio de todos los diseñadores de ordenadores es la posibilidad de poder emular de alguna forma la capacidad humana de poder atender a múltiples cosas al mismo tiempo. Es decir, la filosofía de funcionamiento de la sinapsis neuronal.

En próximos números se desarrollará el concepto de transputer y sus aplicaciones.



CURIOSIDADES INFORMATICAS

ORDENADORES QUE PIENSAN

Por J. Cea

Desde el advenimiento de los ordenadores, uno de los objetivos prioritarios de las ciencias de la computación ha sido, y sigue siendo hoy más que nunca, dotarlos de lo que se ha dado en llamar Inteligencia Artificial. Con sólo pronunciar estas palabras es probable que se haya visto envuelto es una acalorada discusión con su vecino sobre si es o no posible programar una máquina para que sea capaz, en toda situación, de comportarse como un ser humano.

Probablemente la prueba más famosa de Inteligencia Artificial consista en sentar un voluntario frente un monitor y un teclado con absoluta libertad para "charlar" sobre cualquier tema, preguntando y contestando a las preguntas de su interlocutor. Si después de un intercambio lo suficientemente amplio el operario no sabe si ha estado conversando con una máquina o con otro hombre, el ordenador habrá aprobado el examen.

Resulta evidente el hecho de que hay que repetir la experiencia varias veces, intercalando interlocutores humanos e informáticos para intentar reducir al máximo los prejuicios del operador.

Aunque existen desde hace años sistemas informáticos capaces de "conversar" con naturalidad sobre temas determinados, aún está lejos el día en que no podamos saber si nuestro corresponsal al otro extremo de la línea es un ser como nosotros o el fruto de muchos años de estudio. Los inconvenientes para ello son tanto de orden práctico (velocidad, capacidad de almacenamiento) como teóricos. De todos modos, los continuos avances tecnológicos en coprocesadores para tareas específicas, en memorias más veloces y compactas, así como los adelantos en sociología y psicología, y la aparición de lenguajes como el PROLOG o el LISP hacen pensar en un futuro prometedor para este viejo sueño.

Nosotros, pobres poseedores de modestos equipos, no podemos pensar siquiera en conseguir los resultados que consiguen los expertos con ordenadores miles de veces más costosos que los nuestros e infinitamente más sofisticados. Sin embargo, todos hemos visto programas de ajedrez imbatibles (o casi) y nos hemos preguntado cómo un ser que sólo sabe sumar y restar números...

LOS ORDENADORES EN LOS JUEGOS DE INTELIGENCIA

Por regla general, un ordenador realiza su movimiento tras una serie de cálculos (después de todo, debemos recordar que sólo entienden y operan con números) más o menos laboriosos según el juego de que se trate. Dividamos los juegos según el tipo de "scanner" que realice el programa:

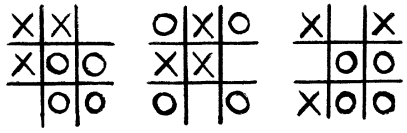
1. JUEGOS DE EVALUACION TOTAL

Son aquellos en los que es factible examinar TODAS Y CADA UNA de las jugadas posibles (debido a que hay muy pocas). El ejemplo más típico es el "Tres en raya". Cualquiera que haya jugado se habrá dado cuenta de que es imposible ganar (o perder) a menos que se cometa un error. En nuestro caso, el ordenador calcula TODOS los posibles movimientos y sus respuestas dentro de un intervalo de jugadas determinado (llamado PROFUNDIDAD) y ejecuta la jugada que considera mejor.

En cada nivel tiene que pensar unas 3^3 jugadas, aunque en realidad son me-

nos dado que muchas producen inmediatamente el fin de la partida y no tienen movimientos "hijos".

No se puede ganar a niveles altos de profundidad (unas 3 jugadas), pero a niveles bajos cabe la posibilidad de "preparar el terreno" para que cuando el ordenador evalúe nuestra posición sea ya demasiado tarde. En el "Tres en raya" existen bastantes estrategias ganadoras, pero el problema es llegar a ellas sin que el ordenador se entere. En los diagramas siguientes, las cruces tienen posición ganadora si las TRES fichas rivales se encuentran en las posiciones señaladas por los círculos:



2. JUEGOS DE EVALUACION PARCIAL

Son los que, existiendo demasiadas variantes, es imposible calcularlas todas. Para solucionar este problema se precisa una rutina encargada de "puntuar" nuestra posición y la del adversario y, de esta forma, escoger el mejor movimiento. El esquema de la partida se parece a un árbol donde cada rama es una jugada, y cada rama se divide en sub-ramas, y éstas en sub-subramas, cada una con una puntuación otorgada por la rutina evaluadora (ver figura 1).

Como es lógico, el nivel de juego del programa depende de forma absoluta de la perfección de la rutina de evaluación. Ésta debe tener en cuenta cosas como la movilidad de las piezas,posición del tablero, piezas atacadas (o atacables), posiciones seguras, etc. Es, con mucho, la parte más crítica del programa.

Éste es el sistema que se utiliza con juegos de árbol poco "frondoso", como el Reversi (también llamado Othello), donde puede calcularse fácilmente la estrategia ganadora. Pero, ¿qué pasa con la "frondosidad" del ajedrez?

Efectivamente, por término medio, en el ajedrez se pueden realizar unos 35 movimientos distintos por turno. Así, una partida típica de 50 jugadas por bando requiere pensar unos $35^{100} = 10^{155}$ jugadas distintas. En el caso de disponer de un super-ordenador capaz de efectuar un CUATRILLON de jugadas por segundo (algo bastante improbable, ciertamente) precisaríamos algo más de 10^{68} años (mucho más de lo que nuestros jefes estén dispuestos a esperar)

Aún en el caso de calcular sólo las 4 primeras jugadas (algo a todas luces insuficiente para una partida de nivel medio), precisaríamos evaluar más de dos billones de movimientos distintos. Esta incapacidad práctica es lo que hace que los programas de ajedrez sean todavía vulnerables. Para ahorrar tiempo sin sacrificar calidad, los programadores ha desarrollado un par de trucos útiles también para otras áreas. Para que éstos sean útiles, se precisa una función evaluadora muy selectiva y que tenga en cuenta no sólo capturar piezas sino también factores como valor y número de piezas amenazadas, posición avanzada de los peones, movilidad general, número de casillas dominadas, casillas centrales, etc.

Cada uno de dichos parámetros tiene un "peso" distinto en la puntuación final de una posición determinada. Una característica del ajedrez es que dicho valor puede cambiar a lo largo de la partida, según la fase del juego en que se encuentre, las tácticas seguidas o el estilo seguido por cada jugador. Algunos programas modernos tienen ésto en cuenta y van variando los coeficientes durante el transcurso de la partida.

1. PODA DEL ARBOL

Consiste en eliminar algunas ramas, generalmente las menos interesantes,

reduciendo así el número de variantes a estudiar. Existen varios métodos para ello, pero sólo describiré la llamada "PODA ALFA-BETA".

La poda alfa-beta evalúa todos los movimientos propios, dando una puntuación a cada uno, y todos las jugadas de respuesta enemigas, puntuándolas también (ver figura 1). Después de hacer ésto, el ordenador hace una lista de aquellas jugadas en las que la resta de nuestra puntuación menos la máxima puntuación enemiga sea más alta. Las otras ramas no se consideran. En una posición típica, evalúa 35^2=1.225 jugadas y escoge, por ejemplo, las diez mejores. De esta forma, pasamos de tener que calcular 10^{155} jugadas a sólo unas 10^{100}. Algo se ha mejorado. Las podas "Alfa", "Alfa-Beta-Alfa", etc, son idénticas a la anterior salvo en el hecho de explorar más o menos ramas antes de podar.

En el caso de la figura, se han considerado sólo 3 ramas para simplificar, pero pueden ser cuantas se quiera. Tras explorar esa sección del árbol de juego, el ordenador despreciará la última rama (donde las puntuaciones optenidas son desastrosas). De esta forma sólo tendrá que considerar, para el nivel siguiente, las dos ramas que quedan.

Este algoritmo de poda nos proporciona un método adecuado para tratar de reducir los ingentes cálculos precisos para jugar una partida media de ajedrez, pero a costa de la calidad. Efectivamente, un ordenador no suele realizar sacrificios espectaculares ni nada por el estilo debido a que sus beneficios sólo son visibles tras muchas jugadas (a no ser que sea alguna jugada obvia) y el ordenador ni siquiera los considera. De la misma forma, un buen método para ganar consiste en realizar movimientos que estén fuera del árbol del ordenador (es decir, ¡¡SORPRENDERLO!!). El único problema es cómo hacerlo...

2. APERTURAS

Durante una apertura existen gran cantidad de movimientos aparentemente del mismo valor (para el ordenador), pero que se demuestran deficientes tras muchas jugadas. Así mismo, al ser una parte donde sólo se despliegan las piezas y poco más, su evaluación correcta es muy difícil (eso si no se podan antes...).

Para evitar ésto, los programadores instalan un "libro" de aperturas que el ordenador sigue hasta el final o hasta que el contrincante se "sale" de la ortodoxia. A partir de ese momento, las rutinas normales de evaluación son las que toman el control.

Normalmente se incorporan cientos de aperturas (para darle un poco de variedad) con muchas de sus variantes. No ocupan casi nada y aseguran un buen principio de partida, avalado por la experiencia acumulada por los maestros ajedrecistas durante cientos de años.

3. EL EFECTO HORIZONTE

Uno de los puntos donde cojean los ordenadores es en el algoritmo de poda, pero no es el único. Otro factor muy importante es el que se suele llamar "Efecto Horizonte".

Como sabréis, el ordenador piensa hasta cierta profundidad para dar la respuesta. ¿Qué pasaría si en el último nivel de análisis el ordenador encontrara una jugada magistral, como hacer peón por dama, pero no se da cuenta de que en la siguiente es mate? El programa descubre que comer la dama es una muy buena jugada, pero no llega a evaluar el mate que le propina el adversario dado que ya ha llegado al tope de profundidad escogido por el usuario. Precisamente en esto se basa la elección del nivel del programa, que indicará la profundidad de análisis. A menor profundidad, más claro y mortífero es este Efecto.

Podemos aprovecharnos de él realizando jugadas "en la frontera", minando lentamente al pobre e indefenso ordenador. Personalmente, considero esta

actuación poco ética. De todos modos, nada hay que cause más satisfacción que ganar después de haber perdido innumerables partidas, aunque sea ensañándose en los defectos (de difícil solución, por otra parte) del programa.

4. FINALES

Los finales de partida es otro momento que crea serias dificultades a cualquier programa de ajedrez. Aquí los efectos perniciosos de la poda y del efecto horizonte actúan conjuntamente para hacerle la vida poco menos que imposible a nuestra querida máquina. El ordenador "desprecia" las jugadas destinadas a conducir al mate (efecto horizonte y poda, dado que son jugadas "a largo plazo"), prefiriendo dedicarse a acosar sin sentido al rival o a comer piezas sin un objetivo determinado.

Así, no es raro que una partida prácticamente decidida se pierda porque el ordenador no sabe dar el mate (esto también suele pasar con los jugadores humanos). Un ordenador con un programa lo suficientemente capaz debería reconocer un final y desconectar la poda, aumentando igualmente la profundidad de análisis para evitar el efecto horizonte. Sin embargo, estos cambios originan un aumento superlativo en los tiempos de cálculo. Además, existe una solución mucho más sencilla.

¿Por qué no usar un libro de finales, al igual que usamos un libro de aperturas? De todos es conocida la famosa máquina de Torres Quevedo, capaz de dar mate en un final de rey y torre contra rey. ¡Y era totalmente mecánica!. ¿Por qué no aplicar los mismos principios en un programa? Efectivamente, los programas modernos suelen traer un libro de finales para ayudarles a rematar a su ya moribundo adversario. Esto crea algunas PARADOJAS muy divertidas como tardar diez movimientos en darte mate con dos alfiles cuando podría coronar un peón y acabar mucho antes...(y digo ésto con muuucha experiencia, Je, Je).

A pesar de todos estos defectos, un programa moderno de ajedrez es un duro rival capaz de poner en aprietos a cualquiera. No tiene piedad, ni escrúpulos, ni remilgos (por lo menos de momento) y acabará contigo en cuanto pueda. Puedes confiar en ello. Algo bastante curioso, y animo a los lectores a que lo hagan, consiste en poner a jugar 2 programas distintos, uno contra otro. Los resultados suelen deparar bastantes e inesperadas sorpresas.

JUEGAN BLANCAS Y MATE EN TRES

Los programas actuales suelen incluir una opción para resolver mates finales hasta una profundidad máxima de unas 7 jugadas. Dado que dichos problemas suelen tener unas soluciones bastante truculentas, el programa opta por desconectar la poda (dado que no puede permitirse eludir sacrificios o combinaciones que normalmente no evaluaría), lo que provoca la comprobación de todas las jugadas posibles. Ésto origina tiempos de cálculo que se traducen en horas a partir de las 3 ó 4 jugadas.

Como comentario añadido diré que de vez en cuando propongo un problema de finales al ordenador y si era de 3 jugadas, a veces ¡¡lo resuelve en DOS!!

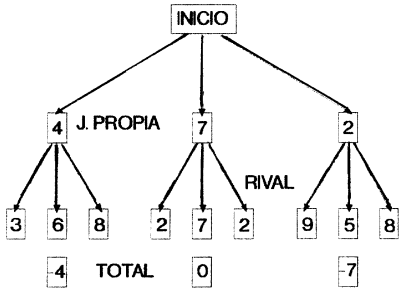


Figura 1

3. JUEGOS DE INFORMACION IMPERFECTA

Hasta ahora hemos visto juegos donde toda la información aparece clara y presente a cada jugador. Sin embargo, no todos los juegos son así. El Póker, por ejemplo, es un juego donde los jugadores no saben lo que tienen los demás y también desconocen las cartas que van a tocarles (a menos que hagan trampas...). Este tipo de juegos no pueden ser abordados de la misma manera que los anteriores; en especial, la estructura arborescente carece de sentido. Además, el hecho de ser un juego de información imperfecta nos brinda la oportunidad de "Farolear" para conseguir información sobre el estilo de juego de nuestro adversario ó proporcionar una falsa imagen del propio. Ésto es algo que los ordenadores deben tener muy en cuenta.

Existen diversas tácticas de juego, pero es un área tan immensa que cualquier descripción peca de superficial. Las técnicas más avanzadas van aprendiendo durante la partida "largando" faroles de vez en cuando para "sondear" al posible incauto. De todos modos, este tipo de juegos no tratables a través de un simple árbol obligan a los programadores a profundizar aún más en los mecanismos en que nos basamos los humanos para tomar decisiones en situaciones donde desconocemos la suficiente información como para tener que "PENSAR" realmente. La experiencia obtenida con este tipo de juegos que podrían calificarse de frívolos es aprovechable en otras áreas más serias como la carrera de armamentos, política publicitaria o en una guerra.

LOS ORDENADORES Y LA CREACION ARTISTICA

Los ordenadores también tienen su corazoncito, aunque sea de frío silicio. Existen innumerables técnicas para "imitar" el comportamiento artístico humano. Éstas van desde la programación de leyes de armonía, contraste, causa-efecto, etc, hasta la aleatorización. Esta es, precisamente, una de las técnicas más sorprendentes que conozco.

La aleatorización (de textos, música, etc) se basa en el análisis estadístico de un original tomado como modelo. En el caso de la música, podemos tener una tabla obtenida por análisis de una partirura original que indique las posibilidades que tiene de salir cada nota después de un Do, un Re... De esta forma, cada nota depende de la anterior, de las 2 anteriores, las 3 ...(cuantas más se tengan en cuenta, más se parece al original).

En un texto, la tabla suele ser bidimensional (letras anteriores, porcentajes) y, aunque resultan textos con una total anarquía sintáctica y semántica, trasciende un belleza indescriptible que contrasta con la total falta de sentido. Así mismo, "se siente" la mano del autor del texto original propuesto para la aleatorización; se nota su estilo aún después de haber destrozado el texto. Es algo inexplicable.

En el caso de la música, la tabla suele ser tridimensional (notas anteriores, longitudes, porcentajes) y funciona igual que en el caso de los textos. En el próximo número, ejemplos de todo esto. Y además trataré de los "FRACTALES".

PRONTUARIOS

TABLA DE CONVERSION DECIMAL-HEXADECIMAL-BINARIO

DEC	HEX	BIN	DEC	HEX	BIN	DEC	HEX	BIN
0	00	00000000	43	2B	00101011	86	56	01010110
1	01	00000001	44	2C	00101100	87	57	01010111
2	02	00000010	45	2D	00101101	88	58	01011000
3	03	00000011	46	2E	00101110	89	59	01011001
4	04	00000100	47	2F	00101111	90	5A	01011010
5	05	00000101	48	30	00110000	91	5B	01011011
6	06	00000110	49	31	00110001	92	5C	01011100
7	07	00000111	50	32	00110010	93	5D	01011101
8	08	00001000	51	33	00110011	94	5E	01011110
9	09	00001001	52	34	00110100	95	5F	01011111
10	0A	00001010	53	35	00110101	96	60	01100000
11	0B	00001011	54	36	00110110	97	61	01100001
12	0C	00001100	55	37	00110111	98	62	01100010
13	0D	00001101	56	38	00111000	99	62	01100011
14	0E	00001110	57	39	00111001	100	63	01100100
15	0F	00001111	58	3A	00111010	101	64	01100101
16	10	00010000	59	3B	00111011	102	65	01100110
17	11	00010001	60	3C	00111100	103	66	01100111
18	12	00010010	61	3D	00111101	104	67	01101000
19	13	00010011	62	3E	00111110	105	68	01101001
20	14	00010100	63	3F	00111111	106	69	01101010
21	15	00010101	64	40	01000000	107	6A	01101011
22	16	00010110	65	41	01000001	108	6B	01101100
23	17	00010111	66	42	01000010	109	6C	01101101
24	18	00011000	67	43	01000011	110	6D	01101110
25	19	00011001	68	44	01000100	111	6E	01101111
26	1A	00011010	69	45	01000101	112	6F	01110000
27	1B	00011011	70	46	01000110	113	70	01110001
28	1C	00011100	71	47	01000111	114	71	01110010
29	1D	00011101	72	48	01001000	115	72	01110011
30	1E	00011110	73	49	01001001	116	73	01110100
31	1F	00011111	74	4A	01001010	117	74	01110101
32	20	00100000	75	4B	01001011	118	75	01110110
33	21	00100001	76	4C	01001100	119	76	01110111
34	22	00100010	77	4D	01001101	120	77	01111000
35	23	00100011	78	4E	01001110	121	78	01111001
36	24	00100100	79	4F	01001111	122	79	01111010
37	25	00100101	80	50	01010000	123	7A	01111011
38	26	00100110	81	51	01010001	124	7B	01111100
39	27	00100111	82	52	01010010	125	7C	01111101
40	28	00101000	83	53	01010011	126	7D	01111110
41	29	00101001	84	54	01010100	127	7E	01111111
42	2A	00101010	85	55	01010101	128	7F	10000000

HISTORIA DE LA INFORMATICA

PARTE PRIMERA

Por Nacho Agulló

Esta serie que aquí comienza pretende narrar el proceso que a través de la Historia nos ha llevado hasta nuestros ordenadores actuales; un proceso que comenzó hace tres mil años con la invención del ábaco y al que no es posible vislumbrar fin. Paso a paso iremos viendo el proceso de la Humanidad construyendo máquinas cada vez más perfectas.

LAS MATEMATICAS EN EL S.XVII

La aparición de grandes matemáticos y filósofos va a dar un fuerte empuje a las Matemáticas en el siglo XVII. De un lado, se realizarán importantisimos descubrimientos matemáticos; y al tiempo, el auge del racionalismo va a dar una gran importancia a las Matemáticas, tomándolas como modelo de saber.

Por otra parte, los avances matemáticos comenzaban a llegar a niveles donde los cálculos se volvían muy complicados, de modo que se hacía muy necesario la invención de algún mecanismo que permitiera simplificar el cálculo.

Por todo ello, como veremos, las máquinas de calcular fueron acogidas con gran interés a medida que fueron apareciendo, lo que favoreció su propagación y perfeccionamiento.

DESCUBRIMIENTOS MATEMATICOS EN EL S. XVII

En 1614 John Napier inventa los logaritmos. Como antecedente hay que citar el "Compás geométrico y militar", de Galileo Galilei, una rudimentaria regla de cálculo. Poco despues, Henry Briggs publicaba el tratado "Arithmethica logarithmica", que contenía tablas con treinta mil logaritmos calculados con catorce cifras decimales. Más tarde, el holandés Adrian Vlacq calcularía los setenta mil restantes hasta completar los cien mil primeros logaritmos.

Podemos suponer el esfuerzo que costó la realización de estos cálculos y el alivio que hubiera supuesto uno de nuestros ordenadores actuales a aquellos matemáticos. Esto nos da una idea de la necesidad que los matemáticos de la época tenían de mecanismos que calcularan.

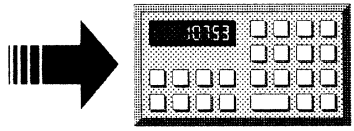
En 1623, Francis Bacon utiliza por primera vez la aritmética en base 2, que tan importante es hoy para la Informática.

LA PRIMERA CALCULADORA MECANICA:"LA PASCALINA"

En 1642, Blaise Pascal inventó la "Pascalina", una máquina que sumaba y restaba, despertando un gran interés. El rey de Francia le concedió una patente para comercializarla. Si bien no fue un éxito de ventas, el interés entre los matemáticos de la época no se extinguió: fueron efectuadas mejoras y modificaciones sucesivamente.

EL CILINDRO DE LEIBNIZ

En 1673 Leibniz presenta en la Royal Society su propia versión de máquina calculadora mejorada mediante el "cilindro de Leibniz", con lo cual consiguió una máquina que podía multiplicar y dividir.



DATOS BIOGRAFICOS

JOHN NAPIER (Edimburgo, 1550-1617)

Publicó dos tratados sobre logaritmos. Se dedicó también a estudiar la trigonometría esférica, inventando las fórmulas de Napier para calcular los ángulos de un triángulo esférico.

HENRY BRIGGS (Warley Wood, Inglaterra, 1556-Oxford, 1631)

Estudió en Cambridge, donde luego enseñó hasta 1596,para hacerlo después en Londres y Oxford. Tuvo amistad con Napier, con quien discutió sobre la elaboración de las tablas de logaritmos decimales. En 1624 publicó "Arithmethica logarithmica". También se dedicó a la Geometría, hallando las fórmulas de Briggs, que permiten hallar los ángulos de un triángulo.

FRANCIS BACON (Londres, 1561-1626)

Autor de múltiples obras filosóficas, se dedicó a la política, llegando a ser Canciller. Acusado de aceptar sobornos, fue condenado a prisión en la Torre de Londres, aunque al poco el rey le perdonó. Desposeído de sus cargos, se dedicó al estudio hasta su muerte.

BLAISSE PASCAL (Clermont-Ferrand, 1623-Paris, 1662)

Muy joven, escribió su "Ensayo sobre las cónicas" e inventó la máquina calculadora. Despues, escribiría "Tratado sobre el peso de la masa del aire" y el "Tratado sobre el equilibrio de los líquidos". Ingresó en la abadía de Port-Royal. A él se debe también el "Teorema de Pascal", de Geometría. A su muerte, sus amigos publicaron "Pensamientos de Pascal", un libro que recogía su forma de pensar.

GOTTFRIED WILHELM LEIBNIZ (Leipzig, 1646- Hannover, 1716)

Estudió Filosofía y Derecho en las universidades de Leipzig y Altdorf. Trabajó como abogado y diplomático; este último trabajo motivó el que se desplazara a París, donde entró en contacto con el ambiente cultural francés, dominado por el cartesianismo. De regreso a Alemania, aceptó el puesto de bibliotecario del duque de Hannover. Viajó por Baviera, Austria y Roma. En 1691 fue nombrado miembro de la Academia de Ciencias de París; en 1700 consejero de Justicia del elector de Brandenburgo, y fue elegido presidente perpetuo de la Academia de Berlín.

En vida mantuvo correspondencia con más de 600 personas; conoció a Espinosa; formuló el principio de "la razón suficiente"; mejoró la calculadora de Pascal consiguiendo que realizara multiplicaciones y divisiones; contribuyó al desarrolllo de la Física conceptualizando la "energía cinética"; inventó, junto con Newton, el cálculo diferencial; desarrolló la teoría del sistema binario, investigando la posibilidad de utilizarlo en aparatos de cálculo... Leibniz nos dejó importantes avances en múltiples ciencias, avances que lo atestiguan como uno de los grandes genios de la Historia.

GLOSARIO

Por E. Cunha

ALGORITMO:

Rutina que sirve para solucionar determinado problema, en la que están descritos todos los pasos necesarios para ello.

BUFFER:

Zona de memoria usada como almacenamiento temporal. Normalmente, al llenarse de datos, su contenido se envía a algún periférico, como la impresora.

BYTE:

Cadena de 8 bits, tomada como unidad en la mayoría de tareas. Cuando un número consta de dos bytes, al primero se le llama BYTE ALTO y al contador de unidades, el segundo, BYTE BAJO. (Ver "Guía del profano" en este número)

CABECERA DE COMPRESION:

Datos que van al comienzo de un bloque de código comprimido, en los que se indican al descompresor los parámetros para que éste realice su tarea.

CELULA DE MEMORIA:

Zona de los chips de memoria compuesta de 8 posiciones de bit, en que se puede almacenar un byte. Estas células (POSICIONES, DIRECCIONES) están numeradas a partir del 0 hasta su término (65535, para 64 Kb; 16777216, para 16 Mb,...)

FLAG:

Es un número que sirve de indicador para algo determinado. También se le llama, en español, BANDERA.

GIGABYTE:

Unidad de capacidad de memoria. Es la menos usada (por ahora) por ser de las más grandes, y por haber pocos sistemas que usen estas capacidades. Equivale a 1024 Mb, 1048516 Kb ó 1073741824 bytes.

HEXADECIMAL:

Nombre dado al sistema de numeración en base 16.

KBYTE (K, Kb):

Unidad de medida de capacidad de memoria. Equivale a 1024 bytes.

MEGABYTE (M, Mb):

Unidad de medida de capacidad de memoria. Equivale a 1024 Kb ó 1048576 bytes.

NYBLE:

Cifra en binario compuesta de una cadena de 4 bits.

OCTAL:

Sistema de numeración en base 8.

PANTALLA GRAFICA:

Zona de memoria destinada al dispositivo externo de visualización; son los datos que conforman la pantalla (textos, dibujos) que vemos en el monitor. En Informática, a cualquier "dibujo" que aparece en pantalla, se le denomina GRAFICO.

PUNTERO:

Valor existente en una posición de memoria o en una variable, que apunta a una dirección absoluta de la memoria total. Cuando tenemos un bloque de datos que puede estar en diferentes zonas de la memoria, en vez de acceder a él directamente, lo hacemos mediante el puntero que lo marca.

RUTINA:

Parte de un programa que realiza una tarea concreta. A veces puede ser un algoritmo, pero no es lo mismo. Una rutina puede ser, simplemente, imprimir un dato en pantalla y regresar al programa principal.