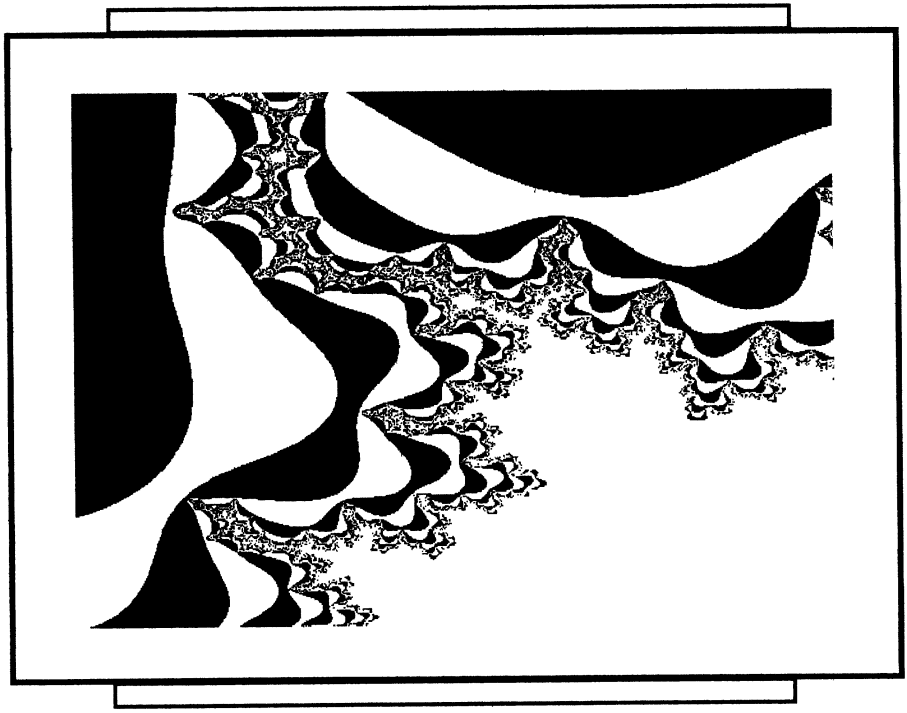
DATA BUS

PUBLICACION EDITADA POR LA ASOCIACION JUVENIL PARA LA INFORMATICA VIGUESA

NUM. 2 *DICIEMBRE* *AÑO 1990*

Dep. Legal VG-87-90



FRACTALES

EMULACION DE LA RECURSIVIDAD

MONTAJES DE HARDWARE

DATA BUS
Núm. 2
Diciembre
Año 1990

DIRECTOR
Jose Manuel Suarez Pousa

REDACTORES
Jesús Cea
Nacho Agulló
Eduardo Cunha

COLABORADORES
Telmo Lago
Pablo Lobariñas
Marcos Vaqueiro Rodríguez

DISEÑO
Eduardo Cunha

MONTAJE
Jose Manuel Suarez Pousa
Eduardo Cunha

EDITA
Asociación Juvenil para la Informática Viguesa
c/ Caldas de Reis 12-6 Izq.
Tel: 23 18 35
41 01 13

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

Publicación íntegramente realizada con equipo de autoedición ATARI

INDICE

EDITORIAL

Con este son ya dos los números de DATA BUS que hemos realizado. Dos números llenos de artículos sobre algoritmos, hardware, aplicaciones, programación,...; en suma, múltiples aspectos de la Informática, que es la ciencia a la que está dedicada esta revista. Pero no vamos a limitarnos tan sólo a los aspectos mencionados.

Por ejemplo, en este número contamos con un extenso artículo sobre un tema técnico como es la aplicación del ordenador a la Robótica, comenzando por algo tan simple como es el control de un LED desde la máquina. Este artículo es la primera (y esperemos que no la última) contribución a nuestra revista de un nuevo colaborador, Marcos Vaqueiro Rodríguez, al que damos la bienvenida.

En este número también contamos con otros artículos técnicos, como puede ser "Música por ordenador", "Procesamiento en paralelo"... Pero no olvidamos otras áreas más simples de la Informática, a las que dedicaremos más espacio en próximos números.

LA REDACCION

Los lectores que estén interesados en enviarnos artículos de Informática para su publicación podrán hacerlo a la dirección de la revista. Garantizamos la publicación de cualquier artículo siempre que tenga un mínimo de calidad.

Así mismo, los lectores que deseen hacer cualquier consulta sobre temas relacionados con la Informática podrán escribirnos a la misma dirección. Estas consultas serán respondidas en la sección de cartas, con la que contaremos próximamente.

FE DE ERRATAS

- En el "Prontuario" apareció duplicado el hexadecimal 62, que parecía corresponder a los decimales 98 y 99. Por ello, los demás hexadecimales a partir del 98 aparecieron desplazados un lugar.
- En la página 7, primera columna, se identificó 2^7 con 1111111 (en realidad tendría que poner 1111111=2^7-1) y 2^8 con 11111111 (11111111=2^8-1).
- En el "Glosario", en la definición de "célula de memoria", se indicaba que para 16 Mb las células se numeraban de 0 a 16777216, cuando en realidad es hasta 16777215.

Eduardo Cunha

EL SISTEMA OPERATIVO

Como se vio en el artículo anterior, tenemos la base para hacer cualquier trabajo que nos plazca (siempre y cuando sea factible realizarlo en un ordenador, claro). El gran problema que surje ahora es su manejo, y, profundizando un poco más, su programación. Este último es un tema que aleja a potenciales usuarios de las máquinas, no tanto por la posible complejidad (que por supuesto, existe), sino por la apariencia misma que pueda tener un listado en cualquier lenguaje. Es muy normal, pues, que un no-iniciado en el mundo de la Informática se quede en la puerta diciendo "esto no es para mí", "esto es muy complicado". Verdaderamente, en parte, son ciertas estas afirmaciones, pero con un poco de interés se le puede coger el gusto a la programación, y no quedarse sólo en el manejo de la máquina, como le sucede a mucha gente; bien por que no le atraiga, bien por que usen el ordenador en su trabajo.

SISTEMAS OPERATIVOS

Lo primero que vemos al conectar la máquina (aparte del led de encendido) es el sistema operativo. Este sistema nos entrega el control de la máquina para usarla como más nos apetezca. De todas formas, hay que distinguir entre los muchos sistemas accesibles por el usuario para ser instalados en su ordenador, pues su variedad es tan amplia como sus prestaciones.

FUNCION DEL SISTEMA OPERATIVO

La principal misión de un sistema operativo es facilitar el manejo de la máquina de forma que el usuario pueda sacar el máximo partido a su ordenador. De esta forma, una máquina (hardware, soporte físico) puede adquirir una "inteligencia" que le permita realizar las tareas básicas de la forma más competente posible. Vimos en el artículo anterior suficiente sobre el funcionamiento interno del ordenador como para poder comprender de una manera clara el funcionamiento del sistema operativo. Pongamos un ejemplo.

Cuando escribimos en el intérprete del sistema que vamos a cargar un programa en memoria, de ahí a que se cumpla parece depender sólo de la unidad de almacenamiento usada. Pero la máquina hace todo el trabajo. Eso sí; el tiempo usado es realmente mínimo. Bien; tenemos tecleada la orden. El sistema de la memoria accede primeramente al disco para hacer un directorio de los ficheros disponibles. En el caso que el fichero requerido esté allí, llama a las rutinas de carga de disco. Todo esto lo hace el controlador de disco. Una vez cargado en memoria, un mensaje en la pantalla nos indicará el éxito o fracaso de la operación.

Un buen sistema operativo debe poder explotar el ordenador hasta el límite de su capacidad. Por ello, el sistema ha de gestionar la unidad central (para avisar en caso de malfunción), los trabajos que sean encomendados (carga y ejecución de ficheros, por ejemplo, asignando tiempo, memoria de buffer, etc, a esas tareas), y la gestión de datos y comunicaciones con el exterior. Además, cada día gana importancia el concepto de "relación con el usuario". Cuanto más fácil de usar, mejor. Y si presenta los comandos de forma intuitiva, mejor todavía.

Esta evolución del entorno de trabajo ha venido pareja con la complejidad interna de los sistemas. En un principio, el sistema operativo era algo muy difuso, que en realidad no existía como tal:

Los ordenadores incorporaban un intérprete de algún lenguaje con el que se controlaba la máquina y se cargaban y ejecutaban programas, función básica de todo sistema operativo. El problema surgía cuando se necesitaban hacer programas compatibles para ordenadores distintos. Cada ordenador necesitaría tener el intérprete del lenguaje de los otros y, prácticamente, su estructura interna a nivel de circuitería. Este método de trabajo (lenguaje en memoria) se vino usando hasta hace muy poco tiempo en microordenadores tan conocidos como los Commodore 64/128, los Oric, los Spectrum, los Amstrad... Incluso se podría hablar de una relación 8 bits-intérprete en ROM, porque poquísimos son los ordenadores de 16 bits que lleven su BASIC en memoria. Además, la mayoría de máquinas de 8 bits trabajaban con un dispositivo externo de cassetes, cosa que imposibilitaba ampliamente el tratamiento de información externa.

Cuando los micros comenzaron a venir de fábrica con unidad de disco, se hizo poco menos que imprescindible, para la correcta gestión de los ficheros almacenados en el disco, la aparición del sistema operativo. Aunque durante un tiempo el sistema fue algo "periférico" al ordenador, que aún era controlado por un intérprete de un lenguaje, hoy día no se concibe una máquina sin su sistema. De todas formas, y volviendo a la diferencia 8-16 bits, en los primeros se estancó su desarrollo, pues tienden a desaparecer, ya que surgen máquinas de 16 bits con una potencia de cálculo muy superior, que exigen sistemas muy complicados para su manejo. La evolución de los sistemas se da, pues, en este campo. La correcta interacción entre usuario y máquina de la forma más sencilla y "amigable" posible es objetivo fundamental de los diseñadores de estos sistemas, dada la complejidad de circuitería y funciones de las máquinas de 16 bits (o incluso más) que son construidas casi a diario.

Y esta investigación se puede apreciar en los resultados que ha dado durante largo tiempo. Hubo de todo desde el primer sistema, pero nos referiremos a la década de la microinformática.

ORDENADOR	*PROCESADOR*		*SISTEMAS OPERATIVOS*
Commodore AMIGA	*68000*	*16*	*AmigaDOS*
AMSTRAD	*Z80A*	*8*	*CP/M*
APPLE II	*6502*	*8*	*ProDOS 1.02, DOS 3.3*
Atari ST/MEGA ST	*68000*	*16*	*TOS, GEM*
CBM 64	*6510*	*8*	*Commodore DOS*
DRAGON 64	*6809E*	*8*	*OS9*
Hewlett-Packard 150	*8088-2*	*16*	*MS/DOS 2.11*
ICL PC15	*8085A*	*8*	*CP/M*
IBM PC	*8088*	*16*	*PC/DOS, MS/DOS, CP/M-86*
Apple MACINTOSH	*68000*	*16*	*Syst. MacIntosh, SMALLTALK*
MSX	*Z80A*	*8*	*MSX/DOS, CP/M*
Olivetti M20	*Z8001*	*16*	*PCOS*
Philips P3500	*Z80A*	*8*	*TurboDOS*
Rank Xerox 820 II	*Z80A*	*8*	*CP/M*
RAINBOW 100	*Z80+8088*	*8, 16*	*CP/M, MS/DOS*
Sinclair SPECTRUM	*Z80A*	*8*	*(Propio)*
Toshiba T300	*8088*	*16*	*MS/DOS, CP/M-86*

El CP/M (Control Program for Microprocessors) fue el primer sistema que alcanzó una enorme difusión en todos los aspectos. Fue exclusivo en el campo de los micros durante algunos años. Luego, con el anuncio de Intel de crear un microprocesador capaz de ser la unidad central de proceso de un micro, los fabricantes llegaron al caso de crear su sistema propio o acogerse a un sistema ya enraizado, haciendo sus máquinas compatibles con las demás. La respuesta fue la instalación del CP/M y su popularidad.

El sistema operativo es el puente entre el usuario y la máquina; llegamos entonces a que si la unidad central cambia, y con ello la circuitería, hay que cambiar el sistema operativo. Surgieron entonces multitud de sistemas para multitud de procesadores que se habían fabricado desde el primer 8008 de Intel, el primer procesador de 8 bits. Estos sistemas fueron, aparte del mencionado CP/M, para los 8008, 8080, 8085 y Z80:

-OASIS, MP/M y MSX-DOS para el Z80,

-ProDOS, DOS 3.3 y SOS para el 6502,

-CP/M-86 y MS/DOS para los 8086 y 8088 (16 bits),

-PICK, XENIX y UNIX para el 68000 (16 bits), nuevos sistemas en los que la multitarea está ya presente.

Datos de 1986. Desde entonces, el uso del MS/DOS ha aumentado considerablemente por el incremento de los PCs.

AMBITO DE LOS SISTEMAS	
CP/M	*58%*
MS/DOS	*29%*
APPLE DOS	*24%*
UCSD P-Sys.	*12%*
CP/M-86	*2%*
UNIX	*2%*
OASIS	*2%*
PICK	*1%*
Otros	*4%*

APARIENCIA DEL SISTEMA

Hoy día se da más importancia a la apariencia externa del sistema que a su funcionamiento interno (por supuesto, sin descuidarlo). Desde que se diseñan los primeros sistemas, en que las órdenes son introducidas por teclado con una especial sintaxis, se sigue una línea muy en ese estilo de forma, variando la sintaxis hasta acercarla al lenguaje humano de una forma muy simple, pero accesible por más usuarios, que veían los ordenadores como máquinas muy difíciles de usar. Esta línea de diseño comienza a finales de los años 40 como simples rutinas en papel perforado que el usuario introducía para realizar la tarea para la que eran destinadas. Más adelante, estas rutinas estaban ya en la memoria del ordenador, ocupando su memoria, y se llamaban con unas órdenes específicas. Y más adelante todavía, dichas rutinas ganaron en complejidad y se requería un intérprete mucho más complicado para acceder a ellas. Todas las órdenes eran accesibles por teclado.

Una línea de diseño aparte nació al tiempo que salía a la venta el famoso Apple MacIntosh. Su sistema operativo organizó un revuelo tal, que hoy día se imponen versiones suyas en todos los ordenadores que salen al mercado.

Muchas son las particularidades de su diseño. En primer lugar, no aparece la pantalla "fría" que se ve en los sistemas con entrada por teclado, sino que ésta está repleta de símbolos o dibujos (iconos) que representan las funciones a las que podemos acceder. Y en segundo lugar, la forma de seleccionar estos iconos. Ya no es por teclado, sino por un indicador que se desplaza en pantalla gracias al "ratón": cuando movemos el "ratón" sobre la mesa hacia la izquierda, el indicador (generalmente una flecha) hace lo mismo en pantalla.

Así tenemos un sistema operativo totalmente nuevo para el usuario. Cuando se quiere ver el directorio de un disco, basta indicar el icono que representa el disco. Entonces se abre una ventana de

información en pantalla que nuestra su contenido. Y aún hay más. Podemos mover el contenido de la ventana, seleccionar un fichero, llevar un fichero a la papelera (borrarlo), al icono de otro disco (para copiarlo), etc, de la forma más fácil, pues todo está representado por un icono sobre el que podemos actuar con nuestro ratón y su indicador. Y si queremos más, podemos seleccionar un icono y luego acceder a la parte alta de la pantalla, en la que un menú desplegable nos indicará qué podemos hacer con ese icono.

El objetivo de estos sistemas es, realmente, alejar al usuario del hardware o sistema físico y aproximarlo a lo que de verdad quiere hacer, de una forma lo más interactiva posible. Este tipo de sistemas de denomina WIMP, iniciales de "Windows, Icons, Mouse and Pull-down menus" (Ventanas, Iconos, Ratón y Menús desplegables), o también WYSIWYG (What You See Is What You Get), o, en español, "Lo que ves es lo que obtienes", para hacer referencia al sistema de elección por ratón e iconos.

Este sistema operativo, desde que fuera usado por el Mac hace varios años, ha conocido múltiples versiones; algunas mejoran el original de Apple y otras no. Verdaderamente útil, la mayoría de los fabricantes tienen su sistema a base de ratón, y raro es el ordenador que se vende sin él. Después del Mac tenemos el Atari ST (1985), que usa el GEM diseñado por Digital Research (creadores del CP/M), y que también se incorpora en los PC; el Commodore Amiga (1986), con su versión especial; el Arquímedes (Francia); el PS/2 de la IBM...

ARITMETICA BINARIA

En esta parte del artículo nos referiremos al cálculo en sistema binario, base 2: suma, resta, multiplicación, división, rotaciones y puertas lógicas, fundamentales en el funcionamiento del ordenador. Este repaso pretende ser sólo eso, ya que, más que nada, podría interesar al programador que use su micro en código-máquina. Un programador que lo haga en otro lenguaje tal vez, simplemente, escriba "PRINT 43.87 +635342.63", sin llegar a mayores profundidades de cálculo.

ARITMETICA BASICA

1. SUMA Y RESTA

Para hacer una suma o una resta en binario sólo hay que seguir las tablas que acompañan este texto, teniendo en consideración el acarreo:

El acarreo es un concepto muy básico en la aritmética del procesador. Consiste en un bit que normalmente está a cero, y marca un uno al término de una operación cuando ha habido un "desbordamiento" (cuando usamos bytes y la representación del resultado ocupa algún bit más de los ocho permitidos). Este concepto lo usamos, como operaciones que puede realizar un microprocesador, en suma y resta. Si ve en la tabla de la suma el resultado de sumar 1+1 (como dígitos binarios, bits), nos da 10. Esto significa que debemos poner un 0 en la posición y "llevar 1", de la misma forma que lo hacemos en una suma corriente, para la siguiente suma

SUMA		*RESTA*	
0 + 0	0	0 - 0	0
0 + 1	1	0 - 1	11
1 + 0	1	1 - 0	1
1 + 1	10	1 - 1	0

parcial. En la resta pasa algo parecido. Al intentar restar 0-1, nos encontramos con que deberíamos poner un -1. Esto lo resolvemos con un 1 en su lugar correspondiente, y un 1 al acarreo para la siguiente resta parcial.

2. MULTIPLICACION Y DIVISION

Estas dos operaciones son más difíciles de realizar en binario. Normalmente, el procesador lo hace por el programador, excepto en el caso de algunos microprocesadores de 8 bits, como es el caso del archiconocido Z80. Hay que inventarse, pues, métodos de multiplicación y división que suplan esta carencia.

Un método que sirve en código-máquina para la multiplicación en binario, por la rapidez del procesador, que no del método, es el de las sumas sucesivas, y, análogamente para la división, las restas sucesivas. El de sumas consiste en añadir a una variable (o registro) que parte de 0 el valor B tantas veces como indique el valor A (al multiplicar A*B; el orden es indistinto). Se puede usar la misma variable B como acumulador, y así usar sólo dos registros para la operación (los dos datos almacenados).

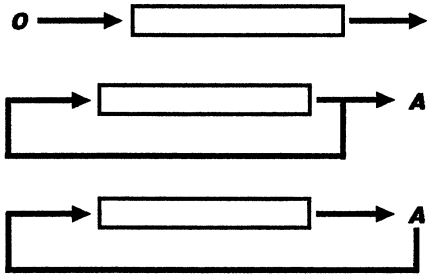
De la misma forma será el de las restas, con la salvedad de que al valor A le será "quitado" el valor B (en la operación A:B) hasta que A sea menor que B, momento en que no se podrá restar más veces. Lo que quede en el registro que contenía A en un principio será el resto de la división, y el número de veces que ha sido restado el valor B será el cociente. (La verdad, no sé si se habrán dado cuenta, pero enseñar a un ordenador es como enseñar a un niño de muy corta edad; así es como aprendemos a multiplicar y dividir).

3. ROTACIONES

Es un tipo de operación que "saben" realizar todos los procesadores. Consiste en desplazar, bien hacia la derecha o bien hacia la izquierda, el contenido de la cadena de bits elegida (de 8 en procesadores de 8 bits, etc), introduciendo, en el hueco que queda libre: o un 0, o lo que sale por el otro lado, o el contenido del acarreo.

Por ello, el acarreo es parte importante también en este tipo de operaciones, ya que, como se ha visto, se puede especificar la clase de rotación dependiendo del acarreo (que actúa como mero acumulador).

Las rotaciones más importantes son las tres indicadas en la figura, que son análogas cuando se trata de rotación al otro lado.



La primera, la más simple, entra un cero y se guarda el saliente. En la segunda, el bit saliente por un lado vuelve a entrar por el otro. Y la tercera es una variación de la segunda, en la medida que no entra el bit saliente de la cadena, sino que entra el que estaba en el acarreo.

La utilidad más inmediata que podemos encontrar para las rotaciones es la multiplicación por 2, que se realiza simplemente rotando hacia la izquierdala cadena, y la división entre 2, rotando hacia la derecha.

4. PUERTAS LOGICAS

Las puertas lógicas son unos microcircuitos lógicos provistos de una o dos entradas y de una sola salida. El bit obtenido por la salida es el resultado de la combinación de los que entran, dependiendo del tipo de puerta. Hay tres puertas lógicas principales, que son

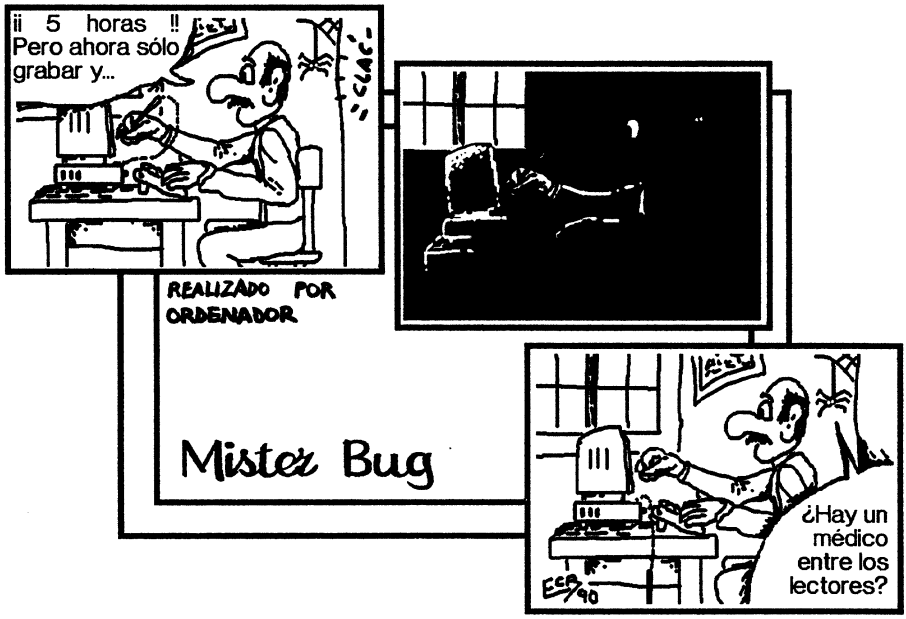
AND			OR			XOR			NOT		NAND			NOR		
0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1
0	1	0	0	1	1	0	1	1	1	0	0	1	1	0	1	0
1	0	0	1	0	1	1	0	1			1	0	1	1	0	0
1	1	1	1	1	1	1	1	0			1	1	0	1	1	0

AND, OR y NOT, y en la mayoría de los micros existe también la puerta XOR o EOR (en el código máquina, no en el BASIC ni en otros lenguajes comunes).

Estas puertas (también son funciones lógicas realizadas por el microprocesador) se pueden ver en la tabla que acompaña a este artículo; por un lado, las entradas, y en el otro, separado, su resultado o combinación.

Las funciones AND, OR y NOT tienen un significado real, no son combinaciones hechas al azar. AND devuelve un 1 sólo si las dos entradas son señales excitadas; OR devuelve 1 con que exista alguna excitada, y NOT devuelve la contraria a la que entra. Las otras son lo que serían NOT AND (NAND) y NOT OR (NOR), y la función especial XOR o EOR (extended OR).

Y veamos un detalle. Supongo que alguna vez se habrá preguntado cómo pueden dos impulsos llegar al mismo tiempo a una puerta por lo intrincado de los circuitos. Pues bien: Todos ellos están sincronizados por un reloj interno que está íntimamemte ligado al procesador de la máquina, que le marca la frecuencia de trabajo correcta para que los impulsos "avancen" sincronizados. Por ello, cuanta más frecuencia del reloj (más impulsos por segundo), más rápido va el ordenador.



HARDWARE

Marcos Vaqueiro Rodríquez

Se abre en este número una serie de artículos sobre la robótica orientada al ordenador. Múltiples montajes tendrán cabida en estas páginas, que intentarán potenciar un poco más los enlaces con el exterior de nuestra máquina.

¿Un sueño? ¿Una afición? ¿Una realidad?... Lo que tú quieras. Dale una nueva vida a tu pequeño ordenador. Todos sabemos que son viejos, que están atrasados, que son lentos... Pero ¿quién te impide darle una nueva forma de actuar, una nueva forma de vida?

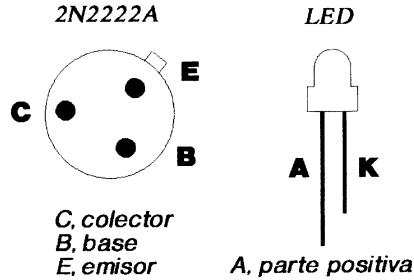
Ese cacharro viejo y que muchos desprecian tiene siempre una salida o conexión al exterior llamada Bus o Puerto de Comunicaciones. A través de él puedes conectar a tu ordenador un sin fín de tarjetas, bien compradas o bien diseñadas por tí... Esta es la aventura que te ofrezco: circuitos para controlar juegos de luces, robots conectados a tu ordenador, tarjetas digitalizadoras, procesadores en paralelo, generador de imágenes y nuevos juegos... Todo lo que siempre soñaste lo puedes intentar con tu maquinita... Todo siempre que te llegue el presupuesto, claro.

Tu primera pregunta es: "¿Riesgos?": Si se hacen bien los cálculos, si se estudia bastante y se verifica cada montaje, entonces el mayor desastre que se debería producir en tu ordenador es el que se le queme el fusible... (Costo de la reparación= 35 pts).

Muchas de las cosas que te voy a contar desde aquí están extraídas de artículos, libros y revistas... Otras saldrán del computador que la naturaleza me ha implantado sobre los hombros, y, el resto, saldrán de tí... ¿Te artreves a intentarlo?

Naturalmente este trabajo requiere un doble conocimiento. Uno informático: estructura de la memoria, de la CPU, de código/máquina, o por lo menos de los POKEs de tu ordenador. Otro electrónico: lo mínimo para saber montar un circuito que veas, y un montón para llegar a diseñarlos, simplificarlos, etc...

En la línea de arriba aparece el primer problema para la mayoría de la gente. Yo, por mi parte, te explicaré, siempre que sea posible, cómo ha sido diseñado el circuito correspondiente.



(Transistor visto desde abajo)

Para empezar esta vez, te voy a contar un par de batallitas sobre dos semiconductores que emplearemos después. Ellos son el transistor y el diodo LED.

El transistor tiene dos sistemas de trabajo: uno lineal (amplificadores, etc) y otro digital (conmutadores, etc). Nosotros vamos a emplear la segunda.

Un transistor que trabaja en forma digital, actúa de "interruptor" de la corriente que entra por dos de sus terminales, según sea la señal que entra por

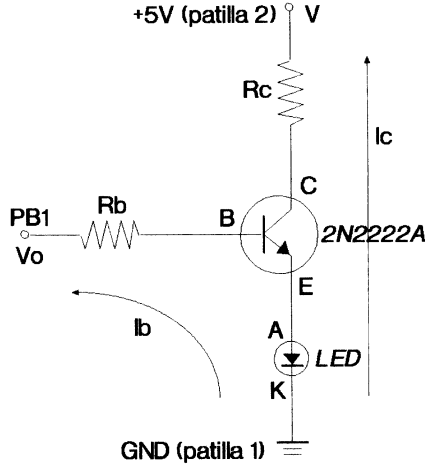
la tercera. Sus patillas son: **colector** (por donde entra la corriente), **emisor** (por donde sale), y **base**, la patilla de control.

El diodo LED es un semicnductor que emite luz cuando pasa corriente a través de él, en una determinada dirección.

El circuito que vamos a diseñar hoy aquí es un interruptor que enciende o apaga un LED según la señal que recibe del ordenador.

Supongo que todos vosotros conoceréis algo de electricidad (es indispensable para poder hacer algo aquí), y que sabréis la Ley de Ohm que dice **I=V/R**, donde **I** es la intensidad, **V** el potencial y **R** la resistencia del circuito.

En la siguiente figura se encuentra el esquema del circuito que vamos a diseñar:



Dif. de potencial AK= 1.7 V
Dif. de potencial BE= 0.7 V
Ic= 20 mA

Una vez fijadas las tensiones **V** y **Vb** que vamos a meter en el circuito, y una vez seleccionada para la tensión **V**, la **hFE** correspondiente (es la ganancia de corriente del transistor, se encuentra en los manuales), y teniendo en cuenta la tensión que consume el LED, y que la intensidad máxima que lo puede atravesar es de 20 mA, vamos a proceder a calcular los componentes **Rc** y **Rb**:

Sea **Ic** la intensidad del colector (20 mA máx.) y sea **Ib** la de la base; entonces: **Ib=2*Ic/hFE**, y como el voltaje que va a consumir para producir la conmutación es de 0.6-0.7 volts, según si es de germanio o de silicio, tenemos, aplicando la ley de Ohm:

Ib=(Vb-0.7)/Rb, luego

Rb=(Vb-0.7)/Ib,
Rb=(Vb-0.7)*hFE/(2*Ic)

Ahora es necesario calcular su potencia. Para ello aplicamos la ley de Joule: **P=Rb*Ib^2**. Si buscamos en el manual la potencia que disipa (W máx), ha de cumplirse que P es mayor a W max para que no se queme el circuito.

Dado que el LED consume 1.7 Volts, por la ley de Ohm, resulta:

Rc=(V-Vl)/Ic = (V-1.5)/Ic,

que también normalizaremos por el proceso anterior.

(El ejemplo para montar venía ya calculado en el tomo II del libro sobre código/máquina del C-64 de F. Montell. Expongo ese circuito por comodidad, pues vienen ya especificadas las conexiones al ordenador.)

Tomamos los datos siguientes:

- **hFE** del 2N2222A como 153.5
- **V=Vb**=5 Volts
- **W max**=0.5 Watts

Por las fórmulas anteriores y tomando Ic=33 mA, tenemos que:

Rc= (5-1.7)/33E-3 =100 Ohms;
P= 100*(33E-3)^2 =0.1089 Watts;

por tanto Rc normalizada será de 100 Ohms 1/2 Watt.

Ib= (2*Ic)/153.5 =0.43 mA, luego
Rb= (5-0.7)/0.43E-3 =10000 Ohms,

y la potencia que disipa

P= 1.849E-3 Watts,

por lo que la normalizamos como 10 Kohms 1/2 Watt ó 1/4 Watt.

Las conexiones al ordenador deben realizarse a través de un conector. **NO SOLDAR NUNCA EN LA PLACA INTERIOR.**

Antes de conectarlo, aplícale tensión con una pila de 5V y verifica que funciona.

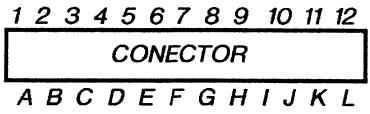
Para el C-64 las conexiones a realizar, si empleamos el puerto de I/O, tal como se especifica en el manual del C-64, las patillas del conector son las siguientes:

Las terminales especificadas en nuestro circuito son la PB1, patilla D; la entrada de potencial +5V, que es la patilla 2; y la toma de masa o GND, que es la patilla 1.

A continuación vamos a teclear las líneas que se hallan a pie de página (presento el listado en BASIC y en código-máquina). En el listado BASIC, en la línea 40, pone POKE 56577,2 porque en binario 2 es 00000010, y el PB1 corresponde al segundo bit de la posición de memoria.

PIN	*TIPO*	*NOTAS*
1	GND	máx 100 mA
2	+5V	
3	RESET	
4	CNTI	
5	SP1	
6	CNT2	
7	SP2	
8	PC2	
9	Ser. ATN IN	
10	9 VAC	máx 100 mA
11	9 VAC	máx 100 mA
12	GND	

PIN	*TIPO*
A	GND
B	FLAG2
C	PB0
D	PB1
E	PB2
F	PB3
G	PB4
H	PB5
I	PB6
J	PB7
K	PA2
L	GND



LISTADO BASIC

```
10 POKE 56579,2 ; sitúa a PB0 en entrada y a PB1 en salida
20 POKE 56577,0 ; pone a 0 la salida PB1; luego el LED se apaga
30 FOR I=1 TO 20: NEXT
40 POKE 56577,2 ; pone a 1 la salida PB1; el LED se enciende
50 FOR I=1 TO 20: NEXT
60 GOTO 20 ; cerramos el bucle
```

LISTADO CODIGO-MAQUINA

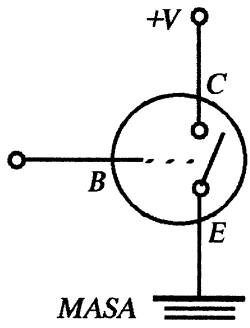
```
C000› LDA #02 ; carga acumulador con el valor 2
C002› STA DD03 ; almacénalo en #DD03= 56579
C005› LDA #00 ; apaga el LED
C007› STA DD01 ; Poke 56577,0; SE APAGA EL LED

C000› LDA #02
C002› STA DD03
C005› STA DD01 ; SE ENCIENDE EL LED
```

Bueno, hasta aquí llega este ejemplo de utilización del puerto de comunicaciones del ordenador, aplicado en este caso al C-64.

Si tu transistor necesita más de 5V para trabajar, no lo emplees, pues no funcionaría el circuito (elige otro modelo). Para trabajar con otras tensiones, necesitamos un Adaptador de Niveles Lógicos, pero eso lo dejamos para otro día...

Cualquier problema que puedas tener con los circuitos aquí presentados, no dudes en consultarlo con la asociación, escribiendo a la dirección indicada en esta revista.



En la figura se ve un esquema del funcionamiento del transistor. El puente entre +V y MASA se mantiene abierto mientras pase corriente por la base.

PROCESAMIENTO EN PARALELO

Telmo Lago

TRANSPUTERS

EL PROYECTO TRANSPUTER

A mediados de 1979, un equipo de investigación perteneciente a una reciente empresa británica llamada Inmos Technologies, compuesto por D. May, I. Pearson, R. Taylor y C. Whitby-Strevens comienza a estudiar la posibilidad de crear un nuevo tipo de componente especialmente dedicado a la arquitectura en paralelo, con posibilidad de interconexión múltiple entre un cierto número de ellos.

En la feria de París de 1983 se presentó una versión preliminar de lo que sería el futuro Transputer. Un chip de 45mm de cada lado. Para los de Inmos parecía ir todo viento en popa, pero en cuanto cayó el Partido Laborista de la presidencia del Reino Unido en favor de los conservadores, la señora Thatcher decidió que ese proyecto (y otros muchos de otra gente) era absolutamente inútil, y retiró toda clase de subvenciones que tenían para la investigación y desarrollo del Transputer. Por suerte, hubo una multinacional americana que se interesó por el proyecto y lo financió. Así, en 1985 se presenta la versión definitiva del Transputer, denominación que después se convertiría en el nombre técnico que designaría a todo componente programable de alta integración con capacidad de ser utilizado como CPU y con posibilidad de interconexión múltiple con otros de idénticas características.

El modelo de Transputer más sencillo que hay en estos momentos es el T414. Es un procesador de 32 bits que permite acceder a una gran cantidad de memoria y estar conectado a cuatro canales de comunicación independientes a

una velocidad máxima de 10 millones de bits por segundo en un programa secuencial. Esta elevada velocidad se consigue gracias a la utilización de una trasmisión unidireccional en serie (los bits van uno a uno). La razón primordial es reducir el número de conexiones.

La última "criaturita" que dió a luz el Inmos es el T800, en el proyecto Super-Node del programa ESPRIT (European Strategic Programme for Research and Development in Information Technology) de la CEE. Aparte de tener una mayor velocidad de proceso, incluye una unidad de cálculo en coma flotante con una velocidad de 2 millones de operaciones por segundo (en el modelo T800-30), y los cuatro canales de comunicación pueden funcionar simultáneamente. En un principio puede parecer difícil de ver cómo es posible de hubicar algo tan revolucionario en un cuadradito de 1cm de lado, pero se debe a que sus diseñadores se basaron en algunos conceptos de la filosofía de las máquinas de tecnología RISC.

EL OCCAM

En vísperas de aparecer el proyecto transputer, ya se estaban ideando nuevos lenguajes de manera que la programación en código-máquina fuera tan sencilla (o casi) como utilizar el lenguaje humano. Lo que debía primar ante todo era la síntesis. El lenguaje con estas características más adecuado para los transputers era el Occam.

Este lenguaje fue diseñado a partir del modelo CSP (Communicating Sequential Processes) de la Universidad de Oxford por el equipo C. A. R. Hoare. El nombre se debe a Guillermo de Ockham, famoso filósofo inglés, y a su "principio de la navaja", consistente en simplificar al máximo y eliminar todo lo suérfluo.

El Occam es algo más que un simple lenguaje de programación, Es un formalismo para el diseño correcto de programas que van a correr sobre máquinas paralelas (entiéndase máquinas con proceso en paralelo), de tal forma que puede llegarse a saber si un programa Occam es "verdadero" o "falso", de la misma forma que si fueran instrucciones lógicas. Además, los programas pueden expresarse como una serie de procesos independientes pero concurrentes, de manera que vayan a ser ejecutados por uno o varios transputers. En estos procesos existe una abstracción tal que lo que es el proceso de programación en sí deshecha todas las interacciones entre los diversos componentes del ordenador.

Un proceso Occam intercambia información a su vez con otros mediante "mensajes" enviados a través de canales. Estos últimos son las conexiones hardware entre transputers. Estos procesos se componen de la combinación adecuada de tres instrucciones: **afectación**, que asigna un valor a una posición de la memoria principal, **entrada**, que espera un mensaje proveniente de otro proceso, y **salida**, que envía un mensaje. Estas instrucciones se pueden combinar de manera que formen uno de estos tres tipos de procesos: secuencial, paralelo y alternativo. Este último permite al procesador que espere un cierto tiempo interno por mensajes en determinados canales.

ASOCIACION DE TRANSPUTERS PARALELISMO MULTIPLE

El reparto de procesos en una máquina paralela puede hacerse de diversos modos. El primero que probablemente le venga a la cabeza a todo el mundo es el de poner un transputer para cada proceso. Es absoluta y totalmente factible, pero en la práctica es tremendamente costoso (a nivel económico). Lo más práctico es interconectar los transputers de manera que formen una red geométrica, siendo así el reparto de procesos óptimo. De este modo, cada transputer hace varios procesos paralelamente, repartiendo su tiempo en cada uno. Para ello se integra en el circuito un

gestor de planificación de tareas (más conocido como "scheduler"), que atribuye a cada proceso el tiempo de su ejecución.

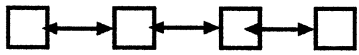
En una máquina de procesadores múltiples existen varios métodos de utilizar el paralelismo. El más difundido es el tratamiento en "pipeline", que consiste en descomponer una tarea en diversas subtareas ejecutadas en cascada por otras unidades de cálculo. Otros métodos menos utilizados son el tratamiento vectorial (procesadores organizados como vectores) y el matricial, en los que cada procesador realiza una tarea determinada sobre cada componente del vector o matriz.

Un enfoque diferente tiene el tratamiento sistólico, en el que los procesadores trabajan a la misma cadencia e intercambian datos en el mismo momento.

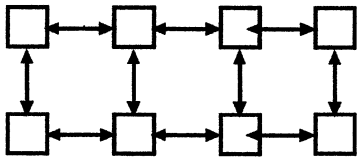
Por último, el tratamiento concurrente o asíncrono, en el que cada procesador hace una tarea de nivel local a su ritmo existiendo sincronismo a la hora de recibir un mensaje. En este tipo de procesamiento hay dos tipos de arquitectura: conexión fija (red física de interconexión) y conexión reconfigurable (los protocolos de interconexión varían según la estructura del programa a ejecutar).

En las máquinas de conexión fija prevalece el propósito de minimizar el número de conexiones entre procesadores y el de procesadores intermedios a utilizar. A partir de este principio se han ideado diversas topologías o "nodos". Cada uno de ellos está compuesto de uno o varios procesadores, una memoria y un circuito de comunicación. Algunas de estas configuraciones están diseñadas para una aplicación específica, como la arborescente, empleada para la Inteligencia Artificial, o la red de mallas, utilizada en tratamiento de imágenes (ver figura). También es posible la estructura de anillo, pero presenta el inconveniente de su lentitud, ya que todos los datos tienen que pasar por todos los transputers aunque no sea necesario, al igual que los transputers en serie. Pero la más universal, sobre todo por su versatilidad (puede adaptarse a todo tipo de aplicaciones) es la estructura llamada "hipercubo", que como dice su nombre es una red de transputers en una configuración cúbica, similar a la estructura cristalina de algunos minerales. Fue utilizada por primera vez en el California Institute of Technology de la mano de C. Seitz y G. Fox. Posteriormente la empezaron a utilizar Intel, en su departamento de ordenadores para aplicaciones científicas, Thinking Machine Corp., para el desarrollo de su Connection Machine, y Floating Point Systems Corp. en su serie T.

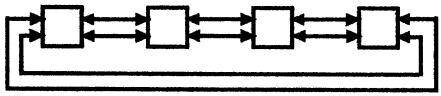
PIPELINE



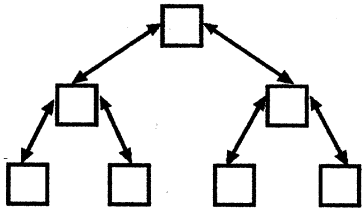
RED DE MALLAS



ANILLO

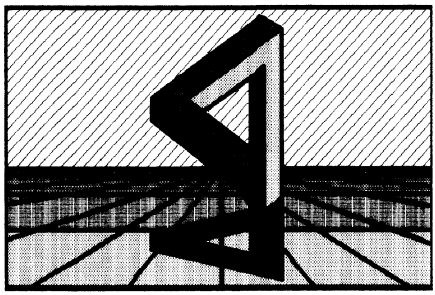


ARBORESCENTE



En esta serie T, se desarrolló un coprocesador matemático para trabajar conjuntamente con los transputers en cada nodo, con lo que cada hipercubo puede llegar a tener la misma potencia de cálculo que el más potente ordenador de la firma norteamericana Cray Research.

Aparte de estas arquitecturas, también se han creado tarjetas con transputers para instalar en ordenadores personales (PC, Apple Macintosh, Commodore Amiga, Atari ST) o diversas estaciones de trabajo.



ALGORITMOS

Jesús Cea

EMULACION DE LA RECURSIVIDAD

Había pensado empezar en este número una serie de artículos sobre programación de objetos en 3D pero, después de releer por enésima vez el artículo de este mes sobre fractales recursivos, decidí aplazar mi proyecto una temporada, y dedicar la sección de este número a explicar cómo se emula la programación recursiva en un lenguaje que carezca de ella, habida cuenta que es una tarea que interesa a mucha gente, dadas sus amplias aplicaciones e implicaciones en todos los campos. Además, no todo el mundo tienen la capacidad y/o el tiempo preciso para este trabajo y no estaría nada mal que alguien les echara una mano.

INTRODUCCION

¿Qué es la programación recursiva? Si ya has leído el artículo de fractales de este número seguro que tienes una idea bastante aproximada del asunto. Si no lo has leído te recomiendo que lo hagas para que luego puedas devorar lo que sigue sin atragantarte.

Básicamente, la programación recursiva consiste en dividir un problema grande en partes más pequeñas y más sencillas de tratar y, a su vez, éstas en subpartes aún más pequeñas y sencillas, y así sucesivamente (si esto no te suena de nada, o no has seguido mi consejo al principio de la introducción, o estás amnésico perdido). Al contrario que los fractales recursivos, donde este proceso sigue hasta el infinito (a costa de una demora infinita, ¿o qué creías?), la ejecución de una rutina recursiva tiende hacia un valor definido llamado base, que marca el fin del proceso de recursión. La elección de este valor base es crítica, y de ello depende que nos salga una virguería o un cristo informático de mucho cuidado. La recursividad también puede ir de las partes al todo o incluso análisis sectorial del todo. La ventaja de este sistema es que, en cualquier nivel en que nos encontremos, el sub-subproblema es semejante a su problema superior y éste, a su vez, es semejante al

problema global a tratar (si esto tampoco te suena, eres un clarísimo caso de diván). Esto posibilita que una misma rutina sea la encargada de analizar tanto el sistema entero como cada uno de sus constituyentes.

MODERNIZATE

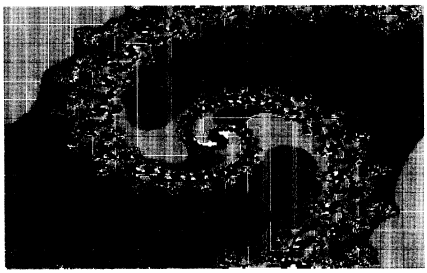
La programación recursiva es un concepto relativamente novedoso. Los primeros lenguajes de ordenador (léase BASIC, FORTH, FORTRAN) carecían total y absolutamente de esta capacidad. Actualmente, la recursividad es un factor determinante de primer orden, y prueba de ello es que todos los modernos lenguajes de programación la contemplan. Resulta evidente que, a medida que los lenguajes han ido evolucionando, tienden a una mayor estructuración, modularidad y recursión (no hay más que pensar en el PASCAL o en el "C"). Esto dice mucho acerca de la importancia del tema.

Paradójicamente, aún hoy en día existen programadores experimentados que prefieren no utilizar la programación recursiva. Parece ser que ello se debe a que tienen dificultades para entenderla. Yo prefiero pensar que se trata más de un desconocimiento del tema (falta de interés) o del hábito adquirido a través del uso de los lenguajes primitivos (¡¡quiero desintoxicarme...!!). Personalmente, entender el concepto me parece sencillo, si bien reconozco que para seguir la pista a una rutina recursiva en pleno funcionamiento se requiere tener las ideas muy claras y haber dormido bien la noche anterior. Pero puedo asegurarte que si no dominas la materia te estás perdiendo una de las facilidades más interesantes y útiles de la informática actual (y de la futura).

Increíblemente y aunque resulte gracioso, todos los libros que tratan el tema (y siempre muy de pasada) utilizan unos ejemplos que, antinómicamente, son más rápidos, cortos y sencillos si se programan por el sistema clásico que por el recursivo; todo lo contrario de lo que pretenden demostrar. El ejemplo más común es el del cálculo factorial, pero hay muchos otros. Dado que estos ejemplos típicos lían la madeja más que clarifican, me inventaré yo unos que sirvan para explicar la idea básica y, además, que sólo sean viables recursivamente.

Como primer ejemplo, un resabio de los últimos exámenes: Supongamos que estamos estudiando Filosofía y encontramos una palabreja cuyo significado se nos escapa (experiencia ésta bastante frecuente, por cierto). Marcamos dónde estábamos leyendo y salimos al galope a por un diccionario (esto ya no es tan frecuente). Encontramos la definición de la palabra (más inusual todavía), pero en ella descubrimos más palabras cuyo significado desconocemos (sin comentarios). Buscamos las siguientes definciones, y nos remitirán a otras palabras, y así hasta que encontremos términos familiares. La longitud del proceso (valor base) queda determinada por nuestro nivel cultural (si tienes un nivel bajo, no me responsabilizo de posibles crisis nerviosas o fracasos conyugales). Una vez que ya hemos encontrado todas las definiciones que precisamos, volvemos cada vez a la definición anterior y la completamos con las que ya tenemos, y esta definición ya completa servirá para completar (si no te gustan las redundancias, búscate un diccionario...) la del nivel superior y ésta, a su vez,... hasta que llegamos de nuevo a la palabra del libro de Filosofía, cuyo sencillo significado, llegado a este punto, ya conocemos (la mayoría de la gente habría optado por el suicidio antes de haber llegado aquí). Si quieres hacerte una idea aún más clara piensa en las muñecas rusas, unas dentro de otras.

Otro ejemplo: Supongamos que nos encontramos inmersos en un laberinto y que sólo disponemos de una cuerda, infinitamente larga, para salir. ¿Qué hacer? Primero creamos un convenio: derecha, centro, izquierda. Luego atamos un extremo de la cuerda en el lugar en el cual nos encontramos, para señalarlo como raíz. Empezamos a caminar hacia delante, desenrollando la cuerda hasta



encontrar una encrucijada. Según el convenio, doblamos a la derecha y seguimos hasta el siguiente cruce, en el que también doblamos a la derecha. El proceso se detiene cuando salimos (que te lo has creído) o hasta que tropezamos con la cuerda que hemos tendido antes (lo que indica que por ahí ya pasamos). En este momento procedemos a recoger la cuerda, desandando el camino hasta la última encrucijada. Una vez en ella, procedemos a explorar la rama del centro y luego la de la izquierda, siguiendo el mismo sistema (recordad el convenio). Cuando terminamos de explorar las tres ramas, recogemos la cuerda hasta la siguiente encrucijada, desde la que tomaremos la rama del medio, de la izquierda, luego recogemos cuerda,... Si el laberinto tiene una salida, el éxito está garantizado, aunque si es muy grande puede llevarnos un buen rato. Si durante nuestras "vuelta atrás" (suelen llamarse podas) llegamos otra vez al punto de partida, podemos estar seguros de que el laberinto no tiene salida (estamos encerrados).

Con estos dos ejemplos creo que ya podemos enumerar las características esenciales de un proceso recursivo: Posibilidad de llamarse a sí mismo y variables locales. Lo último es imprescindible para no perder la información al saltar y regresar del siguiente nivel. Por lo tanto, para emular la recursión tendremos que emular ambos procesos. Vamos por partes:

a) LLamarse a sí mismo: No basta sólo con "ir", sino que hay que dejar un "rastro" para poder regresar luego. La única instrucción BASIC capaz de hacer esto es "GOSUB". Sin embargo, en muchos ordenadores está muy limitado (en el C-64 creo que son unas 7). En estos casos no queda más remedio que hacer las llamadas "a mano": Guardamos en una matriz (array) un puntero a la línea a que tenemos que regresar y saltamos a donde queramos con "GOTO" (y no con GOSUB). Cuando la última llamada termina, recoge el valor en el array y regresa también usando "GOTO" o "ON x GOTO" (y no con RETURN). De esta forma evitamos perder el contacto entre rutinas a distintos niveles. El único límite es la memoria disponible. En caso de que dispongas de un ordenador que no limita las llamadas, tienes medio problema resuelto. (Lógicamente, cuando guardamos un puntero en el array, debemos incrementar el índice del mismo, y, cuando lo leemos, decrementarlo, simulando una pila "LIFO").

b) Variables locales: Al contrario que las globales, las variables locales sólo "se ven" dentro de los procedimientos donde están definidas. Esto es imprescindible en la recursividad, permitiendo tener variables del mismo nombre a distintos niveles sin conflicto alguno (una misma rutina a distintos niveles puede considerarse como rutinas distintas). Además, cuando regresa de un nivel inferior no pierden sus valores. El problema es similar al apartado anterior y está claro que la solución es la misma. Sin embargo, dado que puede ser muy lento operar constantemente con matrices, utilizamos variables

normales, cuyos valores son copiados a/desde arrays sólo cuando se salta o se regresa de otro nivel.

Algo muy importante será que cada programa que hagamos será distinto según lo que queramos preservar y lo que no. La mejor guía es la experiencia.

Como ejemplo ilustrativo, veamos cómo funciona un generador de fractales recursivos como los descritos en este mismo número (copo de nieve).

Lo primero es preguntar al usuario cuántos niveles quiere calcular (Q). Se copia Q en N y se salta a la rutina de trazado con (X,Y), (X1,Y1) como extremos de la línea a calcular.

La primera línea de esta rutina comprueba si N es cero. En caso negativo, calcula el vector correspondiente a la línea, lo divide entre 3 y lo guarda en un array (con índice N). Se recalcula X1, Y1 con el nuevo vector (3 veces más corto) y se salta a una rutina que guarda los valores viejos de X, Y, X1, Y1 en un array de índice N, decrementa N, y salta de nuevo al principio del procedimiento, con los nuevos valores (X,Y), (X1,Y1) y N (decrementado una vez), repitiéndose el proceso.

En caso de que la comprobación del principio sea cierta, dibuja una línea entre los valores actuales (X,Y), (X1,Y1) y salta a una rutina especial encargada de recuperar los valores viejos de (X,Y) y (X1,Y1), el vector V e incrementar N. Se salta a la línea inmediatamente posterior, encargada de la siguiente sección de cálculo de trazado a no ser que N sea mayor que Q, lo que indica que ya se terminó de trazar el dibujo, por lo que se regresa al programa principal.

Como podéis observar, seguir el ritmo de un proceso recursivo precisa mucha imaginación. Sin embargo, es una herramienta inmensamente potente y eficaz. Haz un esfuerzo y sigue en la brecha.

Parece que mi artículo sobre algoritmos de compactación del número 1 levantó bastantes comentarios (bueno, parece que no lo leí yo sólo). Algunos comentan que soy demasiado técnico, mientras otros me dicen que peco de superficial. Dada la poca fiabilidad que me brinda un sólo número, esperaré más comentarios antes de optar por un cambio en uno u otro sentido (aunque este artículo es un experimento).

También se me acusa de tener un cierto estilo "peculiar". En este sentido, subrayar que estoy abierto a toda clase de crítica constructiva que reciba.

Finalmente, señalar un comentario que recibí de un lector sevillano (hola, Javier). Parece ser que conoce unos sistemas similares a mi algoritmo de compactación de bits. Quiero dejar claro que se trata de algoritmos de aplicación específica (compresión de cadenas de texto) que sólo pueden ser utilizadas por el propio programador del fichero fuente. Mi sistema es radicalmente distinto. Tiene un ámbito de programación muy general (no sólo textos, sino programas enteros) y no se requiere ningún co-nocimiento de la organización interna del programa a comprimir (de hecho, es el compresor el encargado de analizar el programa). Además, es destacable el hecho de que el compresor no sólo examina el programa, sino que también lo modifica buscando la estructura más ventajosa. Los algoritmos a los que se refiere Javier probablemente sean los primeros (y más utilizados) en la historia de la Informática. Cualquier parecido es puramente accidental. De todos modos, me alegra recibir puntualizaciones con un mínimo de base. Además, siempre es útil saber que hay alguien pendiente de cualquier error, pero en este caso creo que eres tú quien se equivoca (aunque podría ser yo el equivocado). Aún así, estoy deseando recibir más comentarios de cualquier tipo (al menos hay alguien con un mínimo de interés).

Jesús Cea

FRACTALES (I)

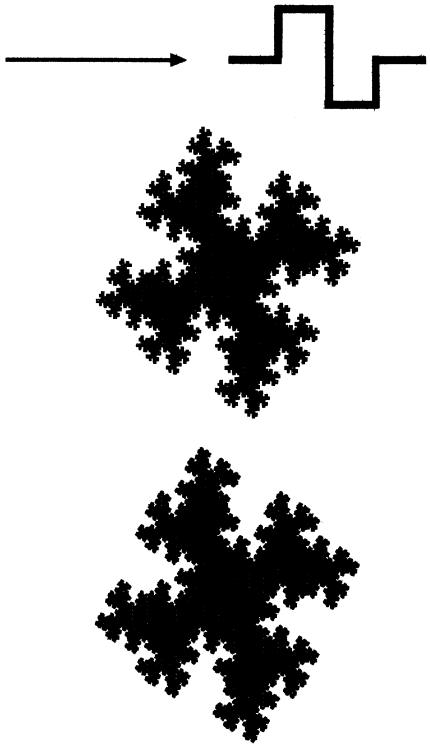
INTRODUCCION

Cojamos un mapa con línea de costa de una escala X y un compás. Si medimos la longitud de la costa y la contrastamos con la de un mapa de la misma zona, pero de una escala menor, comprobaremos espantados que la línea de costa "parece" medir más en el mapa de menor escala. Para comprender este curioso efecto (que, por otra parte, es indudablemente cierto) recurramos a una pequeña simulación mental:

Imaginémonos una pequeña zona de costa vista desde el espacio. Nuestra "regla" de medida tendrá una longitud de kilómetros ó cientos de metros según el poder de resolución de las cámaras que utilicemos. Es obvio que con esta definición se nos escapan cantidad de detalles. Si observamos desde un avión la misma línea de costa, la regla tendrá una longitud de apenas unos metros, apareciendo pedruscos, caletas y otros objetos que pasaban desapercibidos con la escala anterior. Si ahora vamos a pie nuestro compás de medida advierte todos los recovecos y salientes mayores que unos pocos milímetros. Si extrapolamos estos resultados es fácil deducir que, a mayor ampliación, más detalles irán apareciendo. De esta forma, concluimos que la línea de costa tiene una longitud infinita, aunque está "construida" sobre un segmento de longitud finita (el principio y final de la línea de costa).

En realidad esto no es así. Esta sucesión de ampliaciones sucesivas se detiene a nivel de AMSTRONGS, pero es suficiente para que hablar de la longitud de una porción de costa sea absurdo; todo depende de la escala. Al ampliar el doble, la longitud no se hace doble, sino un poco más. El ejemplo de la costa puede generalizarse también para superficies. Pongamos por caso la superficie de una montaña. Al ampliar una ladera, vemos unas pequeñas piedras. Podemos ampliar éstas hasta parecer auténticas montañas (similares a la primera). Estas nuevas montañas tienen laderas que pueden ser ampliadas... El

Figura 1:
Fractal en el que cada lado del cuadrado de partida se divide en cuatro partes, de la forma que se ve en la gráfica:



proceso se repite hasta el infinito (en teoría, claro). También existen volúmenes fractales, hipervolúmenes, etc.

Estos fractales recursivos (se les llama recursivos por razones obvias. Los fractales "de fórmula", infinitamente más bellos y complejos serán estudiados en esta sección dentro de dos números) son mucho más comunes de lo que pudiera parecer. Sólo en el cuerpo humano existen infinidad de ellos. Pensemos en la membrana neuronal, vasos capilares, el árbol bronquial, la estructura cuaternaria proteínica, etc.

Los objetos fractaloides (en realidad los fractales son entes matemáticos puesto que en el mundo real desaparece esta propiedad por encima de una ampliación determinada por el tamaño finito de átomos y moléculas) se nos presentan por igual tanto en formas vivas (árboles) como en sujetos inanimados (un río con sus afluentes) constantemente. Incluso en el caos del movimiento Browniano, en la cresta de una ola al romper en la playa, en el ritmo cardíaco (paradójicamente, la regularidad es símbolo de enfermedad - léase principio de infarto - mientras el caos aparente significa que nuestro corazón marcha como es debido) ó en la frecuencia de goteo de un grifo existe un orden fractal dado a través de los fractales "de fórmula" y sus "ATRACTORS", que serán descritos aquí dentro de dos números.

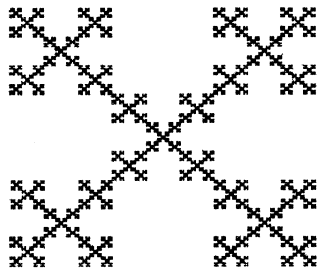


Figura 2: Anticopo de nieve construido con los cuadrados "HACIA DENTRO".

Para finalizar esta larga pero imprescindible introducción (o por lo menos, eso creo), recomendar a los audaces (y temerarios) programadores que se atrevan a ensayar los diseños estudiados a continuación (o los suyos propios) que se consigan un lenguaje con capacidad recursiva (Pascal, C, Logo o similares). De esta forma se ahorrarán infinidad de problemas. Los que no dispongan de estas facilidades pueden optar por escribir los programas en BASIC (iqué vulgaridad!) a costa de sencillez y, sobre todo, de velocidad. Deben llevar una lista de punteros (una matriz) señalando las partes sucesivas del programa cuya misión sea mantener constantemente señalizadores a los puntos a trazar, simulando recursividad. Es una tarea difícil, ardua y proclive a los errores, donde sólo un programador experimentado y, sobre todo, hábil saldrá airoso (leer artículo anterior). Los dibujos de este artículo fueron trazados con un lenguaje recursivo en un Atari ST, pero tuve que escribir las versiones originales en BASIC. Suerte que tenía experiencia en estas lides.

FRACTALES RECURSIVOS

Ni que decir tiene que para estudiar este inmenso mundo los ordenadores son el medio ideal. En contra de lo que pudiera parecer, los fractales recursivos (al contrario que los fractales "de fórmula") no son un producto de la era informática, sino que tienen una historia de casi un siglo. Probablemente todo comenzó con las famosísimas "curvas de Peano" (Giuseppe Peano, lógico y matemático italiano, nació en Spinetta en 1858 y murió en Turín en 1932. Fue profesor de cálculo infinitesimal en la Universidad de Turín. Estudió el cálculo geométrico y creó una lengua internacional llamada "latín sin inflexiones". Sus

obras: "Cálculo diferencial" e "Interlingua"). Dichas curvas fractaloides de construcción recursiva obligaron a los matemáticos de la época a revisar el concepto de curva, entre otras razones, porque las curvas de Peano no tienen derivada en NINGUNO de sus puntos.

Sin embargo,la primera curva fractal definida como tal fue "Snowflake" (copo de nieve), construida en 1904 por el matemático sueco Helge Von Koch. El sistema de construcción es bien sencillo: En el tercio central de cada lado de un triángulo equilátero se coloca otro triángulo equilátero de lado 1/3 del primero, obteniendo una estrella de 6 puntas. En cada uno de los 12 lados resultantes se repite el proceso cuantas veces quera-

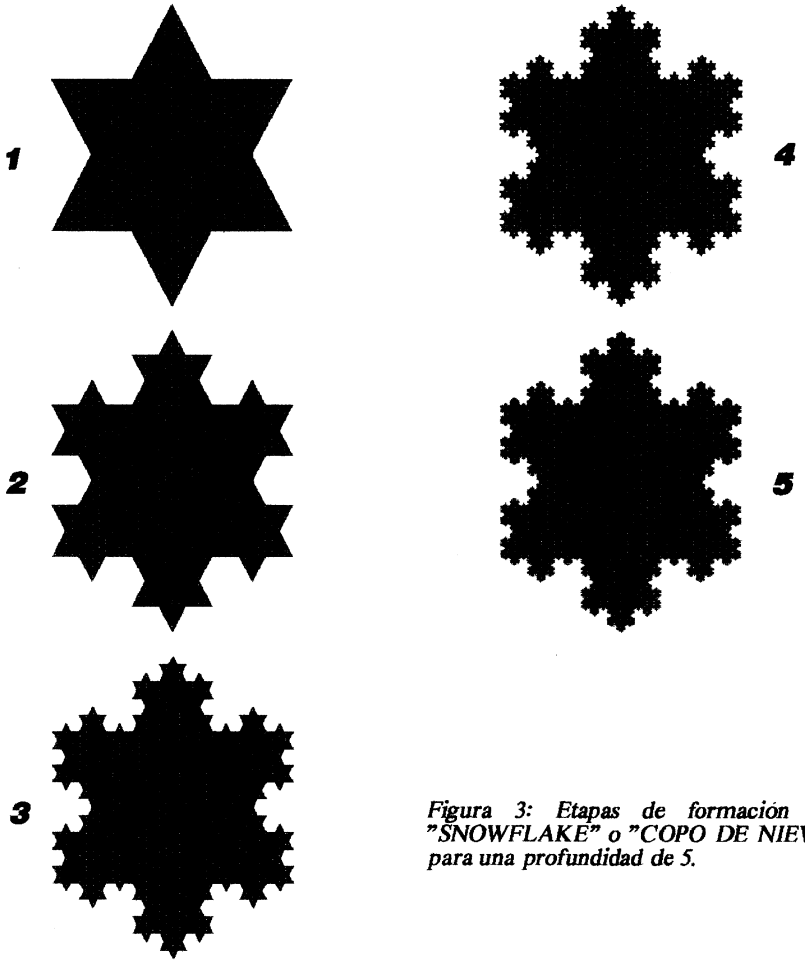


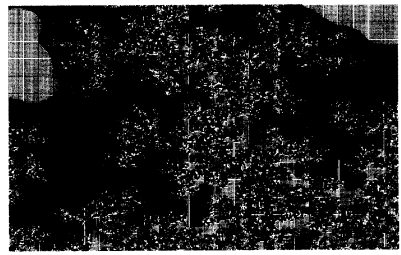
Figura 3: Etapas de formación del "SNOWFLAKE" o "COPO DE NIEVE" para una profundidad de 5.

mos (profundidad). El resultado final del proceso depende en gran medida de la profundidad recursiva que escojamos. Sin embargo, a partir de una profundidad 3, el objeto obtenido es pasmosamente parecido a un copo de nieve (de ahí su nombre). Vemos claramente que cada nivel de iteración incrementa la longitud del perímetro en 4/3. Intuitivamente, la longitud del segmento entre dos puntos de la curva (en el límite) es infinita. Así mismo, observando las gráficas que acompañan este artículo (que corresponden a las etapas de formación del "copo de nieve"), concluimos que cualquier zona de una curva fractal recursiva es semejante a la curva en sí (esta es una de las particularidades de los fractales recursivos).

¿Cuál es el área encerrada por la curva? En el primer nivel se añaden 3 triángulos de área 1/9 del original. En el segundo, 12 de área 1/81 del original... El área que añade cada nivel al anterior es (4/a)"3/4. Para calcular el área en el límite, recurrimos a la fórmula para sumas de términos de una progresión geométrica de |r|<1 (s= a1/1-r). El resultado es exactamente 8/5 del triángulo original.

¿Qué será el anticopo de nieve? Igual que el copo pero construyendo los triángulos hacia dentro. Es una curva infinita que limita un área finita cuyo valor es 2/5 del triángulo original (cada nivel resta (4/a)"3/4). Sustituyendo los triángulos por cuadrados se obtiene una curva fractal que limita un área doble del cuadrado original. Si construimos los cua-drados hacia dentro, obtenemos una curva infinita que no limita área alguna: Dado que añadimos la misma área que sacamos, el área global de la figura 1 será la misma que el cuadrado del principio.

Lo bueno de los fractales recursivos es que se diseñan "a medida" (los fractales de fórmula son impredecibles).



Cualquiera puede hacer el suyo propio. Sólo se requiere imaginación y paciencia.

DIMENSIONES FRACTALES

Desde el descubrimiento de las curvas fractales las Matemáticas vieron claro que, además de las dimensiones típicas (0, 1, 2, 3,...) existían unas dimensiones intermedias fraccionarias (de ahí el nombre de fractales). En las curvas que hemos estudiado aquí podemos decir sin lugar a dudas que son "algo más" que una recta (dimensión 1), pero no llegan a una superficie (dimensión 2). Así, habitan en una tierra de nadie, en una dimensión intermedia fraccionaria, en un mundo fractal. Existen otros universos fractales similares a caballo entre la dimensión 2 y 3, 3 y 4, etc.

El primer matemático en cuantificar estas dimensiones fraccionarias fue el topólogo alemán Félix Hausdorff en 1919, quien definiría lo que habría de llamarse (con toda la razón) dimensión de Hausdorff:

$$\log_{1/D}(L)$$

donde D es el tamaño de la regla (1/2, 1/3, etc.) y L es el número de veces que se repite dicha regla. Para comprender esto veamos algunos ejemplos:

Supongamos una recta de longitud X. Si la dividimos a la mitad tenemos ahora dos trozos iguales.

Si dividimos su lado a la mitad, tendremos 4 cuadrados:

$$\log_{1/2}(4) = \log_2(4) = 2$$

Si tenemos un cubo de volumen X y dividimos su arista a la mitad, obtenemos 8 cubos que suman el volumen original:

$$\log_{1/2}(8) = \log_2(8) = 3$$

En la figura 3,al dividir el lado en 3 partes surge un triángulo que incrementa la longitud de la línea hasta 4. Su dimensión será:

$$\log_3(4)$$

(Como pocas calculadoras admiten una base logarítmica que no sea "e" ó 10, operemos:

$\log_3(4) = \log_3 x * \log_x 4 = (\log_x x/\log_x 3) * \log_x 4 = \log_x 4/\log_x 3$, donde x es una base cualquiera.)

Así, la dimensión del copo de nieve será log 4/log 3= 1.2618595, intermedia entre 1 y 2. La dimensión correspondiente a la misma figura con cuadrados será log 5/ log 3= 1.4649735, y la dimen-sión de la figura 1 es log 8/log 4= 1.5.

La dimensión de Hausdorff es una medida de complejidad fractal e indica el nivel de "fractalidad" de un diseño. La figura 1 es más compleja que 2 y ésta más que 3.

Para dibujar una figura de dimensión comprendida entre 1 y 2, precisamos un medio de 2 dimensiones (la pantalla). Si queremos dibujar un fractal de dimensión entre 2 y , necesitamos un medio tridimensional. Como supongo que habrá pocos lectores con un sistema (equipo) holográfico a mano, tendremos que conformarnos con simular las tres dimensiones en nuestro vulgar monitor bidimensional. El algoritmo a seguir para los cálculos es idéntico al sistema en dos dimensiones (excepto que puntos y vectores tienen 3 componentes en vez de 2). Para la impresión en pantalla podemos optar por imprimir todas las líneas, vistas y ocultas, (quedando un gráfico liadísimo) o imprimir sólo las aristas visibles. Si el lector escoge el segundo sistema, puede diseñar su propio algoritmo de ocultación de caras... o estar atento a la sección de algoritmos de los próximos números.

Por ejemplo, sustituyendo triángulos por tetraedros, la nueva dimensión de Hausdorff es:

$$\log(12)/\log(3) = 2.2618595$$

Si ponemos cubos en vez de cuadrados,

$$\log(15)/\log(3) = 2.4649735$$

La figura 1 en el espacio tendría una dimensión fractal de

$$\log(32)/\log(4) = 2.5$$

Como podéis ver, son idénticas a las de sus homólogos bidimensionales con la excepción de que aumenta la dimensión en 1 (lo mismo que al pasar de cuadrado a cubo, de cubo a hipercubo,...). Podemos rizar el rizo definiendo fractales de 4, 5, 6,... Para calcularlo no hay ningún problema, pero para imprimirlo...

Una curiosidad: Existe una curva fractal diseñada por Gosper que ¡¡Llena toda la superficie que la contiene!! Su construcción es un tanto sofisticada y baste decir que cada nivel de iteración sustituye cada una de las 7 líneas primarias en 49, cada una de longitud $7^{1/2}$ del original:

$$\log(49)/\log(7) = 2$$

(Tiene dimensión 2 porque llena totalmente el área. Si alguien quiere conocer el algoritmo, que me escriba y se lo enviaré con mucho gusto).

Paradójicamente, el borde de la superficie es ¡¡un fractal de dimensión log (3)/ log $(7^{1/2})$= 1.1291501!! (cousas veredes, amigo Sancho).

Como cabría imaginarse, esta curva puede generalizarse a 3 dimensiones, pasando a ocupar todo el volumen en el que está contenido. Su frontera, que también es fractal, tiene una dimensión de (sí, sí, lo habéis adivinado) 2.1291501.

HISTORIA DE LA INFORMATICA

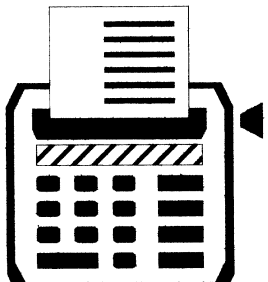
Nacho Agulló

PARTE SEGUNDA

En la parte primera hemos visto cómo fueron los matemáticos los iniciadores de la Informática, buscando máquinas capaces de realizar cálculos mecánicos. Pero no fue hasta que a las Matemáticas se les unió la Automática que nació de verdad la Informática. Con el siglo XVII concluye la etapa de los matemáticos como impulsores de esta ciencia a nivel de máquinas; con la entrada del siglo XVIII serán los ingenieros quienes tomen el relevo.

LA AUTOMATICA EN EL SIGLO XVIII

A comienzos del siglo la Automática contaba ya con muchos ejemplos claros, siendo los más significativos los relojes: Con diverso grado de sofisticación, podían encontrarse relojes con música o escenas animadas. Descartes había hecho, en el siglo anterior, una recopilación de trabajos sobre autómatas en Centro-Europa y el mundo árabe. Esta va a convertirse en referencia para toda la Automática posterior; así mismo, los trabajos de mecánica del propio Descartes, que había construido también autómatas, van a tener influencia. Pero va a ser en el sector textil donde la Automática arranque de verdad.



LA AUTOMATICA EN EL SECTOR TEXTIL

Ya desde aproximadamente el siglo XIII los trabajos de hilar y tejer se habían comenzado a automatizar a base de energía hidráulica o animal. Esta automatización prosiguió lentamente.

En 1666 aparece en Poitiers (Francia) una máquina, diseñada por un obrero anónimo, que, movida por cintas, hacía el trabajo de diez hombres. En 1725 el francés Basilio Bouchon construyó un telar que podía tejer diseños de seda siguiendo las instrucciones codificadas en papel perforado. Este método, antecesor de las "fichas perforadas" tan conocidas, se derivaba de la mecánica de los antiguos relojes medievales. En 1728 otro francés, M. Falcón, perfecciona el trabajo de su compatriota.

Pero estos avances no se restringieron sólo a Francia. Los telares automáticos se extendieron, al tiempo que se mejoraban y perfeccionaban, y así encontramos los ejemplos de las máquinas de James Hargreaves (cuya máquina, "Juana la hilandera", llegaba a montar 80 husos para el trabajo semiautomático), Samuel Crompton y Richard Arkwright, entre 1767 y 1780.

EL DESARROLLO DE LA AUTOMATICA EN EL SIGLO XVIII

A comienzos del siglo XVIII Hans Knaus elaboró el llamado "péndulo de Tournoi"; giraba en torno a un eje con cuatro partes diferenciadas, desarrollando una escena medieval mientras anunciaba las horas, con caballeros y heraldos animados al son de la música de carrillón automática. Más tarde, su hijo elaboraría un reloj péndulo que tocaba el himno imperial mientras desa-

rrollaba toda la coronación de la emperatriz María Teresa de Austria con figuras animadas.

Al finales de siglo, desarrollan su actividad Vauçansons y Pierre Jacquet-Droz, notables constructores de autómatas; desde un flautista mecánico que realmente tocaba la flauta emitiendo aire por su boca y moviendo los dedos sobre los agujeros de la flauta, hasta un zorro que contaba con las vísceras elementales en su interior, y que realmente "digería", disolviendo en su estómago los alimentos, expulsando por su ano los restos.

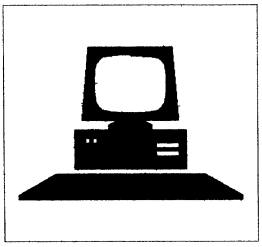
Estos trabajos, influidos por los de Descartes, tuvieron a su vez influencia en su época, la de la revolución francesa. Aunque muchos de aquellos trabajos se perdieron, posteriormente Robert Houdin se ocupó de restaurar algunos.

EL TELAR DE JACQUARD

Joseph-Marie Jacquard, ingeniero francés, añadió a comienzos del siglo XIX una significativa mejora a los telares mecánicos. Jacquard toma el siste-ma de papel perforado de Bouchon y Falcón, y le añade mecanismos de control inspirados en los modelos de Vauçansons. El telar "leía" las tarjetas perforadas, introducidas por una ranura, y realizaba el tipo de tejido indicado por éstas. El telar fue declarado en 1806 "de utilidad pública" en Francia.

A partir de su utilización por Jacquard, el sistema de tarjetas perforadas se extendió rápidamente a muchos dispositivos mecánicos, de los que es un claro ejemplo la pianola.

El telar de Jacquard automatizaba la información, de modo que, en cierto modo, se puede considerar como el comienzo de la auténtica Informática.



DATOS BIOGRAFICOS

JOSEPH JACQUARD, (Lyon, Francia, 1752- 1834)

Inventor del telar textil que lleva su nombre. Fue nombrado agregado del Conservatorio Nacional de Artes y Oficios de París por Napoleón, cargo desde el que pudo dedicarse a su labor de investigación. Su invento, revolucionario en la industria textil, atrajo las iras de los gremios de Lyon. Los obreros veían en él una amenaza para sus empleos.

BIBLIOGRAFIA

Enciclopedia MONITOR (Ed. SALVAT)

Enciclopedia MI COMPUTER (Ed. DELTA)

Gran Enciclopedia de la INFORMATICA (Ed. NUEVA LENTE)

Historia de la INFORMATICA, de Amparo Gil Orihuel e Ignacio Rieiro Martín (Ed. ALHAMBRA)

PROGRAMA SECUENCIAL:

Es el programa "tipo", que corre en la mayoría de ordenadores. Sus instrucciones llegan al procesador una a una y se ejecutan a medida que se reciben.

PROGRAMA PARALELO:

Este tipo de programa está diseñado para que cada secuencia(s) de instrucciones vaya a ejecutarse a un procesador determinado, de manera que cada procesador realiza una parte del trabajo (para más información, releer el artículo del número anterior).

RISC:

Abreviatura de "Reduced Instruction Set Computer. Se refiere a los ordenadores cuyo procesador dispone de muy pocas instrucciones base de código máquina, propiciando así la sencillez de su programación.

TERMINOS COMUNES

ACARREO:

Valor que se añade a un dígito binario en una suma. Este valor procede de la operación anterior. En la resta, se toma un bit de más (acarreo) si no es posible realizar la operación, por ser mayor el divisor.

ACUMULADOR:

Registro usado comúnmente por la Unidad Aritmético-Lógica en operaciones generales.

ALFANUMERICO:

Calificativo otorgado a un dato que es numérico o alfabético, o a un carácter, código, etc.

ANALOGICOS, ORDENADORES:

Ordenadores que manejan señales eléctricas en su funcionamiento, y que su programación está cableada en los propios circuitos.

ANCHO DE BANDA:

(Aplicado a la Informática) Diferencia entre la frecuencia máxima y mínima que puede manejar un circuito.

AND:

Función lógica aplicable a variables lógicas. El resultado es 1 lógico cuando las variables de entrada son todas 1.

ANIDAMIENTO:

Inserción de rutinas, bucles, etc. unos dentro de otros formando una estructura jerárquica.

ARCHIVO:

Conjunto de datos manipulables en conjunto de forma homogénea.