DATA BUS

Número 5 Segunda Epoca Agosto 1.993

EDITA

ASOCIACION JUVENIL PARA LA INFORMATICA VIGUESA

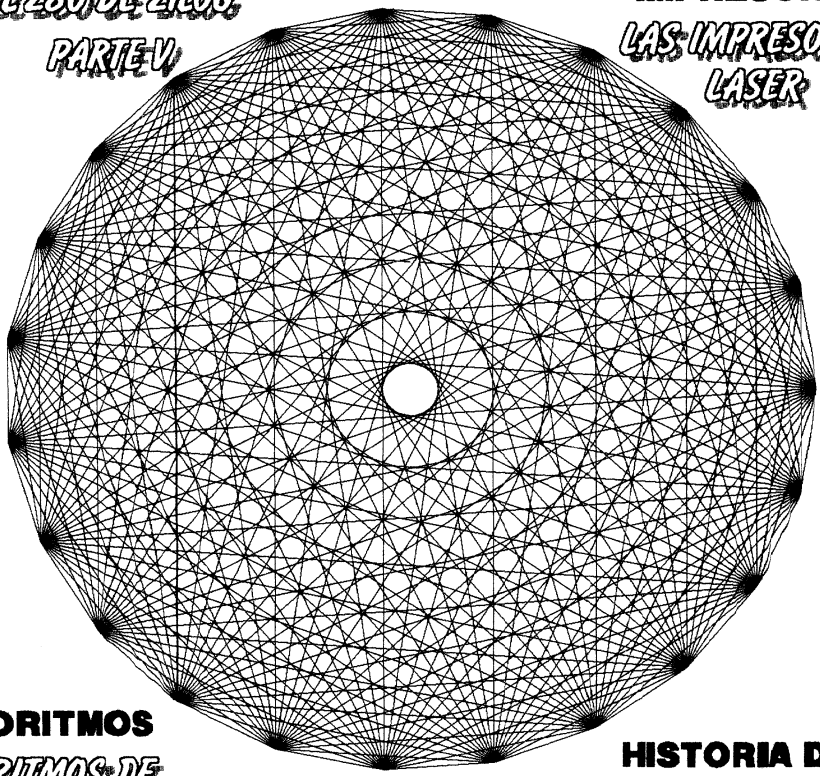
AJV

MICROPROCESADORES

EL Z80 DE ZILOG PARTE V

IMPRESORAS

LAS IMPRESORAS LASER



ALGORITMOS

ALGORITMOS DE CLASIFICACION ALFABETICA

HISTORIA DE LA INFORMATICA

PARTE VIII

DATA BUS

Número 5
Segunda época
Agosto de 1.993

DIRECTOR
Jesús Cea Avión

REDACTORES
Jesús Cea Avión
Nacho Agulló Sousa
Jose Manuel Suárez Pousa

COLABORADORES
Jose Luis Estévez

DISEÑO Y MAQUETACION
Jesús Cea Avión

ASOCIACION JUVENIL PARA LA INFORMATICA VIGUESA
AJV

C/Caldas de Reis, 12-6º Izq.
36209 VIGO
Telf: (986) 23 18 35

Depósito legal
VG-87-90

¡¡¡ ATENCION !!!

Si deseas colaborar con nosotros en *DATA BUS* aportando ideas, críticas, artículos, etc., no tienes más que hacernos llegar tu trabajo. Para ello puedes enviárnoslo a la dirección de la asociación, o bien ponerte en contacto con la persona que te suministró este ejemplar.

¡¡¡ CONTAMOS CON VOSOTROS !!!

INDICE

EDITORIAL

Nuevamente ***DATA BUS*** vuelve a salir con un mes de retraso. Las causas son las de siempre: demoras a la hora de entregar los artículos, tareas escolares, etc. De cualquier forma por fin tenéis aquí el número de verano de esta espléndida revista.

Lamentamos la mala calidad de reproducción del pasado número. Ya en el editorial se adelantó la posible "presencia" de molestas lineas verticales debido al funcionamiento defectuoso de un inyector de la impresora. Paradojicamente eso fue, precisamente, lo que menos se notó. Lo peor de todo fue, sin duda, la baja calidad de las fotocopias ya que nuestro lugar habitual tenía la fotocopiadora estropeada y nos vimos obligados a recurrir a "alternativas" que, como se vió, no daban la talla. Esperemos que no se vuelva a repetir.

Seguimos publicando listados (esta vez sólo en 'C') para acompañar los artículos. Todavía nadie ha dicho nada sobre el particular. ¿Os interesan? ¿En qué lenguajes os gustarían que aparecieran? Si nadie responde, que nadie se queje si los suprimimos...

La serie sobre el Z80 ha tocado a su fin. Inicialmente estaba previsto seguir con el 6502, 68000, 8088, etc., pero la única persona con la que contamos con conocimientos sobre el tema ya tiene sus propias secciones. Si hay alguien de fuera que se anima... En cualquier caso sería interesante que propusierais ideas para reemplazar esta serie, ya que Nacho Agulló quiere seguir colaborando en la revista. Esperamos impacientes vuestras propuestas.

Hemos estado dando muchas vueltas a la perioricidad de ***DATA BUS*** y hemos decidido que, por el momento, sea cuatrimestral. Las razones para ello son bastante evidentes: Falta de artículos y falta de tiempo. A partir de ahora y hasta nuevo aviso ***DATA BUS*** saldrá en Enero, Mayo y Septiembre. Si conseguimos un par de colaboradores fijos quizás podamos pasar a ser trimestrales. Como casi siempre, depende plenamente de vosotros.

Por último, si alguien hecha de menos la serie de "Curiosidades Informáticas" que comprenda que con el monstruo de la página cuatro...

Echad un ojo a la página 24, por favor. Bueno, por nuestra parte nada más. Nos veremos en Enero.

La Redacción

Algoritmos de

clasificación (I)

Jesús Cea

En este número empezamos una nueva serie en la sección de "Algoritmos", sobre algoritmos de clasificación alfabética. Me parece un tema bastante interesante y no demasiado técnico. La serie de Algoritmos Matemáticos queda aplazada indefinidamente ya que, al parecer, se trataba de algo demasiado "esotérico". Los lectores interesados en continuar con ella que hablen conmigo...

Es indudable que una de las areas en las que los ordenadores han tenido mayor éxito (de hecho, una de las razones de que los ordenadores estén donde están) es como gestores de bases de datos. Un problema aflora aquí, sin embargo. Cualquiera que haya trabajado con cantidades importantes de información reconoce la absoluta necesidad de mantener clasificados, de alguna manera lógica, los datos para reducir los tiempos de búsqueda y para simplificar el procesamiento de las bases de datos relacionales (bases de datos independientes pero interrelacionadas). ¿Alguien es capaz siquiera de imaginar para qué podría servir un listín telefónico en el que los usuarios estuvieran completamente desordenados? A buen seguro de que una guía semejante carecería de toda utilidad práctica (realmente no debiera decir tanto, ya que siempre hay gente que encuentra aplicaciones a cualquier cosa...).

El problema de la clasificación de datos, en particular el ordenamiento alfabético, no es tarea trivial. Los algoritmos más obvios presentan unos tiempos más que dilatados, que crecen muy rapidamente con el número de elementos a procesar. En esta serie de artículos veremos los algoritmos de ordenación más conocidos. Por razones de espacio la serie constará de dos artículos. En el primero (el que está leyendo ahora) estudiaremos los algoritmos más simples (e ineficientes), mientras que en el próximo número se publicarán algoritmos más eficaces, aunque quizás no sean tan sencillos de comprender. De todos modos haré todo lo posible por simplificar las descripciones, aislando al lector de los engorrosos detalles de implementación, e intentando que la lectura de la serie redunde en una comprensión real del modo de funcionamiento de los diferentes algoritmos. Habré logado mi objetivo si consigo que el lector entienda lo suficientemente bien los métodos como para ser capaz de programarlos por sí mismo. Como suele decirse, lo importante no es aprender, sino comprender.

Es de destacar que la idoneidad de cada algoritmo depende, en gran medida, de las características del entorno en el que se va a trabajar. No es lo mismo clasificar los datos en memoria que tener que hacerlo en un fichero, ni es lo mismo ordenar cien nombres

que un millon. En esta serie me limitaré a exponer los algoritmos y a compararlos en función del número de operaciones que necesiten. No se tendrá en cuenta que el tiempo de acceso a los datos sea variable según su posición, ni ninguna otra característica ajena al proceso en sí. Esto no es muy realista ya que, incluso aunque los datos se encuentren en *RAM*, es más rápido acceder a elementos consecutivos que andar saltando aleatoriamente de aquí para allá; de cualquier forma los tiempos ofrecidos en este artículo son puramente orientativos y dependientes de la implementación concreta. Rutinas en ensamblador, depuradas y que tengan en cuenta la arquitectura concreta de cada máquina darían unos tiempos completamente diferentes, pero las proporciones existentes entre los diferentes algoritmos deberían ser bastante semejantes.

Empezaremos por los métodos sencillos e instructivos e iremos, poco a poco, entrando en algoritmos mucho más eficientes pero, indudablemente, más sofisticados. En este artículo estudiaremos los algoritmos de clasificación *"pseudocuadráticos"* (aquellos en los que el tiempo de clasificación es, aproximadamente, proporcional al cuadrado del número de datos), en los que clasificar 10.000 valores supone 100 veces más tiempo que ordenar 1.000, aproximadamente. Por tanto, estos algoritmos sólo son utilizables cuando el número de elementos a clasificar sea pequeño, o bien cuando los datos verifiquen ciertas propiedades. Veremos los métodos de *selección, inserción y burbuja.*

En el proximo número, sin embargo, veremos algoritmos *"cuasilineales"* en los que el tiempo de clasificación es, aproximadamente, proporcional al número de datos a procesar. Ordenar 10.000 elementos conlleva, por consiguiente, unas 10 veces más tiempo que clasificar 1.000. Veremos algoritmos *quicksort* (clasificación rápida), *mergesort* (clasificación por mezcla) y *bucket* (clasificación por compartimentos ó bolsillos). También estudiaremos allí el *shellsort*, aunque por sus especiales características no se sabe muy bien donde meterlo. En fin, empecemos por el principio...

SELECCION

```
void selecc(int *datos)
{
int cont1,cont2;
int valor,pos;

  for(cont1=1;cont1<MAXDATOS;cont1++) {
    valor=32767;
    for(cont2=cont1;!(cont2>MAXDATOS);cont2++) {
      if(datos[cont2]<valor) {
        valor=datos[cont2];
        pos=cont2;
      }
    }
    datos[pos]=datos[cont1];
    datos[cont1]=valor;
  }
}
```

SELECCION

Sea un conjunto de *'n'* elementos desordenados. El algoritmo de *selección*, método sencillo y bastante intuitivo, es el siguiente:

En la primera ronda buscamos el elemento más pequeño (de los n) y lo intercambiamos con el primero. En este momento tenemos el dato más pequeño al principio de la lista. Seguidamente buscamos el elemento menor entre todos los que quedan (entre los datos segundo y enésimo) y lo intercambiamos por el segundo elemento. En la tercera ronda buscamos el dato más pequeño entre el tercero y el enésimo y lo intercambiamos por el

```
void inserc(int *datos)
{
int cont1,cont2,cont3,valor;

  for(cont1=1;(cont1>MAXDATOS);cont1++) {
    valor=datos[cont1];
    for(cont2=1;cont2<cont1;cont2++) {
      if(datos[cont2]>valor) break;
    }
    for(cont3=cont1;cont2<cont3;cont3--) {
      datos[cont3]=datos[cont3-1];
    }
    datos[cont2]=valor;
  }
}
```

INSERCION

tercer elemento, etc.

Continuando el proceso, en cada etapa se va clasificando un elemento, así que necesitamos n etapas para tener clasificada toda la lista. En realidad bastaría con n-1 etapas, ya que en la etapa n-ésima sólo tenemos un elemento a intercambiar consigo mismo y, por lo tanto, es superflua. Así mismo, se observa que en cada etapa se tienen que revisar menos elementos (n+1-i, para la etapa i-ésima). Es inmediato deducir, a partir de lo dicho, las fórmulas que nos dan el número de comparaciones y de intercambios: Se realizan $(n^2+n-2)/2$ comparaciones y n-1 intercambios.

Cuando el número de elementos supera cierto umbral (dependiente de las características de los datos y de la implementación del algoritmo), el factor que limita la velocidad es la comparación, por lo el tiempo requerido en el proceso es lo que se llama *"supercuadrático"*. A saber, el tiempo crece "más rápido" que n^2 pero menos que n^3. Ello es debido al factor n-2 de la ecuación. Si el número de elementos (n) es elevado, el factor dominante es n^2 y se puede considerar que el tiempo crece proporcionalmente con el cuadrado del número de elementos a clasificar. Así mismo, este algoritmo presenta una característica única: *el tiempo empleado en el proceso es determinista.*

Efectivamente, en los algoritmos que estudiaremos posteriormente se puede encontrar una cota superior para el tiempo de cálculo, pero resulta imposible precisar el tiempo exacto que consumirá la ordenación de n elementos, ya que depende del orden concreto inicial. En otras palabras, se puede decir que el tiempo de clasificación será *como máximo* de x segundos pero, en general, el tiempo empleado será menor. Con el algoritmo de selección, en cambio, se puede saber *exactamente* cuanto tiempo se va a tardar. Ello es debido a que la velocidad depende exclusivamente (prescindiendo de detalles de implementación) del número de elementos a ordenar y no de sus posiciones iniciales. Como corolario, consume exactamente el mismo tiempo clasificar una lista completamente desordenada que ordenar una lista ya clasificada... Por lo tanto, este algoritmo va muy bien cuando los elementos están desordenados, pero cuando los datos estan "casi" ordenados podemos encontrar sistemas más eficientes.

INSERCION

De este algoritmo podemos encontrar numerosas variantes, según sea o no necesaria la utilización de dos matrices (origen y destino), o si se utilizarán matrices o listas enlazadas. Los detalles concretos son irrelevantes para comprender el funcionamiento del algoritmo, así que cada cual decida según las necesidades concretas de la aplicación.

Como su nombre indica, la idea que hay detrás de este algoritmo

consiste en insertar los elementos en los lugares que les corresponde. Como la mejor forma de entenderlo es con un ejemplo, allá vamos. Estudiemos el caso de dos matrices, una origen y otra destino:

Al principio la matriz destino está completamente vacía. Tomamos el primer elemento de la matriz origen y lo transladamos, sin más, a la matriz destino. Seguidamente cogemos el segundo elemento de la matriz origen y lo *insertamos* en el lugar que le corresponda en la matriz destino (antes o después del único elemento que hay, en este caso). Luego cogemos el tercer elemento de origen y lo insertamos en destino al principio, al final, o en medio de los dos elementos que hay por el momento. Continuando el proceso de leer un dato e *insertarlo* en el lugar que le corresponde, al final tenemos los datos ordenados en la matriz destino. Si nos fijamos bien se observa que la matriz destino está *permanentemente* ordenada. Umm, el coco empieza a trabajar...

A priori este método parece peor que el de selección, ya que hay que "correr" algunos datos para hacer sitio en el lugar adecuado al elemento en curso (los datos se insertan). Considerando el problema más detenidamente, se observa:

a) Si en vez de trabajar con matrices se utilizan listas enlazadas, la inserción de un nuevo dato puede realizarse muy eficientemente.

b) La matriz (o la lista) destino *siempre* está ordenada, a lo largo de todo el proceso.

c) Complicando minimamente el software es posible realizar la clasificación sobre la matriz original, y sin pérdidas de velocidad. Así es como he programado el ejemplo.

La propiedad *b* nos da una pista para hallar una de las aplicaciones más idonea para este algoritmo. Veamoslo con un ejemplo: Tenemos un programa -una base de datos, por ejemplo- que va solicitando nombres que luego se listan por orden alfabético. Para efectuar ésto podemos eplicar, al menos, dos filosofías. En una de ellas, los nuevos nombres son, sencillamente, añadidos al final de la lista (matriz) de nombres viejos. Cuando finaliza la entrada de datos, simplemente se llama a una rutina encargada de clasificar la matriz. La otra filosofía que hallamos consiste en mantener la matriz *permanentemente* ordenada. Para ello, *cada vez* que se introduce un nuevo nombre se procede a insertarlo directamente en su lugar correcto.

Las implicaciones prácticas son muy claras. Con el primer sistema la entrada de datos es muy rápida, pero una vez finalizada ésta debemos esperar un tiempo "x", al regresar al menú, por ejemplo, mientras se ordena la información. Con el segundo método hay un retardo tras la introducción de cada dato (medio segundo, por ejemplo), pero se regresa inmediatamente al menú. Ya que en este caso el tiempo de clasificación se reparte en varios instantes (tras la introducción de cada elemento nuevo) y no se

```
void burbuja1(int *datos)
{
int cont,buff;

  loop:
  for(cont=1;cont<MAXDATOS;cont++) {
    if(datos[cont]>datos[cont+1]) {
      buff=datos[cont];
      datos[cont]=datos[cont+1];
      datos[cont+1]=buff;
      goto loop;
    }
  }
}
```

BURBUJA 1

concentra todo el retraso al final, al operador probablemente le resultará más agradable, siempre y cuando no exista un retardo apreciable. No estamos ante una situación inusual ya que trabajando, incluso, con miles de elementos el tiempo consumido en la inserción de uno nuevo se cuenta en décimas de segundo.

En el supuesto de que el tiempo de inserción fuera apreciable (por estar los datos en disco, por ejemplo) se puede recurrir a lo siguiente: si disponemos de un sistema multitarea podemos *"lanzar"* la rutina de ordenación en *"background"*. De esta forma tenemos todo el tiempo que tarde el usuario en rellenar una nueva ficha para procesar la anterior. Si no tenemos multitarea, siempre quedan las interrupciones...

BURBUJA

El método de la burbuja (bubblesort) es, probablemente, el algoritmo de clasificación más popular (no el más utilizado, sino el que conoce todo el mundo) del mundo de la informática, y uno de los peores. Los métodos vistos hasta ahora son muy *deterministas* en el sentido de que los elementos se mantienen permanentemente ordenados o, al menos, se ve cierta lógica en todo el trasiego de información de aquí para allá. Con los algoritmos de la burbuja (es que hay varios), en cambio, los datos que queremos clasificar van evolucionando de una forma aparentemente caótica hasta que, milagrosamente, encajan en el lugar que les corresponde con un pequeño "click". Para estudiar esta familia de algoritmos comenzaremos por el más simple:

```
#define FALSE 0
#define TRUE !FALSE

void burbuja2(int *datos)
{
int cont,buff,ok;

  do {
    ok=TRUE;
    for(cont=1;cont<MAXDATOS;cont++) {
      if(datos[cont]>datos[cont+1]) {
        buff=datos[cont];
        datos[cont]=datos[cont+1];
        datos[cont+1]=buff;
        ok=FALSE;
      }
    }
  }
}
```

BURBUJA 2

Empezando por el principio, vamos comparando cada par de elementos consecutivos. En caso de aparecer en el orden correcto (el elemento i es menor o igual que el elemento i+1) avanzamos una posición y comparamos los siguientes elementos (en este caso se comparían los elementos i+1 e i+2) y repetimos el proceso. En cuanto encontremos una pareja en orden inverso (el elemento j mayor que el elemento j+1), los intercambiamos y empezamos todo el proceso *desde el principio*. El algoritmo se detiene cuando llegamos al final de la lista, ya que entonces sabemos que está ordenada (si hubiéramos encontrado un elemento descolocado se habría vuelto a empezar desde el principio).

Resulta obvio que este algoritmo, aunque simple, es extremadamente lento, ya que cada elemento puede moverse varias veces antes de llegar a su lugar correcto, y que cada vez que reordenamos un par de elementos volvemos a comparar desde el principio. En el algoritmo de selección que vimos al principio, los datos son movidos directamente a su posición definitiva. En contrapartida, los métodos de burbuja funcionan relativamente bien cuando los elementos de partida están bastante cerca de su posición correc-

ta. Es decir, cuando la matriz está "casi" ordenada.

Es muy posible que el algoritmo comentado en los párrafos anteriores no os suene a nada, y mucho menos al método de la burbuja tradicional que todos conocéis. Lo que ocurre es que, generalmente, se utiliza una variante, con la cual se llega al archifamoso método de la burbuja *"clasico"*. La idea es bastante simple. Si en vez de empezar por el principio cada vez que reordenamos un par de elementos permitimos al programa continuar hasta el final de la lista, aunque señalando de alguna forma (una variable booleana, por ejemplo) que es necesario un "repaso", seguro que lo que resulta os suena bastante familiar. Efectivamente, estamos ante el clásico –y casi mítico– algoritmo de la burbuja, formulado tal como lo conoce el personal... Tras cada pasada se comprueba la variable booleana; si no hubo ninguna reordenación la lista ya está clasificada y se regresa. En caso de que hubiera alguna reordenación (en general, habrá varias), se vuelve a empezar desde el principio.

Estudiando más a fondo lo que ocurre con los elementos a clasificar, se observa que, a pesar del aparente movimiento caótico de la mayoría de los datos, hay uno que se dirige directamente a su lugar, y que no se mueve de allí por muchas pasadas que se realicen. Nos estamos refiriendo al elemento *mayor*. Efectivamente, por el propio algoritmo seguido, tras la primera pasada el dato localizado al final de la lista estará en su lugar correcto, siendo el mayor elemento de todos. Tras la segunda pasada el penúltimo elemento habrá alcanzado su lugar definitivo y el último no se habrá movido de su sitio, por razones obvias (sólo hay intercambio de elementos si el elemento i es

```
#define FALSE 0
#define TRUE !FALSE

void burbuja3(int *datos)
{
int cont,cont2,buff,ok;

  cont2=MAXDATOS;
  do {
    ok=TRUE;
    for(cont=1;cont<cont2;cont++) {
      if(datos[cont]>datos[cont+1]) {
        buff=datos[cont];
        datos[cont]=datos[cont+1];
        datos[cont+1]=buff;
        ok=FALSE;
      }
    }
    cont2--;
  } while(!ok);
}
```

BURBUJA 3

mayor que el i+1). En las siguientes pasadas se irán colocando el resto de los elementos, siempre empezando por el final.

Es obvio, por tanto, que ya que en cada pasada hay *al menos* un elemento que ya ha alcanzado su posición definitiva y que no se va a mover más, se precisan *un máximo* de n pasadas (realmente n–1, por las mismas razones dadas en el algoritmo de selección). Esto contrasta con el algoritmo de burbuja anterior, en el cual el número de pasadas puede llegar a ser enormemente alto. Es posible que dadas las posiciones originales de los elementos no sea necesario efectuar las n–1 pasadas que se han deducido de lo dicho, pero del párrafo anterior podemos extraer una pequeña modificación que disminuye el tiempo empleado en clasificar los *'n'* elementos *a la mitad* –más o menos– comparado con el que emplea el algoritmo de la burbuja clásico. Estamos ante el *algoritmo de la burbuja modificado*.

Efectivamente, si en la pasada i-

ésima (de las n-1 que se necesitan *como máximo*) revisamos sólo hasta el elemento n+1-i, el número de comparaciones se reduce a la mitad. En la primera pasada estudiamos 'n' elementos (es decir, todos), en la segunda sólo los n-1 primeros (todos salvo el último), etc. Los elementos que "pasamos" de analizar son aquellos que, segun hemos visto ya, han alcanzado su posición definitiva y no se moverán más. Pero ojo, el hecho de que un elemento esté en su lugar correcto no implica necesariamente que no se vaya a mover de ahí... Esto sólo se cumple con los elementos que van quedando al final, ya que los que le siguen son *todos* mayores. Esta paradoja (descolocar elementos ya correctamente situados) es una de las razones por las que la familia de métodos de la burbuja es tan lenta y, simultaneamente, la razón de que el algoritmo *shellsort*, estando basado en la burbuja, sea bastante más eficiente. Veremos esto en detalle en el próximo artículo de la serie: *"Algoritmos de clasificación avanzados"*.

```
#define FALSE 0
#define TRUE !FALSE

void burbuja4(int *datos)
{
int cont,cont2,cont3,buff,ok;

  cont2=MAXDATOS;
  do {
    ok=TRUE;
    for(cont=1;cont<cont2;cont++) {
      if(datos[cont]>datos[cont+1]) {
        buff=datos[cont];
        datos[cont]=datos[cont+1];
        datos[cont+1]=buff;
        cont3=cont;
        ok=FALSE;
      }
    }
    cont2=cont3;
  } while(!ok);
}
```

BURBUJA 4

Veamos, para terminar, una última variante. En el último algoritmo comentado nos vamos "olvidando" de los últimos elementos de la matriz, ya que sabemos que ya están ordenados y, por lo tanto, no es preciso revisarlos de nuevo. Las razones para ello ya han sido comentadas unos párrafos más arriba. Observemos con calma el siguiente hecho: supongamos que el último intercambio que hemos realizado ha sido por la mitad de la matriz. Ello implica que los datos situados tras ese "intercambio" no han necesitado ninguna reordenación. Por lo tanto, concluimos que *YA* están ordenados y, consecuentemente, pueden ser obviados en la siguiente pasada. Así pues perfectamente factible que en cada podamos descartar más de un elemento en el análisis. La consecuencia más evidente de todo ésto es, logicamente, el aumento de velocidad.

Personalmente no llego a comprender cómo es posible que los algoritmos de la burbuja sean tan conocidos y populares cuando el algoritmo de selección, por ejemplo, es más rápido y más sencillo de comprender y programar. Supongo que se trata de uno de esos insondables misterios de la existencia, junto a quién mató a Laura Palmer, a la receta del alioli o al porqué los PC's son tan populares...

Sobre los listados

Supongo y espero que a estas alturas ya os habréis fijado en los seis listados (uno para cada algoritmo o variante descrito) que acompañan este artículo. El objetivo de su inclusión no es que los utilicéis directamente en vuestros programas sino el brindaros la oportunidad de que veáis y estudiéis la forma de implementar los sistemas

comentados. Los "expertos" en 'C' se habrán echado las manos a la cabeza al no haber observado ninguna construcción típica del lenguaje, ni haber utilizado "peculiaridades" exclusivas, como los punteros (si, ya sé que el Pascal -¡Puag!- también tiene punteros, pero no son lo mismo ya que, en el colmo de las restricciones, ni siquiera pueden señalar a los elementos de un array...). Las razones de no haber utilizado las "idiosincrasias" propias del 'C' han sido puramente prácticas. Por un lado, no tiene sentido *"criptar"* los listados (aunque con ello los programas fueran más eficientes) cuando lo que se pretende es, precisamente, que los lectores los comprendan. Por otra parte, no todo el mundo conoce el 'C' a la perfección y los programas se han codificado teniendo en mente la adaptación a otros lenguajes quizás más "vulgares" pero, sin duda, mucho más difundidos (lease Basic y Pascal).

Hablando de lenguajes, los ejemplos se han escrito para *"Pure C"*, sin duda el mejor compilador de 'C' para Atari. Es compatible (a nivel de código fuente) con el *"Turbo C"*, así que los usuarios de PC no tendrán que modificar ni una coma. En realidad se trata de *A.N.S.I. C*, por lo que cualquier compilador moderno debería "tragar" los "sources" sin ningún problema. Algún día tengo que escribir un artículo sobre las optimizaciones que es capaz de hacer el *"Pure C"*...

Para aquellos que no dominen el 'C' paso a comentar, muy someramente, algunas peculiaridades de los listados. El *"void"* que aparece al principio sirve para indicar al compilador que la función no retorna ningún valor (se trata de un procedimiento). Los compiladores viejos podrían no aceptarlo, en cuyo caso bastaría conque se eliminara. MAXDATOS contiene el número de elementos a clasificar. En buena lógica debería ser un parámetro de la función, pero lo hice así por razones que no vienen al caso. El valor 32767 que aparece en el primer listado sirve para "normalizar" la rutina. Existe una forma mejor de inicializar el bucle, pero ésta me pareció más clara. El objetivo del bucle consiste en buscar el menor elemento entre los dados. Para ello compara cada dato

TABLA DE TIEMPOS DE EJECUCION

Los tiempos de ejecución vienen dados en segundos y se han calculado para un conjunto de datos completamente desordenado. Los programas que se han utilizado para el cronometraje son los mismos que acompañan el artículo. Los tiempos son meramente orientativos y dependerán, obviamente, del compilador y la máquina. Se observa que los tiempos crecen, aproximadamente, de forma proporcional al cuadrado del número de elementos, por lo que estos algoritmos de denominan *"pseudocuadráticos"*.

Nº Datos	Selección	Inserción	Burbuja 1	Burbuja 2	Burbuja 3	Burbuja 4
100	--	--	1.3	--	--	--
200	--	--	8.6	--	--	--
400	--	--	76.6	--	--	--
500	--	--	146.9	--	--	--
1000	4.4	4.5	--	14.3	8.2	8.3
2000	17.8	18	--	56.9	32.8	33.2

con el valor actual y si es menor se actualiza al valor de *"actual"*. Inicializándolo con 32767 ($2^{15}-1$) se obtiene un funcionamiento correcto.

La admiración (!) conlleva la negación de lo que le sigue. en cuanto a los *"++"* y *"--"*, son los modos de autoincremento y autodecremento, construcciones que considero muy elegantes. En cuanto a *"break"* sirve para salir del bucle en curso de ejecución, siendo idéntico al *"exit if"* del GFA-Basic. La utilización del *"goto"* en el primer listado de la burbuja ha sido por motivos, pura y simplemente, de claridad. podría haber hecho exactamente lo mismo con *"do...while"* y *"break"*, pero opino que así es mucho más claro. No comprendo la razón de que los profesores de informática nos deniegan el derecho a utilizar el *"goto"*. Soy consciente de que el uso abusivo (o el mal uso) de esta instrucción dificulta enormemente el seguimiento de un programa, pero me parece innegable que en determinadas situaciones hace al código mucho más legible. Y si no, que se lo digan a cualquiera que haya necesitado salir de cinco bucles anidados. La cuestión es saber utilizarlo, como ocurre con todo.

En cuanto a los *"#define"*, sirven para declarar constantes. Observar la forma de definir *"TRUE"*. Impresionante, ¿eh? Bueno, creo que entre lo que he dicho, lo que sepáis sobre cualquier otro lenguaje, el sentido común y la indentación podréis desgranar el funcionamiento de las rutinas. Si hay alguna duda, consultad con vuestro profesor de informática o conmigo mismo...

Los tiempos

No hay mucho que decir. Observad que el primer método de la burbuja es, francamente, de risa. debo hacer notar, también, que los tiempos datos son para clasificar una tabla completamente desordenada. Si la matriz estuviera "casi" ordenada los tiempos podrían ser muy diferentes. Ya dije antes que el tiempo para el algoritmo de selección no depende de los datos. Aplicando inserción a un array ordenado no hay que hacer ninguna inserción, mientras que si estuviera clasificado exactamente al revés (orden decreciente) el número de inserciones a efectuar sería máximo. El último algoritmo de la burbuja parece peor que el penúltimo, pero con matrices semiordenadas va mejor, ya que requiere menos comparaciones. Lo mejor es probar los algoritmos en su entorno normal de trabajo.

Algoritmos matemáticos

Tal como dije al principio, el futuro de esta serie es "indefinido". Los artículos publicados son "digeribles" (es la opinión de un lector), pero el que estaba planeado publicar en este número ha sido juzgado, según todas las encuestas, como incomprensible. La razón es simple; he llegado a un punto en donde es necesario meterse muy a fondo en matemáticas y estadística, con lo que los artículos se hacen muy *"cuesta arriba"* incluso para los lectores interesados. Siguiendo varios consejos he procedido a reescribir dos veces el artículo, intentando hacerlo más ameno, pero hay cosas que no se pueden suavizar...

Por si a alguien le interesa, se trata del *"algoritmo CO.R.DI.C."*, capaz de calcular senos, cosenos, tangentes, logaritmos y sus inversas mediante un método iterativo sin ningún producto, sino tan sólo sumas, restas y desplazamientos aritméticos. Y todo ésto con 4 lineas de código y con una precisión arbitrariamente grande... Los lectores interesados en él...

¡¡ Noticias Frescas !!


Jose Manuel Suárez

Un número más, he aquí un pequeño resumen de las noticias más importantes o curiosas acaecidas en el mundillo de la informática en los últimos meses. Esperamos vuestras noticias en la dirección o el teléfono indicados en la página 2.

LOS INVENTORES DEL MORPHING, GALARDONADOS

La prestigiosa *Academy of Motion Picture Arts and Science* americana, equivalente holliwoodiense de Bellas Artes, ha galardonado a la firma *Industrial Light and Magic*, de George Lucas, por la invención del *"morphing"*. Esta técnica permite crear imágenes de síntesis en las cuales los sólidos evolucionan, se estiran y cambian de aspecto de una manera fluida y "natural".

ILM introdujo esta técnica por primera vez en 1988, en la película *"Willow"*, durante las escenas de transformación de seres humanos en animales. Desde entonces la técnica ha sido utilizada con éxito en películas como *"Abyss"* o *"Terminator 2"*, destacables por sus excelentes efectos especiales (pero no os descubro nada nuevo, a menos que vengáis de Marte...). Desde aquello el *morphing* se expandió en la industria de la imagen de síntesis y forma parte de los efectos clásicos, reemplazando los costosos efectos de disolución y de maquetas empleadas hasta entonces (a gran pesar de los hábiles artistas del trucaje, virtuosos de los efectos imagen por imagen, en paro por culpa de simples ordenadores).

Durante una ceremonia organizada el pasado mes de Marzo la *Academy* entregó tres placas como galardón. La primera a *Douglas Smythe*, programador principal de las herramientas de morphing usadas por ILM. La segunda a *Tom Brigham*, inventor del concepto y autor de los primeros trabajos sobre el tema. Y la tercera, por supuesto, se entregó a todo el equipo de ILM por haber sido los primeros en incorporar y usar esta técnica.

VIDEO DIGITAL EN DESARROLLO

Los gigantes de la electrónica Philips, Matsushita y Sony se acaban de asociar para desarrollar la futura norma de videograbadores numéricos. Los trabajos, de por si delicados, se vuelven más difíciles por la preocupación con la compatibilidad de estos futuros aparatos con las normas actuales y venideras de televisión directa por satelite y de TV de alta definición.

Una cosa ya es segura: la electónica de estos videos no será nada simple. Para empezar, las imagenes digitales no estarán grabadas directamente, sino comprimidas. En efecto, una imagen numérica de alta definición exige varios megaoctetos, lo que haría crecer desmesuradamente la longitud de banda magnética necesaria

para la grabación. Una posibilidad consiste en comprimir cada imagen, y otra grabar unicamente la diferencia entre cada fotograma y no las imagenes enteras. En esta última posibilidad se basa el estándar *MPEG (Motion Picture Expert Group)*: fotogramas completos, conteniendo toda la imagen, están dispuestos a intervalos regulares sobre la banda, separados por fotogramas que contienen sólo las diferencias entre imagenes sucesivas. De esta forma se logra que el *"ruido"* no se propage indefinidamente.

Pero los problemas empiezan cuando se quiere obtener una imagen aceptable durante el bobinado o rebobinado rápido. En este caso hay que interpretar sólo los fotogramas completos (y hay sólo uno completo por cada 10, lo que significa que la velocidad mínima hacia adelante será de 10 veces, lo cual es demasiado), o decodificar todos los fotogramas más rapidamente que en lectura normal. Entonces habría que aumentar la potencia y la banda pasante de los circuitos de descompresión de estos videos. Y sabiendo que dichos circuitos son, en realidad, microprocesadores especializados muy rápidos, es difícil pensar en aumentar su velocidad de proceso a bajo coste.

Sin embargo, los japoneses han anunciado la resolución del problema gracias a una serie de artimañas matemáticas que permiten mejorar el proceso codificación-decodificación, lo que significa que la investigación teórica (especialmente en matemáticas) tiene, a menudo, consecuencias comerciales rápidas, aunque disguste a los inconscientes que cortan los créditos en Europa destinados a ese tipo de investigaciones.

MICROSCOPIOS 3D DE RAYOS X

Investigadores del Laboratorio National Lawrence Livermore (California) han puesto a punto una tecnología que podría revolucionar un buen número de actividades, empezando por la medicina. Se trata de un microscopio a rayos X, bajo cuyo nombre bárbaro se esconde la propiedad, de sobra conocida por los fans de Superman, de ver detalles minúsculos en el interior de la materia sólida.

Para ello, se fotografían con rayos X hasta 1000 "trozos" del objeto estudiado. Seguidamente se pone bajo forma numérica las radiografías de estos trozos y se funden en un potente ordenador que reconstruye un modelo en tres dimensiones del objeto. Se puede entonces estudiar el objeto bajo cualquier ángulo, girándolo en la pantalla. También se puede ampliar y "pasearse" dentro de él. Esta técnica permite aislar detalles de hasta un micrómetro. Esto es poco en comparación con un microscopio electrónico, pero es mucho mejor que los tradicionales *escáners* de los hospitales. Además, las técnicas clásicas de microscopio obligan a extraer una muestra del sujeto o aplicarle un tratamiento especial, lo que no ocurre en este caso.

Los primeras areas que se beneficiarán de este nuevo tipo de instrumentación serán la investigación médica y el estudio de nuevos materiales, en particular el de algunas cerámicas. Esta técnica ha sido desarrollada a través del programa de Iniciativa de Defensa Estrategica (o *"guerra de las galaxias"*) que acaba de cancelar la administración Clinton.

UN JUEGO DE REALIDAD VIRTUAL

La compañía *Ixion* (Seattle, EE.UU.) ha anunciado el juego *VR Slingshot*, basado en el enfrentamiento de dos jugadores en un "mundo" virtual.

Cada jugador dispone de un mando de juego analógico y de un par de gafas 3D que son, en realidad, pantallas de cristal líquido (una por cada ojo). El relieve está producido por una ligera variación entre las imágenes que llegan a cada ojo,

según el principio clásico de la estereovisión. El conjunto se conecta a un PC potente o a un Amiga 3000/4000.

David Hon, presidente de *Ixion*, describe *VR Slingshot* como un deporte, en lugar de un juego. Se trata, efectivamente, de gastar las reservas de energía del adversario. Cada jugador pilota una nave equipada de cañones y de un escudo. Disparar sobre el adversario consume energía, pero acelerar y recibir impactos también. Las naves evolucionan bajo una cúpula que contiene generadores de gravedad que interfieren con el vuelo de las naves. El truco consiste, por tanto, en usar las esferas para aumentar tu velocidad sin gastar demasiada energía. El perdedor será el que haya gastado primero toda su energía.

El modo de un jugador permite luchar contra el ordenador pero, sin lugar a dudas, será el modo de dos jugadores el que tendrá más interés, con dos máquinas conectadas entre sí con cable serie null-modem, o bien por modem (1200 baudios o más) y conexión telefónica. Los diseñadores insisten sobre el lado estratégico del juego, que superaría con mucho al aspecto puramente de combate 3D del mismo. Una cosa es segura: los programas de este estilo serán algo habitual en los próximos años, al mismo tiempo que bajan los precios de las máquinas potentes necesarias para visualizar la 3D en tiempo real.

NOTICIAS BREVES

1. Con los problemas de producción aparentemente solucionados (el sobrecalentamiento ocasionaba la destrucción del chip), el Penthium (alias P5, alias Proteus, alias i586), lo último de Intel, ya está en el mercado. De momento, sin embargo, su precio es bastante prohibitivo. Según algunas informaciones que han llegado a nuestra redacción, la compra de 1.000 unidades (obviamente sólo para fabricantes) supondría un desembolso de unas 118.000 Pts. por chip al cambio de hace unos meses, así que ahora que la peseta está tan devaluada... ¡¡Uff!!, preferimos no especular sobre el precio de una placa madre en el mercado nacional.

De momento no hay noticias sobre las frecuencias de trabajo ni las prestaciones, aunque casi todo el mundo coincide en señalar que se queda por detrás del también recien aparecido MC68060 de Motorola. Estamos deseando ver cualquiera de estas bestias en acción...

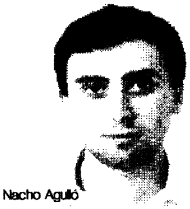
2. El último ordenador de Atari, el Falcon030, ya puede encontrarse en España. Unas cuantas empresas repartidas por toda la geografía española se encargan de importarlo del extranjero, aunque con manuales, ROM y teclado en castellano. Esta fabulosa máquina, que incluye un DSP de serie y un potente 68030, se dió a conocer hace cosa de un año, aunque no ha sido hasta hace seis meses que se distribuyó en grandes cantidades en Francia y Alemania. Los rumores dicen que el Falcon040, con un 68040, saldrá antes de navidades. Desde aquí le deseamos mucho éxito. ¡¡¡Suerte!!!.

3. En cuanto a Commodore, hay rumores de un nuevo Amiga. El Amiga 5000 llevaría, según nuestros confidentes, un 68060 como procesador principal y dos 68040 "reducidos" (sin MMU, FPU, etc.) para apoyar en el manejo de graficos y sonido. El precio, así mismo, sería sorprendentemente bajo para las características y el equipamiento que proporciona. ¿Una serpiente de verano? Hay quien dice que la propia Commodore ha confirmado la noticia...

4. No podemos menos que alabar la política de Apple frente a los estudiantes. Efectivamente los precios de educación de sus productos son más que atrayentes. Desde luego una rebaja de más del 30% frente al precio de venta al público normal no es para desdeñar...

El microprocesador

Z80 de Zilog (V)



Nacho Agulló

Si en las partes anteriores hemos visto el Z80 tanto en su aspecto físico como en su aspecto lógico, y también su funcionamiento, solamente nos resta por ver su juego de instrucciones. A él está dedicado esta última parte de la serie..

TIPOS DE DATOS QUE PUEDE MANEJAR EL Z80

Antes de poder examinar el juego de instrucciones, es preciso distinguir los tipos de datos con los que nos vamos a encontrar.

1. Constante de un sólo byte(n): Número de 8 bits cuyo valor se toma de forma *"natural"*, es decir, de 0 a 255, o 00 a FF hex.

2. Constante de dos bytes(nn): Número de 16 bits cuyo valor se toma de forma *"natural"*, de 0 a 65535, o de 0000 a FFFF hex. Puede ser un dato, o una dirección.

3. Constante de desplazamiento de un byte(d): Número de 8 bits cuyo valor se toma restando 128 a la forma *"natural"*. Así, el valor oscila entre -128 y +127. A este formato numérico se le llama *"complemento a dos"*.

MODOS DE DIRECCIONAMIENTO

El Z80 tiene varias formas de direccionar los datos y las instrucciones, las cuales son:

1. Direccionamiento absoluto: La posición de memoria es expresada por un número de 16 bits (nn).

2. Direccionamiento relativo: La posición de memoria es expresada por un número de 8 bits complementado a dos (d), que se ha de sumar al Contador de Programa. Este tipo de direccionamiento es utilizado solamente en los saltos relativos.

3. Direccionamiento indirecto: La posición de memoria es expresada por un registro de 16 bits (dd), escrito entre paréntesis.

4. Direccionamiento indexado: La posición de memoria es expresada por un registro índice (IX ó IY) complementado con una constante de desplazamiento de un byte (d) y escrito entre paréntesis. La dirección obtenida es entonces (IX+d) ó (IY+d).

CLAVE PARA LA INTERPRETACION DEL JUEGO DE INSTRUCCIONES

b: Dirección de un bit dentro de un byte. Varía de 0 a 7.

p: Dirección de la memoria, múltiplo de 8. Varía desde 0 a 56.

n: Constante de un byte.

d: Constante de desplazamiento.

nn: Constante de dos bytes.

r,r': Registro de 8 bits o posición

de memoria direccionada indirectamente. Puede ser A, B, C, D, E, H, L ó (HL).

m: Registro de 8 bits o posición de memoria direccionada indirecta o indexadamente. Puede ser A, B, C, D, E, H, L, (HL), (IX+d), (IY+d).

s: Constante, registro de 8 bits o posición de memoria direccionada indirecta o indexadamente. Puede ser n, A, B, C, D, E, H, L, (HL), (IX+d), (IY+d).

tt: Registro de 16 bits. Puede ser HL, IX ó IY.

dd: Registro de 16 bits. Puede ser BC, DE, HL ó SP.

pp: Registro de 16 bits. Puede ser BC, DE, IX ó SP.

rr: Registro de 16 bits. Puede ser BC, DE, IY ó SP.

ss: Registro de 16 bits. Puede ser BC, DE, HL, SP, IX ó IY.

cc: Condición. Puede ser relativa a cuatro indicadores (ver figura).

INDICADOR	ACTIVADO	DESACTIVADO
Cero	*Z*(cero)	*NZ*(no cero)
Acarreo	*C*(acarreo)	*NC*(no acarreo)
Paridad o Desbordamiento	*PO*(paridad impar o desbordamiento	*PE*(paridad par o no hay desbordamiento
Signo	*P*(positivo)	*M*(negativo)

EL JUEGO DE INSTRUCCIONES DEL Z80

1. Instrucciones de control del Z80.

NOP: Instrucción sin operación.

HALT: Introduce un ciclo de espera que se repite indefinidamente hasta la recepción de una señal de interrupción.

DI: Desactiva las interrupciones.

EI: Activa las interrupciones.

IM 0: Establece el modo 0 de interrupción.

IM 1: Establece el modo 1 de interrupción.

IM 2: Establece el modo 2 de interrupción.

2. Instrucciones de carga de registros y direcciones.

LD A, (BC): Carga el acumulador con el contenido de la posición indicada por el registro BC.

LD A, (DE): Carga el acumulador con el contenido de la posición indicada por el registro DE.

LD A, I: Carga el acumulador con el contenido del registro I.

LD A, R: Carga el acumulador con el contenido del registro R.

LD A, (nn): Carga el acumulador con el contenido de la posición de memoria nn.

LD (BC), A: Carga la posición de memoria indicada por el registro BC con el valor contenido en el acumulador.

LD (DE), A: Carga la posición de memoria indicada por el registro DE con el valor contenido por el acumulador.

LD I, A: Carga el registro I con el contenido del acumulador.

LD R, A: Carga el registro R con el contenido del acumulador.

LD (nn), A: Carga la posición de memoria direccionada por nn con el contenido del acumulador.

LD m, n: Carga el operando m con el valor n.

LD m, r: Carga el operando m con el registro r.

LD r, m: Carga el registro r con el operando m.

LD ss, nn: Carga el registro ss con el valor nn,

LD ss, (nn): Carga el registro ss con el valor de 16 bits almacenado en la dirección nn.

LD (nn), ss: Carga la dirección nn con el valor de 16 bits contenido en el registro ss.

LD SP, tt: Carga el registro SP con el contenido del registro tt.

PUSH ss: Introduce el registro ss en la pila.

POP ss: Recupera el último valor introducido en la pila y se carga en el registro ss.

3. Instrucciones de Intercambio, Búsqueda y Transferencia.

EXX: Intercambia los contenidos de los Registros de Uso General (B, C, D, E, H y L) con sus correspondientes registros alternativos.

EX AF, AF': Intercambia los contenidos de los registros AF y AF'.

EX DE, HL: Intercambia los contenidos de los registros DE y HL.

EX (SP), tt: Intercambia el valor de 16 bits almacenado en la dirección indicada por SP con el valor contenido en el registro tt.

CPD: Compara la posición direccionada por (HL) con el acumulador y decrementa HL y BC.

CPDR: Repite la operación anterior hasta que el indicador de cero se active, o BC=0.

CPI: Compara la posición direccionada por (HL) con el acumulador, incrementa HL y decrementa BC.

CPIR: Repite la operación anterior hasta que el indicador de cero se active, o BC=0.

LDD: Copia el contenido de la posición (HL) en la posición (DE), decrementando HL, DE y BC.

LDDR: Repite la operación anterior hasta que BC=0.

LDI: Copia el contenido de la posición (HL) en la posición (DE), incrementando HL y DE, y decrementando BC.

LDDR: Repite la operación anterior hasta que BC=0.

4. Instrucciones aritméticas y lógicas.

CCF: Complementa el indicador de acarreo,

SCF: Pone a uno el indicador de acarreo.

CPL: Complementa el acumulador.

NEG: Complementa a dos el acumulador.

DAA: Ajusta el acumulador decimalmente.

ADD A, s: Suma el operando s al acumulador.

ADD HL, dd: Suma el registro dd a HL.

ADD IX, pp: Suma el registro pp a IX.

ADD IY, rr: Suma el registro rr a IY.

ADC A, s: Suma con acarreo el operando s al acumulador.

ADC HL, dd: Suma con acarreo el registro dd a HL.

SUB s: Resta el operando s al acumulador.

SBC A, s: Resta el operando s al acumulador.

SBC HL, dd: Resta con acarreo el registro dd a HL.

AND s: Realiza la operación lógica AND con el operando s y el acumulador, almacenando en éste el resultado.

OR s: Realiza la operación lógica OR con el operando s y el acumulador, almacenando en éste el resultado.

XOR s: Realiza la operación lógica XOR con el operando s y el acumulador, almacenando en éste el resultado.

CP s: Compara el operando s con el acumulador.

INC m: Incrementa el operando m.

INC ss: Incrementa el registro ss.

DEC m: Decrementa el operando m.

DEC ss: Decrementa el registro ss.

5. Instrucciones de rotación y desplazamiento.

RLC m: Rotación hacia la izquierda del operando m. El bit 7 se copia en el indicador de acarreo y en el bit 0. Esta operación equivale a una multiplicación por 2.

RRC m: Rotación hacia la derecha del operando m. El bit 0 se copia en el indicador de acarreo y en el bit 7. Esta operación equivale a una división por 2.

RL m: Rotación hacia la izquierda del operando m con el indicador de acarreo tomado como un bit más; es decir, el bit 7 se copia en el indicador de acarreo y el valor de éste se copia en el bit 0 a su vez. Esta operación equivale a una multiplicación por 2 a la que se suma el valor del indicador de acarreo.

RR m: Rotación hacia la derecha del operando m con el indicador de acarreo tomado como un bit más; es decir, el bit 0 se copia en el indicador de acarreo y el valor de éste se copia en el bit 7 a su vez. Esta operación equivale a una división por 2 a la que se suma el valor del indicador de acarreo multiplicado por 128.

SLA m: Desplazamiento aritmético hacia la izquierda del operando m, copiándose el bit 7 en el indicador de acarreo. Equivale a multiplicar éste por dos.

SRA m: Desplazamiento aritmético hacia la derecha del operando m, permaneciendo el bit 7 invariable. Equivale a dividir éste por dos, sumándole después 128 si el bit 7 estaba activado.

SRL m: Desplazamiento lógico hacia la derecha del operando m, copiándose el bit 0 en el indicador de acarreo. Equivale a dividir éste por dos.

RLD: Rotación a la izquierda de los dos nybbles (mitades de byte –4 bits–) de la dirección (HL), pasando el nybble izquierdo al inferior del registro A, y éste al nybble inferior de la dirección (HL).

RRD: Rotación a la derecha de los dos nybbles (mitades de byte) de la dirección (HL), pasando el nybble derecho al inferior del registro A, y éste al nybble superior de la dirección (HL).

6. Instrucciones de comprobación y manipulación de bits.

BIT b, m: Realiza la comprobación del bit b del operando s.

RES b, m: Pone a cero el bit b del operando n.

SET b, m: Pone a uno el bit b del operando m.

7. Instrucciones de salto.

JP (tt): Salta a la dirección indicada por el registro tt.

JP cc, nn: Salta a la dirección nn dependiendo de si se cumple la condición cc.

JP nn: Salta a la dirección nn.

JR d: Realiza un salto relativo direccionado por d.

JR Z, d: Realiza un salto relativo direccionado por d si el indicador de cero está activado.

JR NZ, d: Realiza un salto relativo direccionado por d si el indicador de

cero está desactivado.

JR C, d: Realiza un salto relativo direccionado por d si el indicador de acarreo está activado.

JR NC, d: Realiza un salto relativo direccionado por d si el indicador de acarreo está desactivado.

DJNZ d: Decrementa el registro B, efectuando un salto relativo direccionado por d si el resultado es cero.

8. Instrucciones de llamada y retorno.

CALL cc, nn: Llamada a la subrutina direccionada en nn dependiendo de si se cumple la condición cc.

CALL nn: Llamada a la subrutina direccionada en nn.

RST p: Llamada a una dirección de la ROM.

RET: Retorno desde subrutina.

RETI: Retorno desde Interrupción Enmascarable.

RETN: Retorno desde Interrupción No Enmascarable.

RET cc: Retorno desde subrutina dependiendo de si se cumple la condición cc.

9. Instrucciones de entrada y salida.

IN A, (n): Carga el acumulador con el byte de datos proporcionado por el puerto de entrada n.

IN r, (C): Carga el registro r con el byte de datos proporcionado por el puerto de entrada direccionado por el registro C.

IND: Carga la posición direccionada por (HL) con el byte de datos proporcionado por el puerto de entrada direccionado por el registro C, decrementa HL y B.

INDR: Repite la operación anterior hasta que B=0.

INI: Carga la posición direccionada por (HL) con el byte de datos proporcionado por el puerto de entrada direccionado por el registro C, incrementa HL y decrementa B.

INIR: Repite la operación anterior hasta que B=0.

OUT (n), A: El valor del acumulador es enviado a través del puerto de salida n.

OUT (C), r: El contenido del registro r es enviado a través del puerto de salida direccionado por el registro C.

OUTD: El contenido de la posición (HL) es enviado a través del puerto de salida direccionado por el registro C, decrementándose HL y B.

OTDR: Repite la operación anterior hasta que B=0.

OUTI: El contenido de la posición (HL) es enviado a través del puerto de salida direccionado por el registro C, incrementándose HL y decrementándose B.

OTIR: Repite la operación anterior hasta que B=0.

Y con esto hemos terminado el juego de instrucciones del Z80 y la serie a él dedicada. Ya sólo queda armarse de un buen ensamblador (*Hisoft*, naturalmente), y ¡A programar!.

Bibliografía

* *Gran Enciclopedia Informática.* Ediciones Nueva Lente.

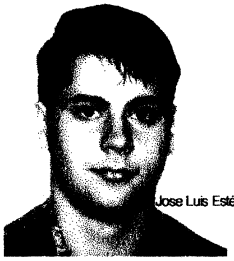
* *Enciclopedia Mi Computer.* Editorial Delta.

* *Construya su propia computadora basada en el Z80.* Steve Ciarcia. Bytes Books. Editorial McGraw Hill.

* *Understanding your Spectrum.* Dr. Ian Logan. Melbourne House Publishers.

Una pequeña introducción a las

Impresoras Láser


Jose Luis Estévez

Hoy se estrena un nuevo colaborador de DATA BUS, con un artículo -interesante y detallado- sobre el funcionamiento de las impresoras láser. ¡¡Espero tu artículo para la próxima DATA BUS, Josse!!. Poco a poco la gente se va animando (menos mal...).

En este artículo, el primero que escribo para *DATA BUS*, pretendo dar a conocer un tipo de impresora que se podría calificar como la gran desconocida de las impresoras, por que aunque la mayoría de la gente conoce la máquina, ha trabajado con ella, conoce su calidad, sus prestaciones y su PRECIO, muy poca conoce como funciona. Utilizaré para ello datos, esquemas y dibujos que pertenecen a la impresora *LASER-PRINTER 4216* de IBM, que es la que más se presta para explicar los diferentes pasos de la impresión láser o ELECTROGRAFIA. Este efecto está basado en el principio físico de atracción y repulsión de cargas eléctricas.

Antes de comenzar he de decir que para explicar el funcionamiento de impresión de las impresoras láser utilizaré palabras técnicas y algunas inglesas que, por diversas razones y para evitar equívocos, es mejor no traducir (al menos no literalmente).

Primeramente, la lógica de la impresora lee los datos que le llegan por el interface (bien paralelo Centronics, serie RS-232, twinaxial o coaxial) y crea una imagen de bit en la memoria. Esta imagen pasa al modulador del láser. Este modulador crea impulsos de luz que llevan el patrón de la imagen y la depositan en un rodillo fotosensible en el que queda una imagen electrostática de la página a imprimir. Veamos cómo.

El rodillo fotosensible es conocido por el nombre de *TAMBOR* y no es más que un rodillo de un determinado diámetro que tiene la particularidad de ir recubierto de un material que reacciona con la luz. Este *tambor* se carga electrotaticamente de una forma muy especial. La superficie del mismo se carga a una tensión MUY elevada y negativa (del orden de -6000V) y en centro o núcleo del mismo se mantiene a una tensión menos negativa (del orden de -100V). Se crea, de esta, manera una superficie homogenea de cargas negativas en la superficie del tambor.

En el tambor incide el haz de láser y provoca que las cargas negativas de la superficie donde incide sean arrastradas hacia el núcleo del tambor, quedando un *"hueco"* de carga en la superficie del mismo. Se crea de esta forma una imagen electrostática en el tambor que no es más que un fiel reflejo de la imagen

que queremos imprimir, de forma que donde hay un *"hueco"* es donde debe haber tinta. Quizás os preguntéis el porqué de la utilización de un modulador de láser y no de una lámpara de luz normal y corriente. El hecho de utilizar un láser es debido a la alta resolución con la que debemos trazar la imagen en el TAMBOR. Con una luz normal y corriente nunca podríamos definir puntos tan pequeños como con un haz de láser. De ahí la alta resolución de este sistema.

Para que ésto se produzca de la mejor manera posible hay máquinas que llevan uno o más hilos (conocidos como hilos o coronas de carga), que sirven para que tanto la superficie como el núcleo se mantengan a una tensión totalmente estable, libre de perturbaciones externas como pueden ser los campos magnéticos que crean las diferentes fuentes de alta tensión que lleva la máquina.

Las impresoras láser no utilizan para imprimir tinta como utiliza una impresora de chorro, o como la que podemos encontrar en una cinta de una impresora matricial. Utiliza una *"tinta"* especial en forma de polvo superfino llamado *TONER*. Este tóner se guarda en un depósito especial para tal fin. Se trata del *DEVELOPER* que posee, además del depósito, una rejilla o hilo de carga y unos rodillos. Veamos su funcion.

La rejilla carga el tóner a una tensión negativa, PERO esa tensión es menos negativa que a la que se carga la superficie del tambor (suele ser del orden de unos -400V), por lo que podemos decir que esta tensión es positiva con respecto a la tensión de la superficie del tambor.

Los rodillos tienen por función sacar el tóner del developer y colocarlo sobre la superficie del TAMBOR. ¿Qué ocurre entonces?

El tóner que está sobre la superficie del rodillo del developer es atraido por la ausencia de carga de los *"huecos"* que creó el láser en la superficie del tambor. El tóner que no es atraido vuelve en el rodillo al depósito del developer.

Tenemos entonces un tambor repleto de cargas muy negativas (blanco) y de cargas menos negativas de tóner (negro) que forman la imagen del texto o gráfico que queremos imprimir. Ahora sólo queda plasmarlo sobre el papel.

Una hoja es alimentada por un juego de rodillos y colocada de forma que pase entre el rodillo y un nuevo elemento llamado *CORONA DE TRANSFERENCIA* (no confundir con corona de carga). ¿Qué es este elemento? Consta de una armadura metálica en forma de caja abierta por la parte superior y en su interior va alojado un hilo de acero muy fino (y muy frágil, os lo digo por experiencia) que es el *HILO DE TRANSFERENCIA*. Este hilo va conectado a una fuente de alimentación que lo pone a una tensión positiva (del orden de unos 400V). De esta forma las partículas de tóner (que permanecían "sujetas" al tambor) son atraidas hacia el hilo de transferencia pero, al estar el papel en medio, quedan depositadas en él. Por otro lado el papel también adquiere un pequeño potencial positivo que es suficiente para que el tóner no se desprenda con el movimiento que sufre la hoja.

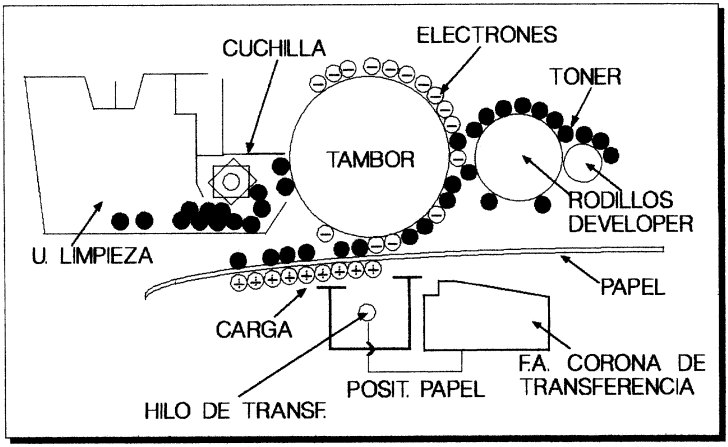
Solo quedan por hacer tres cosas: Borrar la imagen que sigue latente en el tambor, limpiar los restos de tóner del mismo y asegurar la imagen sobre el papel. Vamos por partes.

Para borrar la imagen del tambor se utiliza un haz de luz que incide a

lo largo del tambor y procede de una cadena de led's que van colocados encima del mismo. Este haz de luz *"vela"* el tambor, desplazando las cargas negativas hacia el núcleo mas positivo (-100V), situando la carga de la superficie del tambor a unos 0V.

Limpieza del tambor: Una vez que hemos limpiado el tambor electricamente tenemos que limpiarlo fisicamente (restos de tóner que no han pasado al papel). Para ello se utiliza un elemento al que se le denomina *UNIDAD DE LIMPIEZA*, que consta, principalmente, de una cuchilla que recoge el tóner sobrante y lo deposita en un recipiente de residuos. Hay impresoras que reciclan este tóner limpiándolo de impurezas (polvo de papel y suciedad) y lo introducen de nuevo en el depósito del developer. Las otras simplemente lo acumulan en el recipiente. Este recipiente lleva un sensor que le indica al operador (mediante un testigo luminoso o un código de error) que está lleno y necesita ser vaciado.

Para asegurar el tóner sobre el papel y concluir el proceso de impresión se hace pasar al papel (siempre en posición horizontal) por un elemento llamado *FUSER*, que consta de dos rodillos. Uno es de teflón (el mismo material que las sartenes antiaderentes) al que se le llama *RODILLO DE CALOR o HOT-ROLLER* y que lleva en su interior una resistencia que lo calienta hasta una temperatura lo suficientemente elevada como para fundir el tóner sin quemar el papel. El otro rodillo es de silicona y recibe el nombre de *RODILLO DE PRESION*, y que tiene unos muelles que lo ponen en contacto con el *rodillo de calor*. El papel pasa entre ambos rodillos de forma que el calor funde el tóner sobre el mismo (lo que explica el porqué las hojas salen calientes).



Dependiendo de lo largo de la página y del perímetro del tambor todo este proceso (carga, exposición, revelado, transferencia, neutralización, limpieza y fusión) se repite tres o cuatro veces hasta conformar la página entera. Un conjunto de rodillos llevan el papel a la bandeja de salida concluyendo la impresión.

Bueno, espero que os haya gustado. Espero también no haber sido demasiado técnico. Puede que siga escribiendo para *DATA BUS* y si las

fuerzas no me fallan y me llegan noticias (buenas claro) de la aceptación de este articulo entre los lectores seguiré con algún otro tema que ya estoy pensando...

Invito también a los lectores de *DATA BUS* que quieran añadir algo a este artículo (algo sobre las impresoras láser en color estaría bien) que no se lo callen. Recordad que aquí tenemos un medio perfecto para dar a conocer nuestros conocimientos sobre los aspectos más variopintos de este inmenso mundo de *"locos"* que es la informática. No os cortéis; seguro que cada uno de vosotros tiene información o sabe algo que la mayoría desconoce o que, simplemente, no conoce bien. Animo, *DATA BUS* puede ser tan grande como queramos sus lectores.

VIGO -- 12-07-93 -- JOSSE

NOTA DE LA REDACCION

No podemos menos que agradecer no ya el artículo en sí, sino el párrafo final del mismo. Efectivamente, una revista la hacen sus lectores con sus sugerencias, ideas, artículos, dibujos, etc. Una revista no es un baul cerrado al exterior, sino que necesita la "realimentación positiva" *de sus lectores. Los comentarios y críticas nos inspiran; incluso agradecemos las observaciones negativas, ya que indican que la revista se lee con espíritu crítico, y sirven para mejorar.*

No obstante los comentarios no bastan. Una revista necesita nutrirse de artículos y ahí parece estar el problema. Todo el mundo se anima a aportar ideas para nuevas secciones, nueva maquetación, temas para los artículos, etc., pero ¿quién escribe?, ¿quien arrima el hombro "de verdad"?, ¿quién se va a encargar de esa nueva sección maravillosa que se nos acaba de ocurrir? ¿El que propuso la idea? Yo bastante hago -dice- con dar apoyo moral...

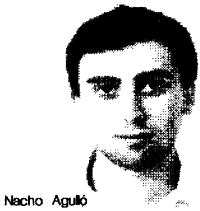
Pues no, no es suficiente. El "apoyo moral" está muy bien pero también necesitamos artículos para salir puntualmente en la fecha señalada. Ultimamente hemos conseguido reducir los retrasos a "solo" un mes pero, y aunque se trata de un logro notable, aún no estamos satisfechos. Nos gustaría poder llegar, algún día, a salir a la calle cuando está planeado. Nos gustaría no tener que retrasar la publicación de ***DATA BUS*** *a causa de ese artículo que no llega y que es imprescindible para cerrar la edición. También nos gustaría publicar noticias (en la sección de "Noticias frescas") remitidas por nuestros lectores, pero hasta el momento ésto ha sido un desierto.*

Además los artículos nos sirven para saber los temas en los que estáis interesados, el nivel de profundidad que deseáis, etc. No queremos que la revista se convierta en un monólogo sin contacto con la realidad. Queremos que sea algo vivo, y que crezca con vuestra ayuda.

Pero basta ya de lamentarse. En honor a la verdad hemos de decir que, poco a poco, los lectores vais respondiendo. En los dos últimos números hemos contado con sendos artículos de colaboradores ajenos a la asociación y de gran calidad. ¿Podremos publicar otro artículo "de fuera" en el próximo número? Eso depende enteramente de vosotros.

¿Alguien se anima? Venga, porfa....

Historia de la informática (VIII)



Nacho Agulló

Por fin, tras siete largas partes, ha llegado el momento que los seguidores fieles de esta sección estábais esperando: la aparición del primer ordenador universal completo.

La tecnología electromecánica

La tecnología electromecánica disponible al comienzo de la Segunda Guerra Mundial, a base de relés y cableados, era ya suficiente para diseñar circuitos lógicos y aritméticos. Sin embargo, faltaba lo más importante: idear cómo construir un ordenador con tales componentes. Muchos problemas cuya solución hoy conocemos de memoria debían todavía ser resueltos por primera vez. Por ello, a pesar de que la tecnología llevaba varias décadas disponible, el éxito se le resistía a aquellos que querían fabricar algo más que una calculadora.

Después de los intentos estudiados en las partes anteriores, como los de Bush, Stibitz o Atanasoff, la ingeniería de la época había llegado hasta las mismas puertas del ordenador. El cálculo automático era un logro ya asentado y más o menos dominado. Faltaba conquistar la mismísima cima: la posibilidad de programar la máquina. La victoria se palpaba cercana.

Las máquinas de Zuse

Va a ser un joven ingeniero aeronáutico alemán, Konrad Zuse, quien tome entonces el relevo. Durante su trabajo en la *Henschel Aircraft Company* tiene ocasión de comprobar cuán laboriosos eran los cálculos de tensiones a altas velocidades necesarios para diseñar las alas de los aviones. Este meticuloso trabajo se realizaba usando reglas de cálculo y las sencillas calculadoras mecánicas de la época. El problema va a captar el interés de Zuse, quien comienza a construir en su tiempo libre una máquina capaz de realizar la pesada tarea.

Trabajando con relés, Zuse descubre que es posible utilizarlos como biestables, utilizando una de las posiciones como *"1"* y la otra como *"0"*. Esto determina el que sus diseños utilicen la notación binaria, como las de Stibitz y Atanasoff. Los trabajos de Zuse obtienen un prometedor éxito inicial al terminar el *Z1*: este dispositivo mecánico es capaz de efectuar las cuatro operaciones elementales, y convertir números de decimal a binario y viceversa.

Zuse continúa su investigación buscando la forma de hacer más versátil su calculadora, adentrándose en un terreno por el que pocos le han

precedido. Algunas de las conclusiones que obtiene son muy similares a las de nuestro viejo conocido Babbage, a pesar de que Zuse desconocía sus trabajos. Pero no se detiene aquí nuestro ingeniero. Su siguiente paso es conseguir que las instrucciones de la calculadora se almacenen en forma binaria al igual que los datos. Con estas características finaliza su segunda máquina, el *Z2*

En plena Segunda Guerra Mundial

El *Z2* se termina de construir en 1941. Como no podía ser de otro modo en plena guerra, el III Reich termina reconociendo el potencial de los descubrimientos de Zuse, y se decide a financiar un nuevo modelo con vistas a emplearlo en el diseño aeronáutico con fines bélicos. Como suele suceder en estos casos, poco podía decir nuestro ingeniero al respecto.

De este modo comienza la construcción del *Z3*, salpicada de innumerables dificultades. Para empezar, su juventud -treinta años escasos- motiva el que le llamen a filas en dos ocasiones y se le envíe nada menos que al frente ruso, para después mandarlo de vuelta para que continúe su trabajo. Por otra parte, los continuos bombardeos que sufre Berlín le obligan a trasladar su taller varias veces. Además, la escasez de materiales le proporciona continuos quebraderos de cabeza que resolver, teniendo que improvisar utilizando piezas de conmutadores telefónicos o películas antiguas en vez de cintas de papel.

Pero Zuse sale airoso de estas dificultades y termina el *Z3*, que es el primer ordenador universal completo. Además de la capacidad de programación y del almacenamiento del programa en la misma memoria que los datos, posee más aspectos novedosos: Almacena 64 palabras de 22 bits, la salida se produce de forma visualizada a través de un conjunto de lámparas montadas en un tablero, y la información se introduce mediante un teclado.

La *Henschel* utiliza un *Z3* para diseñar las bombas volantes no tripuladas que serían conocidas como *V1* y *V2* Mientras, Zuse ya pensaba en el ingenio que construiría a continuación: el *Z4*. Este ordenador aventaja al *Z3* en cuanto al número de bits, que era 32 en vez de 22. Como vemos, los ordenadores de 32 bits puros no son ninguna novedad.

El final de la guerra

Los últimos meses de guerra fueron los peores para la ciudad de Berlín, que tuvo que soportar bombardeos cada vez más duros mientras los aliados se acercaban. El *Z4* va a ser trasladado a Gotinga. El *Z3* fué destruido con su taller durante el bombardeo de Berlín. Finalmente, tras la rendición alemana, el *Z4* será instalado en Basilea (Suiza), donde continuará funcionando hasta 1954.

La postguerra

La destruida Alemania de postguerra no estaba en condiciones de financiar ningún proyecto de investigación, y menos en informática. Sin embargo, Zuse continúo trabajando, aunque sólo en el plano teórico. Desarrolló un sofisticado lenguaje para ordenadores, denominado *Plankakül*, que podía tratar lógicamente la información, no sólo matemática sino de todo tipo. Cuando la recuperación alemana fué suficiente para permitir la inversión en investigación, Zuse continuó su tarea construyendo nuevos modelos.

Biografía

Konrad Zuse (Berlín, 1910 –): Estudió Ingeniería en la Universidad Técnica de Berlín, entrando a trabajar después en la *Henschel Aircraft Company.* El desastre alemán en la guerra supuso una insalvable limitación para sus investigaciones. En cuanto las circunstancias mejoraron, fundó la *Zuse Company,* que habría de ser la fábrica de ordenadores más importante de Alemania hasta su absorción por la Siemens Corporation en 1960.

Bibliografía

- *Historia de la Informática.* Amparo Gil Orihuel e Ignacio Rieiro Martín. Ed. Alhambra.
- *Enciclopedia Monitor.* Ed. Salvat.
- *Enciclopedia Mi Computer.* Ed. Delta.
- *Gran Enciclopedia Informática.* Ed. Nueva Lente.

La precisión ante todo (II)

Si en la última DATA BUS apareció el número PI, aquí tenéis en número e, base de los logaritmos neperianos. Por cierto, que aunque se listan 1230 cifras sólo son exactas las 1229 primeras, ya que la última está redondeada. Si a alguien le resulta útil esta sección, que nos lo diga. No os podéis ni imaginar la alegría que nos daría, oye.

2.71828182845904523536028747135266249775724709369995957496696762772407
663035354759457138217852516642742746639193200305992181741359662904357 2
900334295260595630738132328627943490763233829880753195251019011573834 1
879307021540891499348841675092447614606680822648001684774118537423454 42
437107539077744992069551702761838606261331384583000752044933826560297 6
067371132007093287091274437470472306969772093101416928368190255151086574
637721112523897844250569536967707854499699679468644549059879316368892
300987931277361782154249992295763514822082698951936680331825288693984 9
646510582093923982948879332036250944311730123819706841614039701983767 93
206832823764648042953118023287825098194558153017567173613320698112509961
818815930416903515988885193458072738667385894228792284998920868058257 4
927961048419844436346324496848756023362482704197862320900216099023530
436994184914631409343173814364054625315209618369088870701676839642437 81
405927145635490613031072085103837505101157477041718986106873969655212671 5
468895703503540212340784981933432106817012100562788023519303322474501 58
539047304199577770935036604169973297250886876966403555707162268447162
560798826517871341951246652010305921236677194325278675398558944896970 96
409754591856956380236370162112 05