DATA BUS

Número 2 Segunda Epoca Nov. 1991

EDITA
ASOCIACION JUVENIL PARA LA INFORMATICA VIGUESA
AJIV

REPORTAJE
TRATAMIENTO FOTOGRAFICO EN EL SOL

RUTINAS MATEMATICAS
PRIMERA PARTE

HISTORIA DE LA INFORMATICA
PARTE V

TECNICA
EL Z80 (II)

LA GUERRA NUCLEAR

GRAN CONCURSO

DATA BUS

SEGUNDA EPOCA
NUMERO 2

DIRECTOR
JOSE M. SUAREZ POUSA

REDACTORES
JESUS CEA
NACHO AGULLO

COLABORADORES
TELMO LAGO
PABLO LOBARIÑAS
SIMON GOMEZ

DISEÑO Y MONTAJE
JOSE M. SUAREZ POUSA

AJV ASOCIACION JUVENIL PARA LA INFORMATICA VIGUESA

C/ Caldas de Reis 12-6 Izq.
TEL: 23 18 35
41 01 13
DEPOSITO LEGAL
VG-87-90

SUMARIO

LA GUERRA NUCLEAR

Jesús Cêa nos lo cuenta todo sobre los principios de la guerra nuclear, el primer concurso de programación organizado por A.J.I.V. en colaboración con la Delegacion de Alumnos de la Escuela de Telecomunicaciones del Campus de Vigo, y con participación de la Xunta de Galicia. **p.7**

y además...

EDITORIAL

Hola amigos,

Suponemos que muchos de nuestros lectores, que son estudiantes, ya habrán empezado las clases, y estarán pensando ya en los próximos examenes. Pero antes de empezar a estudiar a fondo para ello, os invitamos a que participeis en el concurso que organiza esta asociación, y que requiere pocos conocimientos de programación. Y quizas podais ganar uno de los super-premios que ofreceremos.

Como vereis, en este número hay más articulos que nunca, y un nuevo colaborador. Os recordamos, y especialmente a los alumnos de la E.T.S.I. Telecom. de Vigo, que pueden mandarnos artículos para su publicación en esta, vuestra revista.

Un saludo

Equipo utilizado para la realización de Data Bus

HARDWARE

ATARI 1040 STF

ATARI STE 2.5 Megabytes

Monitores *SM 124* y *Sony Black Trinitron*

Amiga 2000 3 Megabytes

Impresora de chorro de tinta *Hewlett Packard Deskjet 500*

Digitalizador de video *VidiST*

Video VHS y Cámara *Sony 8mm*

SOFTWARE

Programa de autoedición *Calamus*

Procesadores de texto *Le Rédacteur 3.1* y *First Word+ 3.15*

Programa de Digitalización *VidiST*

Programa de Tratamiento Digital de Imagenes *Retouche 1.0*

Programas de Dibujo Vectorial *Outline Art* y *Didot LineArt*

Programa Vectorizador *Avant Vector*

TECNICA

Nacho Agulló

EL Z80 DE ZILOG (II)

En la parte anterior vimos la historia del Z80, así como las características y funciones de un microprocesador. En esta parte veremos las características físicas concretas del Z80.

UN VISTAZO FISICO AL Z80

Si en la parte anterior veíamos cómo se relacionaba el microprocesador con los restantes elementos del ordenador mediante los buses de datos y de direcciones, ahora veremos en más detalle todas estas pistas que sirven al Z80 para desempeñar su función. En concreto, son cuarenta las pistas que conectan con nuestro preciado microprocesador, a través de otras tantas patillas. Estas cuarenta pistas se agrupan en tres conjuntos principales, los tres buses de los que ya hablamos en la parte anterior; más tres pistas, que corresponden al reloj, a corriente continua de +5 V y a masa. La pista del reloj es utilizada para enviar al microprocesador las señales que marcarán el ritmo del funcionamiento del mismo, desde un reloj interno del ordenador. Una velocidad usual para el Z80 son 3.5 Mhz, lo que significa que el microprocesador recibe 3.500.000 de estas señales por segundo, es decir, una cada 0.0000002857 segundos.

Los tres buses son, como ya vimos en la parte anterior, el de datos, el de direcciones y el de control. Vimos también la función de los buses de datos y de direcciones en la relación del microprocesador con los elementos del ordenador, en una configuración tipo de buses bidireccionales como la que utiliza el Z80. En concreto, el bus de datos del Z80 es de 8 pistas(de ahí el calificativo de "8 bits" que se le da al microprocesador) y el bus de direcciones es de 16 pistas, pudiendo direccionar hasta 65536 direcciones simultáneamente. En cuanto al bus de control, consta de trece pistas(con las cuales se completan las cuarenta pistas con las que cuenta el Z80), que vamos a ver a continuación.

Estas trece pistas se dividen a su vez en tres grupos, a saber: de Control del Sistema, de Control del Microprocesador y de Control del Bus. Vamos a verlas a continuación:

Pistas de Control del Sistema.

Son seis, todas de salida, activas a nivel bajo.

$\overline{M1}$ (MACHINE 1) Activa en dos situaciones: a) Durante un ciclo de busqueda de código de operación perteneciente a una instrucción, ya conste ésta de uno o dos

bytes de código, y b) Durante el ciclo siguiente a una interrupción(ciclo de acuse), junto con IORQ.

$\overline{\text{MREQ}}$ (MEMORY REQUEST) Triestado, activa mientras el bus de direcciones retiene una dirección válida para una operación de lectura o escritura en la memoria.

$\overline{\text{IORQ}}$ (INPUT/OUPUT REQUEST) Triestado, activa en dos situaciones: a) Mientras la mitad inferior del bus de direcciones retiene una dirección válida para una operación de Entrada/Salida, y b) Durante el ciclo siguiente a una interrupción (ciclo de acuse), junto con M1.

$\overline{\text{RD}}$ (READ) Triestado, activa mientras el bus de datos está siendo utilizado en una operación de lectura de la memoria o de un dispositivo de Entrada/Salida.

$\overline{\text{WR}}$ (WRITE) Triestado, activa mientras el bus de datos está siendo utilizado en una operación de escritura a la memoria o a un dispositivo de Entrada/Salida.

$\overline{\text{RFSH}}$ (REFRESH) Activa cuando los siete bits inferiores del bus de direcciones contienen una dirección de refresco para memoria dinámica y la señal MREQ en curso debe emplearse para realizar una lectura de refresco sobre todas las memorias dinámicas.

Podemos observar que el bus de datos bidireccional que utiliza el Z80, cuya función vimos en la parte anterior, está regulado por dos "semáforos", como son las líneas $\overline{\text{RD}}$ y $\overline{\text{WR}}$, que indican en que sentido se está utilizando el bus de datos.

Pistas de Control del Microprocesador.

Son cinco, de las cuales son cuatro de entrada y una de salida.

$\overline{\text{HALT}}$ (HALT) Salida, activa a nivel bajo durante la ejecución por parte del microprocesador de una instrucción HALT. Esta instrucción provoca una espera indefinida por parte del Z80, durante la cual se ejecutan instrucciones NOP(sin operación), para mantener la actividad de regeneración de la memoria. La espera termina cuando se recibe una señal de interrupción.

$\overline{\text{WAIT}}$ (WAIT) Entrada, activa a nivel bajo. Es utilizada por los dispositivos de Entrada/Salida para indicar al microprocesador que no está preparados para una transferencia de datos. El Z80 introduce entonces estados de espera mientras la señal continúa activa; de esta manera la memoria de los dispositivos de Entrada/Salida puede sincronizarse con el microprocesador central.

$\overline{\text{INT}}$ (MASKABLE INTERRUPTION) Entrada, activa a nivel bajo. Esta señal es enviada por los dispositivos de Entrada/Salida para solicitar la ejecución de un vector de interrupción(un programa alternativo). La petición será aceptada si está activado el indicador de validación de interrupción controlado por el software interno y si no está activada la señal $\overline{\text{BUSRQ}}$. Cuando la petición es aceptada, se genera un ciclo de acuse de recibo, durante el cual se activan $\overline{\text{M1}}$ y $\overline{\text{IORQ}}$.

$\overline{\text{NMI}}$ (NON MASKABLE INTERRUPTION) Entrada, disparada por flanco negativo. Se identifica siempre al terminar la instrucción en curso, independientemente

del estado del indicador de validación de interrupción, excepto si está activa la señal BUSRQ, que la anula. Funciona de forma similar a INT, pero su prioridad es más alta; provoca el equivalente a una llamada a la dirección 0066H, donde se sitúa el correspondiente vector de interrupción.

RESET Entrada, provoca el salto del microprocesador a la primera dirección de la memoria(0000H), que es también la que se ejecuta cuando se enciende el ordenador. Se utiliza para reinicializar el ordenador.

Pistas de Control del Bus.

Son dos:

BUSRQ (BUS REQUEST) Entrada, es utilizada por los periféricos para requerir al microprocesador el uso de los buses de datos y direcciones. Cada activación de esta línea produce el "congelamiento" de la actividad del microprocesador durante un ciclo, a fin de no interferir la utilización de los buses por parte de los periféricos.

BUSAK (BUS ACKNOWLEDGE) Salida, activa tras la recepción de una señal BUSRQ. Es utilizada para indicar que el requerimiento de utilización de los buses se está atendiendo y el periférico solicitante tiene vía libre para la utilización de los mismos.

Y con esto hemos finalizado el vistazo físico al Z80.

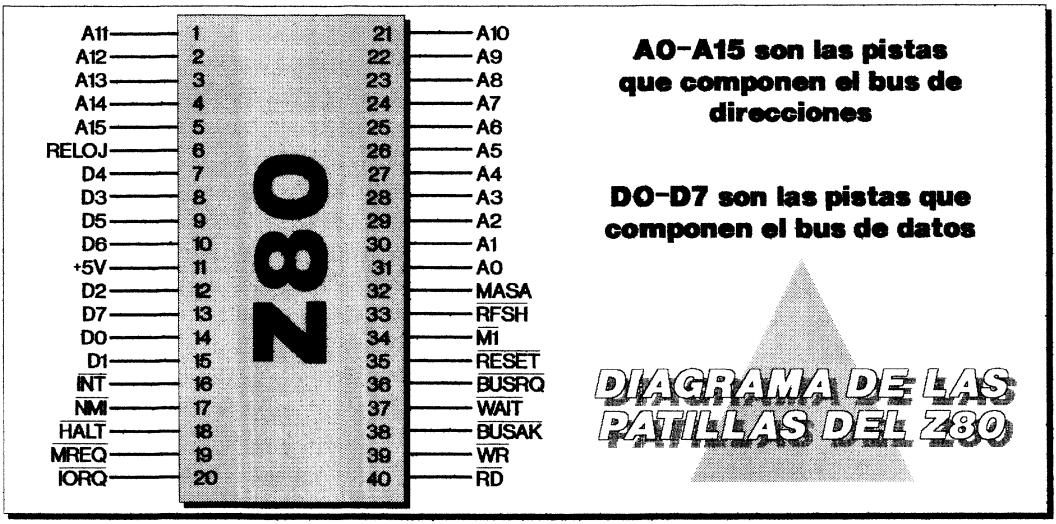
BIBLIOGRAFIA

- 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.*



CURIOSIDADES INFORMATICAS

Jesús Cea

LA GUERRA NUCLEAR

No, señores. No se trata de un holocausto atómico, aunque los resultados para los combatientes vienen a ser los mismos. En un universo plagado de cráteres y minas se emboscan los dos rivales, en una lucha sin cuartel. Su lema es: Sólo puede quedar uno (esto me suena de algo...).

La guerra nuclear consiste en un universo en forma de anillo (no tiene principio ni fin) en el cual se ponen a luchar dos programas escritos en un lenguaje especial denominado *REDCODE*. El intérprete del REDCODE se denomina *MARS* (Memory Array Redcode Simulator). El objetivo de cada programa es eliminar al adversario al tiempo que se protege de los ataques enemigos. Para ello, cada programa debe desplegar una estrategia de combate lo bastante buena como para sobrevivir a ese mundo salvaje y sin reglas denominado matriz de juego.

Al parecer el nombre de guerra nuclear se debe a que los combates se desarrollan en una memoria, y las viejas memorias de ordenador estaban constituidas por núcleos de ferrita, de ahí el nombre. Personalmente me parece evidente que dicha explicación se dió a posteriori, al igual que el significado de *MARS*. De todos modos, no deja de tener ciertas connotaciones, ¿verdad?

La idea original de este artículo así como del consiguiente programa de ordenador se tomó de varios artículos aparecidos en la revista "Investigación y Ciencia" desde 1.984. Sin embargo, a la hora de crear mi propio intérprete *MARS* he realizado multitud de modificaciones tanto para facilitar la programación como para aumentar el interés por el juego. Todo lo que se diga aquí corresponde a mi versión particular del *MARS*. De todos modos os recomiendo que leais los artículos aparecidos en la ya mencionada revista. Valen la pena...

Para poder jugar a la guerra nuclear se necesitan: un ordenador, un intérprete *MARS* para dicho ordenador y dos programas combatientes. El intérprete *MARS* puede traer ya incluido un editor para simplificar el desarrollo y depuración de los programas, como es mi caso. En resumen, se trata de un programa que va leyendo instrucciones alternativas de ambos códigos rivales y ejecutando las operaciones correspondientes asociadas. Los programas luchadores viven y mueren en la matriz de juego, auténtica arena de circo digna de los mejores gladiadores. El intérprete *MARS* debe manejar convenientemente dicha zona de juego para simular un anillo sin fin. En mi versión de la guerra nuclear se trabaja con un tablero de 8192 celdas (el original eran 8000). Así, para acceder a cualquier casilla se realiza un *AND 8191*, lo que nos asegura que obtendremos tendremos valores válidos 0-8191. Otra modificación adicional es que un programa no puede acceder simultaneamente a toda la matriz de juego, sino tan sólo a las 4096 casillas que le rodean. Con esa limitación se intenta dar prioridad a la movilidad de los programas.

Como ya comenté al principio, los programas se escriben en un lenguaje pseudo-ensamblador denominado REDCODE. En la figura 1 podéis ver los comandos disponi-

bles con los modos de direccionamiento permitidos. Se ha intentado crear un REDCODE facilmente asimilable incluso por gente sin conocimientos de programación. Para ello se ha limitado el número de instrucciones a 10, y a 3 los modos de direccionamiento, todos ellos muy sencillos. Para todas aquellas personas sin conocimientos de ensamblador, voy a explicar el funcionamiento de cada modo:

El primero de todos es el modo inmediato. Este modo sólo puede aparecer como fuente, nunca como destino (la fuente es el primer argumento y el destino el segundo). Produce que el valor indicado sea introducido en la casilla señalada por destino. En el caso del comando *DAT*, el valor indica el dato que se almacena es esa misma casilla. Este es el modo más sencillo de entender y manejar. Se reconoce porque aparece un simbolo "#" antes del dato. Los valores permitidos (al igual que en los otros modos) cubren desde -2048 hasta 2047 (son las 4096 casillas que dije antes).

El segundo modo es el relativo. Como ya expliqué la matriz de juego no tiene principio ni final, lo que hace absurdo referirse a casillas de forma absoluta. La única forma de referirse a las celdas consiste en especificar desplazamientos de la forma "la casilla situada 3 celdas más adelante", o "la celda situada 15 casillas hacia tras de la instrucción actual". Este modo puede aparecer tanto en la fuente como en el destino. Así, un *MOV 0,1* copiaría la celda actual en la siguiente. Por convenio los desplazamientos son relativos a la instrucción que se está ejecutando actualmente. Se trata de el modo más simple de acceder a la memoria. Se reconoce porque el parámetro se indica directamente, sin poner ningún símbolo delante.

El tercer modo se denomina indirecto y es, con mucho, el más complicado. Como contrapartida es el más útil de los tres. Se reconoce porque el argumento se pone entre paréntesis. El argumento indica una casilla cuyo valor se toma como un nuevo desplazamiento. Es como si dicha casilla contuviera un puntero a la posición real que se desea acceder. En la figura 2 tenéis un par de ejemplos para que entendáis perfectamente su funcionamiento.

En cuanto a las instrucciones, su funcionamiento se indica en la figura 1. *Inm* significa direccionamiento inmediato y *mem* implica direccionamiento a memoria, tanto relativo como indirecto.

Cuando un programa intenta ejecutar una instrucción *DAT*, *MARS* reconoce que se trata de una instrucción ilegal, y aborta su ejecución. En el caso de que no se estén

Figura 1:

INSTRUCCION		***FUNCION ASOCIADA***
DAT	inm	Sirve para almacenar datos tales como punteros. No es ejecutable.
MOV	inm, mem mem, mem	Permite copiar celdas o bien asignar valores concretos a las casillas.
ADD	inm, mem mem, mem	Suma (o resta) un valor, inmediato o depositado en la memoria, a una casilla.
JMP	mem	Salta a una casilla determinada.
JMZ	mem, mem	Salta a la casilla destino si la fuente tiene valor cero.
JMN	mem, mem	Salta a la casilla destino si la fuente tiene un valor distinto de cero.
PRT	mem	Protege una casilla determinada de ataques enemigos.
SPL	mem	Divide el programa en varios subprocesos independientes, realizándose una multitarea entre ellos.
DJN	mem, mem	Decrementa la celda fuente. Si su valor no es cero después de dicha modificación, salta al destino. Si es cero continua con la siguiente instrucción.
CMP	mem, mem	Compara dos celdas de memoria. Si sus valores NO coinciden, se salta la instrucción siguiente.

ejecutando más programas por ese bando, se declarará perdedor. De ello se deduce que para eliminar un programa enemigo éste debe ser bombardeado con instrucciones *DAT*. Esto puede hacerse utilizando los comandos *MOV*, *ADD* y *DJN*, ya que convierten las casillas destino en instrukciones *DAT* automaticamente. (*MOV* sólo lo hace cuando se utiliza una fuente inmediata). Al empezar, todas las casillas no ocupadas se inicializan como *DAT* #-2048.

Figura 2:

```
            MOV   #0, (puntero)     ;Pone un DAT #0
            ...                     ;30 casillas después
puntero:    DAT   #30               ;de puntero.

            Otro ejemplo:
            MOV   (puntero), (puntero2)   ;Copia la casilla situada
            ...                           ;20 celdas después de puntero
puntero:    DAT   #20                     ;en la casilla que está 800
puntero2:   DAT   #-800                   ;casillas antes de puntero2.
```

Resulta trivial ver que para restar se utiliza el comando *ADD* con un argumento negativo. *PRT* crea una "coraza" sobre la casilla indicada. Dicho escudo SOLO resistirá un impacto. La protección desaparece cuando se intenta escribir sobre la celda, por lo que estará desguarnecida si se intenta modificar por segunda vez. El comando *CMP* compara dos casillas enteramente (se comprueba si coinciden tanto la instrucción como sus parámetros si los tuviera).

La instrucción *SPL* sirve para dividir la ejecución de un programa en dos, ejecutandose alternativamente una instrucción de cada parte. Se permiten hasta 32768 programas ejecutándose simultaneamente (por bando). Puede parecer que el número máximo de programas simultáneos está limitado al número de casillas (8192), pero nada impide que haya 100 programas ejecutando la misma celda a la vez... Si un programa supera dicho límite es declarado perdedor automaticamente. Si uno de los subprogramas es destruido, se anula su ejecución pero los demás ni siquiera se enteran. Así, para eliminar a un jugador hay que destruir todos sus subprogramas. Como contrapartida, cuantos más programas se estén ejecutando tanto más lentos correrán (1/n donde n es el número de programas de un bando). Ley de vida...

Independientemente de si tienen o no parámetros, todos los comandos ocupan una casilla y tardan lo mismo en ejecutarse. La automodificación no entra (por ahora) dentro de la filosofía del juego. El resultado usual de intentar modificar una casilla es, generalmente, su conversión a *DAT* a no ser que se use *MOV mem,mem*.

Cuando se inicializa *MARS*, éste procede a situar en posiciones aleatorias a los dos contendientes poniendo buen cuidado en que no se solapen. Seguidamente se ejecuta alternativamente una instrucción de cada bando y se muestran en pantalla las modificaciones sufridas por la matriz de juego. Si se encuentra un comando *SPL* se procede a añadir a la lista de ejecución del jugador en cuestión un nuevo programa. Si se encuentra un comando *DAT*, se borra de la lista de ejecución el programa dañado, y si era el último que quedaba, se declara perdedor.

La ejecución continúa hasta que se eliminan todos los programas de un jugador o hasta que se llegue al límite de instrucciones por bando. Dicho límite evita que *MARS* entre en un ciclo infinito como el que se produciría si se estuvieran ejecutando dos trasgos (situación frecuente. Ver más adelante). Si se alcanza el límite y todavía hay programas funcionando, se declara un empate. En mi versión de *MARS* dicho límite se selecciona entre 32768, 65536 y 131072 intrucciones por bando. Dado que he escrito el programa 100% en ensamblador se ejecutan más de 4000 instrucciones por segundo y por bando, por lo que los resultados de los encuentros se conocen rapidamente. Como el resultado de una sóla batalla no es significativo mi programa permite seleccionar torneos de 1, 15 o 31 combates al cabo de los cuales muestra las estadísticas totales de partidas ganadas, perdidas o empatadas.

El formato en que se deben entregar los

programas es como sigue (en mi caso, el editor, el compilador y el intérprete forman parte del mismo programa, pero no se puede decir lo mismo de versiones que escriban otra gente). Primero, una etiqueta de un máximo de 8 caracteres empezando por una letra y siguiéndole indistintamente cifras o letras. El final de la etiqueta se indica con dos puntos ":". Si no hay etiqueta no se pone nada. Luego va el opcode de 3 letras seguido del direccionamiento que se desee (siempre que sea válido). Los espacios innecesarios no se tienen en cuenta. Si queremos poner un comentario, se pone con un punto y coma ";" delante. La longitud máxima del comentario es de 32 caracteres más el punto y coma (si se requieren comentarios más largos hay que dividirlos en varias lineas). Está TERMINANTEMENTE PROHI-

Figura 3:

```
trasgo:  MOV  trasgo, trasgo2   ;Copia esta instrucción
trasgo2:                        ;en la siguiente celda

        O lo que es lo mismo:
        MOV  0, 1    ;Hace lo mismo
```

BIDO insertar lineas en blanco. Si se requiere una linea vacía habrá que conformarse con poner un punto y coma sin ningún comentario. Cada linea debe terminar con un retorno de carro ($0D). Los saltos de linea ($0A) son opcionales y debieran ser ignorados al cargar el listado. Sin embargo, es conveniente incluir alguno por si se requiere sacar una copia por impresora. No existe diferenciación mayúsculas/minúsculas.

Es responsabilidad del programador comprobar si el fichero que se indica para compilar es o no correcto tanto en el formato en sí como en etiquetas no válidas o inexistentes, direccionamientos incorrectos, longitudes ilegales, etc. Es muy importante recordar que el intervalo de valores válidos para todos los direccionamientos van desde -2048 hasta 2047. Valores fuera de este rango debieran provocar un error (recordad también que un programa no puede acceder a toda la matriz de juego a la vez). Aconsejo a todo aquel que desee escribir un intérprete *MARS* que incluya también un editor y que se ponga en contacto conmigo para evitar posibles incompatibilidades. Por cierto, ¡¡¡la longitud de los programas NO DEBE superar las 255 lineas!!!. De todos modos, un programa tan grande parece ser (a priori...) demasiado vulnerable como para ser factible.

Ahora vamos a ver un par de ejemplos de programas "luchadores". Sin lugar a dudas, el programa más sencillo es el que se muestra en la figura 3. Se llama *trasgo* y mide una sóla celda. Funciona copiando la instrucción actual en la casilla siguiente. Cuando *MARS* incrementa su *PC* (Program Counter), se encuentra con la misma instrucción y así se repite el proceso indefinidamente. Trasgo barre toda la memoria muy rapidamente, y la única forma de pararlo es acertarle con un "bombazo" de lleno (algo dificilillo por constar de una sóla instrucción) o ponerle una casilla protegida delante. Como contrapartida, trasgo no gana casi nunca sino que empata. Veamos esto:

Cualquier programa "normal" consta de una serie de comandos que se ejecutan secuencialmente y de una instrucción que devuelve la ejecución hacia atrás originando un bucle (normalmente un *JMP* o un *DJN*). Supongamos que se ejecuta el cuerpo principal del bucle y trasgo empieza a "machacarlo" desde atrás. Cuando el programa ejecute la instrucción que devuelve el control al principio del bucle, ¿qué es lo que se encuentra? Pues la "estela" que trasgo fue dejando tras de sí. En este momento el programa es como si sufriera una mutación. Ahora tenemos dos trasgos persiguiéndose por la matriz de juego. Esta persecución acaba cuando se llega al ya comentado límite de instrucciones, declarándose un empate.

Como puede verse, trasgo es un ejemplo de programa muy agresivo pero no por ello ganador. El siguiente paso podría ser el programa mostrado en la figura 4a. Se llama *enano*. Debo hacer notar que la versión propuesta en "Investigación y ciencia" sumaba 5 en vez de 4 (porque eran 8000 celdas) y era mucho más efectivo al poder acceder a toda la matriz a la vez, y no a sólo 4096 casillas, como aquí. ¿Como funciona?

Las primeras tres instrucciones crean un bucle que se encarga de poner un *DAT* #0 cada 4 celdas. Una duda nos asalta: ¿Qué ocurre cuando el puntero llega a 2047 (el límite superior) ? Veamos: 4, 8, 12, ..., 2040, 2044, 2048. Pero el valor 2048 no en válido. *MARS* automaticamente realiza un "clipping" para solucionar esto. La secuencia real sería 2040, 2044, -2048, -2044, ... Problema resuelto (en realidad es exactamente igual que trabajando en ensamblador). Pero surge otro problema; ¿No se machacará a sí mismo? Siguiendo la secuencia: -2044, -2040, ..., -8,-4. Enano bombardea 4 casillas antes de puntero. "Casualmente", es exactamente la celda anterior a la primera instrucción del programa. La siguiente granada cae "exactamente" sobre puntero. A partir de aquí todo se repite como antes. ¡¡Uff!!, estuvo cerca, ¿verdad?

Enano sería un programa muy efectivo si no fuera porque sólo puede acceder a la mitad del area de juego. Como dije al principio, esto está hecho a propósito para obligar a los programas a moverse constantemente de aquí para allá. Pero enano tiene otro talón de Aquiles. ¿Qué ocurre si enfrentamos al enano contra su mortal enemigo, el trasgo? Los enfrentamientos enano-trasgo casi siempre acaban en empate por las razones expuestas anteriormente (aunque enano barre la memoria, sólo lo hace de 4 en 4 celdas y trasgo puede escurrir el bulto con relativa facilidad). Craso error. Habida cuenta de que el trasgo es un enemigo peligroso y facilmente "utilizable" (comando *SPL*), cualquier programa que se precie debe ser a pruebas de trasgos y demás alimañas similares. La figura 4b muestra como enano puede aprender de sus errores. Se trata de *enano2*, hermano de sangre del enano original. Cuando trasgo tropieza con la casilla protegida no puede copiarse. Esa casilla esta vacía con casi total seguridad, y trasgo morirá al intentar ejecutar un *DAT*. Otra cosa distinta en que cuando se colocaron los programas al principio, trasgo coincida precisamente en dicha casilla (una posibilidad remota, ciertamente) en dicho caso parece que se da la situación de empate de siempre, ¿o no? Pues no.

Veamos más de cerca esa situación. Tras unos pocos movimientos tenemos a trasgo barriendo la memoria seguido de cerca por un veloz enano2 reconvertido. La situación parece clara, pero veamos que ocurre cuando están a punto de cumplir la primera vuelta al circuito y trasgo va ganando por un par de celdas a enano2. ¡¡¡Trasgo se estrella contra la casilla protegida del principio!!!. Algo totalmente inesperado, ¿verdad? (esto debe mostraros que aquí NUNCA hay que dar nada por sentado). Por lo tanto, enano2 tiene un porcentaje de efectividad contra trasgo del 100%. Esto es otra cosa, ¿eh?

Como detalle, enano2 destruye la primera casilla de sí mismo (el *PRT*). Ese no es ningún problema dado que dicha celda sólo se ejecuta la primera vez y luego es totalmente superflua. Otro detalle es que ponga *PRT* -1 y no *PRT* 0, que sería lo más normal para que trasgo no machacara a enano2. Examinemos cuidadosamente esa situación.

Supongamos que trago es capaz de sortear exitosamente las andanadas de enano2 y llega a la casilla protegida (el *PRT* del principio). Trasgo intenta copiarse pero no tiene éxito. De todos modos, cuando le toque ejecutar la instrucción siguiente se encontrará con el propio programa enano2. ¡¡¡Ahora tenemos a ambos contendientes ejecutando el mismo programa!!!. Una situación curiosa, cuando menos. Para que se dé esta situación es necesario que trasgo alcance a enano2 ANTES de que este bombardee UNA vez la instrucción *PRT*. Si trasgo alcanza a enano2 después de que

Figura 4:

a)

```
go:       MOV  #0, (puntero)
          ADD  #4, puntero
          JMP  go
puntero:  DAT  #4
```

b)

```
go:       PRT  -1
go2:      MOV  #0, (puntero)
          ADD  #4, puntero
          JMP  go2
puntero:  DAT  #4
```

éste haya trasformado el *PRT* en un miserable *DAT* (o simplemente, si trasgo alcanza a enano2 después de que éste haya "desprotegido" el *PRT*, valga la redundancia), iiise produce la persecución de siempre porque ahora trasgo no se encuentra con ninguna casilla protegida!!!. En realidad lo que ocurre es que cuando enano2 destruyó el *PRT*, se trasformó en un enano corriente y moliente y claro, pasa lo que tiene que pasar...

Ahora que sabemos como producir y como eliminar trasgos, conjuguemos todo esto en un programa de lucha (figura 5). Yo le llamo lanzatrasgos por razones evidentes. Lanzatrasgos va enviando trasgos cada cierto tiempo a barrer la memoria. Así mismo, protege una casilla constantemente para evitar ser víctima de sus propios secuaces. Hay que experimentar con el valor de retardo para obtener unos resultados satisfactorios. Si se envían los trasgos demasiado juntos, el programa se enlentece sobremanera y puede no alcanzar al enemigo antes de cumplir el límite de instrucciones, además de convertirse en un blanco fácil. Si se mandan los trasgos demasiado separados, cualquier adversario que se proteja utilizando un bucle estará a salvo. Enfrentando a enano2 y a lanzatrasgos a lo largo de 31 partidas, lanzatrasgos ganó 26 y perdió 2. No está mal.

Cuando no se utilizan códigos autodesplazantes tipo trasgo, es necesario que el programa se vaya moviendo para poder así acceder a toda la memoria de juego. En la figura 6 muestro un programa capaz de copiarse así mismo 211 casillas después de puntero2 y de pasarle el control a la nueva copia. iiiNo se trata de un programa de lucha porque los enemigos que "arrolla" pasan a ejecutar el mismo listado!!!. Sólo se trata de un ejemplo para que veais como crear códigos capaces de moverse por ahí, algo bastante necesario. Comentaros que se puede hacer exactamente lo mismo pero con menos instrucciones (y más rápido), pero no voy a deciros como. Venga, iiia darle al coco!!!.

Para finalizar voy a daros una serie de consejos que considero importantes. Espero que no caigan en saco roto:

Figura 5:

```
loop:  MOV  #551, -1
loop2: PRT  -3
       DJN  -3, loop2
       SPL  loop
       MOV  0, 1
```

Figura 6:

```
       MOV  #-8, puntero
       MOV  #211, puntero2
       MOV  #8, contador
loop:  MOV  (puntero), (puntero2)
       ADD  #1, puntero
       ADD  #1, puntero2
       DJN  contador, loop
       JMP  213
puntero:  DAT  #0
puntero2: DAT  #0
contador: DAT  #0
```

1. Siempre que sea posible, escribir los programas con el propio editor *MARS* (si es que existe).

2. Los programadores que deseen escribir su propio *MARS* debieran ponerse en contacto conmigo primero para evitar incompatibilidades no deseadas.

3. Seguir escrupulosamente el formato de listado explicado anteriormente. Si diseñáis los programas con un editor *MARS* directamente, éste debiera hacerlo automaticamente.

4. Hacer los programas de lucha lo más cortos que sea posible para que sean menos vulnerables.

5. Cuanto más rápido se ejecuta un programa menos riesgos corre.

6. NO utilizar nunca características especiales de un *MARS* determinado, como podría ser el hecho de que las instrucciones *ADD* y *DJN* transforman el destino en un *DAT*. En mi programa funcionaría, pero quizás no en otras implementaciones. Además, es posible que en futuras versiones se incorpore la automodificación (si se hace esto, probablemente se añadan nue-

vas instrucciones para mantener la compatibilidad).

7. Un empate siempre es mejor que perder.

8. Cuidado con el número de programas que se ejecutan simultaneamente con la instrucción *SPL*, no por el límite de 32768 (muy difícil de alcanzar) sino porque la velocidad de ejecución decrece rapidamente.

9. Leer los artículos aparecidos en "Investigación y ciencia". Seguro que os dan más de una idea.

10. Todas aquellas personas interesadas en el tema pueden ponerse en contacto conmigo para disponer de una información más extensa. Cualquier idea constructiva será bien acogida y tendrá su recompensa.

11. Guerra nuclear puede servir de enorme ayuda para introducir a la gente en el apasionante mundo de la programación. Es muy fácil de aprender y desarrolla en gran medida el razonamiento lógico, la intuición y, sobre todo, despierta el interés el mundo de los ordenadores en general y de la programación en particular.

Bien, esto es todo por este mes. Espero que haya conseguido despertar el gusanillo por la guerra nuclear "pacífica". Si así es, me sentiré plenamente satisfecho.

Mi versión de MARS funciona en cualquier Atari ST/STE, incluso en el 520. Trabaja perfectamente en las tres resoluciones básicas de este ordenador. Incluye en un sólo programa, 100% en ensamblador, un editor, un compilador y un intérprete de MARS. Permite escribir dos programas simultaneamente (los dos contendientes) y el editor comprueba cada vez que se introduce una linea que el formato sea correcto. Si queréis más detalles, sólo tenéis que pedírmelos. Mi programa es de dominio público y está a disposición de todo aquel que lo solicite.

Un compañero de la universidad está escribiendo una versión de MARS para PC compatibles. Esperamos que esté disponible para Octubre, porque los exámenes están encima y hay que empezar a estudiar. Ya os informaremos puntualmente de las novedades al respecto.

Si podemos contar con dicho programa para Octubre, tenemos pensado realizar un campeonato de Guerra Nuclear en la universidad viguesa. En principio, la participación será libre y gratuita, y habrá importantes premios para los mejor clasificados (es una actividad subvencionada por la Xunta de Galicia). Esta es una idea que me ronda la cabeza desde hace tiempo y iiide este año no pasa!!!...

Para aquellas personas que se decidan a leer los artículos originales aparecidos en la revista "INVESTIGACION Y CIENCIA" a partir de 1984, cito a continuación las diferencias entre el intérprete comentado en dicha revista y el mio propio:

1. El tablero original media 8.000 cassillas, mientras que mi versión dispone de 8.192.

2. Los programas escritos para el MARS original podian acceder simultaneamente a toda la memoria, mientras que aquí, un programa sólo puede acceder directamente a las 4.096 casillas más cercanas.

3. La forma de señalar el direccionamiento indirecto en dicha revista era mediante una aroba. En mi versión se especifica indicando el argumento entre paréntesis.

4. La sintaxis de los comandos originales era primero destino, y luego fuente, mientras que yo utilizo el convenio más usual de primero fuente y luego destino.

Como podéis ver los cambios son más bien para trabajar con un código mucho más familiar para aquellas personas que cuenten con conocimientos previos de ensamblador.

REPORTAJE

J.M.SUAREZ

EL TRATAMIENTO DE LA IMAGEN FOTOGRAFICA EN "EL SOL"

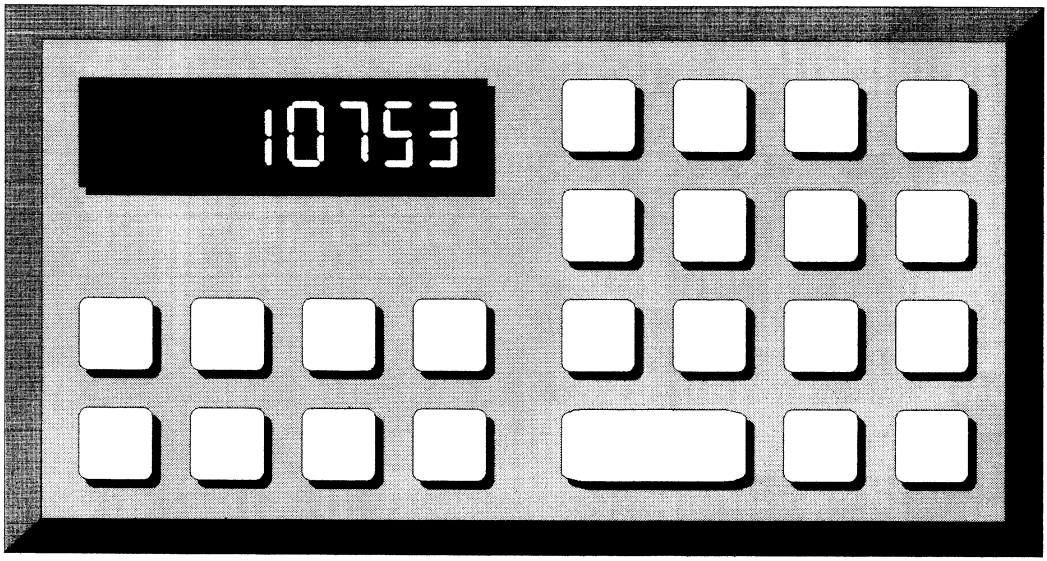
Con tan solo hechar un vistazo a la variedad de diarios nacional en nuestro país, hay algo que inmediatamente captamos: las fotografías, y sobre todo la calidad de las mismas. En este aspecto "El Sol" se lleva la palma. La nitidez que muestrán sus imagenes sólo es comparable con la de otros diarios de reciente creación en todo el mundo, como son el "National", de Estados Unidos, el londinense "European", o el canadiense "Le Droit".

La clave está en su avanzado sistema de digitalización fotográfica y transmisión por fibra óptica, que permiten un tratamiento de la imágen sin apenas perdida de calidad. Aunque en ningún momento debemos descartar la gran labor que realizan los fotógrafos de "El Sol", coordinados por Miguel González, vamos a centrar nuestra atención en el tratamiento de la imagen, tras la realización y el revelado de la foto. Para ello este diario cuenta con una red tecnológica altamente sofisticada que permite tratar la imagen electrónicamente, al igual que se hace con los textos, siempre a través de las pantallas del Macintosh y gracias a un programa especial de digitalización de imagen, denominado "Photo Shop".

Pues bien, el proceso de entrada de una foto para su tratamiento comienza una vez que ésta ha superado esos primeros pasos esenciales para su "existencia"- salida del fotógrafo, disparo, impresión, revelado del negativo y copiado- y trás haber sido seleccionada para su publicación se envía a la sala de fotocomposición, donde se acomoda en el apartado correspondiente, segun el tema de que trate, junto con las dimensiones en que será reproducida.

La sala de fotocomposición está equipada con seis pantallas todas ellas conectadas con éscaneres, elemento principal para el tratamiento de la imagen. Los éscaneres pueden ser de diferentes tipos y modelos según su uso; unos sirven únicamente para diapositivas, como el Nikon 35 mm Film Scaner LS-3500; otros para opacos de grandes dimensiones, o simplemente para copias en blanco y negro, como el ECRM Autokon 2000; y otros, como el Color Getter Optitronics, admiten cualquier tipo de soporte, incluido el mismo negativo.

El tratamiento de la imagen comienza cuando la foto, diapo o negativo se introduce en el éscaner, e inmediatamente se visiona en la pantalla del Macintosh. El primer paso es la lectura de la imagen mediante un foco luminoso que la recorre de izquierda a derecha, al tiempo que va extrayendo toda la información contenida en sus pixels. Una vez realizada la lectura el programa informático "Photo Shop" permite adaptar la imagen al gusto del consumidor -lo que ha concluído por llamarse "escanear"-. Pero antes, eso sí, hay que darle las dimensiones correctas para su posterior inserción en la página que corresponda. Las posibilidades



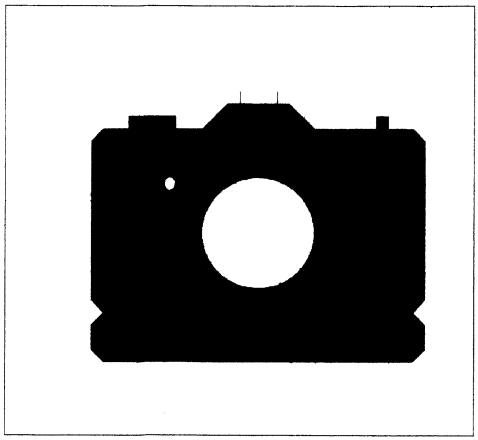
que ofrece este sistema son múltiples: desde la selección de zonas, ampliaciones, reducciones, recortes,... e incluso posibles retoques o manipulaciones, algo que, evidentemente, no está permitido si se trata de material para publicar, ya que podría tergiversar el sentido real de la información.

Una vez que la imagen ha sido tratada, y tras haberse llegado a los resultados deseados, se devuelve al redactor pertinente o bien se envía directamente a los talleres para la inserción en su página. Todo el proceso es posible a través de las redes informáticas que conectan a todos los trabajadores de "El Sol" mediante 200 terminales, y que tras la descomposición de la imagen en dígitos, le hace circular por las redes del mismo modo que la información escrita.

De la agencia al disco óptico

También las fotos de agencia entran directamente al sistema de forma digitalizada. "El Sol" recibe diariamente cientos de imágenes procedentes de las agencias contratadas -Efe, Reuter, Epa y Associated Press- pero en lugar de obtenerlas en opacos, como sería el método tradicional, las recibe y visualiza directamente en pantallas, una para cada agencia proveedora. Al mismo tiempo que una foto llega a la pantalla por vía telefónica, la imagen se reproduce en papel de baja calidad, en el que se indica un número de referencia. Mientras la imagen original es procesada electrónicamente y almacenada en discos ópticos, la mala copia en papel sirve para recuperarla en cualquier momento y de forma inmediata a través del número de referencia. Si bien podríamos preguntarnos por qué no trabajar directamente con el original de pantalla antes de ser archivado, la respuesta esta en que, por un lado cada foto contiene infinidad de información y retener demasiada podría colapsar la red, y por otro lado porque evita la necesidad de comprobar si son válidas todas y cada una de las imágenes que llegan. De esta manera los redactores o el editor pueden visualizar la foto en la mala copia y recuperarla en segundos si interesa. Para ello sólo hay que teclear el número de referencia e inmediatamente se recupera en pantalla, y del

mismo modo que las fotos aportadas por los fotógrafos "de la casa" se pueden modificar las luces y sombras, realzar detalles, etcétera, etcétera. Tanto en blanco y negro como en color pueden ser escaneadas, o lo



que es lo mismo, tratadas para mejorar su calidad.

Trabajar con una imagen de color es siempre algo más complicado, ya que en lugar de una sola copia, como es el caso del blanco y negro, son necesarias cuatro; cada una de ellas contiene uno de los colores básicos: cyan, magenta, amarillo o negro. Para que el resultado del proceso sea perfectamente satisfactorio, lo primero -tras recuperar la imagen color a color- es hacer coincidir exactamente las cuatro copias en pantalla. Una vez conseguido, ya sólo queda modificarla como se desee. Además de otras facilidades, el programa contiene en densímetro que, como su nombre indica, mide las densidades del color en cada zona, a la vez que permite retocarlo, bien en cada zona, bien tono a tono.

Del mismo modo, "El Sol" está preparado para recibir las fotos que envíen los corresponsales, gracias a los receptores de telefoto, procesarlas, archivarlas o permitir su recuperación para ser tratadas, o acceder a las que procedan de bancos de imágenes; e incluso acepta la posibilidad de tratar imágenes televisivas mediante un adaptador especial al sistema.

Vemos pues como el proceso fotográfico en el diario "El Sol" antes de su paso a las rotativas no termina con el revelado, sino que la imagen es escaneada y tratada por el personal especializado, siempre en vistas a ofrecer la mejor calidad.

HISTORIA

HISTORIA DE LA INFORMATICA

Nacho Agulló

PARTE V

En la parte anterior hemos visto como Hermann Hollerith inauguraba la era de la Informática de Gestión. A continuación veremos como al tiempo que sigue avanzando esta vertiente de la Informática, ya regida por las leyes del mercado y de la competencia, se continúan produciendo avances cualitativos que más tarde serán recogidos por la informática comercial.

LA INFORMATICA DE GESTION A COMIENZOS DEL SIGLO XX

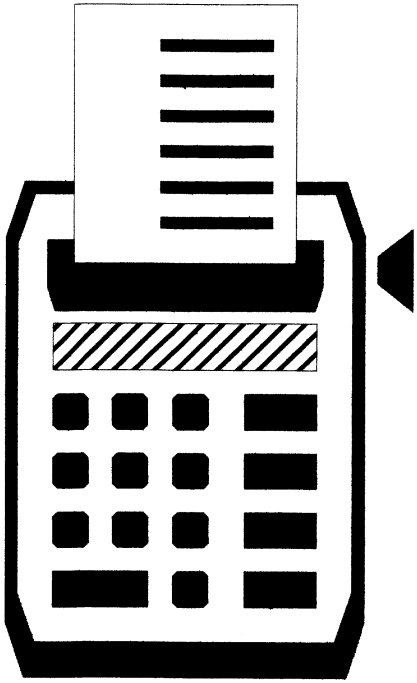
En la parte anterior habíamos visto como la Informática de Gestión comenzaba a ser utilizada para el proceso de datos a gran escala en el ámbito comercial, todavía únicamente a través de los equipos de Hollerith. A continuación veremos cómo se inició la competencia por primera ver en el mercado de la informática.

Las patentes que poseía Hollerith le permitían cobrar precios elevados por los equipos que necesitaba la Oficina de Censos de los EE. UU., la cual no podía prescindir de ellos por imposibilidad práctica de realizar el trabajo manualmente. Se generó un conflicto económico que terminó con el abandono de la Oficina de Censos por Hollerith en 1903. Sin Hollerith, la Oficina de Censos se veía reintegrada al estado de 1889. La solución estaba en la aparición de equipos alternativos - competencia-, y así la Oficina de Censos se dedicó a fomentar la creación de equipos de proceso de datos alternativos, que además no fueran simples copias de los de Hollerith, para no violar sus patentes. No deja de ser significativo que esta tarea de fomento estuviera inspirada, no por el interés por el progreso y el amor a la ciencia, como las subvenciones del Parlamento inglés a Charles Babbage, sino por motivos exclusivamene económicos - los precios de Hollerith - y prácticos - la imposibilidad de realizar el trabajo sin los equipos de proceso de datos -.

Aunque en 1905 comenzaron a caducar las patentes de Hollerith, parte de ellas se mantenía aún en vigor, lo que constituía una limitación para la aparición de equipos alternativos. No obstante, en 1910 James Powers, ingeniero estadístico de Nueva York, ofreció un sistema de tabuladoras mecánicas basadas en las de Hollerith, con la diferencia de que el sistema utilizado para la lectura de las tarjetas era mecánico; de esta manera, no violaba las patentes de Hollerith. Así fué como se inauguró la competencia en el ámbito de la informática comercial. Sucesivamente, Powers patentó una forma de reoresentar datos alfanuméricos en fichas perforadas mediante una perforación a diferente altura para un número y una combinación de perforaciones para una letra, y Hollerith respondió con

una nueva tarjeta standard de 80 columnas y 12 filas.

Las máquinas de Powers carecían de la capacidad de programación que poseían las de Hollerith, por haber inventado éste en 1902 un conmutador de clavijas similar a los



utilizados con las líneas telefónicas que permitía seleccionar las columnas de las fichas perforadas a agregar; no obstante, Powers fué un duro competidor. En 1911 abandonó la Oficina de Censos, en la que había trabajado al igual que Hollerith, para crear la "Compañía Powers de Máquinas de Contabilidad", que fué comprada en 1927 por la "Remington Rand Corporativa". La competencia continuó, ya bajo los nombres de Remington e IBM, acaparando entre ambas la mayoría del mercado durante muchos años, más allá de la muerte de Hollerith y Powers, como si su rivalidad hubiera impregnado a los sistemas que diseñaron.

Posteriormente, se produciría una evolución y especialización del mercado, con la apariciòn de nuevas empresas. Algunos ejemplos son la empresea del noruego Fredik Rosnig Bull, reconvertida en 1934 en Francia en la "Cie. des Machines Bull", actualmente "Bull"; la "British Tabulating", hoy ICL, que se especializó junto con IBM en tabuladoras, NCR y "Burroughs", que se esoecializaron en máquinas de contabilidad.

La tabuladora fué sustituida finalmente por el ordenador; pero durante los años treinta y cuarenta vivió su época dorada en la informática comercial. Unos grandes almacenes de Pittsburgh, por ejemplo, utilizaron un sistema de ventas con 250 terminales distribuidas por el establecimiento. IBM patentó una capaz de llevar los registros de las transacciones de hasta 10000 cuentas bancarias. Y no sólo en la informática comercial: las tabuladora fueron adaptadas a aplicaciones de cálculo, análisis de ondas y astronomía. En 1930 fueron utilizadas en la identificación del planeta Plutón.

LA INTRODUCCION DEL CALCULO ELECTROMAGNETICO

Va a ser un español, Leonardo Torres Quevedo, quien a principios del siglo XX, realice el salto histórico que supone la introducción de la tecnología eléctromagnética en la Informática. Ya a principios de siglo Valdenar Poulsen había sido pionero en la utilización de memorias magnéticas; Torres Quevedo va a preconizar el ordenador electromagnético.

Torres Quevedo, en su función de ingeniero, se familiariza con las plantas industriales de la Europa de principio de siglo; fué durante esta experiencia cuando comenzó a pensar en el aspecto *automatizable* del trabajo, más allá del aspecto

meramente mecánico, sino en el aspecto intelectual. Así, se esfuerza por discernir los trabajos en los que el pensamiento humano podría ser reemplazado por un automatismo. Los autómatas manejados por Torres Quevedo ya no eran meramente mecánicos, como los de la época de Descartes y Jacquard, sino de funcionamiento electromecánico.

Fruto de esta dedicación, inventa en 1911 el primer jugador de ajedrez verdaderamente autómata(no como el de Maelzel). Las piezas eran movidas por electroimanes situados debajo del tablero. Este jugador de ajedrez estaba programado para ganar partidas sencillas.

Pero su trabajo en el ámbito de electromecánica no se detuvo aquí. Dándose cuentas de las amplias posibilidades de ésta en el procesamiento matemático, en 1914 publica un trabajo en el que demostraba la viabilidad de los ingenios de Babbage mediante la electromecánica. En este mismo trabajo, propone una forma nueva de almacenar los números: la "aritmética de coma flotante". Este método es utilizado hoy en nuestros ordenadores de forma generalizada.

Pero Torres Quevedo no se contentó con sólo demostrar la viabilidad del proceso numérico electromecánico. Se dedicó a construir su propio calculador con sus conocimientos, y en 1920 la terminó. Utilizaba una máquina de escribir adaptada para la introducción de los números, y los resultados eran impresos automáticamente. Aunque no desarrolló más su diseño, llegó a prever la posibilidad de conectar varias terminales a la calculadora central.

BIOGRAFIA

Leonardo Torres Quevedo(Santa Cruz, Cantabria, 1852 - Madrid, 1936)

Cursó estudios en el Instituto de Bilbao y en la Facultad de Ingeniería de Madrid. Científico e ingeniero polivalente, además de sus aportaciones a la Informática, realizó investigaciones en campos muy diferentes. Suyo es el funicular de las cataratas del Niágara, que continúa en uso; suyo fué un diseño de dirigible cuya armadura reunía las propiedas de los dirigibles rígidos y flexibles; también realizó un prototipo de barco por control remoto mediante ondas hertzianas, que exhibió ante el rey Alfonso XIII en 1906.

Sus trabajos significaron un avance significativo, sobre todo en Informática, sirviendo de base a los trabajos posteriores, empezando por los de Hermann Hollerith.

En su longeva vida no hubo de faltar el reconocimiento a su labor; fué condecorado por la Academia Francesa de Ciencias y nombrado presidente de honor vitalicio de la Academia de Ciencias de Madrid.

BIBLIOGRAFIA

- 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.*

TECNICA

Telmo Lago

ANOMALIAS EN LOS ORDENADORES

Alguna vez nos tiene ocurrido que al acabar un programa no haga lo que deseemos. Es ovbio, habrá que depurarlo; pero esto a lo que me estoy refiriendo puede darse también después de el ya mencionado anteriormente proceso de depuración, y el programa estar perfectamente hecho. ¿Es que la culpa va a ser siempre del programador, que si es un inútil que no sabe hacer las cosas, y tal? Quiero romper una lanza en favor de los programadores z decir que la culpa también puede ser de causas ajenas totalmente a ellos.

Primer culpable: los virus. Hoy en dia esas horrendas criaturas informáticas (ya comentadas en un número anterior de esta revista) ya no son aquellas simples curiosidades que se limitaban a poner en el momento más inoportuno algún gracioso mensaje haciendo referencia a los que lo habían hecho, sino que en el colmo del refinamiento (en cuanto a las perradas, si se me permite la expresión) llegan a introducirse en el reloj del ordenador. Me explico: Por regla general, los ordenadores suelen traer un reloj interno alimentado con una batería de Ni-Cad (níquel y cadmio), del cual una serie de programas (generalmente todos los de gestión) extraen la fecha y hora actual. Esos datos de fecha y hora están almacenados en unos pocos bytes de la memoria del sistema, y en esa pequeña memoria reservada es el sitio idóneo para instalar alguno de esos virus. Efecto producido: el programa puede hacer las cosas más extrañas imaginables, cuando no destroza algún fichero (como mínimo empieya a pisar lo que hay en memoria.

Otras causas ya pueden ser atribuidas al propio hardware del sistema. Lo primero y más evidente: que el ordenador tenga algún componente dañado. Esto puede darse de muy diversas formas. Defectos de fabricación de algún componente. No es muy frecuente, ya que en el proceso de fabricación de un integrado se cuidan mucho las condiciones de higiene y los controles de calidad son bastante estrictos. De todas formas, un fallo de este tipo siempre suele aparecer al poco tiempo de trabajar con el equipo. Es una de las razones por las que siempre recomiendan hacer un uso exhaustivo del equipo cuando se acaba de comprar, aunque es frecuente que se de la famosa Ley de que se estropea nada más quedarse sin garantía.

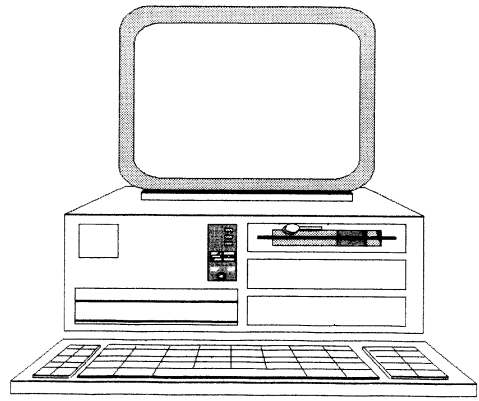
Agentes externos. (Nada de garrotazos a la placa). En este apartado se encuentran las vibraciones z cambios bruscos de temperatura. Unas vibraciones fuertes pueden llegar a provocar la mala conexión de componentes o conectores, dando lugar incluso a la inutilización total del equipo. Este tipo de perturbación suele eafectar sobre todo a los periféricos del tipo de almacenamiento magnético.

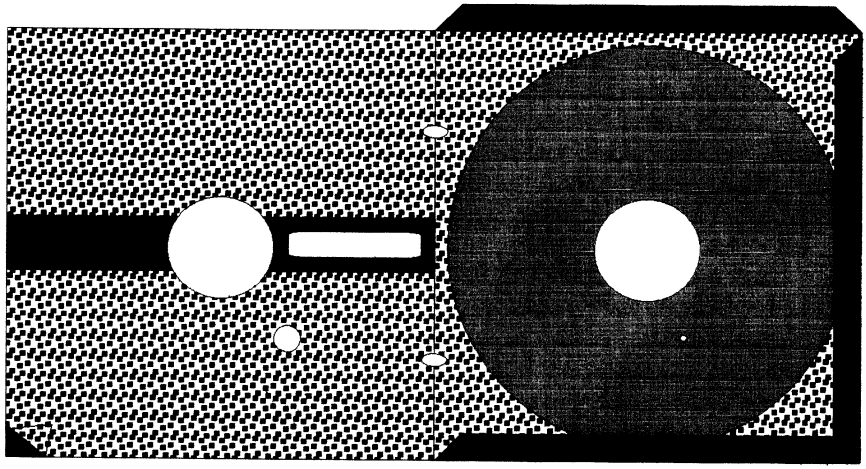
Las diferencias extremas de tempe-

ratura pueden provocar tensiones en las pistas del circuito (debido a dilataciones y contracciones) lo suficientemente grandes como para dañarlas con fisuras y/o cortes totales. Incluso en algunos ordenadores se da el caso de que tienen integrados altamente sensibles a los cambios de temperatura. Por ejemplo, el AMIGA de Commodore Business Machines, tiene un integrado (que curiosamente, es la base del sistema) que ¡¡¡LE PUEDE COGER EL FRIO!!!. Sí, sí. Has leído bien. Si sufre un bajón muy brusco de temperatura hay que levantarlo un poquito del zócalo para que vuelva a funcionar normalmente. Otro detalle: el chip ULA del viejo Sinclair Spectrum se podía arreglar METIENDOLO EN EL CONGELADOR (auténtico).

Otra cosa muy dada a amargarnos la vida son las fluctuaciones de corriente eléctrica. Son de diversa naturaleza. Por lo general aparecen cuando se encienden dispositivos periféricos. Por ello es muy recomendable que lo último en encender sea siempre la unidad central. En los casos de bajón brusco en la línea general también pueden conllevar alteración de la memoria, con el consiguiente desfase del programa, cuando no el borrado total.

Por último, hablar de un tipo de fallos de los que no tiene la culpa ni el hardware ni el programador. El influjo de los rayos alfa procedentes del espacio exterior pueden alterar el contenido de las memorias. No se sabe exactamente por qué les afectan. A este tipo de fenómeno se le denomina soft error. Estadísticamente es normal que en una máquina se produzca uno cada mil horas de trabajo, aproximadamente. Lo que suele ocurrir es que debido a la naturaleza de los rayos alfa (iones positivos de helio) se alteran cargas de los circuitos, cambiando algunos bits de 0 a 1 y viceversa, con el consiguiente caos.





ALGORITMOS

Jesús Cea

RUTINAS MATEMATICAS

PRIMERA PARTE

Si para algo se usan los ordenadores es, sin lugar a dudas, para el cálculo numérico. A pesar de que los precios del hardware caen día a día no todo el mundo puede permitirse el lujo de adquirir un coprocesador matemático. En esta serie voy a explicaros como emular por software las rutinas matemáticas más utilizadas.

Para empezar por algo "asequible" para todos, vamos a iniciar nuestro estudio repasando algunos algoritmos de tratamiento de números enteros. En el próximo número explicaré la creación de rutinas matemáticas para operar con números en coma flotante.

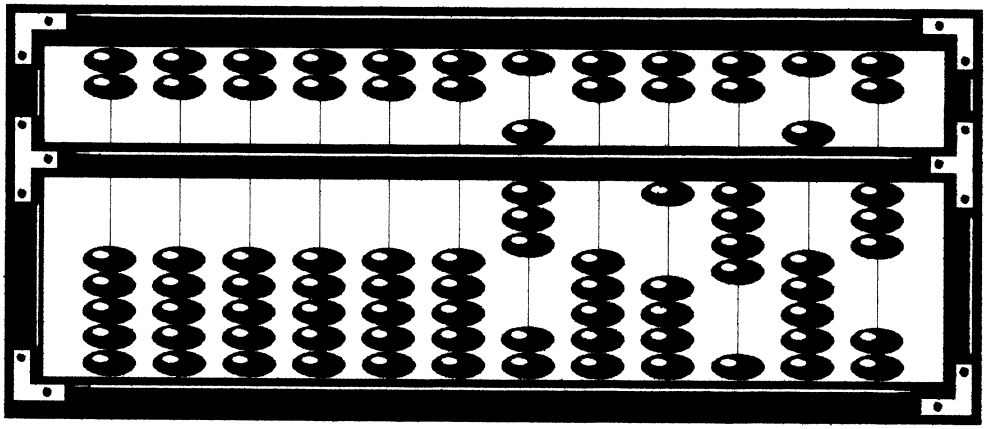
OPERACIONES CON ENTEROS

Como todos sabéis, los algoritmos de suma y resta entera son extremadamente sencillos. Además no debemos olvidar que hasta el procesador más arcaico dispone de instrucciones apropiadas para estos fines. Por todo ello, y dado que ya han sido tratados en el número 2 de DATA BUS, me limitaré a mostrar sus tablas lógicas (ver figura 1).

MULTIPLICACION Y DIVISION

Al igual que en el caso anterior, en el número 2 ya se dieron algoritmos para calcular productos y cocientes. Sin embargo, dichos métodos distan mucho de ser eficaces. ¿Alguien se imagina calcular 65535^2 siguiendo los pasos dictados por Eduardo Cunha? ¿Y que tal dividir 65535 entre 1? Evidentemente, todo es mejorable en este mundo (o casi) y estos algoritmos pueden optimizarse más que ningún otro. La verdad es que Eduardo ya se podría haber esmerado un poco más. Mejor que revisar dichos algoritmos, he optado por pensarme otros nuevos. Y la verdad es que tampoco son tan nuevos, como podréis deducir de lo que sigue.

Bien pensado los algoritmos explicados por Eduardo son (más o menos) los mismos que se explican a los alumnos de párvulos. Sin embargo, sabemos perfectamente que nadie opera así después de aprobar 3º de E.G.B. (Para quien no tenga el número 2, decir que los algoritmos de Eduardo consisten en sumar repetidas veces para multiplicar y restar sucesivamente para dividir. ¡ARGG!). En realidad, todos utilizamos un algoritmo mucho más inteligente, como se demuestra del hecho de no requerir un billón de años en hacer 9E300*2E1000 a mano (para los lectores que no estén famirializados con la notación científica, 9E300 significa $9*10^{300}$ o, lo que es lo mismo, un 9 seguido de 300 ceros).



La sorpresa surge al intentar convertir ese método de decimal a binario. En la figura 2 se muestra una multiplicación en binario. A la vista de dicha figura surge un algoritmo extraordinariamente sencillo y fantasticamente eficaz:

1. Se inicializa a cero el registro que contendrá el resultado final.

2. Se toma el multiplicador y se comprueba si es cero. En caso afirmativo ya terminamos.

3. Si no es cero, se realiza una operación *LSR* con el multiplicador (se divide por 2). El bit que sale se pasa al ***Carry*** (esto lo hace automaticamente cualquier microprocesador). El bit que entra se pone a cero (esto también es automático).

4. Si el ***Carry*** es cero (el multiplicador era par), saltamos al paso 6.

5. Si el ***Carry*** no es cero (el multiplicador era impar), sumamos el multiplicando al registro que contendrá el resultado.

6. Se multiplica el multiplicando por 2 (un simple *LSL*), y saltamos al paso 2.

El hecho de realizar desplazamientos lógicos (los *LSL* y *LSR*) se debe a que estamos trabajando en binario. Cuando trabajamos en decimal, si encontramos un cero multiplicamos todo por 10 (la base). Si se utiliza binario, al encontrar un cero se multiplica todo por 2. Ni que decir tiene que multiplicar (o dividir) por 2 es una de las operaciones incluidas en cualquier microprocesador que se precie de tal...

Como puede verse, este algoritmo es mucho mejor que el propuesto en el número 2, y es muy sencillo. Todo se basa en convertir el método utilizado normalmente por nosotros mismos a la base binaria.

En la figura 3 podéis ver una división en binario. Al igual que en el caso anterior todo se reduce a adaptar el método usual a la base 2. El algoritmo es el siguiente:

1. Comprueba si el divisor es cero. En ese caso se indica una "división por cero".

2. Se ponen a cero los registros que contendrán al cociente y al resto.

3. Se comprueba si el dividendo es menor que el divisor. En ese caso pasa el dividendo al registro de resto y regresa.

4. Hace *LSL* con el divisor hasta que sea mayor o igual que el dividendo. Se toma nota del número de desplazamientos efectuados.

5. Hace un *LSL* con el registro de cociente.

6. Si el dividendo es mayor o igual que el divisor, suma uno al registro que guarda el cociente y resta el divisor al dividendo.

7. Se efectua un *LSR* al divisor y se decrementa el contador inicializado en el paso 4. Si se acabaron los desplazamientos, pasa el dividendo al registro de resto y acaba.

8. Salta al paso 5.

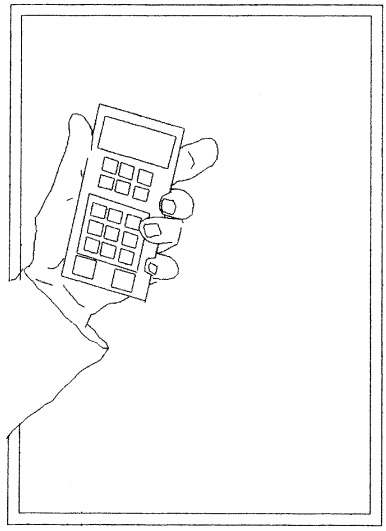
Este método parece más complicado que el de multiplicación, pero es que también es más difícil (al menos para mí) dividir que multiplicar... Podemos ahorranos el paso 4 si multiplicamos directamente por 2^n y asignamos el valor n al contador de desplazamientos. El número n corresponde al tamaño máximo del dividendo en bits. Valores usuales son 16 y 32.

Si queremos operar con signo es bien sencillo. Si tomamos *0* como positivo y *1* como negativo, basta efectuar un *XOR* para obtener el signo correcto tanto en los productos como en las divisiones.

En realidad desarrollé estos algoritmos cuando trabajaba con los "viejos" 6502 y 8502. Los procesadores de 16 bits actuales (68000 por ejemplo) incorporan instrucciones para multiplicar y dividir números enteros con o sin signo, por lo que estos sistemas pueden parecer reliquias de museo. Sin embargo, nunca se sabe si van a hacer falta...

Veamos ahora otro caso. Supongamos que estamos trabajando con un procesador capaz de multiplicar directamente dos números de 16 bits, dando un resultado de 32 bits (sean A y B el número de bits del multiplicando y del multiplicador. El número MAXIMO de bits del producto es A+B). ¿Qué pasa si queremos multiplicar dos números de 32 bits cada uno?

Podemos considerar ambos número divididos en dos mitades de 16 bits cada una. Sea A el primer número y B el segundo. Entonces:



$A=2^{16}*a_{hi}+a_{lo}$.

$B=2^{16}*b_{hi}+b_{lo}$.

$A*B=(2^{16}*a_{hi}+a_{lo})*(2^{16}*b_{hi}+b_{lo})$.

Operando convenientemente se obtiene:

$A*B=2^{32}*a_{hi}*b_{hi}+2^{16}*(a_{hi}*b_{lo}+a_{lo}*b_{hi})+a_{lo}*b_{lo}$.

Como puede verse, todo se reduce ahora a productos 16*16 bits calculables con las instrucciones estándar y a desplazamientos lógicos. Este método se puede generalizar a 48 o 64 bits sin más que añadir términos a las ecuaciones. El inverso también es posible (utilizar menos bits). En el caso extremo, si suponemos un procesador capaz de multiplicar dos números de un sólo bit (un simple *XOR*) y aplicamos la generalización explicada, obtenemos milagrosamente el algoritmo expuesto. Que no se diga...

Evidentemente existen más métodos para multiplicar/dividir. El más común es el de las tablas. Supongamos una tabla de 256 bytes que da el resultado de multiplicar dos números cualesquiera de 4 bits cada uno. Si aplicamos la generalización podremos multiplicar rapidamente números de

¡TE HA LLEGADO EL TURNO!

PARTICIPA TU TAMBIEN EN NUESTRO CONCURSO "LA GUERRA NUCLEAR" Y GANA UNO DE ESTOS ESTUPENDOS PREMIOS:

1er PREMIO: RADIO CASSETTE FM DIGITAL AKAI Y CINTAS DE AUDIO SONY

2ndo PREMIO: CONSOLA DE VIDEOJUEGOS PORTATIL NINTENDO GAMEBOY

3er PREMIO: WALKMAN SONY MEGABASS Y CINTAS DE AUDIO SONY

4rto PREMIO: AGENDA ELECTRONICO CASIO

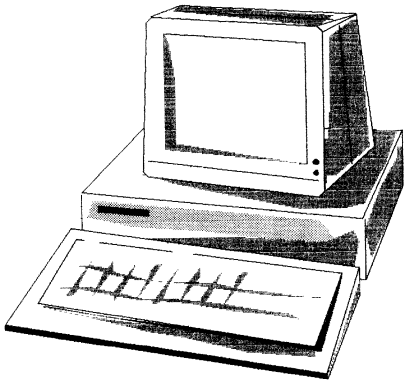
5nto PREMIO: LOTE DE CINTAS DE CROMO

PARA MAS INFORMACION, SOLICITA LAS BASES DEL CONCURSO Y EL PROGRAMA A AJIV.

ASOCIACION JUVENIL PARA LA INFORMATICA VIGUESA

AJIV

cualquier longitud. Se pueden crear tablas de productos de 8 bits, pero ocuparía 128Kbytes. Una salida elegante (y muy rápida) sería una tabla de 8*4 bits (8Kbytes), por ejemplo. ¿Por qué utilizar tablas si tenemos ya instrucciones especializadas? La respuesta es que dichas instrucciones suelen ser más lentas que buscar simplemente



el resultado en una lista. Otra ventaja de las tablas es que las podemos crear "a medida". ¿Qué tal una tabla con los primeros 256 multiplos de 40, o una lista con todos los desplazamientos necesarios para unas rutinas gráficas? De todos modos, cuidado con todo esto. Os aseguro que el empleo de tablas puede llegar a ser un autentico vicio. En el caso que dije antes de la tabla con los múltiplos de 40, sale más "barato" (en velocidad y en memoria) realizar una multiplicación por 8, guardarlo, multiplicar por 4 y sumar todo (8*4+8=40). Las multiplicaciones (divisiones) por potencias de 2 son facilmente calculables utilizando desplazamientos lógicos y deberíais utilizarlas siempre que fuera posible.

LA RAIZ CUADRADA

Al contrario que los casos anteriores, no conozco ningún procesador que disponga de instrucciones directas para el cálculo de raices cuadradas. Debido a ello, no tenemos otra salida que diseñar y programar nuestras propias rutinas para tal fin. Veamos algunos algoritmos para hacerlo:

Rutina 1:

Se basa en que las funciones n^2 y $n^{1/2}$ son ambas funciones monótonas, y en el método de dicotomía. Para simplificar, supongamos que estamos trabajando con números de 32 bits. Veamos ahora la descripción algorítmica del método:

1. Como trabajamos con números de 32 bits, la raiz cuadrada de cualquier número tendrá un máximo de 32/2=16 bits. Inicializamos un registro (registro de offset) con el valor $2^{16}/2=2^{16-1}=2^{15}$. Copiamos el valor de ese registro en el registro que contendra el resultado.

2. Hacemos *LSR* con el registro de offset. Si dicho registro es cero, acaba.

3. Calculamos el cuadrado del valor contenido en el registro del resultado y lo comparamos con el número cuya raiz queremos hallar.

4. Si coinciden, ya acabamos.

5. Si el cuadrado que calculamos es mayor que el número original, resta el registro de offset al registro del resultado. Pasa al punto 2.

6. Suma el registro de offset al registro del resultado y salta al paso 2.

Con este método se precisan un máximo de 32/2=16 iteraciones para obtener la mejor aproximación ENTERA posible de la raiz cuadrada de cualquier número. Sin embargo se podrían realizar hasta 16 productos 16*16 bits (aunque luego doy un método distinto para calcular cuadrados), algo muy lento incluso aunque nuestro procesador disponga de instrucciones apropiadas para ello. Seguidamente voy a explicar un sistema alternativo, aunque su verdadera utilidad se da al trabajar en coma flotante. En el caso entero incluso es mejor el método que acabo de explicar. Veámoslo, de todos modos:

Rutina 2:

1. Introducimos en el registro del resultado el valor n/2, donde n es el número del que queremos obtener su raiz cuadrada.

2. Copiamos el valor del registro de resultado en el registro de redondeo.

3. Asignamos al registro del resultado el valor (x+n/x)/2, donde x era su antiguo valor.

4. Comparamos el registro de resultado y el de redondeo. Si son iguales, acaba.

5. Salta al paso 2.

En realidad este método tiene una convergencia rápida pero difícil. En la mayoría de los casos se inicia un bucle infinito debido a una alternancia en el valor del último bit del resultado provocada al calcular la raiz de un número que no sea cuadrado perfekto. Para evitar esto, bastaría ejecutar el algoritmo K veces, tras las cuales suponemos que ya alcanzamos el valor final. Este sistema tendría el inconveniente de que frenaría muchísimo la rutina en los casos de convergencia. Otro sistema sería el comparar los registros de resultado y de redondeo sin tener en cuenta el último bit, pero ello acarrearía pequeñas imprecisiones en los casos de convergencia lenta. En realidad todo esto es debido a que queremos aproximar una raiz no entera con valores enteros. Como ya dije, este método se muestra especialmente interesante cuando se aplica a números en coma flotante.

Rutina 3:

Los dos algoritmos explicados requieren de operaciones "lentas" como son los productos y las divisiones. El método que sigue se inspira en la misma filosofía de todo el artículo: Pasar los métodos que utilizamos normalmente de decimal a la base binaria. Como ocurrió en los casos del producto y el cociente, al trabajar en binario se simplifican notablemente las operaciones a realizar. En la figura 4 se muestra el cálculo de una raiz cuadrada en binario. Estudiemos ahora los pasos a seguir:

1. Se inicializa un registro (registro de desplazamientos) con el valor n/2, donde n es el número de bits del radicando. En el caso de que n sea impar, se incrementa en uno (n siempre debe ser par).

2. Se inicializan con el valor cero el registro de resultado y el registro de resto.

3. Se comprueba si el registro de desplazamientos es cero. En caso afirmativo, regresa.

4. Se decrementa en uno el registro de desplazamientos y se realizan dos *LRL* con el registro del radicando. Los bits que salen por la izquierda del radicando se introducen por la derecha del registro de resto.

5. Se calcula a parte el valor 4*resultado+1. Ese valor se compara con el registro de resto.

6. Si el resto es mayor o igual que dicho valor intermedio, se resta dicho valor al resto. En el resultado se pone 2*resultado+1. Pasa al apartado 3.

7. Se pone en el resultado el valor 2*resultado y se salta al paso 3.

Resulta evidente que esta rutina es muchísimo más rápida que las anteriores, algo bastante necesario. No se requieren operaciones lentas, sino que sólo se ejecutan operaciones simples y muy rápidas tales como desplazamientos lógicos, sumas, restas, decrementos, etc. Una joya, vamos.

Tenía pensado explicaros este mes como manejar la aritmética en coma flotante, pero por razones de espacio y tiempo no ha podido ser. De todos modos me comprometo a hablaros de todo esto en el próximo número.

En el número anterior aparecieron numerosas erratas en el artículo de algoritmos (y en los otros también). Espero que nadie se haya apresurado a decir que mis fórmulas estaban equivocadas...
