Introducción práctica a un poderoso computador

Libro 1

JUNIOR COMPUTER

INGELEK, S. A.

Junior Computer
Libro 1

Introducción práctica a un poderoso ordenador monoplaca ampliable

A. Nachtmann
G. H. Nachbar

INGELEK, S. A.
Ginzo de Limia, 48
MADRID-29

Ingelek, S. A. Primera edición, 1980.

Depósito legal: M-38867-1980.
ISBN: 84-85831-02-0.
Imprime Gráficas ELICA. Bóyer, 5. Madrid-32.

Prólogo

Con este libro queremos demostrar que cualquiera puede construir y utilizar un computador.
El Junior Computer es un computador monoplaca, que ha sido diseñado pensando en que fuera sencillo, barato y con grandes posibilidades de programación. Como CPU (Unidad Central de Proceso) se ha utilizado el microprocesador 6502, que goza de merecida fama entre los expertos en microordenadores. Por otra parte, el sistema básico descrito en este libro, tiene muchas posibilidades de ampliación.
Este primer libro dedicado al Junior Computer consta de cinco capítulos.
El capítulo 1 está dedicado a la descripción del Junior Computer. Asimismo, este capítulo es una introducción general a los ordenadores.
El capítulo 2 explica las reglas y procesos utilizados por el Junior Computer para realizar operaciones aritméticas y lógicas.
El capítulo 3 constituye una breve introducción a los organigramas que son una descripción gráfica de las operaciones de un programa.
El capítulo 4 nos descubre prácticamente los principios de la programación.
Finalmente, en el capítulo 5 proponemos una serie de programas tipo que preparan el camino hacia el segundo libro dedicado al Junior Computer.
Bueno, esperamos que con todo lo que acabamos de decir hayamos despertado su curiosidad y que la lectura de este libro resulte de su agrado.

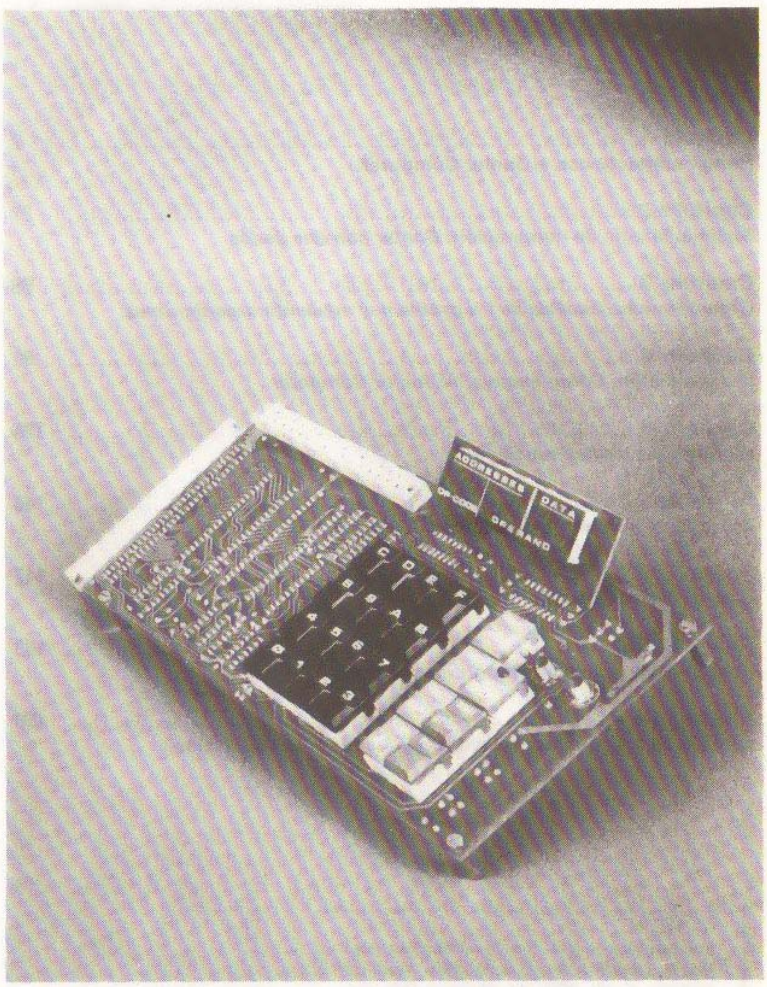
Los autores

El Junior Computer, libro 2, es la continuación de este libro y está dedicado a los siguientes temas:

- La Unidad de Entrada/Salida y programación.
- El programa monitor y todas sus posibilidades.
- El editor hexadecimal.
- El ensamblador hexadecimal.
- Listado del programa monitor.

Los circuitos impresos mencionados en este libro pueden obtenerse a través del servicio de EPS de la revista Elektor o por medio de los establecimientos de electrónica distribuidores de la revista.

Indice


ADDRESSES
DATA
OP-CODE
OPERAND

1

Primer contacto con el Junior Computer

El término Junior parece implicar que este computador está pensado sólo para niños o aficionados. Nada más lejos de la realidad. Al diseñarlo, el Departamento Ténico de ELEKTOR, ha intentado que fuera barato, fácil de construir y que, sin embargo, tuviera la capacidad de los grandes sistemas.
Así, pues, aunque «pequeño de talla», el Junior Computer y su juego de instrucciones, forman un potente sistema de programación, ideal para profesionales y aficionados; de hecho, el 6502 es la CPU de muchos microordenadores profesionales. Por supuesto, el usuario puede ampliar la unidad de base, añadiendo las tarjetas de ampliación (RAM, ROM, teclado ASCII, vídeo, etc.) que sucesivamente iremos publicando.

Mucha gente piensa en los microprocesadores como aparatos de gran complejidad y creen que su construcción y manejo es materia para especialistas en electrónica; sin embargo, nosotros no opinamos así y para demostrarlo hemos diseñado el Junior Computer.
En principio, un microcomputador es algo muy simple. Para su construcción sólo son necesarios algunos conocimientos básicos de electrónica. Simplemente se trata de poner las «cajitas negras» en su emplazamiento correcto. ¿Quién no ha hecho alguna vez un rompecabezas?
Uno de los aspectos más importantes de cualquier sistema electrónico es saber «qué se puede hacer», más que «cómo lo hace». En el caso de un microcomputador, el juego de instrucciones será el que nos informe de sus posibilidades.
El juego de instrucciones es el conjunto de sentencias y direcciones (dadas por el programador) que el microprocesador puede «entender». Por tanto,

aquí el problema fundamental no es comprender el funcionamiento de los sistemas electrónicos puestos en juego, sino aprender a usar el juego de instrucciones, para comunicarse con el microprocesador y decirle lo que queremos que haga. Es como llevar un coche; no es imprescindible saber repararlo para poder conducirlo. La cuestión esencial aquí es cómo decirle al computador qué es lo que deseamos hacer (en lenguaje comprensible para él) e interpretar su «respuesta».

Cómo funciona un microcomputador

Aunque acabamos de decir que no es necesario conocer el funcionamiento interno de un microprocesador para poder utilizarlo, daremos una breve descripción de sus distintas secciones, lo cual nos ayudará a comprender éste capítulo y los siguientes.
Como puede verse en el diagrama de bloques de la figura 1, un microcomputador se divide en tres secciones básicas:

- La unidad central de proceso (CPU).
- Sección de entrada-salida (I/O).
- Sección de memoria.

La información circula entre estos tres bloques mediante las líneas de información llamadas *buses: bus de direcciones, bus de datos* y *bus de control.* Un bus es, sencillamente, una línea (conductora) o un conjunto de ellas que interconecta a dos o más secciones.

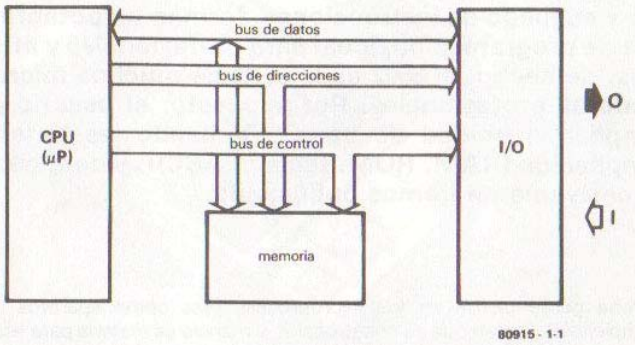


Figura 1. El diagrama básico de un computador se compone de tres bloques y tres buses. Estos últimos interconectan los bloques entre sí.

Un microporcesador sólo trabaja con información o datos que tengan forma comprensible para él (concretamente pulsos digitales). Como su nombre indica, el bus de datos lleva esta información de una a otra sección (o bloque) del computador. Normalmente el *bus de datos* se compone de ocho conductores, y, por tanto, es capaz de transmitir ocho *bits* al mismo tiempo. El *bit* es la unidad de información binaria (en inglés, BInary digiT). Normalmente, a los grupos de ocho bits se les conoce con el nombre de *byte.*

La CPU (Central Processing Unit = Unidad Central de Proceso) es el «cerebro» de un ordenador. Su función consiste en controlar el funcionamiento de todos los restantes elementos del ordenador y en procesar (realizar operaciones) con los datos.
Todo computador (por perfecto que sea) es un «artefacto» inútil a menos que disponga de un medio para comunicarse con el exterior. Aquí es donde entra en escena la unidad de entrada-salida (I/O = Input/Output). Para que el operador pueda entender lo que «dice» el microprocesador y viceversa se hace necesario algún sistema que traduzca los mensajes en ambos sentidos. Este sistema es comúnmente un teclado y un terminal de vídeo (pantalla de televisión) o alguna forma de visualizador (display). Naturalmente, esto no quiere decir que la comunicación se limite a estos dos sistemas.
La *memoria* es simplemente un almacén que el computador utiliza para guardar (o extraer) los datos e instrucciones necesarios para realizar un determinado trabajo (programa). Esto no quiere decir que los computadores sean «inteligentes» (al menos todavía no, que nosotros sepamos). Con un computador hay que ser perfecta y estrictamente exactos (no supone nada por sí solo) al indicarle lo que queremos que haga y en qué orden. Los datos se almacenan en compartimentos individuales de la memoria o *posiciones*. Cada una de estas posiciones posee su propia (y única) *dirección*. Por medio del *bus de direcciones,* el microprocesador puede seleccionar cualquier posición de memoria, y, por tanto, el dato requerido. Este bus también se utiliza para acceder a la entrada o salida correspondiente (I/O), según indique el programa.
Finalmente, pero no por ello menos importante, tenemos el *bus de control.* Este permite a la CPU regular las distintas funciones internas, como por ejemplo, indicar al bus de datos la forma en que debe ser transferida una información (es decir, si hacia la CPU, al exterior o a la memoria).

Organización interna del Junior Computer

Una vez visto el diagrama de bloques «básico» de un microcomputador, estamos en condiciones de abordar el diagrama de bloques del Junior Computer (fig. 2). Veamos, primeramente, los tres buses.

Buses

El bus de direcciones está formado por dieciséis líneas y es independiente de los otros dos buses (el de datos y el de control). Mediante estas dieciséis líneas, la CPU es capaz de direccionar hasta 2^{16} (ó 65.536) posiciones distintas de memoria. De este modo, las 65.536 posiciones (ó 64 k.) de memoria están a disposición del microprocesador (suponiendo, claro está, que se haya ampliado la memoria de base, que es de 1 k RAM).
El bus de datos se compone de ocho líneas, pero a diferencia del anterior, éste es bidireccional, esto quiere decir que los datos pueden ser transferidos desde o hacia el microprocesador. Por supuesto, los datos que circulan por este bus sólo pueden hacerlo en una dirección al mismo tiempo. La dirección del dato a transferir es determinada por el bus de control. Si el computador se dispone a *leer* (read) un dato, el bus de control debe permitir que los datos sean transferidos desde la memoria (o cualquier otra fuente) hacia la CPU. Contrariamente, cuando el computador *escribe* (write), el bus de control per-

mite que la información circule desde la CPU a la memoria (o cualquier otro destino).
El bus de control realiza esta función por medio de «buffers» (circuitos amplificadores/separadores de ganancia unidad) bidireccionales que permiten el paso de los datos en un sentido u otro, dependiendo de las señales del bus de control.

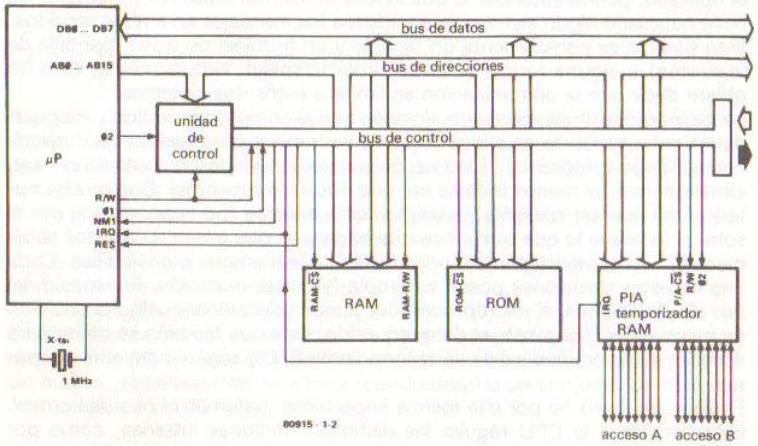


Figura 2. Versión detallada del diagrama de la figura 1 aplicado específicamente al Junior Computer.

La memoria

Como se indica en el diagrama de la figura 2, existen dos tipos de memorias principalmente: la RAM y la ROM. Hay, por otra parte, dos tipos de informaciones o datos:
— Permanentes (como el programa *monitor* almacenado en la ROM).
— Temporales (como la mayoría de los programas del *usuario*, que se almacenan, normalmente, en la RAM).
Los datos permanentes, únicamente podrán ser leídos; en cambio, los datos temporales pueden escribirse y leerse.
Cuando se habla de «leer» y «escribir» en la memoria se hace referencia al acto de «ver» los datos que contiene o «introducir» otros nuevos (auque pueden ser los mismos), respectivamente. La memoria permanente o de Sólo Lectura se llama ROM (Read Only Memory = Memoria de Sólo Lectura) y es invariablemente donde se memoriza el programa monitor. La memoria que admite «lectura» y «escritura» se conoce como RAM (Random Access Memory = Memoria de Acceso Aleatorio). Una RAM puede almacenar datos, resultados intermedios y programas en funcionamiento (es decir, el programa que se está ejecutando). Es por ello que a las memorias RAM se les llama a veces *memoria de trabajo*. La señal que controla el bus (bidireccional) de datos mencionado anteriormente, informa también a la *memoria* cuando se van a «leer» o «escribir» datos, de ahí el término Read/$\overline{\text{Write}}$.

Conviene resaltar el hecho de que la información contenida en la memoria puede leerse tantas veces como sea preciso, ya que el proceso de lectura no borra la información existente. Esto es lógico, ya que el hecho de leer estas páginas no varía su contenido. La información leída es enviada al «cerebro» (CPU) del ordenador para, a continuación, ser procesada. Las memorias RAM tienen un pequeño «defecto», por decirlo de alguna manera: la información que contiene sólo se mantendrá si ésta se mantiene conectada continuamente a la tensión de alimentación, de forma que, si sufre un corte de corriente, toda la información en ella contenida se borrará automáticamente. Cuando esta información quiere conservarse por más largos períodos de tiempo se deberá emplear otro sistema de almacenamiento; por ejemplo, cinta de cassette o floppy disc (disco magnético), que en este caso es más conveniente y (sobre todo) seguro, que «dejarlo» en una RAM, ya que si falta la corriente, aunque sea por un instante, perderemos la información almacenada.

Sección de entrada-salida (I/O)

El bloque marcado como I/O es el encargado de relacionar al computador con el mundo exterior. En la figura 2 se le ha representado con las siglas PIA (Peripheral Interface Adapter = Etapa Adaptadora de Periféricos).
Al igual que en las memorias RAM, necesariamente, ésta es una sección en la cual los datos circulan en forma bidireccional. La PIA tiene dos marcos (puerta A y puerta B) de entrada-salida, con ocho líneas cada uno. La puerta (A ó B) que se activa (para leer o escribir) es determinada por el bus de direcciones. Cada línea puede trabajar como entrada o salida, independientemente de las otras 15 y en el momento en que lo deseemos.
Una cierta cantidad de datos pueden ser memorizados en la PIA por cortos períodos de tiempo (nótese que las siglas RAM figuran también en el bloque de la PIA). Esto es útil cuando la CPU tiene que realizar alguna operación a la vez que está transfiriendo datos por una de las puertas. La información, tanto de entrada como de salida puede ser almacenada en esta memoria, pero sólo en un solo sentido cada vez (no pueden almacenarse simultáneamente datos de entrada y de salida).
El bus de direcciones informa a la PIA sobre la puerta por donde circulará el dato que queremos transferir y de si debe seguir o no la transferencia de datos.
Hay tres buses más que van al exterior (las tres flechas que apuntan hacia la derecha en la figura 2), pero estos son para una futura ampliación más que para la comunicación con el exterior. Por lo que al JC se refiere, el mundo exterior es todo aquello que está más allá del teclado y del visualizador.

La CPU: Centro de la actividad

Como ya hemos dicho antes, la Unidad Central de Proceso (CPU) es el «cerebro» de todo sistema microcomputador, y sus funciones son: controlar a las demás unidades y procesar los datos. La CPU posee un cierto número de posiciones de memoria (registros) en las que se almacenan temporalmente datos, direcciones e instrucciones para su decodificación y/o manipulación. También contiene un *contador de programa,* que sencillamente cuenta los «pasos» del programa; su salida puede conectarse al bus de direcciones para tener acceso a la memoria y poder extraer así las instrucciones del programa. Para determinar la siguiente dirección a la que deberá dirigirse el programa, la CPU analiza conjuntamente la última instrucción ejecutada y el contador de

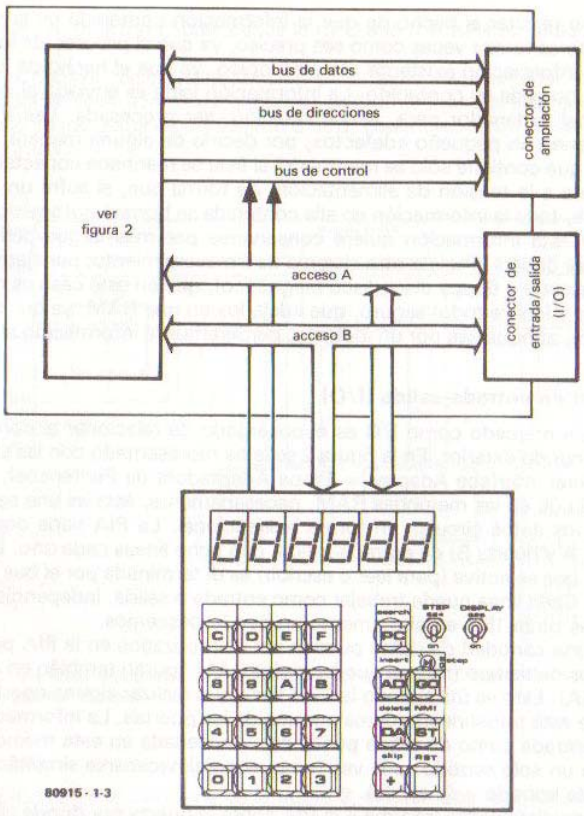


Figura 3. Detalle del diagrama de la figura 2: la comunicación con el exterior es posible gracias al teclado (entrada = I en I/O) y al visualizador (salida = = O en I/O).

programa. El procedimiento exacto de cómo se realiza esta operación es obvio que sale fuera del marco de este capítulo.

El microprocesador tiene una especie de «marcapasos» o *reloj* interno que se utiliza para generar los pulsos de tiempo, alrededor de los cuales se efectúan todas las operaciones. Este generador de reloj produce dos señales que llamamos Ø1 y Ø2, defasadas 180 °entre sí. Sin ellas el sistema sería totalmente inútil.

Existen otras tres señales que se muestran en la figura 2 con las siglas RES, IRQ y NMI. RES es la señal de reposición y no necesita mayor explicación. Esta señal indica al JC que se ponga en la condición de comienzo. Las otras dos señales, IRQ (Interrupt Re-Quest = Solicitud de Interrupción) y NMI

(Non-Maskable Interrupt = Interrupción No Evitable) se usan para hacer modificaciones en un programa o «pasarlo» por partes mientras se ejecuta. En este caso, el computador debe recibir información exterior que le indique la siguiente sentencia. Esto puede ser muy útil, sobre todo cuando se usa el microprocesador con periféricos lentos (como el hombre, por ejemplo). Téngase en cuenta que el microprocesador puede realizar alrededor de medio millón de operaciones por segundo, mientras que una persona en el mismo período de tiempo solamente realizará 3 ó 4. Una vez concluida la interrupción, el computador continuará con el programa principal desde donde se produjo la «parada». En el caso de que se utilicen ambas funciones al mismo tiempo (IRQ, NMI), tendrá prioridad la que indique el programa. Nótese que la sentencia IRQ puede ser controlada por el programa, mientras que NMI, como su nombre indica, no.
En el bloque de la PIA (fig. 2) podemos ver también un temporizador programable, al cual se le prestará una mayor atención en el capítulo 5 del segundo libro.

Periféricos

El sistema se completa con el teclado y el visualizador (figura 3). El teclado se compone de 23 teclas y dos interruptores, de las cuales 16 teclas se utilizan para introducir información (en código hexadecimal) en el microprocesador; el resto del teclado tiene diferentes funciones de control. El visualizador se compone de seis unidades LED, de siete segmentos y es el encargado de mostrar al exterior los datos, direcciones, etc. (en código hexadecimal igualmente).
El teclado y el visualizador se conectan al computador a través de los accesos A y B. El acceso A es bidireccional, mientras que el B es unidireccional. Hay dos señales que se conectan directamente al teclado y al bus de control: RES y NMI. Estas proceden de las teclas RST y ST, respectivamente (hablaremos más a cerca de esto cuando se vean sus funciones). Las dieciséis líneas de los accesos A y B se llevan a un conector que tiene 31 contactos para permitir futuras ampliaciones.
Se puede acceder a los buses de direcciones, datos y control, a través de un conector de 64 contactos. La razón de estos conectores es simple; un teclado hexadecimal y un visualizador LED de siete segmentos es el sistema más elemental y barato para comunicarse con el microprocesador.
Hay muchas otras formas de I/O, mejores y más complejas que la actual, pero también son más caras. La mayoría de los aficionados no pueden comenzar haciendo el desembolso que supone una unidad de I/O más sofisticada. Los aficionados, generalmente, prefieren practicar hasta adquirir suficiente experiencia, por lo que el JC ha sido diseñado para ayudar en este aprendizaje, hasta convertirse en un «experto» en computadores.
El conector de 64 contactos se utilizará para una posterior ampliación de la memoria, de modo que permita introducir programas más largos y complejos. Esto se hace prácticamente necesario si queremos ampliar la «potencia» de nuestro computador (a usted le parecerá siempre que la memoria es insuficiente).
Ahora, echemos una mirada algo más detallada a la circuitería del JC. Puede decirse que el *hardware* es como el «cuerpo» del computador, y los programas (*software*) su «personalidad».

Circuito del Junior Computer

El diagrama del circuito se presenta en la figura 4. La CPU es el microprocesador 6502. A los lectores poco familiarizados con los diversos tipos de microprocesadores podemos asegurarles que el 6502 es un microprocesador rápido y de alta calidad, que tiene un «poderoso» juego de instrucciones con una gran variedad de posibilidades de programación.
Todo microprocesador necesita de algún dispositivo para hacer fluir la sangre por sus venas: es decir, necesita un generador de reloj. En este caso está formado por N1, R1, C1, D1 y un cristal de 1 MHZ. El reloj genera dos señales (Ø1 y Ø2); una, para el bus de direcciones, y otra, para el de datos. El bus de direcciones comprende las líneas AØ...A5, y el de datos las líneas DØ...D7.
Las señales eléctricas que circulan por estas líneas están codificadas como información digital, ¿y cuál es este código? Imaginemos un sistema numérico con dos dígitos únicamente: el Ø* y el 1 (en contraposición al decimal: Ø...9). Los números cero y uno se representan por Ø y 1 (no cambian), pero el número 2 lo representaremos por 1Ø. El tres es 11, el cuatro es 1ØØ, etc. A continuación, se muestra la tabla de equivalencia para los números del Ø al 15.

número decimal	equivalente binario
Ø	ØØØØ
1	ØØØ1
2	ØØ1Ø
3	ØØ11
4	Ø1ØØ
5	Ø1Ø1
6	Ø11Ø
7	Ø111
8	1ØØØ
9	1ØØ1
1Ø	1Ø1Ø
11	1Ø11
12	11ØØ
13	11Ø1
14	111Ø
15	1111

Obsérvese, que con cuatro dígitos sólo son posibles 16 combinaciones, mientras que en base decimal se puede contar hasta diez mil (cero a 9.999). Como se dijo anteriormente, el bus de direcciones posee 16 líneas o lo que es lo mismo, 2^{16} combinaciones diferentes. Por tanto, la «limitación» impuesta al trabajar en sistema binario no representa ningún problema.

* Nota: Se utiliza la notación Ø para el número cero, con el fin de poder distinguirlo de la letra O.

Organización de la memoria

La organización de la memoria del Junior Computer se irá comprendiendo a medida que vayamos viendo el papel que desempeña la memoria dentro de un computador.

En la figura 4 se presenta la memoria, que se compone de dos bloques de RAM y uno de EPROM (Erasable Programmable Read Only Memory = ROM Reprogramable). La EPROM (IC2) se usa para memorizar datos de modo permanente (ejemplo: el programa monitor) y las RAMs (IC4 e IC5) como memoria de trabajo. La transferencia de datos se realiza en bloques de ocho bits (un byte) simultáneamente. La EPROM contiene 1.024 de estos grupos de ocho bits; en terminología de computadores, esto se conoce como 1 kbyte de EPROM (k, significa kilo o mil). Los diferentes bytes son accesibles mediante diez líneas de direccionamiento (conectadas al chip), con lo que en total tenemos 2^{10} = 1.024 direcciones distintas. ¡Justo el mismo número que el de la capacidad de la memoria.

Las RAMs tienen una capacidad de 1.024 «medios bytes», pero como hay dos, la capacidad completa es de 1.024 bytes «completos» (igual que la EPROM). Las líneas del bus de datos están organizadas de forma que la primera mitad del dato (4 bits) se encuentra en IC4 y la segunda mitad en IC5.

La memoria también recibe señales del bus de control, por ejemplo, la señal de selección. La EPROM y la RAM están conectadas a las mismas líneas de dirección, por lo que cabe preguntarse, ¿cómo podemos seleccionar la memoria correcta en el momento adecuado? Ambos «chips» de memoria poseen una entrada de selección de «chip» ($\overline{CS}$). Cuando esta entrada está a nivel alto (+ 5 vol. = 1), la memoria es inaccesible, contrariamente si está a nivel cero (masa = 0), la memoria se hará accesible. La señal $\overline{CS}$ es generada por el decodificador de direcciones IC6 (línea K7 para la EPROM y KØ para las RAMs).

La memoria sólo utiliza diez líneas de dirección, quedando seis libres, que hacen un total de 2^6 = 64 direcciones. Esto nos permite utilizar estas seis líneas de dirección como líneas de selección de «chips», y, por tanto, en futuras ampliaciones podremos seleccionar hasta 64 bloques de 1 k-byte. Luego, con seis líneas de direccionamiento (o selección) de «chip» y las 10 líneas del bus de direcciones, podremos acceder (en bloques de 1 k = 2^{10} = 1.024) a toda la memoria. En la versión «base» del JC, el decodificador de direcciones sólo trabaja con los ocho primeros bloques de memoria (de KØ a K7), para lo cual, tres líneas (AØ...A12) del bus de direcciones alimentan al decodificador, que se encarga de obtener las ocho líneas de salida necesarias (una para cada bloque).

La tabla que figura a continuación nos lo aclarará mejor.

A15 . . . A13	A12	A11	AØ	A9 . . . AØ	salida activa	bloques de memoria
X	Ø	Ø	Ø	X	KØ	1 k RAM (IC4, IC5)
X	Ø	Ø	1	X	K1	1 k de RAM, ROM externa
X	Ø	1	1	X	K3	1 k de RAM, ROM externa
X	1	Ø	Ø	X	K4	1 k de RAM, ROM externa
X	1	Ø	1	X	K5	1 k de RAM, ROM externa
X	1	1	Ø	X	K6	RAM en la PIA (IC3)
X	1	1	1	X	K7	1 k EPROM (IC2)

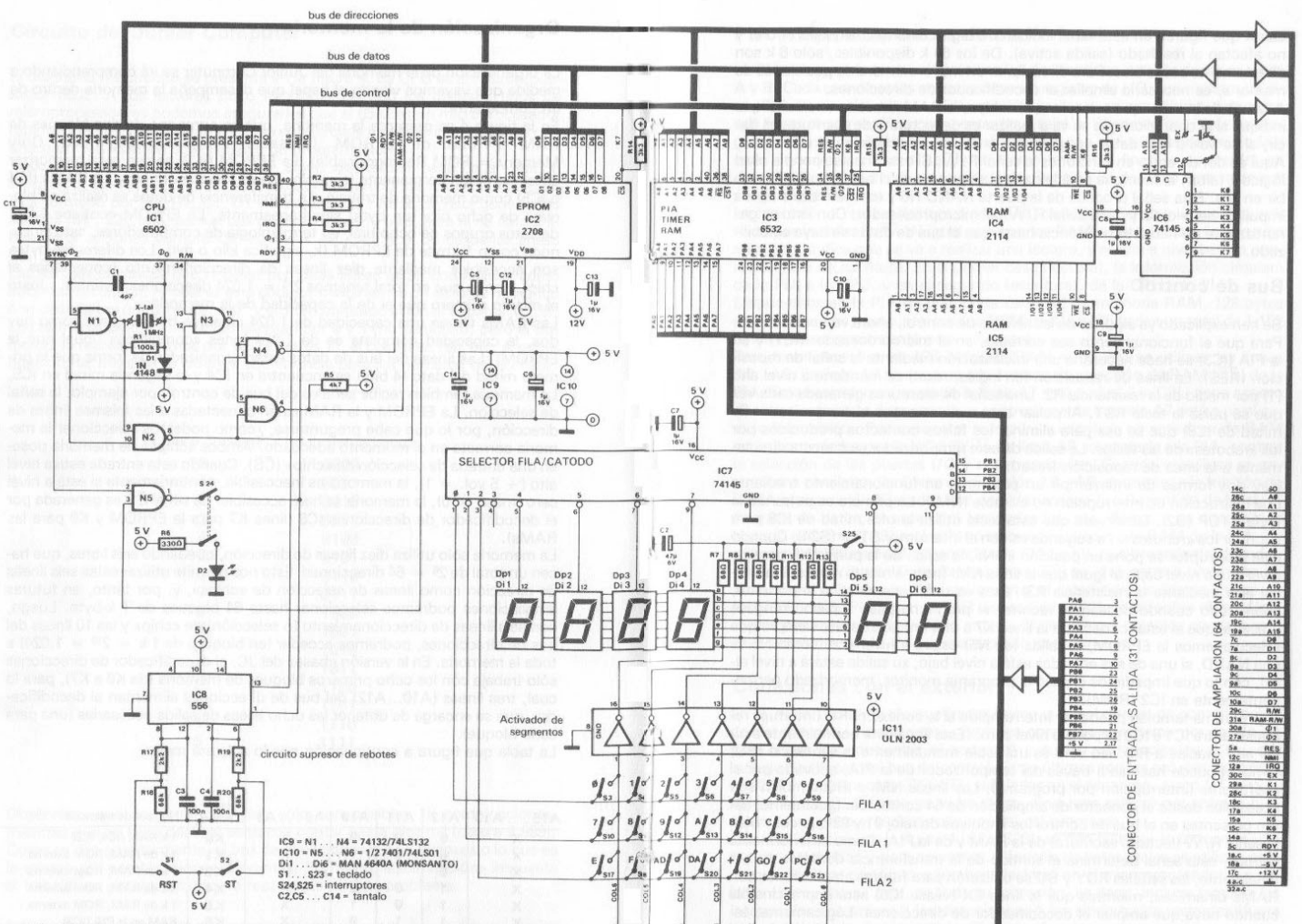


Figura 4. Circuito completo del Junior Computer. Para simplificar el esquema se han representado los buses como una sola línea.

Las X que figuran en esta tabla indican un dígito binario cualquiera (1 o Ø) y no afectan al resultado (salida activa). De los 64 k disponibles, sólo 8 k son directamente accesibles. Para direccionar un mayor número de posiciones de memoria, es necesario ampliar el decodificador de direcciones.
Además de la señal que selecciona el «chip», la RAM necesita otra señal que indique si la operación que se va a realizar es de lectura o de escritura, es decir, si se van a leer datos ya grabados o se van a escribir otros diferentes. Aquí es donde entra en funciones la señal R/$\overline{W}$. Si esta línea se pone a nivel lógico 1 (alto), la memoria será leída, y si se pone a nivel bajo se podrá escribir en ella. Esta señal procede de la puerta NAND N6 y es una mezcla de los impulsos de reloj Ø2 y de la señal R/$\overline{W}$ del microprocesador. Con esto se garantiza que no habrá transferencia hasta que el bus de datos se haya estabilizado.

Bus de control

Se han explicado ya algunas de las señales de control, ahora veamos el resto. Para que el funcionamiento sea correcto, en el microprocesador (IC1) y en la PIA (IC3) se hace necesaria una inicialización mediante la señal de reposición (RES). La línea de reposición (en inglés, reset) se mantiene a nivel alto (1) por medio de la resistencia R2. Una señal de «reset» es generada cada vez que se pulsa la tecla RST. Al pulsar esta tecla se activa el temporizador (la mitad de IC8) que se usa para eliminar los falsos contactos producidos por los «rebotes» de las teclas. La salida de este temporizador se conecta directamente a la línea de reposición (reset).
Hay dos formas de interrumpir un programa en funcionamiento mediante una instrucción de interrupción no evitable (NMI). La primera es pulsando la tecla STOP (S2). Obsérvese que esta tecla utiliza la otra mitad de IC8 para suprimir los «rebotes». La segunda es con el interruptor STEP (S24). Cuando este interruptor se pone en posición «ON», la salida de la puerta N5 pasa de nivel alto a nivel bajo, al igual que la línea NMI (normalmente mantenida a nivel alto mediante la resistencia R3). Esta es una característica importante, sobre todo cuando deseamos recorrer el programa paso a paso. Téngase presente que al estar conectada la línea K7 a una entrada de N5, siempre que seleccionemos la EPROM, la salida (de N5) estará a nivel alto (como N5 es una NAND, si una de sus entradas está a nivel bajo, su salida estará a nivel alto), con lo que impedimos el paso al programa monitor, memorizado permanentemente en IC2 (ROM).
El programa también puede ser interrumpido si la conexión IRQ (interrupt request) entre IC1 e IC3 se pone a nivel cero. (Esta línea está normalmente a nivel alto gracias a R4). No sólo es utilizable manualmente la sentencia IRQ, también puede hacerse a través del temporizador de la PIA, activado por el «software» (interrupción por programa). Las líneas NMI e IRQ también son accesibles desde el conector de ampliación de 64 contactos. Igualmente, están presentes en el bus de control los impulsos de reloj Ø1 y Ø2 que controlan la señal R/$\overline{W}$ (lectura/escritura) de la RAM y de la PIA. Como anteriormente dijimos, esta señal determina el sentido de la transferencia de datos.
Finalmente, las señales RDY y SO se utilizarán para futuras ampliaciones con RAMs dinámicas, mientras que la línea EX (véase IC6) será imprescindible cuando haya que ampliar el decodificador de direcciones. Lógicamente, estas últimas líneas (RDY, SO y EX:) no tienen utilidad en la versión básica del Junior Computer.

PIA (etapa adaptadora de periféricos)

La PIA es capaz de transferir datos en los dos sentidos a través de los accesos A y B. Cada uno de estos accesos tiene un registro de I/O y un registro de dirección de datos (8 bits). La información presente en este último determina qué líneas se van a utilizar como entradas o salidas. Si uno de los bits de este registro se pone a nivel 1, se ordenará a la PIA que active la salida asociada á este bit, mientras que si se pone a cero la misma línea se convertirá en entrada. El contenido de los registros de dirección de datos se determina por programa (software).

La operación a realizar (lectura o escritura) con un cierto dato, viene determinada por la señal R/$\overline{W}$. Al igual que con las RAMs, cuando esta línea está a nivel alto, indica que se va a realizar una lectura, y si está a nivel bajo la operación será de escritura. En el primer caso (lectura), la información circulará de la PIA a la CPU, y en el segundo (escritura), de la CPU a la PIA.

Disponemos en la PIA de una cierta cantidad de memoria RAM, 128 bytes para ser exactos, que junto con los 1.024 de IC4 e IC5 hacen un total de 1.152 bytes (más que suficiente para empezar).

Para tener acceso a los 128 bytes de RAM en la PIA se utilizan las líneas AØ...A6. La línea A7 se conecta a la entrada de selección de RAM ($\overline{RS}$) de la PIA. En otras palabras, cuando la línea A7 está a nivel 1 la CPU tiene acceso directo a esta parte de memoria, y contrariamente cuando A7 está a nivel cero lo imposibilita. Por supuesto, la lectura y escritura en este área de RAM están controladas por la señal R/$\overline{W}$. La línea A9, conectada a la PIA, controla selección de las puertas I/O y el temporizador. Cuando esta línea se encuentra a nivel alto, las líneas de dirección AØ...A6, determinan la función de la PIA que se está utilizando. Por ejemplo: acceso A ó B, sentido de la transferencia de datos, el acceso al temporizador, etc. Téngase en cuenta que cuando esta línea (A9) está a 1, no se puede acceder a la RAM de la PIA. Finalmente, la última línea conectada a la PIA es K6, que procede del decodificador de direcciones. Cuando esta línea (K6) se encuentra a nivel 1 se interrumpe el acceso a la PIA (sin importar el estado de las demás líneas).

Resumiendo, la PIA es un versátil dispositivo multifunción sobre el que fácilmente podría escribirse un libro entero. Sin embargo, creemos que con esto será suficiente, en lo que a este capítulo concierne.

Conexiones con el exterior

En el JC, el teclado y el visualizador hacen el papel de vías de comunicación con el exterior (o periféricos). Estos se conectan al sistema por medio de siete líneas pertenecientes al acceso A y cuatro pertenecientes al B, además de dos líneas del bus de control. Estas últimas son las señales de parada (STOP) y puesta a cero (RESET), señaladas en el esquema, como ST y RST. El interruptor S24, como ya se ha dicho, sirve para seleccionar entre la función paso-a-paso y el desarrollo normal del programa.

Las demás teclas, en la figura 4 (S3 a S23), se han distribuido formando una matriz de tres filas y siete columnas. Dieciséis de estas teclas se utilizan para introducir datos codificados (en hexadecimal) en el microprocesador. La palabra «dato» se usa aquí, en toda su extensión, es decir, incluye también las informaciones relativas a las direcciones. A las cinco teclas restantes (fuera de la matriz) se les han asignado diferentes funciones de control que se estudiarán con más detenimiento en el capítulo 4.

Tanto la información enviada al visualizador como la que entra por el teclado se transfieren a través de las siete líneas del acceso A. Este es un caso en el que podemos ver la utilidad de un bus bidireccional. La información que muestra el visualizador es controlada por el programa monitor, que también se encarga de identificar las teclas pulsadas. El integrado IC7 (decodificador BCD-a-decimal) decodifica la información presente en las líneas PB1, PB2, PB3 y PB4 del acceso B. La información decodificada se utiliza para multiplexar el visualizador y comprobar el estado (lógico) de las teclas, informando si alguna de ellas ha sido pulsada. Las tres primeras salidas de IC7, sirven para «escrutar» el teclado. Si se pulsa una tecla mientras una de estas salidas se encuentra a nivel alto se transmitirá a la CPU el código de la tecla en cuestión a través del acceso A. La tercera salida de IC7 no se usa (realmente es que no se necesita). Cuando una de las seis restantes salidas de IC7 está a nivel alto se activa el dígito correspondiente del visualizador, y entonces es cuando la CPU tranfiere hacia los activadores de segmentos (IC11), y a través del acceso A, la información codificada relativa a los segmentos para pasar, finalmente, al dígito activado. Este proceso de selección sucesiva de los dígitos del visualizador es el multiplexado, un método elegante de reducir el número de elementos de un sistema de visualización, si bien implica un aumento del programa monitor. Sin embargo, el coste de la memoria extra necesaria es inferior al de los componentes necesarios para realizar un visualizador no multiplexado.
El visualizador puede utilizarse de dos formas distintas. Normalmente los cuatro dígitos de la derecha (Dp1 a Dp4) indicarán una dirección, y los dos restantes (Dp5 y Dp6), el dato correspondiente a esa dirección. Como segunda posibilidad, los dos primeros (Dp1 y Dp2) indicarán el código hexadecimal de una instrucción (OP-CODE) y los otros cuatro (Dp3 a Dp6) mostrarán la dirección del dato correspondiente a esa instrucción. Esta última modalidad de visualización hace más cómoda la introducción de datos. Queda por mencionar el interruptor de encendido del visualizador S25, que permite reducir el consumo de la fuente de alimentación cuando el JC trabaja con un terminal de vídeo u otro periférico.

Alimentación

Incluso el más sofisticado computador necesita una fuente de alimentación. En el Junior Computer son necesarias tres tensiones de alimentación. El circuito de la fuente de alimentación se muestra en la figura 5. Estas tensiones son: + 5 V. (para todos los integrados y el visualizador), — 5 V y + 12 V (para la EPROM). Estas tensiones son proporcionadas por los reguladores de tensión integrados IC1, IC2, IC3. Para asegurar el necesario desacoplo se ha provisto a cada regulador de sus propios condensadores.
Una vez aclaradas las bases teóricas y prácticas del JC podemos empezar su construcción.

Construcción

¿Por dónde empezar? Si queremos hacer un buen montaje y, en la medida de lo posible, evitar los problemas propios de un circuito de este tipo, es imprescindible leer este apartado, prestando la máxima atención.

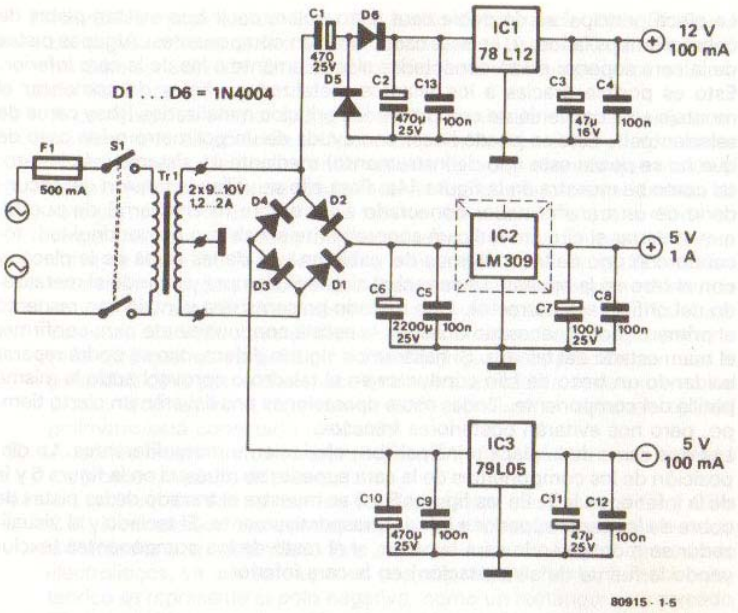


Figura 5. La fuerte de alimentación del Junior Computer proporciona tres tensiones estables.

El montaje se dividirá en tres partes. Primeramente montaremos los componentes en las placas de circuito impreso. Es desaconsejable que el lector intente construirse la placa principal del circuito, ya que se trata de una placa de doble cara, que requiere algunas técnias especiales para su realización. Para solucionar este problema se comercializan a través de la revista ELEKTOR todas las placas necesarias para este montaje (80089-1, 80089-2, 80089-3). Una vez montados todos los componentes correctamente, viene la segunda etapa: la comprobación; ésta nos llevará poco tiempo si los componentes son buenos. La tercera y última etapa es el ensamblaje mecánico: introducir el Junior Computer en una caja.

Placa principal

La construcción del Junior Computer es realmente simple, ya que se trata de un computador *monoplaca.* Esto quiere decir que todos los componentes electrónicos se montan en la misma placa de circuito impreso. En nuestro caso, cabe preguntarse: ¿Porqué se le llama monoplaca, si realmente hay tres? La respuesta es muy simple: una de las placas es la de la alimentación y no se tiene en cuenta; quedan, por tanto, la placa principal y la del visualizador. Esta última es una pequeña placa que va colocada sobre la principal, y se monta formando un ángulo de 45°, con el solo fin de facilitar así la lectura de datos.

La placa principal es de doble cara, esto quiere decir que existen pistas de cobre a ambos lados, y, en este caso, también componentes. Algunas pistas de la cara superior están conectadas eléctricamente a las de la cara inferior. Esto es posible gracias a los taladros metalizados. Antes de comenzar el montaje será conveniente comprobar los orificios metalizados (¡hay cerca de seiscientos!). Esto se puede hacer con ayuda de un polímetro o (en caso de que no se posea este tipo de instrumento) mediante un sistema más barato, tal como se muestra en la figura 14a. Para ello se utiliza la tensión del secundario de un transformador conectado a un timbre (o chicharra) de puerta, que al cerrar el circuito indicará sonoramente si existe o no continuidad: tocando con uno de los extremos del cable en una de las caras de la placa, y con el otro en la opuesta se cerrará el circuito (siempre y cuando el metalizado del orificio sea correcto). Este método presenta una ventaja con respecto al primero: no es necesario observar la escala continuamente para confirmar el buen estado del taladro. Si halláramos alguno defectuoso se podrá reparar soldando un trozo de hilo conductor en el taladro o aprovechando la misma patilla del componente. Todas estas operaciones nos llevarán un cierto tiempo, pero nos evitarán posteriores fracasos.
Las dos caras de la placa principal son, obviamente, muy diferentes. La disposición de los componentes de la cara superior se muestra en la figura 6 y la de la inferior en la 7. En las figuras 8 y 9 se muestra el trazado de las pistas de cobre de las caras superior e inferior, respectivamente. El teclado y el visualizador se montan en la cara superior, y el resto de los componentes (excluyendo la fuente de alimentación) en la cara inferior.

Montaje de componentes

Ahora podemos comenzar el montaje «en serio». El soldador utilizado debe ser de tipo «lápiz», con una potencia de unos 20 W. Los componentes se deberán montar en el siguiente orden (ver figura 7 y lista de componentes de la placa principal):

1. Las resistencias R1...R20 se montarán en primer lugar. Una vez soldadas se cortará la longitud excedente de las patillas lo más cerca posible de la placa. Esto se hace como medida de seguridad para prevenir cortocircuitos entre pistas adyacentes. Para los lectores que no estén familiarizados con el código de colores de las resistencias, facilitamos a continuación la lista de los valores utilizados:

   100 k: marrón, negro, amarillo (oro).
   3k3: naranja, naranja, rojo (oro).
   4k7: amarillo, violeta, rojo (oro).
   330Ω: naranja, naranja, marrón (oro).
   68Ω: azul, gris, negro (oro).
   2k2: rojo, rojo, rojo (oro).
   68k: azul, gris, naranja (oro).

   El primer anillo es el que se encuentran más cerca de uno de los extremos. El cuarto (entre paréntesis en nuestra lista) indica la tolerancia (o desviación del valor estandar) del componente. La mayoría de las resistencias poseen un anillo dorado (oro) que indica una tolerancia de ± 5 por 100, aunque también es posible encontrarlas del 1 por 100 (anillo de

tolerancia marrón) o del 2 por 100 (anillo de tolerancia rojo). Si alguna de las resistencias tuviera el anillo «plateado» (10 por 100 de tolerancia) no debe ser utilizada. En este caso, se deberá sustituir por otra resistencia del 5 por 100 o inferior.

2. A continuación se montará el diodo D1, poniendo especial cuidado en no insertarlo en posición errónea. La polaridad en estos componentes viene determinada por un anillo en uno de sus extremos que indica el catodo y se corresponde con la línea vertical que figura en el símbolo de este componente en el circuito teórico. En el caso de que el diodo no tenga anillo o esté situado exactamente en el centro, la única forma positiva de identificar su polaridad es utilizando un polímetro. Si se utiliza este sistema encontraremos que el diodo presenta una gran resistencia en uno de los sentidos y resistencia casi nula en el contrario. Esto únicamente nos indica que el diodo está en buenas condiciones, pero no mos dice cuál es su polaridad. Para averiguarla es preciso conocer la polaridad de la punta de prueba que se ha aplicado a cada extremo del diodo. Normalmente, cuando el diodo conduce el terminal rojo del polímetro está conectado al cátodo.
3. Ahora se montarán los condensadores C1, C3 y C4 (cortando también los trozos de patilla sobrantes).
4. A continuación se montan los condensadores electrolíticos C2 y C5 a C14. Normalmente los condensadores y resistencias pueden montarse sin atender a su polaridad. Este no es el caso de los condensadores electrolíticos, ya que éstos tienen una polaridad definida. En el circuito teórico se representa el polo negativo, como un rectángulo sombreado (▮) y el positivo con un rectángulo en blanco (□). Los condensadores que no llevan marcado el signo más se pueden identificar de dos formas: algunos llevan una hendidura indicando el polo positivo mientras que en otros se indica con una señal de color rojo.
5. Se montan ahora los zócalos de los circuitos integrados. Es recomendable utilizarlos para todos los ICs, aunque represente un gasto extra, si bien los más pequeños (y baratos) pueden montarse directamente. Sólo será realmente imprescindible los zócalos para IC1, IC2 e IC3. En la fotografía de la placa (figura 7) se ha dibujado una hendidura en la silueta de cada IC. Si usted mira la representación de un circuito integrado con la hendidura en la parte superior, la patilla número uno será la superior del lado izquierdo, aunque para evitar equivocaciones también se ha indicado (un 1) en la placa. Así, pues, la numeración de las patillas en los IC empieza por la parte izquierda, como antes dijimos, y se sigue hacia la parte inferior izquierda, de modo que en un integrado de catorce patillas la inferior izquierda será la número siete, y la inferior derecha la número ocho, correspondiéndole a la patilla superior derecha el número catorce. La hendidura de identificación puede presentar varias formas. Normalmente, consiste en un pequeño semicírculo en el centro de la parte superior, pero también puede ir impreso en el cuerpo del IC. Cuando va impreso, el signo utilizado es un punto que indica la patilla número uno, y entonces el circuito debe ser montado haciendo coincidir el citado punto con el número uno de la serigrafía. Conviene señalar que los zócalos, en su inmensa mayoría, tienen también marcada de alguna manera la patilla número uno.
6. El cristal de 1 Mhz puede montarse directamente o utilizando un zócalo adecuado para el mismo.

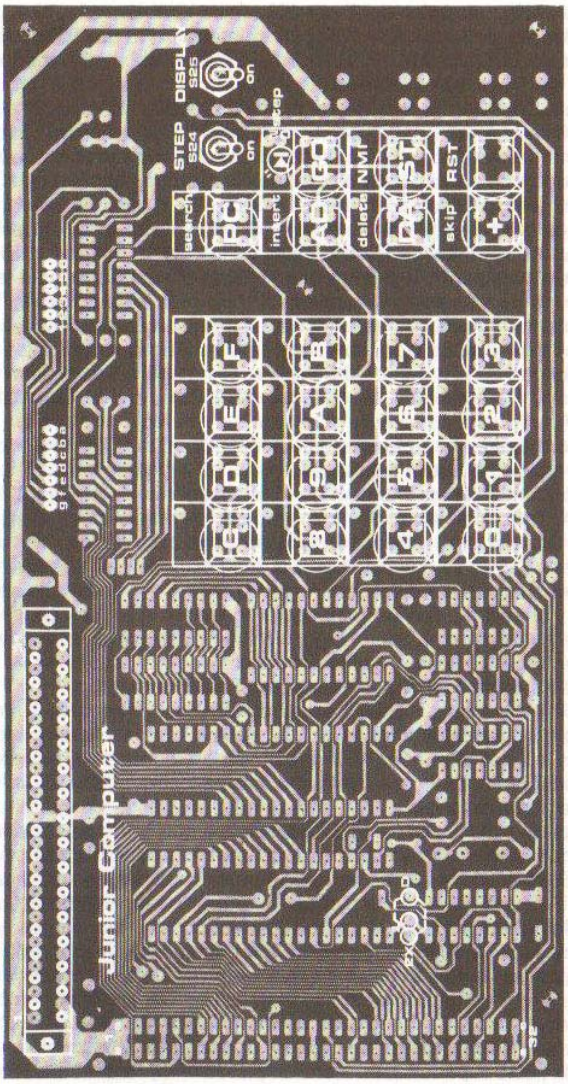


Figura 6. Distribución de componentes en la cara del teclado de la placa principal (EPS 80089-1).

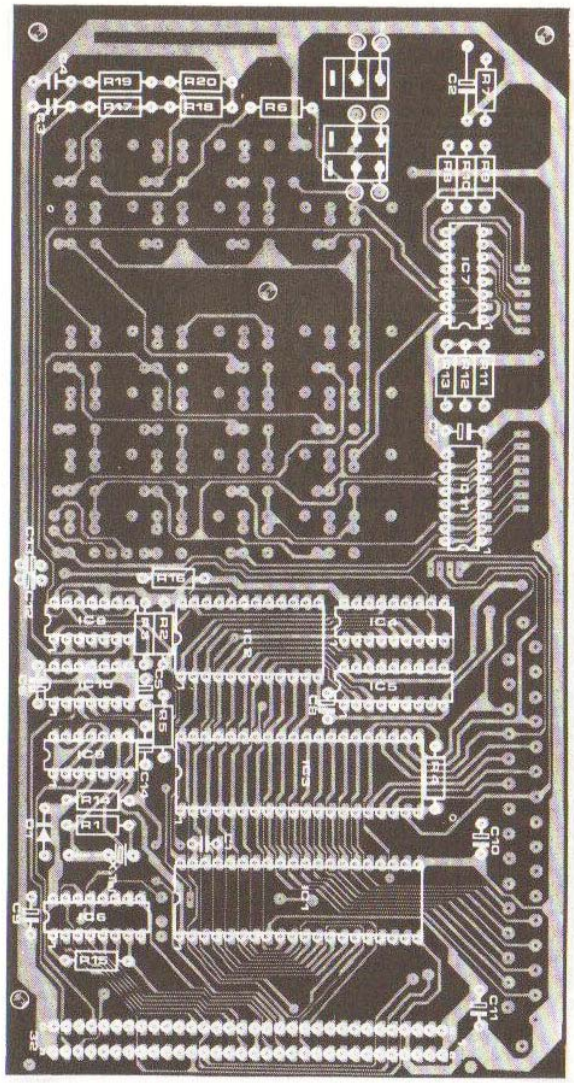


Figura 7. Distribución de componentes en la otra cara de la placa princial.

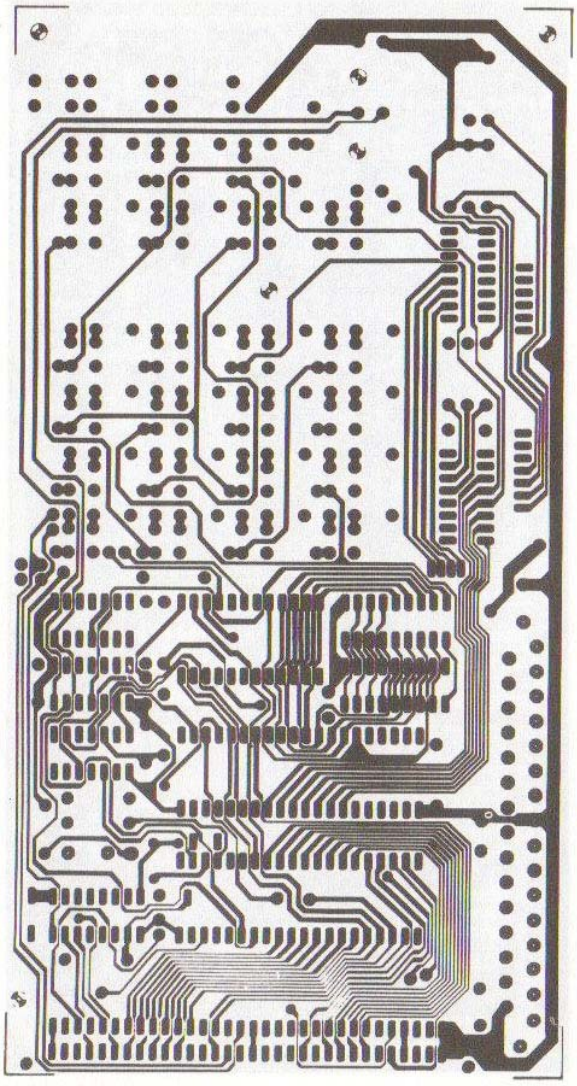


Figura 8. Trazado de las pistas en la cara superior de la placa principal.

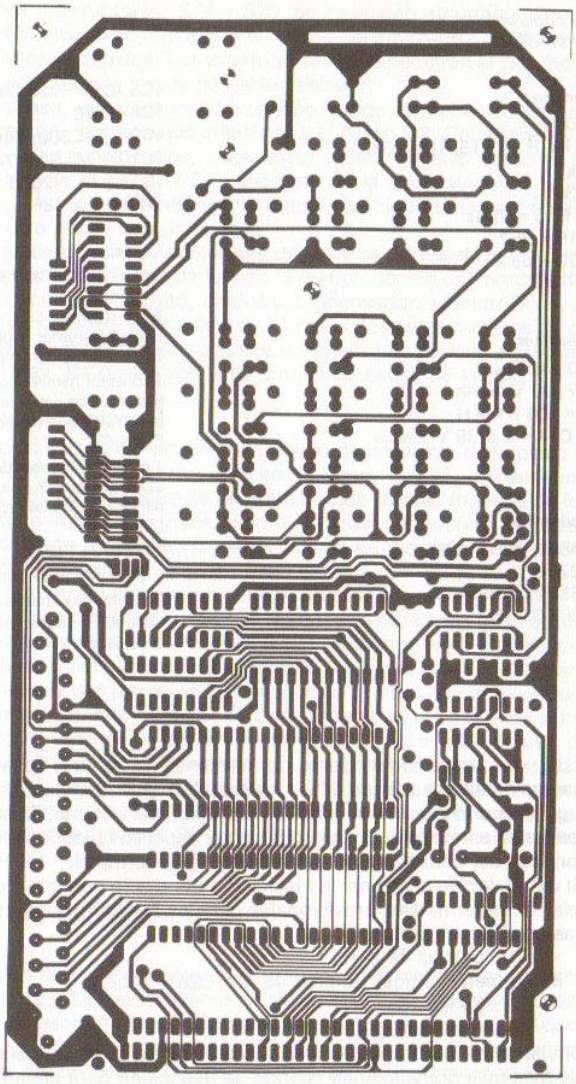


Figura 9. Trazado de las pistas en la cara inferior de la placa principal.

Lista de componentes de la placa principal.

Resistencias:

R1 = 100 k
R2,R3,R4,R14,R15,R16 = 3k3
R5 = 4k7
R6 = 330 Ω
R7 . . . R13 = 68 Ω
R17,R19 = 2k2
R18,R20 = 68 k

Condensadores:

C1 = 10 p cerámico
C2 = 47 µ/6 V tántalo
C3,C4 = 100 n MKH
C5 . . . C14 = 1 µ/35 V tántalo

Semiconductores:

IC1 = 6502 (Rockwell)
IC2 = 2708
IC3 = 6532 (Rockwell)
IC4,IC5 = 2114
IC6,IC7 = 74145, 74LS145
IC8 = 556
IC9 = 74LS00, 7400, 74LS132
IC10 = 74LS01, 7401
IC11 = ULN2003 (Sprague)
D1 = 1N4148

Varios:

S1 . . . S21, S23 = teclas (Shadow o similar)
S22 = tecla con LED
S24 = interruptor doble
S25 = interruptor simple
1 conector hembra de 64 contactos con salida paralela al circuito impreso DIN 41612
1 contector hembra de 31 contactos con salida paralela al circuito impreso DIN 41617
1 Cristal de 1 MHz
1 zócalo de 24 patillas
2 zócalos de 40 patillas
placa de circuito impreso (EPS 80089-1)

7. Este paso es opcional. Como de momento no se va a necesitar el conector de ampliación de 64 contactos, éste puede instalarse en último lugar o cuando se lleve a cabo la ampliación del sistema. Cuando vaya a realizarse esta operación se ha de tener especial cuidado en apretar los tornillos del conector antes de soldar algún terminal.
   Si usted decide posponer la instalación del citado conector, deberá instalar unos terminales provisionales para las líneas de alimentación, cuyas conexiones son:
   + 5 V: patillas 1a ó 1c.
   Masa (cero voltios): patillas 4a, 4c, 32a y 32c.
   − 5 V: patilla 18a.
   + 12 V: patilla 17c.
   Obviamente, utilizar terminales como solución eventual obliga a tomar las máximas precauciones cuando se desuelden para efectuar la sustitución por el conector, ya que las pistas de cobre son muy delgadas y podrían despegarse si se les aplicara excesivo calor.

Con esto completamos el montaje de componentes en la parte superior de la placa, seguidamente montaremos el teclado en la otra cara.

8. Sólo hay un puente a realizar en la placa principal. Este se hará soldando un trozo de hilo conductor entre los puntos marcados como «D» y «⊥» (masa).

9. Los dos interruptores, S24 y S25, se instalarán a continuación. Estos deben montarse de forma que el cuerpo del interruptor quede en la parte inferior de la placa. Los interruptores se conectarán al circuito principal mediante seis trozos de cable (aislado).
10. Si se desea, ya puede instalar el conector de 31 contactos en su lugar.
11. Finalmente, montaremos el teclado y el diodo D2. Debe tenerse cuidado al colocar las teclas, pues éstas deben hacerse coincidir con la serigrafía de la placa. D2 se montará en la parte superior de la tecla «GO». Téngase presente que se trata de un diodo LED, que como cualquier otro posee una polaridad.

Con esto queda concluido el montaje de la placa principal. Evidentemente, no representará una pérdida de tiempo la comprobación del montaje de cada uno de los circuitos integrado, diodos y condensadores electrolíicos, así como los demás componentes pasivos. También nos aseguraremos de que no haya ningún «puente» (debido a restos de estaño) entre pista adyacentes, ni soldaduras frías, las cuales son muy a menudo causa de averías.

Placa del visualizador

La distribución de componentes y el trazado de las pistas de la placa del visualizador (EPS 80089-2) se presenta en las figura 10 y 11, respectivamente. La construcción de esta placa se realizará en dos etapas: montaje de los seis dígitos de siete segmentos y conexión de la misma al circuito principal. Los dígitos no deberán presentar ningún problema de montaje, ya que sólo hay una forma de insertarlos en la placa (para ello sus terminales se han dispuesto asimétricamente). La conexión del visualizador a la placa principal* se hace mediante trece conductores de hilo rígido: uno para cada segmento (siete en total) y uno para el cátodo de cada dígito (seis en total). Estas conexiones están marcadas en la placa principal junto a la parte superior del teclado. La distancia entre las dos placas será alrededor de 5 mm. Los trece conductores serán de una longitud aproximada de 2 cm. y deberán sobresalir de la cara de cobre de la placa del visualizador. Es conveniente hacer una revisión de terminales antes de conectar ambas placas. Primeramente se montará la placa del visualizador sobre la principal (introduciendo los hilos en los orificios correspondientes, pero sin soldarlos) y, seguidamente, le daremos una inclinación de aproximadamente 45° con respecto a la placa principal. Ahora ya podemos soldar los hilos y cortar los extremos sobrantes.

Placa de la fuente de alimentación

La distribución de componentes y el trazado del circuito impreso se muestra en las figuras 12 y 13, respectivamente. Al igual que la placa del visualizador, ésta no debe presentar ningún problema en su montaje. Unicamente se deberá tener cuidado en respetar la polaridad de los diodos y condensadores electrolíticos. No olvidemos que se debe proveer del refrigerador adecuado a IC2 (LM 309 K).

---

* Nota: Si se desea, el visualizador puede instalarse en otro emplazamiento diferente del que se ha previsto en la placa principal; para ello el uso de cable paralelo múltiple puede resultar ventajoso. Este cable está formado por varios conductores de diferente color, formando una «cinta» (lo cual es idóneo para nuestros propósitos). Obviamente, se ha de tener cuidado de conectar el otro extremo del cable (el que va a la placa principal) correctamente, es decir, cada color a su orificio.

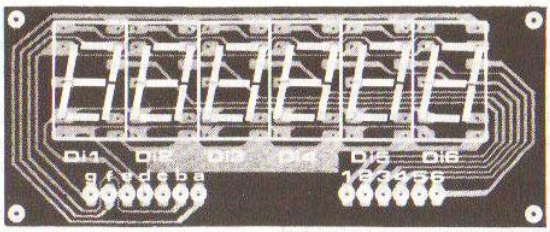


Figura 10. Disposición de componentes (EPS 80089-2) en la placa del visualizador.

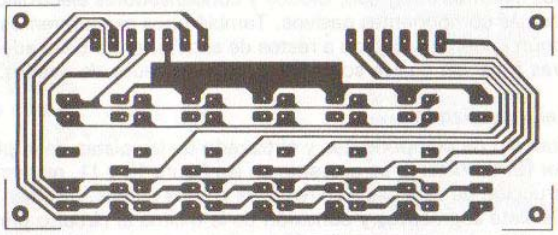


Figura 11. Trazado de las pistas en la placa del visualizador.

Lista de componentes de la placa del visualizador.
Semiconductores:
Di1 . . . Di6 = MAN 4640A
cátodo común (Monsanto)

Varios:
Una placa de circuito impreso
referencia EPS 80089-2

¿Funciona?

Por fin, las tres placas están ya montadas. El siguiente paso será conectar el transformador Tr1 a la fuente de alimentación, a través del interruptor de alimentación S1 y el fusible F1. Dado que esta fase de comprobación es breve se deberán tomar todas las precauciones posibles. Antes de expezar será conveniente, una vez más, revisar la colocación de los componentes. Es mejor descubrir un error a tiempo que lamentarlo después. A veces, es conveniente que otra persona eche una mirada al circuito (¡cuatro ojos ven más que dos!).
Por supuesto, lo primero que debemos comprobar es la fuente de alimentación (para lo cual no debe estar conectada al circuito principal). Demos tensión al circuito. ¿No sale humo? Es una buena señal, ¿o, tal vez, nos hemos olvidado de accionar el interruptor S1? Una vez solucionado esto último, repetimos nuevamente la pregunta, ¿sale humo? Si la respuesta es no, podemos decir que la fuente de alimentación ha pasado su primera prueba. Mida-

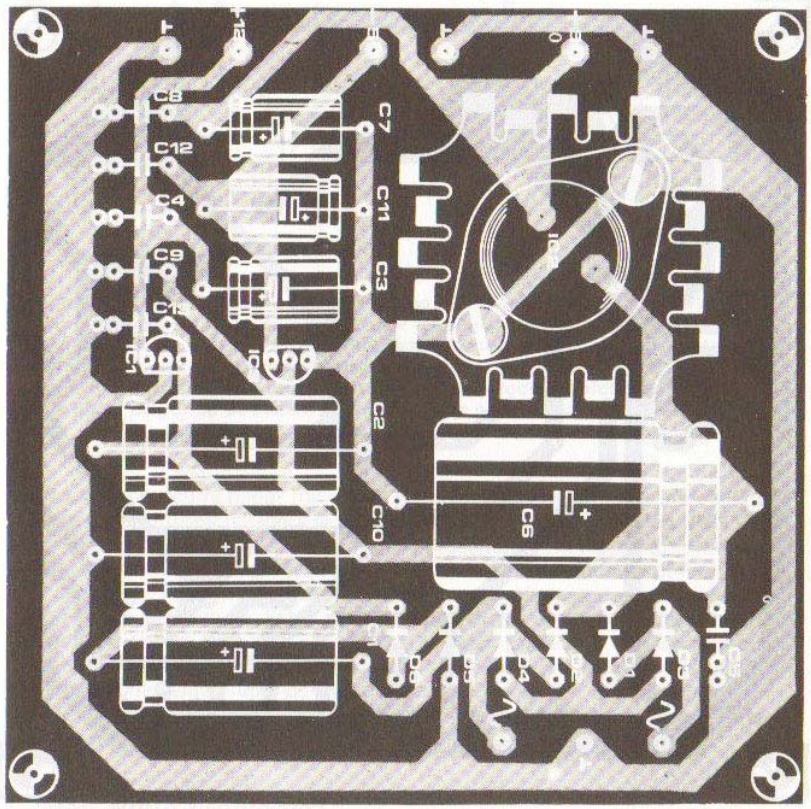


Figura 12. Disposición de los componentes de la placa de la fuente de alimentación (EPS 80089-3).

Lista de componentes de la fuente de alimentación.

Condensadores:

C1, C2, C10 = 470 μ/25 V
C3, C11 = 47 μ/25 V
C4, C5, C8, C9, C12, C13 = 100 n MKH
C6 = 2200 μ/25 V
C7 = 100 μ/25 V

Semiconductores:

IC1 = 78L12ACP (5%)
IC2 = LM 309K
IC3 = 79L05ACP (5%)
D1 . . . D6 = 1N4004

Varios:

Tr1 = Transformador
primario 220 V
secundario 2 × 9 ... 10 V/1,2 ... 2 A
S1 = interruptor doble
F1 = fusible 500 mA con portafusibles
1 circuito impreso EPS 80089-3
1 disipador para IC2

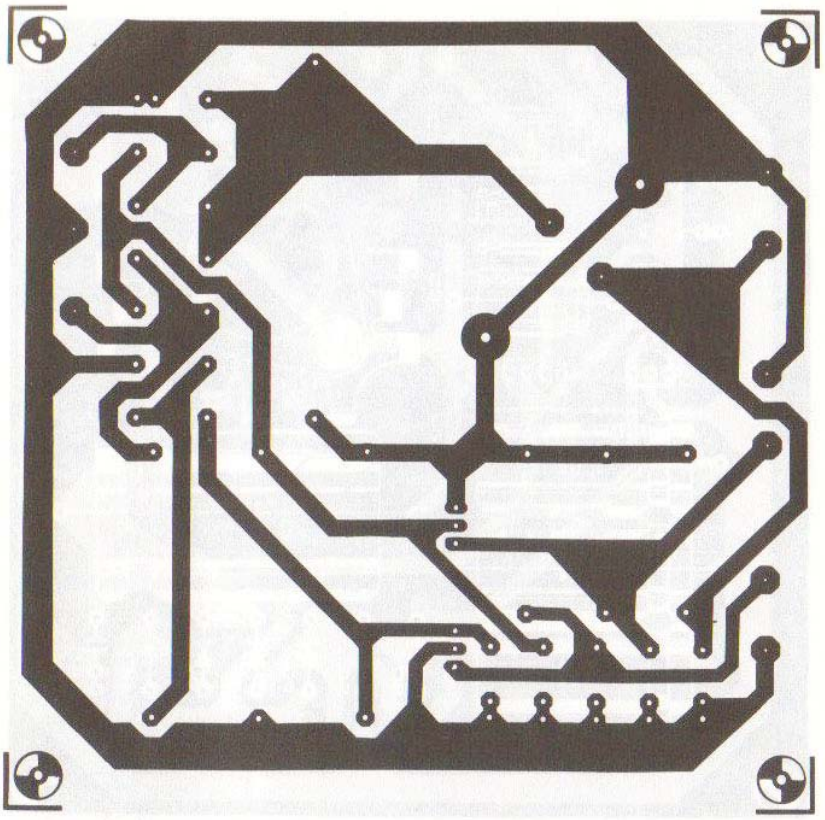


Figura 13. Trazado de las pistas de la placa de la fuente de alimentación.

mos ahora (con el polímetro en la escala correspondiente de tensiones continuas) todos los voltajes de salida de la fuente de alimentación. Las lecturas deberán estar dentro del 5 por 100 de la tensión especificada en el texto. Si no fuera así quiere decir que hay algún fallo. Esto último es bastante improbable, ya que el circuito es muy simple y utiliza componentes de elevada calidad.

Si todas las comprobaciones respecto a la fuente de alimentación dan resultado positivo, seguidamente podremos conectar los cables entre la fuente de alimentación y el circuito principal. En este punto deberemos tener certeza absoluta, sobre la exactitud de las conexiones realizadas, no importa el número de veces que se repitan las comprobaciones, si en esta etapa se produjera algún fallo, se deberá únicamente a un descuido. Una vez seguros de que cada conexión está en su sitio, pongamos el interruptor S24 en posición «OFF» (desconectado) y S25 en «ON» (conectado). Ahora demos tensión al

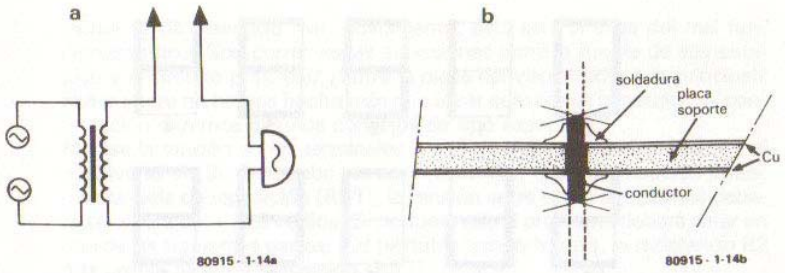


Figura 14. Los taladros metalizados no se han de comprobar necesariamente con un polímetro. Esto se puede hacer mediante un transformador y un (barato) timbre de puerta. Importante: este método sólo se utilizará si todavía no se han soldado los componentes. La figura b, muestra cómo realizar la conexión entre las dos caras de una placa «autoconstruida» en el caso de que no se disponga de sistema de metalizado para los taladros.

circuito, ¿no ocurre nada? ¡no cunda el pánico! esto es lo que se supone que debe pasar. Pulsemos ahora la tecla de puesta a cero o reposición (RST). Si todo está en orden, el visualizador debe presentar un número hexadecimal y para comprender su significado será necesario haber estudiado previamente el capítulo 2. Por el momento, solamente compare los signos del visualizador con los de la figura 15. Estos signos deben formar un conjunto aleatorio de números y letras (que aquí representan también números). Los lectores que ya hayan leído (y comprendido) los capítulos 2 y 3, identificarán los citados signos como algún tipo de instrucción de bifurcación. Si el visualizador muestra algo de lo dicho hasta ahora, podremos «saltar» al siguiente apartado, y empezar directamente con el ensamblaje mecánico.

¿Hay algún problema?

Nosotros esperamos sinceramente que no haya necesidad de leer este apartado, pero en el caso de que algo «vaya mal», a continuación damos la lista de los problemas más comunes.
El primer punto a comprobar es las tensiones de la fuente de alimentación. Suponemos que esto ya se habrá hecho varias veces, pero puede que se haya cometido algún error en los últimos momentos (las prisas nunca llevan a buen fin). Si no es así, la causa de los problemas estará en la placa principal:

- *Cortocircuitos entre soldaduras.* Revisar, si existe, alguna soldadura que forme un «puente» con las de alrededor y lo mismo con las pistas que se encuentren muy próximas. Esto, no siempre está tan claro como puede parecer. Algunas veces un delgado «hilo» de estaño (causado por una salpicadura) es el responsable de los puentes «invisibles».
- *Soldaduras «frías».* Esto puede sucederle a cualquiera. Las soldaduras frías se reconocerán inmediatamente por presentar una superficie poco «brillante» y un contacto eléctrico pobre con la pista de circuito impreso. Si encontráramos alguna, el problema tiene fácil solución: se le aplica la

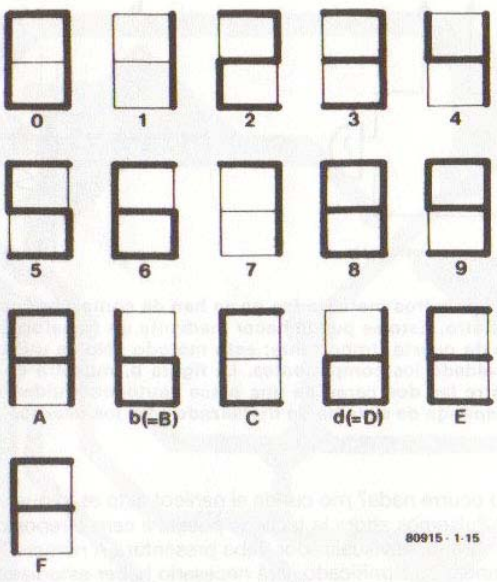


Figura 15. Cuando se pulsa la tecla RST aparece en el visualizador una combinación aleatoria (aparentemente) de números hexadecimales. Esto indica que el Junior Computer funciona correctamente.

punta del soldador y simultáneamente se añade algo más de estaño (justo lo imprescindible).

— *Mal contacto en los zócalos de los ICs.* No es tan difícil que esto suceda. El efecto es una resistencia parásita entre la patilla del IC y el terminal (o simplemente ausencia total de contacto). Sólo una cuidadosa inspección del circuito nos revelará si existe alguna patilla que no haya «entrado» en el alojamiento correspondiente del zócalo. No es muy frecuente (pero sí posible) que algún zócalo tenga los alojamientos sucios, pero si al presionar el IC contra el zócalo, el montaje recobra su funcionamiento normal, evidentemente, ésta era la causa. Algunas veces, esto se puede remediar frotando la parte superior del zócalo con un algodón ligeramente empapado en alcohol, permitiendo su entrada en los alojamientos de las patillas del circuito integrado.

— *Mala calidad de las pistas en los circuitos autoconstruidos.* Esto, generalmente, se produce cuando en la etapa de corrosión por ácido no se emplea el tiempo correcto (ya sea por exceso o por defecto), formándose con ello diminutas pistas de cobre, que difícilmente se descubrirán a simple vista. Los problemas de conexión entre las dos caras (taladros metalizados) se habrán eliminado si se efectuarón las comprobaciones anteriormente citadas.

— *Montaje incorrecto de diodos, condensadores electrolíticos o ICs.* Si algún circuito integrado se ha montado en posición equivocada o simple-

mente se ha insertado mal, obviamente, ésta será la cusa del mal funcionamiento. ¿Son correctas las conexiones entre la fuente de alimentación y el circuito principal? ¿Entre la placa del visualizador y la principal? Hasta ahora no hemos hecho más que tocar cuestiones generales. A continuación daremos algunos consejos de tipo especial:

- Mídase la tensión en los terminales 7 y 13 de IC8 (el negativo es el 7 y el positivo es el 13). Esta debe ser de + 5 voltios. Si a continuación pulsamos la tecla de reposición (RST), la tensión entre estos dos puntos deberá ser ahora de + 0,5 voltios. Si no fuera así, el problema deberá estar en una de las siguientes partes: IC8 (el doble temporizador), la resistencia R2 o la propia tecla de reposición (RST).
- Si los puntos anteriores resultan correctos, entonces midamos la resistencia entre la patilla 12 y masa de IC6 (por supuesto, para realizar esta medida deberá estar desconectada la tensión de alimentación). Esta deber ser cero ohmios, de lo contrario esta conexión es incorrecta.
- El generador de reloj constituye «el corazón» del microprocesador. Si poseemos un osciloscopio de doble traza podremos monitorizar las señales de reloj (Ø1 y Ø2) a través del conector de ampliación en los terminales 30a y 27a. La conexión se realizará de la siguiente manera: el terminal de tierra del osciloscopio se conectará a los terminales de masa del computador (4a ó 4c), la entrada A (ó Y1) al terminal 30a y la B (ó Y2) al 27a. En la pantalla deberán aparecer dos señales desfasadas 180°. Es decir, cuando una presenta su nivel máximo la otra el mínimo y viceversa. La tensión pico a pico de estas señales debe estar entre tres y cinco voltios. El aparato no debe encontrar dificultad en mantener estáticas las señales en la pantalla, o sea, deben ser estables. Si esto no sucediera habrá que revisar los componentes C1, IC9 y D1.

Esperamos que alguna de las soluciones dadas hasta aquí puedan resolver su problema; como siempre, si no consiguiera poner en funcionamiento el Junior Computer puede dirigirse al departamento técnico de ELEKTOR, remitiéndonos el mayor número de datos posibles (para más detalles sobre consultas técnicas, vea la última revista).

La caja

La caja cumple tres funciones básicas: protege el circuito de los elementos, facilita el manejo y da al computador un aspecto funcional.
Normalmente, hay dos métodos para construir la caja de un montaje: La típica «caja de cigarros» y las cajas comerciales prefabricadas. Las cajas comerciales para montajes de este tipo, suelen disponer de un panel transparente para el visualizador. La placa del visualizador del JC puede montarse detrás del citado panel, una vez que se hayan hecho los taladros pertinentes. También será necesario practicar los taladros adecuados en la parte superior de la caja, para el teclado y los interruptores. El portafusibles y el conector de red se colocarán en la parte posterior de la caja.
Si se decide utilizar una «caja de cigarros» el diseño se hará según el propio criterio.
Téngase en cuenta que si se piensan realizar las diferentes ampliaciones del sistema, la caja debe tener capacidad para admitir estos módulos.

Las placas de circuito impreso se montarán en la caja utilizando «separadores». Debemos tener cuidado al hacer la instalación mecánica del teclado, de forma que las teclas puedan pulsarse líbremente, evitando así entradas de datos erróneas.

2

Sistema binario de numeración

Contar con dos dedos

Probablemente, el hecho de que el homo sapiens **tenga diez dedos es la razón por la que se cuentan las cosas por decenas. Disponemos de diez cifras distintas y hemos formado un código que nos permite manejarlas. Análogamente, el Junior Computer también trabaja y maneja números, pero sólo tiene dos «dedos» con los que contar. Esto que puede parecer una desventaja a primera vista, en realidad no lo es y este capítulo quiere demostrarlo profundizando en el sistema binario (base dos) de numeración.**

Coja un 1 un 9 un 8 y otro 1 y escríbalos uno al lado del otro, de izquierda a derecha.

1981

Evidentemente, se trata de un número que puede corresponder a un año, a un precio, etc. Utilizado los códigos matemáticos aprendidos en la escuela puede significar muchas cosas más. Puede representar también un número de teléfono. En este último caso representa un código que junto con el código correspondiente a la central determina la posición de varios relés y otro material eléctrico y electrónico.
Si el número 1981 se introduce en el computador tendrá una expresión bastante distinta:

11110111101

Como ya se dijo en el capítulo uno, los círculos con una diagonal que los cruza representan el número cero. Esto permite diferenciarlos de la letra mayúscula O. Dado que sólo hay dos cifras en el sistema de numeración utilizado por los computadores, éstas se repetirán mucho más a menudo que en el sistema normal de numeración.
Debemos señalar aquí que el número 11110111101 no es el número decimal 11.110.111.101. Este número resulta un poco extraño, porque en él sólo se

tienen ceros y unos. Con el sistema de numeración de diez cifras, cada una de ellas tiene un 10 por 100 de posibilidades de aparecer en una posición determinada, y la posibilidad de que el número 11 110 111101 aparezca en la aritmética normal es muy pequeña (según nuestros cálculos, dos millonésimas partes del 1 por 100). El número de diferentes posibilidades, que pueden obtenerse utilizando el mismo número de cifras que antes (o sea, once) es:

— Para base 10 (sistema decimal de numeración)

$10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 1010 = 10^{11} = 100.000.000.000.$

— Para base dos (sistema binario de numeración)

$2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^{11} = 2.048.$

¡Hay una pequeña diferencia!

La estructura de un número

Todo número puede considerarse como una suma de números más pequeños. Un número decimal puede dividirse en unidades, decenas, centenas, millares y así sucesivamente. Puede, por tanto, definirse, un número, como unidades multiplicadas por otro número. Por ejemplo: 300 = 3x100. Para el número 1981 podríamos escribir:

1000 = un millar	$= 1 \times 10^3$	$= 1000 \times 1$
900 = nueve centenas	$= 9 \times 10^2$	$= 100 \times 9$
80 = ocho decenas	$= 8 \times 10^1$	$= 10 \times 8$
+ 1 = una unidad	$= 1 \times 10^0$	$= 1 \times 1$
1981		1981

Pero también podríamos escribir:

1024 = una vez	$1024 = 1 \times 2^{10}$	$= 1024 \times 1$
512 = una vez	$512 = 1 \times 2^9$	$= 512 \times 1$
256 = una vez	$256 = 1 \times 2^8$	$= 256 \times 1$
128 = una vez	$128 = 1 \times 2^7$	$= 128 \times 1$
0 = ning. vez	$64 = 0 \times 2^6$	$= 64 \times 0$
32 = una vez	$32 = 1 \times 2^5$	$= 32 \times 1$
16 = una vez	$16 = 1 \times 2^4$	$= 16 \times 1$
8 = una vez	$8 = 1 \times 2^3$	$= 8 \times 1$
4 = una vez	$4 = 1 \times 2^2$	$= 4 \times 1$
0 = ning. vez	$2 = 0 \times 2^1$	$= 2 \times 0$
+ 1 = una vez	$1 = 1 \times 2^0$	$= 1 \times 1$
1981		

Un uno indica la presencia de una potencia de dos y un cero indica su ausencia.

11110111101

Como puede verse, en este caso, los números no están divididos en potencias de 10, sino en potencias de dos. Conviene recordar, que estamos hablando todavía del número 1981. En esta tabla binaria se han utilizado números de base 10 (todos los números que no están formados por ceros y

unos). Pero debe quedar claro que cuando se trabaja con el sistema binario de numeración sólo son posibles dos símbolos: 1 y Ø.

¿Por qué no utilizar el sistema decimal?

El lector se preguntará porqué razón hemos dejado nuestro probado y eficiente sistema decimal, para pasar a un sistema disparatado que sólo utiliza dos cifras. La respuesta es bastante sencilla. Si damos otro vistazo al equivalente binario de 1981 nos encontraremos de que se trata de responder sí o no a la pregunta: ¿Está o no está presente una potencia de dos en esta posición del número? Un sí estará indicado por un uno y un no por un cero.
Los números deben ser interpretados, trasladados y menajeados de una u otra manera por los circuitos electrónicos interiores del computador, esto significa que cada número activará algún tipo de dispositivo electrónico. Si el computador trabaja con un sistema de numeración de base dos, entonces sólo será necesaria la relación «activado/desactivado» de un dispositivo electrónico. Estas dos posibilidades son los llamados estados lógicos y se definen como sigue:
1 = Uno lógico. Está presente una cierta tensión positiva (o estado alto).
Ø = Cero lógico. No hay tensión (o estado bajo)*.
La conclusión obvia de esto es que es mucho más sencillo diseñar un circuito electrónico para que funcione en sistema binario (sólo dos estados de salida posibles) que uno para que funcione en sistema decimal (diez estados de salida posibles).
Existe también otra ventaja, si se utiliza un sistema binario, que se hace patente a la hora de tomar decisiones. Más adelante, en este capítulo, utilizaremos organigramas y al desarrollarlos nos encontraremos con que muchas veces hay que tomar decisiones. Algo así como: ¿Es ABC igual a XYZ? Si la respuesta tuviera que darse en un sistema de base diez podríamos elegir entre diez posibilidades diferentes. Pero en un sistema binario la respuesta tiene sólo dos posibilidades: sí o no.
Visto de esta manera, creemos que es posible decir que el número binario 1111Ø1111Ø es más sencillo que el número decimal 1981. El primero tiene una sola elección por posición: hay potencia de dos o no hay potencia de dos. La última, por el contrario, tiene diez factores (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) posibles, para la potencia de diez, por posición. Así, pues, queda claro que aunque los números sean más largos con el sistema binario el ordenador puede funcionar mucho más eficazmente.

Bits y Bytes

La palabra bit se ha acuñado para hacer referencia a un dígito individual dentro de un número binario. El origen de la palabra está en los términos ingleses BInary digiT. Un bit puede tomar los valores cero y uno. Los bits están casi siempre en grupos (o «palabras») del mismo modo que en esta página se utilizan letras para formar palabras. Si la palabra está formada por

* Nota: Esto se denomina lógica positiva. La inmensa mayoría de los circuitos lógicos funcionan de este modo, pero existen algunos que utilizan lógica negativa que es exactamente lo contrario de lo que acabamos de ver, es decir, el nivel positivo de tensión se presenta por un cero y la ausencia de tensión por un uno.

8 bits, al grupo se le suele denominar *byte* (existen sistemas más largos en los que se utilizan palabras de 16 bits). Las palabras con sólo 4 bits suelen denominarse, a veces *nibbles*. El lector recordará del capítulo primero que el bus de datos del Junior Computer tiene una anchura de 8 bits. Esto significa que el bus de datos envía información byte a byte. Por el contrario, en el bus de direcciones, que tiene 16 bits se envían simultáneamente dos bytes.
Las palabras bit y byte se utilizan también en:

- Las instrucciones introducidas en el computador por el usuario.
- El código de 4 bits utilizado para definir en binario los números de base 10. Este código se denomina BCD (binay-coded-decimal) y se verá más adelante.
- El código de direcciones de 16 bit que define una posición de memoria o la dirección de un periférico.
- El código ASCII (American Standards Code for Information Interchange). Este código representa en binario todas las letras del alfabeto, los números, los signos de puntuación y otros muchos símbolos.

Todos estos códigos (y otros muchos que no hemos mencionado) están formados por palabras de un determinado número de bits, cada uno de los cuales puede ser cero ó uno.

Hexadecimal

Nosotros aceptamos ya que un sistema de numeración con sólo dos dígitos es ideal para el computador, pero, ¿qué sucede con la intercomunicación entre el computador y los seres humanos? Si la información tiene que ser introducida en el computador en forma de unos y ceros el único resultado posible es un caos total. Piénsese en el direccionamiento por dos bytes: ¡16 cifras para definir una posición! En la práctica ese sistema no resulta funcional, porque los errores son inevitables.
Para evitar este problema es necesario un sistema de numeración más sencillo, es decir, un sistema que pueda ser fácilmente interpretado por el computador y (mucho más importante) por el operador. La elección obvia es el sistema hexadecimal (hexadecimal significa sencillamente dieciséis números). Antes, cuando discutíamos la conversión de números decimales al sistema binario, veíamos que la cantidad de cifras necesaria para expresar un número en binario aumentaba vertiginosamente. Es lógico suponer, por el contrario, que si se utiliza una base mayor que diez (en este caso 16) la cantidad de cifras necesarias para representar un número disminuirá. Con un sistema de base 16 la longitud de los números binarios se divide por cuatro. En otras palabras, un byte de 8 bits puede representarse por sólo dos símbolos. En el sistema decimal se utilizaban diez símbolos (1. 2, 3, 4, 5, 6, 7, 8, 9, 0). En el sistema binario sólo eran necesarios dos (uno y cero). Por tanto, en un sistema de base 16 será necesario de disponer de 16 símbolos. Como en el sistema de numeración decimal se dispone sólo de diez símbolos deberán crearse seis nuevos símbolos, ya que los números 1Ø a 15 no pueden utilizarse en este caso, porque darían lugar a confusión. El sistema hexadecimal utiliza, por tanto, los símbolos Ø...9 y las letras A, B, C, D, E, F. En la siguiente tabla se da la correspondencia entre hexadecimal, decimal y un código binario de 4 bits.

Ø = Ø = ØØØØ	4 = 4 = Ø1ØØ	8 = 8 = 1ØØØ	C = 12 = 11ØØ
1 = 1 = ØØØ1	5 = 5 = Ø1Ø1	9 = 9 = 1ØØ1	D = 13 = 11Ø1
2 = 2 = ØØ1Ø	6 = 6 = Ø11Ø	A = 1Ø = 1Ø1Ø	E = 14 = 111Ø
3 = 3 = ØØ11	7 = 7 = Ø111	B = 11 = 1Ø11	F = 15 = 1111

Hasta aquí todo parece bastante sencillo, pero, ¿qué sucede a la hora de utilizar estos números? La verdad es que resultan bastante fáciles de manejar. Un número binario se divide en grupos de 4 bits (nibbles), de derecha a izquierda. Cuando el último grupo a la izquierda tenga menos de cuatro cifras se completará con ceros a la izquierda hasta completar el grupo de 4 bits. Por ejemplo:

$$\underbrace{(Ø)11Ø1Ø1Ø111ØØ}_{6\quad A\quad B\quad C} = 6ABC_{16}$$

El subíndice 16 detrás de 6ABC indica que estamos utilizando cógido hexadecimal. Este índice se utiliza muy pocas veces, pero lo hemos utilizado aquí para familiarizar a los lectores con él.

Para evitar confusiones, muy pronto toda la información y especificaciones de los compuadores se dieron en código hexadecimal, incluso sin tratarse de un número binario. Debe resaltarse, por tanto, que el sistema hexadecimal no es un binario abraviado, sino que se trata de un sistema completo de numeración de base 16. El número $6ABC_{16}$ puede desarrollarse como sigue:

$6 \times 16^3 + A \times 16^2 + B \times 16^1 + C \times 16^0 = 6 \times 4096 + 10 \times 256 + 11 \times$
$\times 16 + 12 \times 1 = 27.324_{10}$ (decimal).

Dejamos al lector el comprobar que se obtiene el mismo resultado si se pasa el número binario del que procede 6ABC, directamente a decimal.

La razón por la que todo «funciona» tan maravillosamente es porque 16 es una potencia de dos, dos es la base del sistema binario y $2^4 = 16$. Obsérvese aquí la potencia de cuatro y que el número binario fue dividido en grupos de cuatro.

Existe también un sistema ya algo antiguo, llamado octal, que dividía el número binario en grupos de tres. Aquí la base no es 16 sino que, como el nombre sugiere, es ocho. Los símbolos utilizados son los números cero a siete. Si observamos, una vez más, las potencias de dos, veremos que $2^3 = 8$. En el futuro es posible que se utilice un sistema de base 32, pero esto supondrá un considerable esfuerzo por parte del operador, ya que deberá memorizar 22 nuevos símbolos (los siguientes a los Ø...9 utilizados normalmente).

BCD

Como se ha dicho anteriormente, BCD es una abreviación de Binary Coded Decimal. En este código a cada símbolo decimal se le asigna un número binario de 4 bits, tal como se indica a continuación:

Ø = ØØØØ
1 = ØØØ1
2 = ØØ1Ø
3 = ØØ11
4 = Ø1ØØ
5 = Ø1Ø1
6 = Ø11Ø
7 = Ø111
8 = 1ØØØ
9 = 1ØØ1

La expresión de un númnero decimal en código BCD consiste simplemente en sustituir cada cifra por su correspondiente número binario de 4 bit. Por ejemplo: el número 1981 expresado en BCD tendrá la forma

0001	1001	1000	0001
1	9	8	1

Como puede verse, éste número no tiene nada que ver con el equivalente binario del mismo número que, como recordaremos, era:

11110111101

Dividiendo el número BCD en grupos de cuatro es muy sencillo deducir el número decimal que representa. Una desventaja importante del sistema BCD radica en la dificultad de realizar cualquier operación matemática. En la figura 1 se da una tabla de todos los sistemas de numeración que el operador utilizará en el Junior Computer.

binario	decimal	hexa-decimal	nibble	BCD
0	0	0	0000	0000
1	1	1	0001	0001
10	2	2	0010	0010
11	3	3	0011	0011
100	4	4	0100	0100
101	5	5	0101	0101
110	6	6	0110	0110
111	7	7	0111	0111
1000	8	8	1000	1000
1001	9	9	1001	1001
1010	10	A	1010	
1011	11	B	1011	
1100	12	C	1100	
1101	13	D	1101	
1110	14	E	1110	
1111	15	F	1111	
10000	16	10		
10001	17	11		
.	.	.		
.	.	.		
.	.	.		
.	.	.		

Figura 1. Diversos códigos de numeración utilizados por el operador del Junior Computer. En el sistema binario se han eliminado los ceros a la izquierda por no ser necesarios. Exactamente lo mismo se ha hecho con el sistema decimal, escribiendo 7 en vez de 07, etc.

Conversión binario/decimal-decimal/binario

Muy a menudo es necesario convertir un número decimal en su equivalente binario y viceversa. Pasar de binario a decimal es muy sencillo. Como sabemos, cada vez que un uno está en el número binario le corresponde una potencia de dos. Todas las potencias de dos así obtenidas se suman y el resultado obtenido es el número decimal equivalente. Veamos un ejemplo:

$$11\emptyset1\emptyset111 = 2^{\emptyset} + 2^{1} + 2^{2} + 2^{4} + 2^{6} + 2^{7}$$
$$= 1 + 2 + 4 + 16 + 64 + 128 = 215$$

La conversión de decimal a binario es un poco más complicada, como puede verse en la figura 2. Básicamente se trata de dividir por dos el número y sucesivamente los cocientes obtenidos mientras se pueda, y escribir en una línea aparte los restos de cada división. La secuencia de restos así obtenida representa el equivalente binario del número. Señalemos aquí que aunque, por supuesto, es posible convertir directamente un número decimal en su equivalente hexadecimal y viceversa, resultará mucho más fácil obtener primero el número en binario para pasar a continuación a hexadecimal.

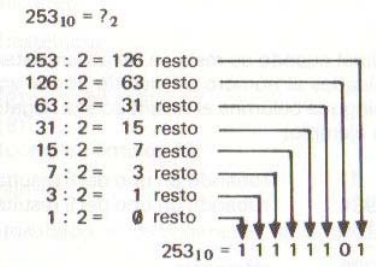


Figura 2. La conversión de un número decimal a binario se hace dividiendo repetidamente el número decimal por dos. El resto de la división, dependiendo de si es, uno o cero, se escribe en forma binaria en la parte inferior.

Aritmética binaria

Los números binarios pueden ser sumados, restados, multiplicados o divididos como cualquier otro número. Los principios de operación son exactamente los mismos que en el sistema decimal; sin embargo, dado que en el sistema binario la base es dos se producen ciertas simplificaciones.

Suma

Cuando se suman dos números decimales se produce un acarreo si la cantidad obtenida en una columna sobrepasa el valor 9.

```
  1        se acarrea un uno de la columna de unidades
 129
+243
----
 372      suma
```

Del mismo modo se producen acarreos en la suma binaria. Sólo que esto sucede siempre que la cantidad obtenida en una columna excede de uno (éste es un sistema de base dos no de base diez). Otro ejemplo aclarará esto:

```
  1111 1        se acarrea un uno de la columna precedente
   10110 1
+   1010 1
  1000010       suma
```

Lógicamente se aplican todas las reglas normales de las matemáticas:

```
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 y 1 de acarreo = 10
```

Estas reglas básicas pueden verse bastante claramente en la columna binaria de la figura 1: cada número es igual al precedente más uno.

Resta

En el sistema decimal cuando se resta se utiliza un sistema de «pedir prestado» o «robar» unidades al número adyacente por la izquierda con el fin de asegurar que en ninguna columna el resultado sea negativo, tal como se indica en el siguiente ejemplo:

```
    11      robando un uno del 8 resultan 14 unidades
  1984      robando un uno del 9 resultan 17 decenas
-  199
------
  1785      diferencia
```

Con números binarios el sistema es el mismo:

```
   111111       cantidades «robadas»
  1100000 1
-   111111 0
-----------
  0100001 1     diferencia
```

Las reglas de la sustracción aprendidas en la escuela siguen siendo válidas.

```
0 - 0 = 0
1 - 0 = 1
1 - 1 = 0
0 - 1 = 1 después de «robar», ya que 10 - 01 = 01
```

De nuevo en este caso pueden comprobarse estas reglas en las columnas binarias de la figura 1. Partiendo del extremo inferior y siguiendo hacia arriba se observará que cada número se obtiene restando uno del anterior.
Este método de sustracción es muy sencillo para una persona, pero bastante complicado para el computador. Existe un método mucho más elegante en el

que la sustracción puede realizarse como una suma, ¿le parece imposible? Siga leyendo. El método que utiliza el computador se llama complementación y suma. El complemento de un número binario se obtiene invirtiendo cada bit de ese número. Así, el complemento de 1001 es 0110. El computador realiza la sustracción de dos números complementando el sustraendo (número que va a ser restado), y sumando el resultado al minuendo (número del que debe restarse). El resultado del acarreo obtenido al sumar los dos bits más significativos los utiliza como se indica a continuación: si el bit de acarreo es uno quiere decir que el resultado es un número positivo y que el resultado final se obtiene sumando el bit de acarreo al bit menos significativo (lado derecho); por otra parte, cuando el bit de acarreo es cero el resultado de la resta es negativo y se obtiene complementando la suma obtenida (resultando intermedio). Un par de ejemplos servirán para clarificar la situación.

```
  1001  (9) – 0011 (3)
  1001  (9)
  1100  (complemento de 3)
 10101  (suma)
     1  (acarreo)
 +0110  (resultado)
```

Respuesta = +6

```
  1000  (8) – 1100 (12)
  1000  (8)
  0011  (complemento de 12)
 01011  (suma)
 -0100  (resultado = complemento de la suma con signo menos)
```

Respuesta = –4

Esto puede parecer bastante pesado para un ser humano, pero resulta ser muy sencillo para el computador y, por tanto, es el método más rápido y eficaz de sustracción que puede utilizar.

Multiplicación

Se aplican aquí también las mismas reglas que para la multiplicación decimal. Para empezar, multiplicaremos entre sí los números decimales 147 y 231:

```
  147  multiplicando
x 231  multiplicador
-----
  147
 441
+294
-----
=33957
```

El resultado de la multiplicación de 147x1 se escribe justo debajo del multiplicador. El producto de 147x3 es como si fuera el producto de 147x30 y el resultado se desplaza un lugar a la izquierda; finalmente, el resultado de 147x2

se desplaza dos lugares a la izquierda porque en realidad es el resultado de multiplicar 147x200. Estos tres resultados parciales, colocados en la situación que acabamos de indicar, se suman para producir el resultado final (33.957). Si un dígito del multiplicador es uno (excepto en las unidades) lo único que hay que hacer es colocar desplazado el multiplicando un lugar a la izquierda. Si el dígito del multiplicador es un cero, el resultado parcial es también cero y normalmente no se escribe, pero el siguiente resultado parcial se desplaza dos lugares hacia la izquierda. Como en el sistema binario sólo existen ceros y unos la multiplicación se reducirá a sumar y desplazar el multiplicando. Veamos un ejemplo en el que se multiplican los números binarios 1011 y 1010.

```
    1011   multiplicando
  x 1010   multiplicador
  ------
    0000
   1011
  0000
 1011
+ 1                 1 de acarreo por la suma
-------
 1101110   producto
```

El computador realizará esta operación paso a paso como sigue:

	1011 (11) × 1010 (10)		
bits del multiplicador	multiplicando	1011	
LSB = 0	escribe ceros		0000
bit 2 = 1	desplaza el multiplicando	10110	
	Suma		10110
bit 3 = 0	desplaza el multiplicando no suma	101100	10110
MSB = 1	desplaza el multiplicando	1011000	
	suma		1101110
	resultado		1101110

División

Veamos, en primer lugar, un ejemplo de división decimal; se trata de dividir el número 2091 por el número 17.

```
2091 : 17 = 123
17
--
 39
 34
--
  51
  51
  --
  00
```

La división binaria es exactamente igual, pero mucho más sencilla. Si el resto, después de «bajar» la siguiente cifra, es mayor que el divisor, se coloca un uno en la correspondiente posición del cociente, por el contrario, si el resto

es menor que el divisor, en el cociente se escribe un cero. Como ejemplo, el número 100010010101 (= 2197) se dividirá por 1101 (= 13), para dar el resultado 10101001 (= 169).

```
1000100101Ø1 : 1101 = 10101001
 1101
 ----
 001000
  0000
  ----
  10000
   1101
   ----
   000111
    0000
    ----
    01110
     1101
     ----
     000011
      0000
      ----
      00110
       0000
       ----
       01101
        1101
        ----
        0000
```

En el ejemplo anterior se han incluido todas las operaciones con el fin de aumentar la claridad. Sin embargo, en la práctica, se omiten aquellos resultados en los que el producto es cero y se «bajan» tantos números del dividendo como sea necesario hasta obtener un número mayor que el divisor. De este modo, el cálculo anterior puede escribirse mucho más sencillamente como sigue:

```
1000100101Ø1 : 1101 = 10101001
 1101
 ----
  10000
   1101
   ----
    1110
    1101
    ----
     1101
     1101
     ----
     0000
```

Conviene señalar aquí que si hay que realizar operaciones en el sistema hexadecimal suele ser más sencillo pasar primero a sistema binario, realizar la operación y volver al sistema hexadecimal.

Números negativos

Hasta ahora, hemos realizado todos los cálculos con números positivos; sin embargo, puede suceder (y de hecho sucede) que tengamos que manejar números negativos.

No tiene absolutamente ningún sentido colocar un signo menos (—) delante de un número binario, porque el computador no lo entiende. Lógicamente, sin embargo, existen diversos métodos de representar los números binarios negativos. Nosotros nos limitaremos a describir el método más cómodo, y que es el utilizado en el Junior Computer.

El microprocesador 6502 utilizado en el Junior Computer utiliza palabras de 8 bits de longitud (un byte). Esto significa que pueden manejarse 256 combinaciones diferentes de ceros y unos, es decir, todas las comprendidas (e incluidas) entre 00000000 y 11111111 (en la hexadecimal entre 00 y FF, y en decimal entre 0 y 255). Los números 00000000...11111111 se representan en la figura 3 de modo semejante a una cinta métrica en la que la distancia entre cada dos números es la misma y la máxima que puede existir entre dos números es de 255 unidades. La distancia total no puede, pues, exceder de 11111111, porque no existe un noveno bit. En el capítulo 3 veremos que a veces se utiliza un noveno bit llamado bandera (flag) de acarreo.

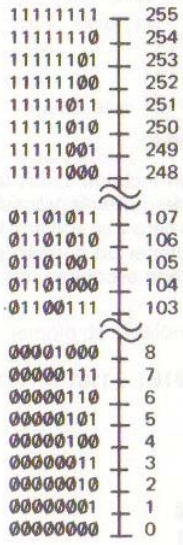


Figura 3. Línea de números para todos los números de ocho bits posibles.

¿Y qué tiene que ver esto con los números negativos?, puede usted preguntarse. Coja siete bits de los ocho que se disponen en un byte. Tendremos, de esta manera, las combinaciones posibles entre 00000000 y 01111111 (ambas incluidas); es decir, un total de 128 combinaciones (justo la mitad de las que se obtenían con una palabra de ocho bits). Estas combinaciones se utilizan para representar los números positivos del sistema binario.

Los números negativos se representan como sigue:
Partiendo de cero y restando la unidad se obtiene:

```
  00000000
- 00000001
----------
  11111111   = cero menos uno = -1
- 00000001     (restamos otra vez)
----------
  11111110   = -2
- 00000001     (y otra vez)
----------
  11111101   = -3
```

Continuando de este modo se obtendrá una línea con su centro en cero y cuyos dos extremos serán + 127 y — 128 (ver figura 4). Los números negativos así obtenidos están dados en la notación llamada «complemento de dos».

No es intención de este libro confundir al lector con fórmulas maravillosas y fantásticas, por tanto, nos limitaremos, en lo que sigue, a mostrar cómo se realizan las diversas operaciones matemáticas. Un número complemento de dos se obtiene primero realizando la complementación del número binario y añadiendo después un uno al resultado (como se ha dicho antes, complementar un número bianario consiste simplemente en invertir cada bit de un número). Como un ejemplo, el número decimal positivo 3 se representa por 00000011, y al sustituir todos los ceros por unos y viceversa (inversión) resulta 11111100. Si ahora sumamos 1 (00000001) obtendremos 11111101 que es el equivalente de —3 en binario (ver figura 4).

```
01111111 ┼  127   $7F
01111110 ┼  126   $7E
01111101 ┼  125   $7D
         ≈
00000111 ┼  7     $07
00000110 ┼  6     $06
00000101 ┼  5     $05
00000100 ┼  4     $04
00000011 ┼  3     $03
00000010 ┼  2     $02
00000001 ┼  1     $01
00000000 ┼  0     $00
11111111 ┼  -1    $FF
11111110 ┼  -2    $FE
11111101 ┼  -3    $FD
11111100 ┼  -4    $FC
11111011 ┼  -5    $FB
11111010 ┼  -6    $FA
11111001 ┼  -7    $F9
11111000 ┼  -8    $F8
         ≈
10000010 ┼  -126  $82
10000001 ┼  -127  $81
10000000 ┼  -128  $80
```

Figura 4. Línea de números para todos los números de ocho bits (positivos y negativos) realizada con el método de la complementación. El signo dólar ($) precede a los equivalentes hexadecimales.

De este modo, es posible producir números negativos a costa del octavo bit (como sabemos, el número total de variaciones que se pueden realizar depende del número de bits disponibles). Los números binarios negativos siempre empiezan con un uno (el bit situado más a la izquierda) mientras que los números binarios positivos con igual longitud de palabra siempre empiezan por un cero (este cero puede eliminarse si se desea). De este modo, el bit más significativo (MSD = Most Significant Bit) puede asociarse al signo positivo-negativo. En la figura 5 se dan varios números binarios negativos junto con su correspondiente código hexadecimal.

binario	decimal	hexa-decimal
11111111	– 1	FF
1111111Ø	– 2	FE
1111110Ø	– 3	FD
111111ØØ	– 4	FC
11111Ø11	– 5	FB
11111Ø1Ø	– 6	FA
11111ØØ1	– 7	F9
11111ØØØ	– 8	F8
11110111	– 9	F7
11110110	–1Ø	F6
11110101	–11	F5
11110100	–12	F4
11110011	–13	F3
11110010	–14	F2
11110001	–15	F1
11110000	–16	FØ
11101111	–17	EF
1110111Ø	–18	EE
11101101	–19	ED
1110110Ø	–2Ø	EC
11101011	–21	EB
1110101Ø	–22	EA
11101001	–23	E9
1110100Ø	–24	E8
11100111	–25	E7
11100110	–26	E6
11100101	–27	E5
11100100	–28	E4
11100011	–29	E3
1110001Ø	–3Ø	E2
11100001	–31	E1
11100000	–32	EØ
11011111	–33	DF
1101111Ø	–34	DE
11011101	–35	DD
1101110Ø	–36	DC
11011011	–37	DB
1101101Ø	–38	DA
11011001	–39	D9
1101100Ø	–4Ø	D8

Figura 5. Esta tabla contiene una serie de números binarios negativos junto con sus equivalentes decimal y hexadecimal.

Ejercicios

Con los siguientes ejercicios podremos medir cuanto hemos aprendido sobre los sistemas de numeración binario y hexadecimal.

1. *Pasar los siguientes números decimales al sistema binario.*
   - *a) 16*
   - *b) 24*
   - *c) 125*
   - *d) 513*
   - *e) 756*
2. *Pasar los siguientes números binarios al sistema decimal.*
   - *a) 0111*
   - *b) 1001*
   - *c) 1100101*
   - *d) 1011011*
   - *e) 1110010101*
3. *Pasar los siguientes números decimales a BCD.*
   - *a) 12*
   - *b) 37*
   - *c) 128*
   - *d) 412*
   - *e) 3762*
4. *Pasar los siguientes números a BCD a decimal.*
   - *a) 1001*
   - *b) 0101*
   - *c) 10000110*
   - *d) 00111000*
   - *e) 100101110010*
5. *Pasar los siguientes números binarios a hexadecimal.*
   - *a) 00101111*
   - *b) 11111*
   - *c) 101000111*
   - *d) 110101010*
   - *e) 001011*
6. *Pasar los siguientes números hexadecimales a binario.*
   - *a) 132*
   - *b) A014*
   - *c) 0356*
   - *d) C5E1*
   - *e) ABBA*
7. *Realizar las siguientes operaciones binarias.*
   - *a) 01001111 + 11000111*
   - *b) 1110011 + 11111111*
   - *c) 11111111 + 1*
   - *d) 11110000 + 1111*
   - *e) 10101010 + 1010101*
   - *f) 01110100 – 1101*
   - *g) 11110000 – 1111*
   - *h) 10111000 – 10000001*

i) 10101111 – 10101111
j) 100 – 11111011
k) 11110001 x 01111
l) 101 x 11111111
m) 1010 x 1010
n) 11 x 11111111
o) 10000000101 ÷ 111
p) 110100000000 ÷ 1101
q) 100110111101100010 ÷ 1001110

8. Realizar las siguientes operaciones hexadecimales.
a) A + B
b) D3 – 3E
c) ABBA x 4
d) B9A0 ÷ 0B

Soluciones

1: a) 10000
b) 11000
c) 1111101
d) 1000000001
e) 1011110100

2: a) 7
b) 9
c) 101
d) 91
e) 917

3: a) 00010010
b) 00110111
c) 000100101000
d) 010000010010
e) 00110111011000010

4: a) 9
b) 5
c) 86
d) 38
e) 972

5: a) 2F
b) 1F
c) 147
d) 1AA
e) 0B

6: a) 000100110010
b) 1010000000010100
c) 0000001101010110
d) 1100010111100001
e) 1010101110111010

7: a) 100010110
b) 101110010

c) 100000000
d) 11111111
e) 11111111
f) 1100111
g) 11100001
h) 110111
i) 0
j) 100001001 (= —247 en notación complemento a dos de 9 bits)
k) 111000011111
l) 10011111011
m) 1100100
n) 101111101
o) 10010011
p) 100000000
q) 1111111111

8: a) 15
b) 95
c) 2AEE8
d) 10E0

3

Organigramas

Análisis de los problemas mediante organigramas

Para realizar cualquiera de las múltiples funciones que puede ejecutar (juegos, contabilidad, control de aparatos domésticos, etc.) el Junior Computer debe ser previamente programado por el usuario. Y antes de programar el computador es necesario un análisis detallado de cada problema en particular. Para programas sencillos y cortos es posible pasar directamente del problema al programa utilizando el conjunto de instrucciones del microprocesador (ver capítulo 4), pero para programas más complicados y largos esto no es posible, y se recurre a los organigramas.

Un *organigrama* (al que se designa también con los nombres de ordinograma y diagrama de flujo) sirve para dar una visión de conjunto del proceso de solución del problema, desde el principio hasta el final. Asimismo indica todos aquellos puntos del programa en los que hay que tomar decisiones o realizar comprobaciones; también indica las operaciones que hay que realizar para obtener un determinado resultado, etc. En los puntos de toma de decisiones puede haber diversos caminos que pueden tomarse dependiendo del resultado de una operación. Todos estos caminos pueden indicarse claramente en el organigrama. En la figura 1 se dan los símbolos más corrientemente utilizados en los organigramas, junto con su significado. El primer paso en cualquier organigrama está contenido en el llamado símbolo *terminal* con la palabra *comienzo* en su interior. El mismo símbolo con la palabra *final* se utiliza para indicar la conclusión del programa. Las *operaciones* de un programa están contenidas en un rectángulo y el texto interior a ese rectángulo define la operación a realizar. Este texto debe ser lo más corto posible y debe contener los mínimos términos necesarios para comprender la operación. En el ejemplo dado en la figura 1, a la letra A se la asigna el valor de la suma B + C. Esto queda suficientemente indicado con A = B + C, y los pasos como llamar

a A, llamar a B, etc., se omiten. Una vez realizada una operación puede ser necesario tomar una decisión sobre el resultado. Las *decisiones* se indican en el organigrama mediante rombos. En el ejemplo, el valor de A se compara con el valor de D y se toma una decisión dependiendo de si los dos valores son iguales o no. La decisión puede dirigirnos al siguiente símbolo de la figura 1, que corresponde a una instrucción de *entrada-salida*. Esto se indica en el organigrama mediante un paralelogramo. En esta etapa se puede introducir o sacar información del computador. Por ejemplo, el operador puede soli-

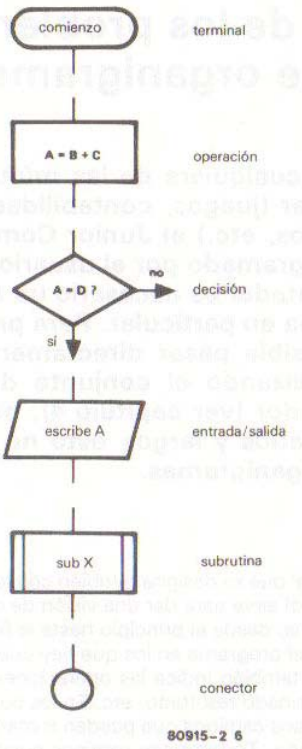


Figura 1. Símbolos más comunes utilizados en los organigramas. Existen muchos más, pero la mayoría de los programas sólo requieren la utilización de los aquí presentados.

citar del computador un resultado intermedio o el computador puede necesitar un nuevo valor para una determinada variable.

Hay ocasiones en que una cierta operación o serie de operaciones tiene que repetirse varias veces. Cuando esto sucede es usual encerrar la secuencia en un pequeño programa o *subrutina* dentro del programa principal. El símbolo

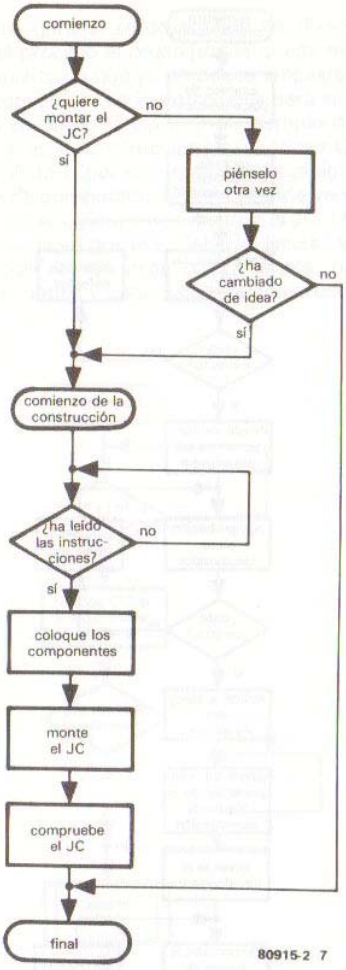


Figura 2. Organigramas general de la construcción del Junior Computer.

de una subrutina es un rectángulo con dos líneas verticales a cada lado. Cada subrutina, por supuesto, tendrá su propio organigrama. Lógicamente, los programas largos y complicados tendrán organigramas que pueden ocupar varias páginas. Las distintas partes de un organigrama pueden unirse mediante el símbolo llamado *conector* que se muestra también en la figura 1. Utilizando estos símbolos es posible esbozar cualquier programa. Una vez

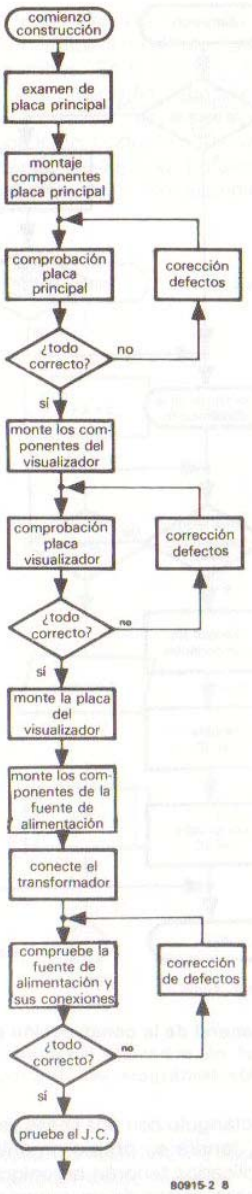


Figura 3. El organigrama de la figura 7 ha sido desarrollado hasta llegar a los pasos de construcción mencionados en el capítulo 1.

obtenido el organigrama general se desarrollan las diversas subrutinas. A medida que continúa el proceso el organigrama queda más y más definido hasta que llega un momento en que ya no puede ampliarse más y los distintos símbolos pueden convertirse en instrucciones para el Junior Computer. A modo de ilustración se da en la figura 2 un ejemplo de organigrama. El problema estudiado en él es la construcción de un Junior Computer (JC), según se explica en el capítulo 1 (por supuesto este organigrama nunca se traducirá en un programa de computador). Cómo puede verse la primera decisión a tomar se refiere a si se quiere o no construir el JC. Una vez tomada esta decisión y si la respuesta es positiva, deberán leerse las instrucciones de construcción para colocar los diversos componentes, para realizar el ensamblaje de las distintas partes y para realizar la construcción del sistema; si

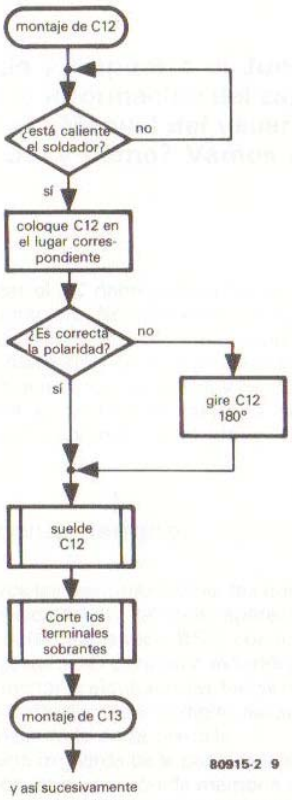
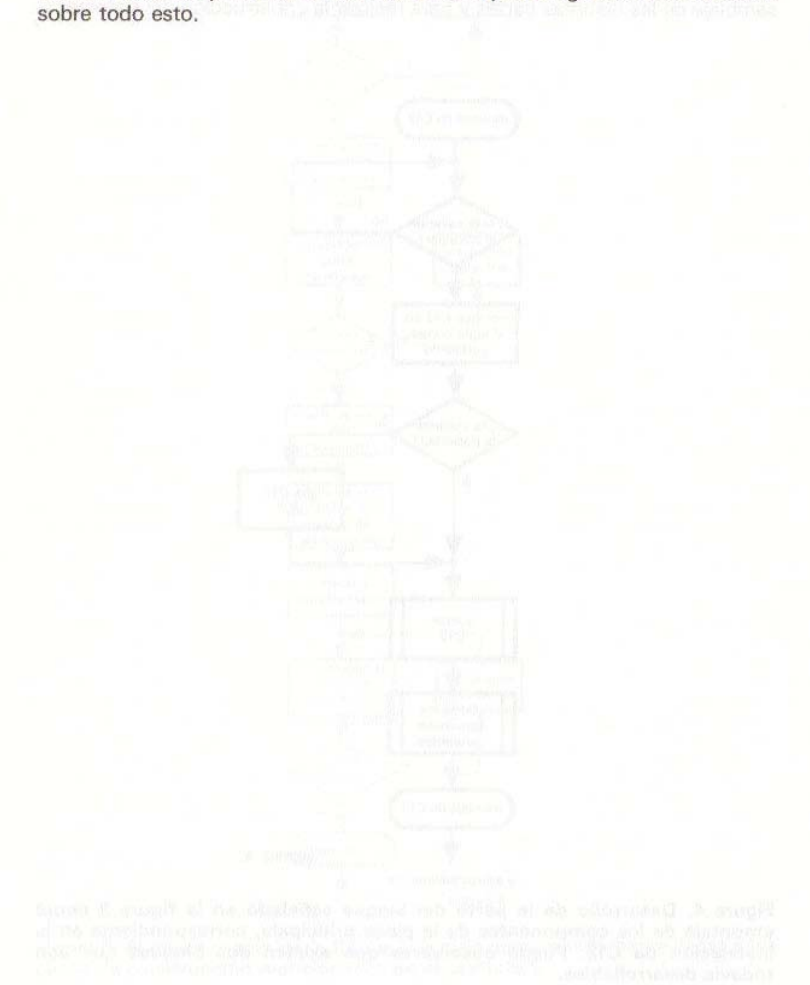


Figura 4. Desarrollo de la parte del bloque señalado en la figura 3 como «montaje de los componentes de la placa principal», correspondiente en la instalación de C12. Puede observarse que existen dos bloques que son todavía desarrollables.

la respuesta ha sido no, el organigrama invita al lector a reconsiderar su decisión. Si la respuesta sigue siendo negativa, el lector es enviado directamente al final del organigrama. Por otro lado, si el lector ha cambiado de manera de pensar es reenviado a la corriente principal del programa.

El organigrama de la figura 2 es solamente un primer esbozo. En la figura 3 está ya más desarrollado. Puede verse que todos los pasos mencionados en el capítulo 1 aparecen en el organigrama de la figura 3. Avanzando un paso más, podemos obtener un organigrama que indique la instalación de cada componente en la tarjeta principal, tal como se indica en la figura 4. En ella pueden verse dos indicativos de subrutina que indican que la operación indicada no está completamente definida. En el capítulo siguiente abundaremos sobre todo esto.



4

Programación

Cómo manejar el Junior Computer

Una vez completado y dispuesto el Junior Computer y supuesta asimilada la información del capítulo segundo, vamos a entrar en el «Manual del usuario» ¿Qué podemos hacer con el JC y cómo? Vamos a verlo en este capítulo.

Para ser capaces de utilizar el JC deberemos adquirir previamente algunos conocimientos de programación. No obstante, no hay que preocuparse: programar será bastante divertido. Al final de este capítulo le será posible desarrollar sus propios (de momento, cortos) programas y conseguir del JC más o menos lo que usted quiera. El capítulo quinto, por otra parte, trata de programas más complejos y da también algunos esquemas lógicos de programación que nos serán muy útiles en el futuro. Pero primero...

Familiaricémonos con el terreno

El terreno será para nosotros las veintitrés teclas, los dos conmutadores y los seis dígitos de la versión básica del JC, tal como aparece en la figura 1a. Una vez conectado el sistema pulsemos la tecla RST, con lo cual se inicia el funcionamiento del microprocesador. El programa monitor comienza entonces a sondear el teclado para detectar si alguna de las teclas ha sido pulsada (y, en su caso, cuál). Si se pulsa cualquier tecla de dígito hexadecimal (Ø...F), el carácter en cuestión será presentado en la pantalla.
Los cuatro dígitos de la parte izquierda de la pantalla nos mostrarán, en código hexadecimal, la dirección de la posición de memoria de que se trata en cada caso y los dos del lado derecho nos darán el contenido de esta posición de memoria. En principio, es posible utilizar todas las direcciones desde ØØØØ hasta FFFF.
Supongamos que queremos introducir algunos datos en un área dada de me-

moria. A partir de la dirección 0200 se desea introducir los datos 18, A9, 03, etc. Las secuencias de operaciones sería como sigue:

				dirección	dato				
RST				xxxx	xx				
AD				xxxx	xx				
0	2	0	0	0200	xx				
DA				0200	xx				
		1	8	0200	18	0200	1	8	CLC
+		A	9	0201	A9	0201	A	9	LDA
+		0	3	0202	03	0202	0	3	
+		6	9	0203	69	0203	6	9	ADC
+		0	7	0204	07	0204	0	7	
+		8	D	0205	8D	0205	8	D	STA
+		0	0	0206	00	0206	0	0	
+		0	3	0207	03	0207	0	3	
+		4	C	0208	4C	0208	4	C	JMP
+		0	0	0209	00	0209	0	0	
+		0	3	020A	03	020A	0	3	
AD						020B	X	X	
1	A	7	A	1A7A	xx	020C	X	X	
DA				1A7A	xx	1A79	X	X	
		0	0	1A7A	00	1A7A	0	0	
+		1	C	1A7B	1C	1A7B	1	C	
						1A7C	X	X	

¿Qué hemos hecho realmente? En primer lugar, la tecla RST inicializa el programa monitor. Presionando AD hemos informado al ordenador que deseábamos cargar datos en unas ciertas direcciones de memoria. Las cruces (x) en distintos puntos del esquema indican que el dato correspondiente es irrelevante (x = don't care = irrelevante); en otras palabras, el estado de las líneas de datos correspondientes puede ser o alto o bajo, sin que ello tenga la menor importancia para el resultado final.

Hemos querido empezar en la dirección 0200. Para ello pulsamos las teclas 0, 2, 0 y 0 para obtener en la pantalla la dirección correcta. Esta dirección se halla ahora preparada para recibir datos. Tras pulsar DA seguido por 18, la dirección de memoria 0200 queda cargada con el número hexadecimal 18 (en realidad se trata de una instrucción). Hablando estrictamente «cargar» no es el término correcto puesto que la dirección 0200 no estaba vacía. En realidad la instrucción 18 reemplazó el contenido previo de la posición de memoria 0200.

La dirección en pantalla se incrementa (avanza) en una unidad al presionar la tecla (+). Los datos siguientes resultarán, por tanto, cargados en las direc-

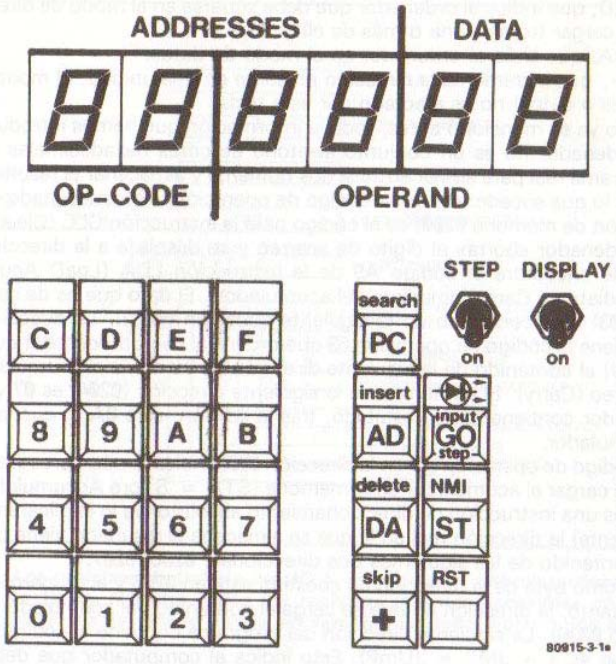


Figura 1a. El teclado del JC, junto a las 16 teclas hexadecimales y las 7 teclas de control, existen dos interruptores de función. Los textos en letras pequeñas indican las funciones de edición que el JC puede realizar. En condiciones normales los cuatros dígitos más a la izquierda muestran una dirección de memoria determinada y los dos de la derecha el contenido de esa dirección.

ciones de memoria inmediatamente siguientes a 0200. No es necesario presionar DA otra vez, salvo si los datos deben situarse en cualquier otra dirección no correlativa de memoria. Si, por ejemplo, el dato siguiente tuviera que situarse en la dirección 1A7A tendríamos que presionar primero la tecla AD.
En la parte derecha del esquema tenemos un plano de memoria que nos muestra el contenido de cada una de las direcciones que intervienen en este programa concreto. Las dos columnas de cajas representan: la dirección de memoria (a la izquierda) y el contenido de ella (a la derecha).
Ahora nos son ya familiares las siguientes teclas:
— 0...F para introducir datos y direcciones *.
— RST, que inicializa el microprocesador y activa el programa monitor y la pantalla.

* Nota: Como la mayoría de las calculadoras de bolsillo, las direcciones y datos se introducen de izquierda a derecha (en el sentido de lectura), pero la pantalla aparenta desplazarse de derecha a izquierda.

— AD, que indica al ordenador que debe situarse en el modo de direcciones para cargar (tomar) una o más de ellas.
— DA, que sitúa al ordenador en el modo de datos.
— +, que incrementa la dirección presente en una unidad. El modo (direcciones o datos) no es afectado por esta tecla.

Como ya se mencionó antes, toda la información que hemos introducido en el ordenador no es un conjunto aleatorio de cifras hexadecimales sino un programa real para sumar entre sí dos números y almacenar el resultado. Esto es lo que sucede: el primer código de operación, 18 (almacenado en la dirección de memoria 0200) es el código para la instrucción CLC (Clear Carry). El ordenador «borra» el dígito de acarreo y se desplaza a la dirección 0201 donde encuentra el código A9 de la instrucción LDA (LoaD Acumulator inmediate = Carga inmediata del acumulador). El dato que ha de ser cargado (03) está contenido en la siguiente dirección (0202). La dirección 0203 contiene el código de operación 69 que ordena al microprocesador que sume (AdD) el contenido de la siguiente dirección de memoria al acumulador con acarreo (Carry). El contenido de la siguiente dirección (0204) es 07 y el acumulador contiene 03; el resultado, tras la adición, será 0A, y quedará en el acumulador.

El código de operación 8D en la dirección 0205 indica al microprocesador que debe cargar el acumulador en la memoria (STA = STore Accumulator). Como es una instrucción de direccionamiento absoluto (ya lo explicaremos más adelante) la dirección real en la que se almacena el resultado viene dada por el contenido de las siguientes dos direcciones, 0206, 0207.

El último byte de la dirección en cuestión está en 0206 y el primero en 0207. Por tanto, la dirección en que se carga el contenido del acumulador es 0300 (y no 0030). La siguiente dirección del programa contiene el código de operación 4C (= JMP = JUmP). Esto indica al computador que debe saltar (jump, en inglés) a la dirección contenida en las dos direcciones siguientes (0209 y 020A), que en este caso es 0300. Allí es también donde hicimos cargar el resultado de la suma. El ordenador realizará entonces la operación correspondiente al código de operación 0A que es ASL-A (Aritmetic Shift Left Accumulator), es decir, desplazar el contenido del acumulador un lugar hacia la izquierda. Finalmente, como no hay instrucción en la dirección 0301, el ordenador no sabrá que hacer a continuación y se interrumpirá el programa.

Es importante darse cuenta que la operación de un programa ha de ser clara y concisa; debemos cuidar siempre al desarrollar programas que el ordenador tenga siempre algo que hacer y que ello sea lo que nosotros queramos. Programar no es más que seleccionar varias instrucciones y colocarlas en el orden correcto de forma que el computador realice unos ciertos cálculos hasta producir el resultado deseado. Como siempre podemos aplicar el viejo refrán: Haz una pregunta tonta y obtendrás una respuesta tonta.

Registros del 6502

Antes de que sigamos tratando de las posibilidades de software del JC, tales como el conjunto de intrucciones o los diferentes modos de direccionamiento es recomendable echar una ojeada a la estructura interna de registros del microprocesador 6502. El contenido de los registros internos de la CPU está continuamente a disposición del programador y puede ser manejado a través

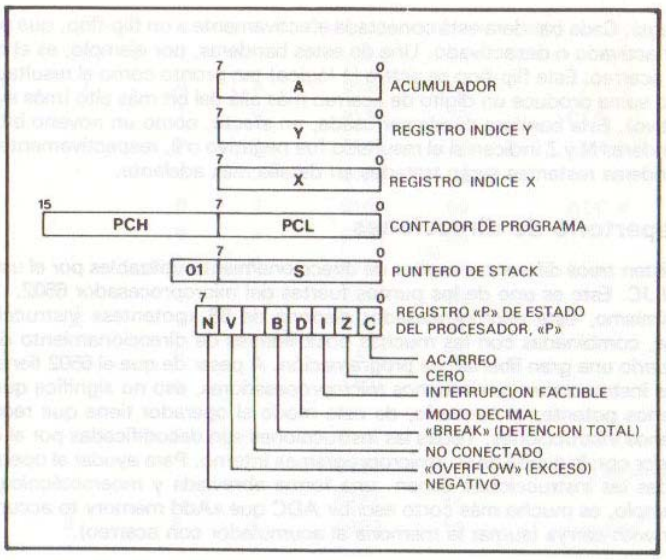


Figura 1b. Los registros de trabajo del JC o, mejor dicho, del microprocesador 6502. La figura muestra todos los registros internos accesibles por el usuario. El registro de estado podría ser también llamado registro «de bandera».

del programa. El procesador contiene seis registros internos programables, como puede verse en la figura 1b.

El rectángulo marcado A en la figura es el acumulador de ocho bits. Este registro debe utilizarse para manejar datos y transferir indicaciones desde y hacia la memoria. Se utiliza a menudo como estación de espera para datos que circulan entre posiciones de memoria.

Los registros índice X e Y (también de ocho bits) contienen datos con los cuales cualquier posición puede ser direccionada indirectamente. (Ampliaremos esto posteriormente).

El contador de programa, PC, es un registro de dieciséis bits que contiene la dirección de memoria en donde está la siguiente instrucción. El contador de programa se divide en dos registros, el PCL (L = Low) y el PCH (H = High), que contienen, respectivamente, los bytes de dirección de menor y mayor orden.

El puntero, S, se utiliza para almacenamiento temporal de direcciones de memoria. El puntero es también un registro de ocho bits, lo que significa que el stack (zona de memoria para elmacenamiento temporal de datos) tiene como máximo 256 bits. El procesador siempre considera Ø1 como el byte de mayor orden de una dirección de almacenamiento temporal, lo que significa que las direcciones de memoria Ø1ØØ...Ø1FF están asignadas permanentemente al stack.

Por último, trataremos el registro de estado P. Este registro contiene información que refleja el resultado de varias operaciones en forma de «banderas»

(flags). Cada bandera está conectada efectivamente a un flip-flop, que puede ser activado o desactivado. Una de estas banderas, por ejemplo, es el dígito de acarreo. Este flip-flop se activa (1 lógico) tan pronto como el resultado de una suma produce un dígito de acarreo más allá del bit más alto (más significativo). Esta bandera puede ser usada, en efecto, como un noveno bit. Las banderas N y Z indican si el resultado fue negativo o Ø, respectivamente. Las banderas restantes serán tratadas en detalle más adelante.

Repertorio de direcciones

Existen trece diferentes modos de direccionamiento utilizables por el usuario del JC. Este es uno de los puntos fuertes del microprocesador 6502.
Asimismo, este microprocesador dispone de 56 «potentes» instrucciones que, combinadas con las muchas posibilidades de direccionamiento dan al usuario una gran libertad de programación. A pesar de que el 6502 tiene menos instrucciones que muchos microprocesadores, eso no significa que sea menos potente; al contrario, de este modo el operador tiene que recordar menos instrucciones. Todas las instrucciones son decodificadas por el ordenador con la ayuda de su «microprograma» interno. Para ayudar al operador, todas las instrucciones tienen una forma abraviada y mnemotécnica; por ejemplo, es mucho más corto escribir ADC que «Add memory to accumulator with carry» (sumar la memoria al acumulador con acarreo).

Direccionamiento inmediato

Las instrucciones de direccionamiento inmediato se aplican a datos en la memoria de trabajo que han de ser manejados tan pronto como la instrucción es conocida. La instrucción consiste en dos bytes; el primero, para la instrucción en sí (código de operación, y el segundo, para el dato con que se opera. El símbolo # se utiliza para indicar que el número siguiente es un dato, como podremos ver en los siguientes ejemplos:
LDA # 7A significa: Cárguese el acumulador con 7A.
LDX # 3B significa: Cárguese el registro X con 3B. También podría haberse usado el registro Y y la instrucción hubiera sido: LDY # 3B.
ADC # (byte) significa: Súmese el contenido de la siguiente dirección de memoria al existente en el acumulador con dígito de acarreo. Esto puede expresarse también como: A + M + C → A. Como es lógico, el acarreo es sumado o no al resultado dependiendo del estado de la bandera de acarreo. Esta bandera debe ser siempre desactivada con la instrucción CLC antes de emprender la suma, por ejemplo. El programa puede ser:

```
CLC     Clear Carry = Borrar el contenido de la bandera de acarreo
        (C = Ø1).
LDA 13  Cargar el acumulador con 13.
ADC Ø8  Sumar 8 al acumulador.
BRK     Detenerse tan pronto como la adición se complete.
```

El último paso se necesita para completar el programa y para informar al ordenador que la tarea ha sido llevada a cabo.
Para ejecutar este programa en el JC necesitaremos primero determinar los códigos de instrucción (ver tabla al final del libro). Encontraremos que CLC = 18, LDA = A9, ADC = 69, BRK = ØØ. Una buena dirección de partida podría ser Ø1ØØ, como puede verse en la secuencia de operaciones siguientes:

				dirección:	pantalla:	
RST	AD			xxxx	xx	
Ø	1	Ø	Ø	Ø1ØØ	xx	
DA		1	8	Ø1ØØ	18	CLC
+		A	9	Ø1Ø1	A9	LDA #
+		1	3	Ø1Ø2	13	
+		6	9	Ø1Ø3	69	ADC #
+		Ø	8	Ø1Ø4	Ø8	
+		Ø	Ø	Ø1Ø5	ØØ	BRK
AD				Ø1Ø5	ØØ	
1	A	7	E	1A7E	xx	
DA		Ø	Ø	1A7E	ØØ	
+		1	C	1A7F	1C	
AD						
Ø	1	Ø	Ø	Ø1ØØ	18	
GO				Ø1Ø7	xx	ejecución del programa
AD				Ø1Ø7	xx	
Ø	Ø	F	3	ØØF3	1B	resultado

Hay algunas cosillas que conviene aclarar. ¿Por qué el salto brusco desde Ø1Ø5 a 1A7E? ¿Por qué conservar el resultado en la dirección ØØF3? Aclarémoslo diciendo que cuando el microprocesador encuentra una instrucción Break retrocede hacia el programa monitor hasta el punto indicado por los contenidos de las direcciones 1A7E y 1A7F, en este caso 1CØØ. Esta sección del programa monitor contiene una rutina de almacenamiento que asegura el contenido de cada registro es cargado en direcciones de memoria (RAM) específicas. El resultado es como sigue:

ØØEF almacena el contenido del registro PCL.
ØØFØ almacena el contenido del registro PCH.
ØØF1 almacena el contenido del registro P.
ØØF2 almacena el contenido del registro S.
ØØF3 almacena el contenido del registro A.
ØØF4 almacena el contenido del registro Y.
ØØF5 almacena el contenido del registro X.

El contenido del acumulador se almacena en ØØF3 y puesto que el acumulador contiene el resultado de la suma (en este caso 1B), ØØF3 es el lugar en que encontraremos el resultado.

La resta de números es también posible (no ocurre así con algunos pequeños microprocesadores). En este caso, la instrucción a utilizar es SBC # (byte) que significa: Restar la memoria del acumulador con acarreo invertido (ver en cap. 2 la resta de dos números binarios). Esto puede escribirse A-M-$\overline{C}$→A. Recuérdese que el JC utiliza el método de resta del complemento a dos. Para obtener el resultado completo, la bandera de acarreo debe de estar activada, con lo que C = 1, $\overline{C}$ = Ø. Esto puede ser realizado por la instrucción SEC (SEt Carry flag = Activar bandera de acarreo). El programa para restar dos números queda entonces como sigue:

SEC Código de operación 38.
LDA # 13 Código de operación A9.
SBC # 08 Código de operación E9.
BRK Código de operación ØØ.

Como antes, podemos utilizar Ø1ØØ como dirección de partida. El programa puede introducirse a través del teclado como sigue:

				dirección	dato	
RST	AD			xxxx	xx	
Ø	1	Ø	Ø	Ø1ØØ	xx	
DA		3	8	Ø1ØØ	38	SEC
+		A	9	Ø1Ø1	A9	LDA #
+		1	3	Ø1Ø2	13	
+		E	9	Ø1Ø3	E9	SBC #
+		Ø	8	Ø1Ø4	Ø8	
+		Ø	Ø	Ø1Ø5	ØØ	BRK
AD				Ø1Ø5	ØØ	
1	A	7	E	1A7E	xx	ver nota
DA		Ø	Ø	1A7E	ØØ	
+		1	C	1A7F	1C	
AD						
Ø	1	Ø	Ø	Ø1ØØ	38	ejecución del programa
GO				Ø1Ø7	xx	
AD				Ø1Ø7	xx	
Ø	Ø	F	3	ØØF3	ØB	resultado

Nota: Las direcciones 1A7E y 1A7F no necesitan ser cargadas si contienen todavía los datos del programa ejemplo previo; esto es, si el interruptor de red no ha sido desconectado entre tanto.

Como antes, la dirección ØØF3 contiene el resultado de la resta: 13 (19 en decimal) — Ø8 = ØB (11 en decimal).

Funciones lógicas

El procesador no es sólo capaz de realizar operaciones aritméticas, sino que también puede realizar funciones lógicas. La función OR es una operación lógica bien conocida que puede ser implementada (realizada) utilizando la instrucción ORA # (byte) que significa: Disyunción lógica entre memoria y acumulador (AUM → A). El código de operación para esta instrucción es Ø9. Veamos un corto ejemplo.

LDA # AA Cárguese al acumulador con AA.
ORA # ØF Realícese la función OR bit por bit con ØF.
BRK Alto (stop).

AA (en hexadecimal) = 1Ø1Ø1Ø1Ø (en binario).
ØF (en hexadecimal) = ØØØØ1111 (en binario).
Resultado tras ejecutar la función OR = 1Ø1Ø1111 (AF).

Siempre que un bit particular en el acumulador O (OR) en la dirección de memoria (segunda mitad de la instrucción) es un 1 el resultado será 1. Cuando ambos bits son Ø el resultado será también Ø. Una vez más, el resultado se almacenará en el acumulador. La figura 2a ilustra este principio vía «Hardware» (puertas lógicas OR).

Otra operación lógica bastante común es la función AND. Esta se realiza con la instrucción AND (A∩M→A). Veamos un programa corto:

LDA # AA Cárguese el acumulador con AA.
AND # ØF Conjunción de cada bit de AA con ØF.
BRK Stop.

AA en hexadecimal = 10101010.
ØF en hexadecimal = ØØØØ1111.
Resultado tras la conjunción (AND) = ØØØØ1Ø1Ø (ØA).

Siempre que dos bits correspondientes de memoria y acumulador sean 1 el resultado será 1; si cualquiera de ellos fuese Ø el resultado será Ø. La figura 2b ilustra el funcionamiento con una puerta lógica AND.

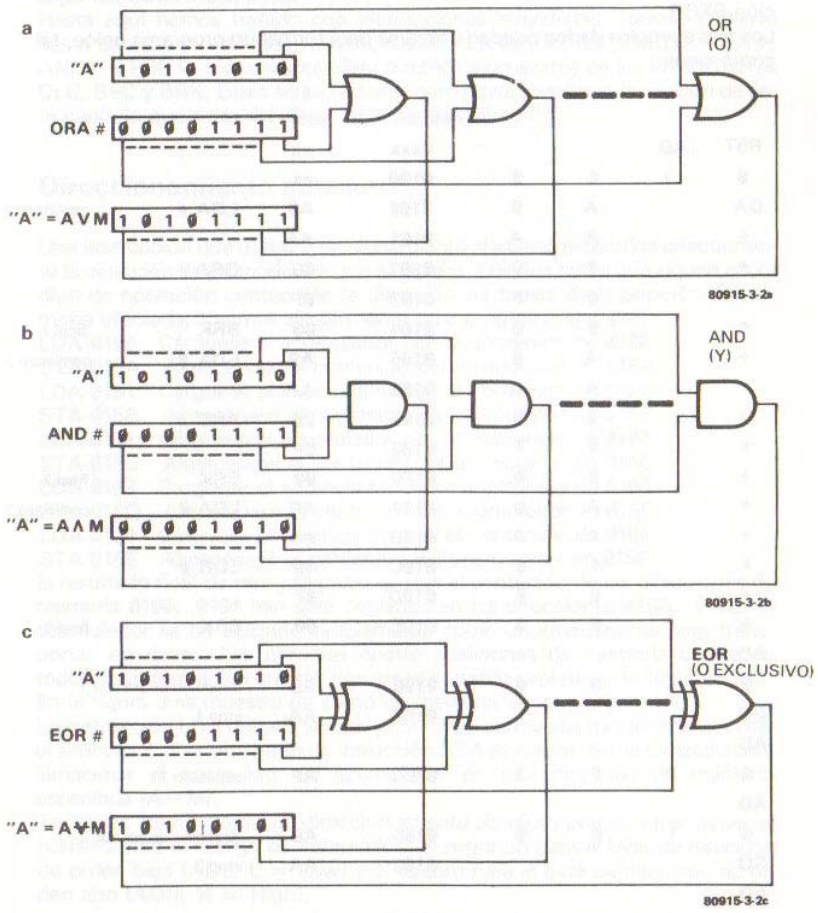


Figura 2. Esquemas simbólicos de puertas lógicas para ilustrar el uso de las funciones lógicas O (OR), Y (AND) y EOR (Exclusive OR).

La tercera función lógica que consideraremos es el O Exclusivo (función EXOR). La instrucción EXOR puede ser representada por: A $\forall$ M → A. El código de operación es 49.

LDA # AA Cárguese el acumulador con AA.
EOR # ØF O exclusivo de cada bit del acumulador con ØF.
BRK Stop.

AA (en hexadecimal) = 1Ø1Ø1Ø1Ø.
ØF (en hexadecimal) = ØØØØ1111.
El resultado tras la función EXOR = 1Ø1ØØ1Ø1 (A5).

Cuando ambos bits tienen el mismo valor el resultado es Ø; en caso contrario, el resultado es 1. Puede verse que esta instrucción es útil para invertir algunos bits en el acumulador. La figura 2c ilustra el funcionamiento de la función EXOR.

Los tres ejemplos dados pueden utilizarse para formar un programa único, tal como sigue:

				dirección	dato		
RST	AD			xxxx	xx		
Ø	1	Ø	Ø	ØlØØ	xx		
DA		A	9	ØlØØ	A9	LDA #	comienzo 1
+		A	A	Ø1Ø1	AA		
+		Ø	9	Ø1Ø2	Ø9	ORA #	
+		Ø	F	Ø1Ø3	ØF		
+		Ø	Ø	Ø1Ø4	ØØ	BRK	final 1
+		A	9	Ø1Ø5	A9	LDA #	comienzo 2
+		A	A	Ø1Ø6	AA		
+		2	9	Ø1Ø7	29	AND #	
+		Ø	F	Ø1Ø8	ØF		
+		Ø	Ø	Ø1Ø9	ØØ	BRK	final 2
+		A	9	Ø1ØA	A9	LDA #	comienzo 3
+		A	A	Ø1ØB	AA		
+		4	9	Ø1ØC	49	EOR #	
+		Ø	F	Ø1ØD	ØF		
+		Ø	Ø	Ø1ØE	ØØ	BRK	final 3
AD							
Ø	1	Ø	Ø	ØlØØ	A9	dirección de partida 1	
GO				Ø1Ø6	AA	stop 1	
AD							
Ø	Ø	F	3	ØØF3	AF	resultado 1	
AD							
Ø	1	Ø	5	Ø1Ø5	A9	dirección de partida 2	
GO				Ø1ØB	AA	stop 2	
AD							
Ø	Ø	F	3	ØØF3	ØA	resultado 2	
AD							

```
 Ø    1    Ø    A     Ø1ØA    A9    dirección de partida 3
GO                    Ø11Ø    xx    stop 3
AD
 Ø    Ø    F    3     ØØF3    A5    resultado 3
```

Como hay tres «BREAKS» en el programa, las tres partes que lo componen deberán funcionar una tras la otra. Asegúrese de introducir la dirección de partida correcta antes de presionar «GO».
Las tres instrucciones lógicas tienen utilizaciones muy importantes. Por ejemplo, pueden servir para introducir o modificar algunos bits en un byte y dejar los otros inalterados.
Hasta aquí hemos tratado con instrucciones inmediatas. Hemos cubierto hasta ahora ocho de ellas; LDA#, LDX#, LDY#, ADC#, SBC#, ORA#, AND# y EOR#. Hemos aprendido también algo acerca de las instrucciones CLC, SEC y BRK. Estas serán tratadas con mayor detalle en la sección de este capítulo que trata del direccionamiento indirecto.

Direccionamiento absoluto

Una instrucción que utilice direccionamiento absoluto especifica directamente la dirección de memoria a la que se refiere. Los dos bytes que siguen al código de operación contendrán la dirección verdadera de la posición de memoria afectada. Veamos simplemente otro programa ejemplo:

```
LDA-Ø1ØØ  Cárguese el acumulador con el contenido de Ø1ØØ *.
STA-Ø15A   Almacénese el contenido del acumulador en Ø15A.
LDA-Ø1Ø1   Cárguese el acumulador con el contenido de Ø1Ø1.
STA-Ø15B   Almacénese el contenido del acumulador en Ø15B.
LDA-Ø1Ø1   Cárguese el acumulador con el contenido de Ø1Ø2.
STA-Ø15C   Almacénese el contenido del acumulador en Ø15C.
LDA-Ø1Ø3   Cárguese el acumulador con el contenido de Ø1Ø3.
STA-Ø15D   Almacénese el contenido del acumulador en Ø15D.
LDA-Ø1Ø4   Cárguese el acumulador con el contenido de Ø1Ø4.
STA-Ø15E   Almacénese el contenido del acumulador en Ø15E.
```

El resultado final de este programa es que el contenido de las direcciones de memoria Ø1ØØ...Ø1Ø4 han sido copiados en las direcciones Ø15A...Ø15E. El acumulador se ha utilizado simplemente como un «mensajero» para transportar los datos. Las primeras cuatro posiciones de memoria contienen todavía la información original (los datos han sido copiados, no transferidos). En la figura 3 se muestra un plano de memoria de este proceso.
La instrucción LDA debería sernos ya familiar, aunque se ha utilizado aquí sin el símbolo #. Por otra parte, la intrucción STA es nueva. Se ha utilizado para almacenar el contenido del acumulador en una dirección de memoria específica (A→M).
Todas las instrucciones de direccionamiento absoluto precisan tres bytes: el primero para el código de instrucción, el segundo para el byte de dirección de orden bajo (ADL; L = Low) y el tercero para el byte de dirección de orden alto (ADH; H = High).

---

* Nota: El guión entre la instrucción y la dirección de memoria muestra que estamos utilizando direccionamiento absoluto.

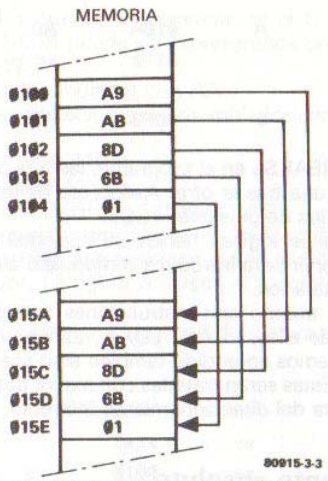


Figura 3. Plano de memoria del resultado de la transferencia de datos de un área de memoria a otra.

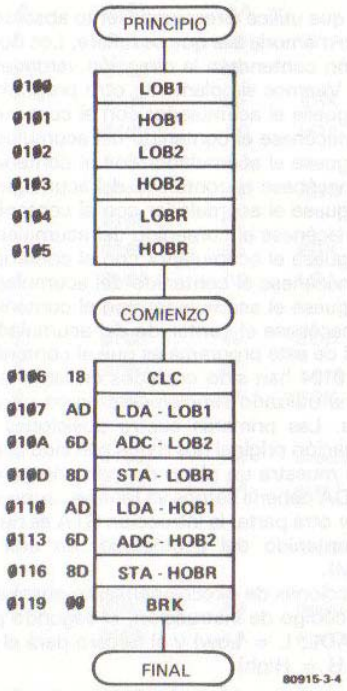


Figura 4. Organigrama (diagrama de flujo) del programa que suma dos números de 16 bits y almacena el resultado.

Vamos a echar una ojeada a un programa en el que se suman dos números de 16 bits utilizando direccionamiento absoluto. Ambos números contienen dos bytes (HOB = Byte de orden alto = High Order Byte; LOB = Byte de orden bajo = Low Order Byte). El resultado será de nuevo un número de 16 bits, pero precisará de un dígito de acarreo junto al MSB (Bit más significativo = El bit que se encuentra más a la izquierda).
Antes de entrar en el programa real es recomendable examinar el organigrama (diagrama de flujo) de la figura 4. Como puede verse, se han reservado seis direcciones de memoria (Ø1ØØ...Ø1Ø5) para los dos números de 16 bits y el resultado. El programa real no comienza hasta la dirección Ø1Ø6 y continúa hasta Ø119 inclusive.
Los dos números que vamos a sumar entre sí son Ø4EF y 23AB.

(STEP: desconectado; DISPLAY: conectado)					dirección	dato	
RST	AD				xxxx	xx	
1	A	7	A		1A7A	xx	
DA		Ø	Ø		1A7A	ØØ	rutina STEP dispuesta
+		1	C		1A7B	1C	
++					1A7D	xx	
+		Ø	Ø		1A7E	ØØ	rutina BRK dispuesta
+		1	C		1A7F	1C	
AD					1A7F	1C	
Ø	1	Ø	Ø		Ø1ØØ	xx	
DA		E	F		Ø1ØØ	EF	LOB1
+		Ø	4		Ø1Ø1	Ø4	HOB1
+		A	B		Ø1Ø2	AB	LOB2
+		2	3		Ø1Ø3	23	HOB2
+					Ø1Ø4	xx	reservado para LOBR
+					Ø1Ø5	xx	reservado para HOBR
+		1	8	CLC	Ø1Ø6	18	borra flag de acarreo
+		A	D	LDA-	Ø1Ø7	AD	
+		Ø	Ø		Ø1Ø8	ØØ	LOB1 en A
+		Ø	1		Ø1Ø9	Ø1	
+		6	D	ADC-	Ø1ØA	6D	
+		Ø	2		Ø1ØB	Ø2	A+LOB2+→A
+		Ø	1		Ø1ØC	Ø1	(incluyendo bandera de acarreo)
+		8	D	STA-	Ø1ØD	8D	
+		Ø	4		Ø1ØE	Ø4	resultado (=LOBR) a
+		Ø	1		Ø1ØF	Ø1	la dirección Ø1Ø4
+		A	D	LDA-	Ø11Ø	AD	
+		Ø	1		Ø111	Ø1	HOB1 en A
+		Ø	1		Ø112	Ø1	

```
+        6  D  ADC-   0113  6D  }  A+HOB2→A
+        0  3         0114  03  }  (incluyendo bandera de acarreo)
+        0  1         0115  01  }
+        8  D  STA-   0116  8D  }  resultado ( = HOBR) a
+        0  5         0117  05  }  la dirección 0105
+        0  1         0118  01  }
+        0  0  BRK    0119  00     fin del programa
AD
+     1  0  6         0106  18     dirección de partida
GO                    011B  xx     programa en funcionamiento
AD
0    1  0  4         0104  9A     resultado: LOBR
+                     0105  28     resultado: HOBR
```

Nota: Los contenidos de las direcciones 1A7A...1A7F se refieren a las rutinas de interrupción contenidas en el programa monitor que serán tratadas en detalle más adelante.

Ahora vamos a examinar el programa con más detalles. Los dos números a sumar eran:

```
ØDEF (hexadecimal) 00000100  11101111.
23AB (hexadecimal) 00100011  10101011.
                   ←HOB→    ←LOB→
```

En primer lugar, la bandera de acarreo es desactivada en la dirección 0106 y tras cargar el acumulador con LOB1 y sumarle LOB2 la bandera G (o dígito de acarreo) es activada:

```
  11101111   LOB1
  10101011   LOB2
+ 111 1111   acarreo
-------------
[1]10011010  LOBR
 ←9→←A→
```

La bandera del dígito de acarreo permanecerá activada hasta después del almacenamiento de LOBR (en la dirección 0104) y de la carga de HOB1 en el acumulador. Pero cuando añadimos HOB2 la bandera de acarreo debe de ser desactivada una vez más:

```
  00000100   HOB1
  00100011   HOB2
         1   acarreo procedente de LOBR
+      111   acarreo
-------------
[0]00101000  HOBR
 ←2→←8→
```

El resultado de la suma puede encontrarse en las posiciones 0104 y 0105.

La mayor parte de las instrucciones del programa que acabamos de dar utilizaban direccionamiento absoluto. Los tres bytes de estas instrucciones son fácilmente reconocibles. Hay, por supuesto, muchas otras instrucciones que utilizan direccionamiento absoluto. A continuación damos una lista de las más comunes:

Instrucciones referentes a la memoria:

LDA-Código de operación AD (M→A). Cárguese el acumulador con la memoria.
LDX-Código de operación AE (M→X). Cárguese el índice X con la memoria.
LDY-Código de operación AC (M→Y). Cárguese el índice Y con la memoria.
STA-Código de operación 8D (A→M). Almacénese el acumulador en la memoria.
STX-Código de operación 8E (X→M). Almacénese el índice X en la memoria.
STY-Código de operación 8C (Y→M). Almacénese el índice Y en la memoria.

Instrucciones aritméticas:

ADC-Código de operación 6D (A+M+C→A). Súmese la memoria al acumulador con acarreo.
SBC-Código de operación ED (A−M−$\overline{C}$→A). Réstese la memoria del acumulador con acarreo invertido.
INC-Código de operación EE (M+1→M). Increméntese la memoria en 1.
DEC-Código de operación CE (M−1→M). Disminuye la memoria en 1.
Las dos últimas instrucciones ocasionan simplemente un incremento (INC-) o decremento (DEC-) de una unidad en la memoria.

Instrucciones lógicas:

ORA-Código de operación ØD (AUM→A). Disyunción de memoria con acumulador.
AND-Código de operación 2D (A∩M→A). Cunjunción de memoria y acumulador.
EOR-Código de operación 4D (A∀M→A). Exclusivo entre memoria y acumulador.

Ejecución paso a paso

Hay dos maneras de ejecutar un programa en el Junior Computer. La primera y más obvia es introducir la dirección de partida del programa y presionar la tecla «GO» (con el interruptor STEP en posición desconectado). El programa se ejecutará entonces hasta que el procesador encuentre una instrucción BRK. El otro método es colocar el interruptor STEP en posición conectado, con lo que se encenderá el LED en la tecla STEP/GO y el programa podrá ser ejecutado instrucción por instrucción, con tal que las direcciones 1A7A y 1A7B contengan ØØ y 1C, respectivamente. Cada vez que pulsemos la tecla

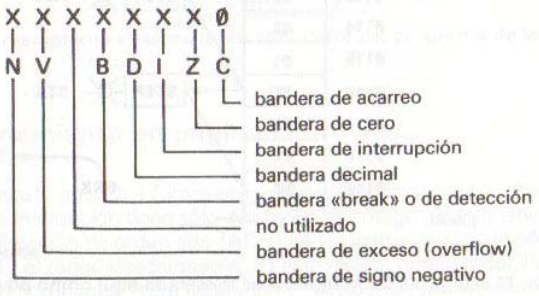


Figura 5. Disposición del registro de estado.

STEP/GO la siguiente instrucción del programa será ejecutada. Esto puede verse en el plano de memoria de la figura 6a.
Una vez que la dirección de partida (Ø1Ø6) ha sido introducida se pulsa la tecla STEP/GO y se ejecuta la primera instrucción. En este caso la instrucción es borrar el bit de acarreo. Para asegurarse de que el bit de acarreo ha sido borrado basta examinar el registro P. Como ya se mencionó anteriormente, el contenido del registro de estado está también almacenado en la dirección ØØF1. Pulsando la tecla AD seguida de esta dirección la pantalla reflejará el contenido del registro de estado.
Como puede ser que no quede bastante claro lo que la información de la pantalla realmente significa; echaremos una mirada a la disposición interna del registro de estado (fig. 5).

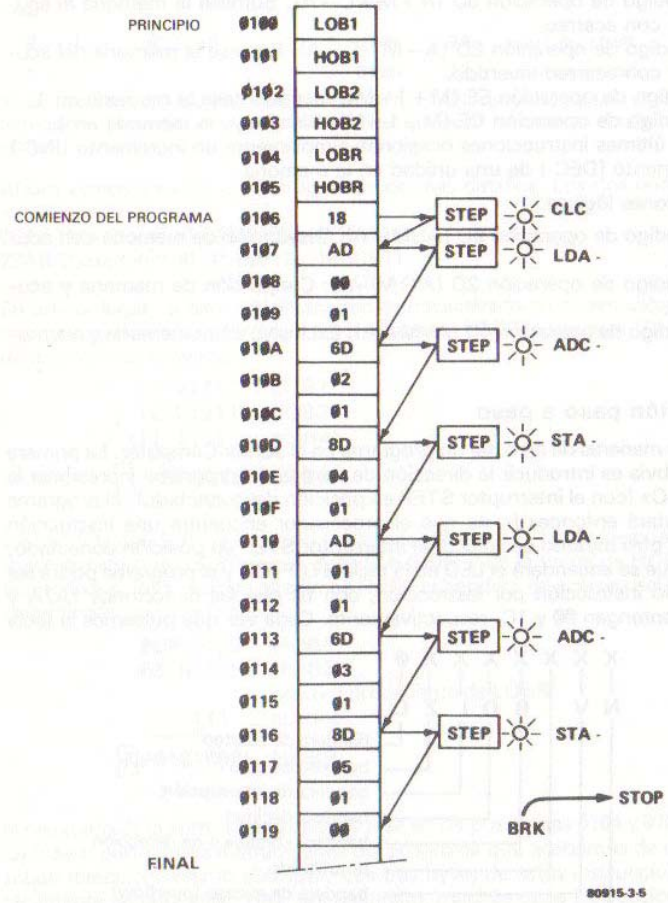


Figura 6a. El programa de la figura 4 es mostrado aquí como plano de memoria y es operado utilizando la función STEP.

Las banderas que por el momento nos son indiferentes se han señalado con una X. Nótese que la bandera de acarreo o exceso es el LSB (bit menos significativo) del registro de estado; de esta forma resulta extremadamente simple comprobar si la bandera de acarreo está o no activada, simplemente observando si el número contenido en el registro de estado es par o impar.
Volviendo atrás para recorrer el programa paso a paso, al pulsar la tecla PC, observaremos en la pantalla la cifra Ø107 AD. Como sabemos, AD es el código de operación de LDA—, la siguiente instrucción en la secuencia. Pulsando la tecla STEP/GO aparecerá ahora Ø10A 6D en la pantalla. Esto muestra que la instrucción LDA— ha sido ejecutada y el valor de LOB1 (EF) debería estar ahora en el acumulador. De nuevo podemos comprobar rápidamente esto último, si no acabamos de creérnoslo del todo.
El programa completo puede ser ejecutado paso a paso de este modo: Queda bastante claro que este modo de operación es una útil herramienta para corregir errores de programa y al mismo tiempo una gran ayuda cuando se utiliza el sistema con fines didácticos.

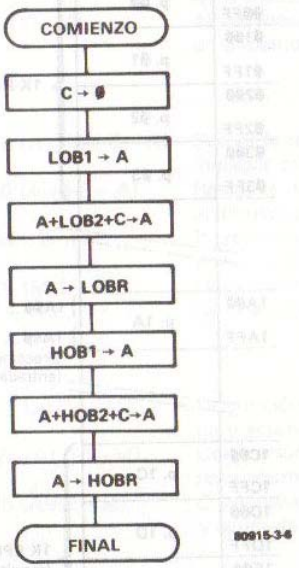


Figura 6b. Organigrama «basto» (pero completo) del programa de la figura 4.

Direccionamiento en página cero

Vamos a discutir aquí una forma especial de direccionamiento absoluto: En este caso, la instrucción tiene sólo dos bytes de código característico, ya que el byte de dirección de orden alto (ADH) de las instrucciones de página cero es siempre 0. El rango direccionable, de este modo, queda reducido a las direcciones comprendidas entre ØØØØ y ØØFF. Estas 256 posiciones de memoria

pertenecen a la «página cero». Los siguientes 256 bytes pertenecen a la página 1, etc. Esto nos lleva a la figura 7, que nos muestra la estructura en páginas de direcciones del JC.
Las páginas Ø...3 contienen la memoria RAM o memoria de trabajo. La página 1A pertenece al PIA y está dividida en RAM, direccionamiento I/O y temporizador.
Las páginas 1C...1F se reservan para el programa monitor. En el JC básico sólo pueden direccionarse las páginas ØØ...1B debido a la decodificación incompleta de las direcciones. Esto significa que la posición de memoria más alta que es posible direccionar es la 1FFF.
Puesto que el ordenador sabe que el byte de mayor orden de cada instrucción de página cero es ØØ, podemos, en consecuencia, ahorrar una posición de memoria de las tres utilizadas con direccionamiento absoluto.

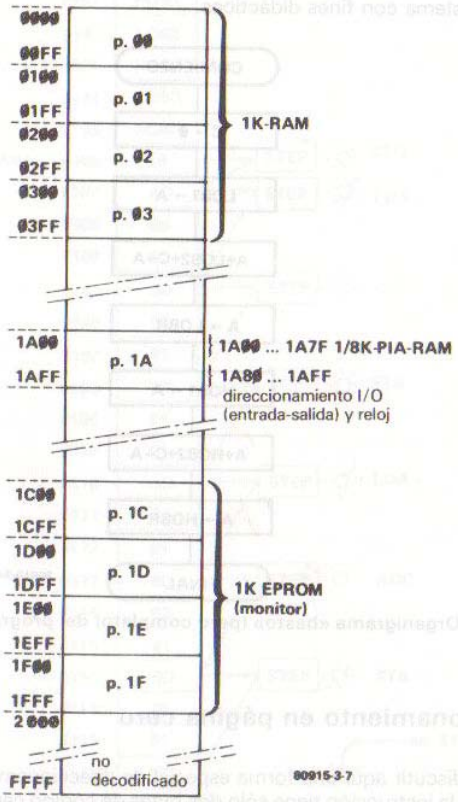


Figura 7. La estructura de direccionamiento de páginas del JC. Los bytes de orden alto (el primer byte de cada número) de todas las direcciones de una misma página son iguales.

Los códigos nemotécnicos utilizados para las instrucciones de página cero son, asimismo, utilizados para direccionamiento absoluto añadiéndoles una Z al final. De momento, nos resultan bastante familiares los siguientes:

Instrucciones referentes a la memoria:

LDAZ Código de operación A5 (M→A)	Cárguese el acumulador con la memoria.
LDXZ Código de operación A6 (M→X)	Cárguese el índice X con la memoria.
LDYZ Código de operación A4 (M→Y)	Cárguese el índice Y con la memoria
STAZ Código de operación 85 (A→M)	Almacénese el contenido del acumulador en la memoria.
STXZ Código de operación 86 (X→M)	Almacénese el índice X en la memoria.
STYZ Código de operación 84 (Y→M)	Almacénese el índice Y en la memoria.

Instrucciones aritméticas:

ADCZ Código de operación 65 (A+M+C→A)	Sumar la memoria al acumulador con acarreo.
SBCZ Código de operación E5 (A-M-C→A)	Restar la memoria del acumulador con acarreo.
INCZ Código de operació E6 (M+1→M)	Increméntese la memoria en una unidad.
DECZ Código de operación C6 (M-1→M)	Decreméntese la memoria en una unidad.

Instrucciones lógicas:

ORAZ Código de operación Ø5 (AUM→M)	Disyunción entre memoria y acumulador.
ANDZ Código de operación 25 (A∩M→M)	Conjunción entre memoria y acumulador.
EORZ Código de operación 45 (A∀M→M)	O exclusivo entre memoria y acumulador.

Direccionamientos relativos

Este modo de direccionamiento se utiliza solamente para instrucciones de ramificación, de las cuales hay dos tipos diferentes: condicional e incondicional. Las instrucciones de ramificación incondicional causan siempre un salto a otra instrucción, mientras que con las instrucciones de ramificación condicional el ordenador tiene en cuenta alguna o algunas cosas antes de decidir o no el salto. Estas decisiones son el reflejo de las condiciones que imponemos para las ramificaciones del organigrama que se hace al desarrollar un programa. La mayoría de los símbolos de los organigramas se convierten en instrucciones concretas al expresarlas en lenguaje escrito. Veamos, por

ejemplo: Si la bandera X está activada salta a la subrutina A; si la bandera Y está desactivada incrementa el contador, etc.
Las instrucciones de ramificación condicional tiene dos bytes de código característico. El segundo byte es tratado como un número binario asignado de 8 bits, que es sumado al contador de programa una vez que el contenido del PC ha sido incrementado a la dirección de la siguiente instrucción del programa. A continuación damos una lista de las instrucciones de ramificación condicional:

1. BCC Código de operación 90 — Salta si el acarreo es nulo (C = Ø) (Branch if Carry Clear).
   BCS Código de operación B0 — Salta si el acarreo está activado (C = 1) (Branch if Carry Set).
2. BNE Código de operación D0 — Salta si no es igual a Ø (Z = Ø) (Branch if Not Equal to zero).
   BEQ Código de operación F0 — Salta si es igual a Ø (Z = 1) (Branch if EQual to zero).
3. BPL Código de operación 10 — Salta si es mayor que Ø (N = Ø) (Branch if PLus).
   BMI Código de operación 30 — Salta si es menor que Ø (N = 1 (Branch if MInus).
4. BVC Código de operación 50 — Salta si no está activado el exceso (Overflow = V = Ø) (Branch if oVerflow Clear).
   BVS Código de operación 70 — Salta si está activado el exceso (Overflow = V = 1) (Branch if oVerflow Set).

Podemos utilizar ya estas instrucciones para desarrollar programas. Las instrucciones de ramificación incondicional será discutidas más tarde en este capítulo.
Echemos una mirada al diagrama de flujo de la figura 8. Al comienzo del programa (Ø200) el registro Y es cargado con el valor ØA. La siguiente instrucción es DEY que reduce el valor del registro Y en una unidad, con lo que ØA se queda en Ø9. Avanzando hacia abajo en el diagrama de flujo llegamos a una instrucción de ramificación condicional (BNE). De lo anterior descubrimos que el microprocesador solamente ejecuta una ramificación si el contenido del registro Y no es igual a Ø. En este caso, el registro Y contiene Ø9, con lo que el procesador saltará a la dirección que contenga la instrucción DEY. Tan pronto como el valor del registro Y resulte ser cero el procesador abandonará la ramificación y el programa se detendrá en la última instrucción (BRK). Este programa no hace más que decrementar continuamente el contenido del registro Y hasta que el valor contenido allí resulta ser Ø. Rutinas de este tipo utilizan a menudo como lazos de retardo.
En el lado izquierdo del símbolo romboidal que contiene la instrucción de ramificación hay dos números hexadecimales, D0 y F2. El primero es (por supuesto) la instrucción mientras que el segundo es el valor del desplazamiento que se utiliza para calcular la dirección efectiva. Si el programa es para saltar hacia atrás (como en este caso) el valor del desplazamiento (salto) deberá ser negativo. Un salto hacia adelante requerirá un valor de desplazamiento positivo.
El equivalente del hexadecimal FD es 11111101 que es, en notación de complemento a 2, —3. Puesto que el paso o desplazamiento es relativo, la ramificación nos llevará tres posiciones de memoria hacia atrás, calculadas

desde la dirección inmediatamente siguiente a la que contiene el valor de desplazamiento (0205—3 = 0202). Todo esto resultará un poco más claro en el programa siguiente:

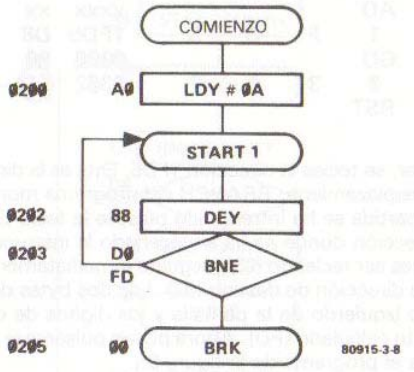


Figura 8. Ejemplo de lazo de retardo utilizando una instrucción de ramificación condicional.

```
0200  A0  LDY
0201  0A
0202  88  DEY
0203  D0  BNE
0204  FD  valor de desplazamiento
0205  00  BRK
```

(Puede que usted no se haya dado cuenta, pero acabamos de descubrir una nueva instrucción —DEY— que reduce el contenido del registro Y en una unidad).
La dirección efectiva a la que se llega se calcula desde la posición de memoria siguiente a la que contiene el valor de desplazamiento o salto, tal y como si el contador de programa estuviera apuntando hacia ella una vez que el programa ha alcanzado la dirección 0204. Recuerde que el contador de programa es incrementado *antes* de que la siguiente instrucción sea realizada.
La dirección efectiva (dirección a la que se salta) puede estar situada hasta un máximo de 127 pasos hacia adelante (+ 127: en hexadecimal 00...7F) ó 128 pasos hacia atrás (—128: en hexadecimal FF...80). Esto nos da, en definitiva, las 256 posibilidades de un solo byte.

Calcular desplazamientos con el programa monitor

Hay dos aspectos importantes a tener en cuenta cuando se calculan desplazamientos para instrucciones de ramificación: el lugar donde la ramificación se origina y el lugar donde debe terminar. Esto es bastante fácil de ver en un

organigrama como el de la figura 8, pero cuando hay numerosas instrucciones de ramificación en un programa es más cómodo dejar al JC que calcule todos los saltos.
Utilizando la figura 8 como ejemplo, esto puede realizarse como sigue:

```
AD                    xxxx  xx
 1    F    D    5     1FD5  D8
GO                    0000  00
 0    3    0    2     0302  FD   — desplazamiento
RST
```

En primer lugar, se teclea la dirección 1FD5. Esta es la dirección de partida de la rutina de desplazamiento BRANCH del programa monitor. Una vez que la dirección de partida se ha introducido púlsese la tecla GO. El byte de orden bajo de la dirección donde se ha almacenado la instrucción de ramificación puede entonces ser tecleado (03), seguido inmediatamente por el byte de orden bajo de la dirección de destino (02). Los dos bytes de dirección aparecerán en el lado izquierdo de la pantalla y los dígitos de datos presentarán el desplazamiento calculado (FD). Ahora puede pulsarse la tecla RESET e introducir después el programa de la figura 8.

```
AD
 0    2    0    0     0200  xx
DA                    0200  A0   LDY #
 +         0    A     0201  0A
 +         8    8     0202  88   DEY
 +         D    0     0203  D0   BNE
 +         F    D     0204  FD   desplazamiento
 +         0    0     0205  00   BRK
```

El programa puede ejecutarse en el modo normal, pero sería interesante ver exactamente cómo se desarrolla el proceso en el modo STEP.

Un contador mediante software

Con todos estos conocimientos sobre instrucciones de ramificación veamos si somos ahora capaces de construir un contador sin utilizar componente alguno aparte del JC. En este proceso conoceremos, además, una instrucción nueva: CMP.
Este contador particular comenzará desde 0 y se detendrá tan pronto como alcance el límite preseleccionado de 2000 (hexadecimal). El diagrama de flujo para el programa se muestra en la figura 9. Se requieren dos bytes para el contenido del contador de sofware y éstos se almacenan (utilizando direccionamiento de página 0) en las posiciones de memoria 0000 (COUNTL) y 0001 (COUNTH).
Inicialmente se carga el acumulador con 00 y se activa el contador (se almacena 00 en COUNTL y en COUNTH). A continuación, el contenido de COUNTL se incrementa en una unidad y alcanzamos la primera instrucción de ramificación (BNE). Esta instrucción comprueba el estado de la bandera de cero (Z); si esta bandera está activada, el programa salta a BEG2.

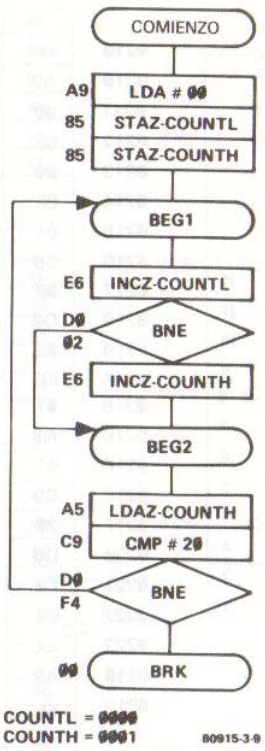


Figura 9. Organigrama del programa contador.

Se carga entonces el acumulador con el contenido de COUNTH y se compara, mediante la instrucción CMP, con el valor 20 (el valor final preajustado para el byte de mayor orden). Esta nueva instrucción (código de operación C9) activa la bandera Z si el contenido del acumulador (COUNTH) y el contenido del byte siguiente (20) son los mismos. Inicialmente, por supuesto, el valor de COUNTH es todavía 0. De esta forma el programa continuará hacia la siguiente instrucción de ramificación y de ella saltará hacia atrás hasta BEG1. Cuando el contenido de COUNTH alcanza el valor 20 la bandera Z es activada, con lo cual se impide la ramificación y el programa se detiene.
En otras palabras, COUNTL resulta ser 00 cada 256 ciclos. Esto incrementa el valor de COUNTH en una unidad hasta que su valor alcance 20. El programa real es como sigue:

```
AD                  xxxx   xx
 1  F   D  5        1FD5   D8   dirección de partida de la
GO                  0000   00   rutina de desplazamiento
 1  8   1  C        181C   02   salto para el primer BNE
 2  0   1  6        2016   F4   salto para el segundo BNE
```

```
RST  AD
 0   2    1   Ø                 Ø21Ø   xx   dirección de partida del programa
DA        A   9                 Ø21Ø   A9   LDA #
 +        Ø   Ø                 Ø211   ØØ
 +        8   5                 Ø212   85   STAZ-
 +        Ø   Ø                 Ø213   ØØ
 +        8   5                 Ø214   85   STAZ-
 +        Ø   1                 Ø215   Ø1
 +        E   6                 Ø216   E6   INCZ-
 +        Ø   Ø            12   Ø217   ØØ
 +        D   Ø            11   Ø218   DØ   BNE
 +        Ø   2            10   Ø219   Ø2   salto
 +        E   6             9   Ø21A   E6   INCZ-
 +        Ø   1             8   Ø21B 1 Ø1
 +        A   5             7   Ø21C 2 A5   LDAZ-
 +        Ø   1             6   Ø21D   Ø1
 +        C   9             5   Ø21E   C9   CMP #
 +        2   Ø             4   Ø21F   2Ø   byte para CMP #
 +        D   Ø             3   Ø22Ø   DØ   BNE
 +        F   4             2   Ø221   F4   salto
 +        Ø   Ø             1   Ø222   ØØ   BRK
AD                               Ø222   xx
 Ø   2    1   Ø                 Ø21Ø   A9   dirección de partida
GO                               Ø212   xx   ejecución (Run)
```

Inicialmente, los valores de desplazamiento para ambas instrucciones de ramificación fueron calculados utilizando el programa monitor. Esto produjo los dos valores F4 y Ø2. No es estrictamente necesario realizar este procedimiento cada vez, pero el hacerlo ayuda a familiarizarnos con el teclado y el programa monitor.

Instrucciones de ramificación incondicional

Como su propio nombre indica, estas instrucciones se ejecutan sin necesidad de que se verifique ninguna condición previa. La primera instrucción de ramificación incondicional que describiremos es la instrucción JMP—. Esta instrucción utiliza (fíjese en el guión) direccionamiento o absoluto o indirecto. Cuando se utiliza direccionamiento absoluto su código de operación es 4C. Es, por tanto, una instrucción de tres bytes, de los cuales los dos últimos contienen la dirección a la que el programa debe saltar.

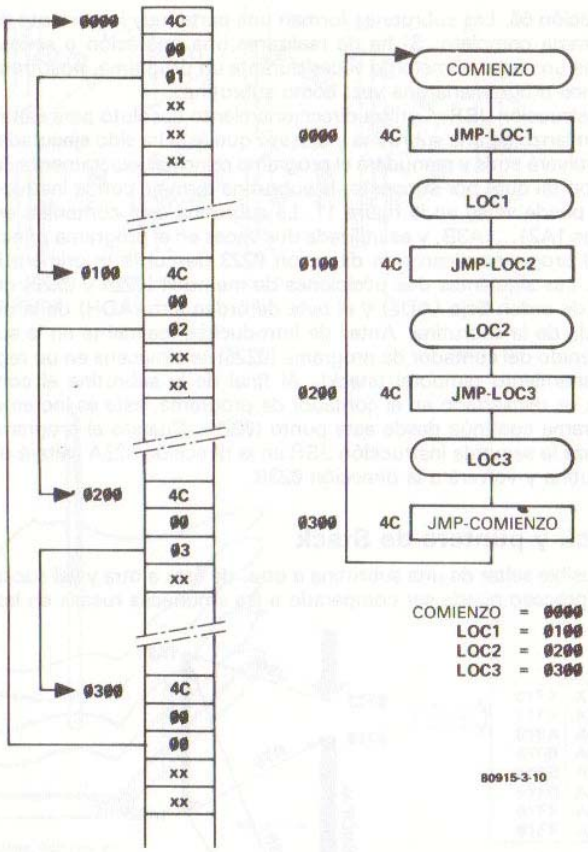


Figura 10. Un programa simple para ilustrar la utilización de instrucciones de salto incondicional.

La figura 10 muestra un diagrama de memoria y un organigrama de un programa simple que sólo contiene instrucciones de ramificación incondicionales (saltos). El programa saltará desde START hasta las posiciones de memoria 1, 2 y 3 (por orden), volviendo luego a START. Puede parecer bastante inútil, pero les garantizamos que se trata de un programa bastante serio.

JSR y RTS, entrada y salida de subrutinas

Conozcamos ahora dos instrucciones más de ramificación incondicional: JSR = Salta a subrutina (Jump to SubRoutine) con código de operación 20 y RTS = Retorno de la subrutina (ReTurn from Subroutine) con código de

operación 60. Las subrutinas forman una parte muy importante de cualquier programa completo. Si ha de realizarse una operación o series de operaciones un cierto número de veces durante un programa, nosotros sólo necesitamos programarla una vez: como subrutina.
La instrucción JSR— utiliza direccionamiento absoluto para saltar y situarse al comienzo de una subrutina y una vez que ésta ha sido ejecutada el ordenador volverá atrás y reanudará el programa principal exactamente donde lo dejó (con tal que, por supuesto, la subrutina termine con la instrucción RTS). Esto puede verse en la figura 11. La subrutina está contenida en las direcciones 1A21...1A3B, y es utilizada dos veces en el programa principal. Cuando el programa alcanza la dirección 0223 descubre la primera instrucción JSR. Las siguientes dos posiciones de memoria (0224 y 0225) contienen el byte de orden bajo (ADL) y el byte de orden alto (ADH) de la dirección de partida de la subrutina. Antes de introducirse realmente en la subrutina, el contenido del contador de programa (0225) se almacena en un registro de almacenamiento temporal (stack). Al final de la subrutina el contenido del stack es reinsertado en el contador de programa, éste es incrementado y el programa continúa desde este punto (0226). Cuando el programa principal alcanza la segunda instrucción JSR en la dirección 023A saltará otra vez a la subrutina y volverá a la dirección 023B.

Stack y puntero de Stack

Es posible saltar de una subrutina a otra, de ésta a otra y así sucesivamente. Este proceso puede ser comparado a las «muñecas rusas» en las que cada

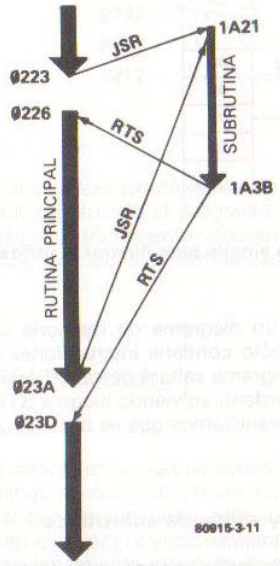


Figura 11. En esta figura puede verse esquemáticamente cómo se utilizan las instrucciones de *salto hacia* y *retorno* de una subrutina; puede saltarse a una subrutina desde cualquier parte del programa principal.

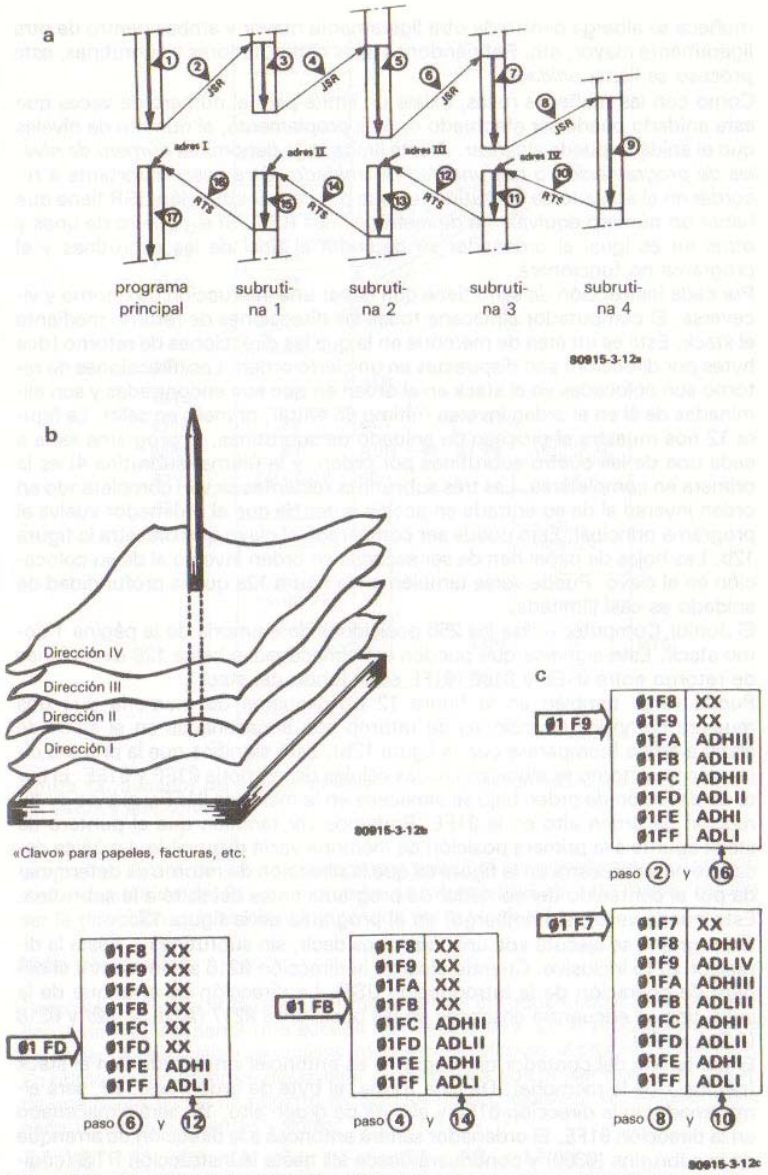


Figura 12. El anidado de subrutinas (12a) puede ser comparado al uso del clavo para papeles, facturas, etc., de la figura 12b. El plano de memoria de 12c nos muestra el contenido del stack y del puntero de stack para cada instrucción de salto y retorno.

muñeca se alberga dentro de otra ligeramente mayor y ambas dentro de otra ligeramente mayor, etc. Refiriéndonos a los computadores y subrutinas, este proceso se llama *anidado.*

Como con las muñecas rusas, existe un límite para el número de veces que este anidado puede ser efectuado o, más propiamente, al número de niveles que el anidado puede alcanzar. A este límite se le denomima *número de niveles de programación* o *profundidad de anidado.* Otra cosa importante a recordar en el anidado de subrutinas es que por cada instrucción JSR tiene que haber un número equivalente de instrucciones RST. Si el número de unas y otras no es igual el ordenador se detendrá al final de las subrutinas y el programa no funcionará.

Por cada instrucción de salto tiene que haber una instrucción de retorno y viceversa. El computador almacena todas las direcciones de retorno mediante el stack. Esta es un área de memoria en la que las direcciones de retorno (dos bytes por dirección) son dispuestas en un cierto orden. Las direcciones de retorno son colocadas en el stack en el orden en que son encontradas y son eliminadas de él en el orden inverso (último en entrar, primero en salir). La figura 12 nos muestra el proceso de anidado de subrutinas. El programa salta a cada una de las cuatro subrutinas por orden, y la última (subrutina 4) es la primera en completarse. Las tres subrutinas restantes se van completando en orden inverso al de su entrada en acción antes de que el ordenador vuelva al programa principal. Esto puede ser comparado al clavo que muestra la figura 12b. Las hojas de papel han de ser sacadas en orden inverso al de su colocación en el clavo. Puede verse también en la figura 12a que la profundidad de anidado es casi ilimitada.

El Junior Computer utiliza las 256 posiciones de memoria de la página 1 como stack. Esto significa que pueden ser almacenadas hasta 128 direcciones de retorno entre Ø1FF y Ø1ØØ (Ø1FF es el fondo del stack).

Puede verse también en la figura 12 un esquema de memoria que nos muestra como las direcciones de retorno son almacenadas en el stack de abajo a arriba (compárese con la figura 12b). Esto significa que la primera dirección de retorno se almacena en las células de memoria Ø1FF y Ø1FE. El byte de dirección de orden bajo se almacena en la memoria Ø1FF y el byte de dirección de orden alto en la Ø1FE. Podemos ver también que el puntero de stack apunta a la primera posición de memoria vacía disponible. La única cosa que no se muestra en la figura es que la dirección de retorno es determinada por el contenido del contador de programa antes del salto a la subrutina. Esto puede verse, sin embargo, en el programa de la figura 13.

El programa se ejecuta «de una vez» —es decir, sin subrutinas— hasta la dirección Ø215 inclusive. Cuando alcanza la dirección Ø216 se encuentra el código de operación de la introducción JSR. La dirección de arranque de la subrutina se encuentra entonces en las posiciones Ø217 (ADL = ØØ) y Ø218 (ADH = Ø3).

El contenido del contador de programa es entonces «inyectado» en el stack (página 1 de la memoria). De esta forma, el byte de orden bajo, Ø2, será almacenado en la dirección Ø1FF y el byte de orden alto, 18, será almacenado en la dirección Ø1FE. El ordenador saltará entonces a la dirección de arranque de la subrutina (Ø3ØØ) y continuará desde allí hasta la instrucción RTS (código de operación 6Ø), situada en la dirección Ø3Ø6. Esta instrucción utiliza direccionamiento «implicado» (ya hablaremos más sobre ello), lo que significa que la dirección de retorno se hallará en la «cumbre» del stack. Esta dirección de retorno se elimina entonces del stack y es reemplazada en el contador de

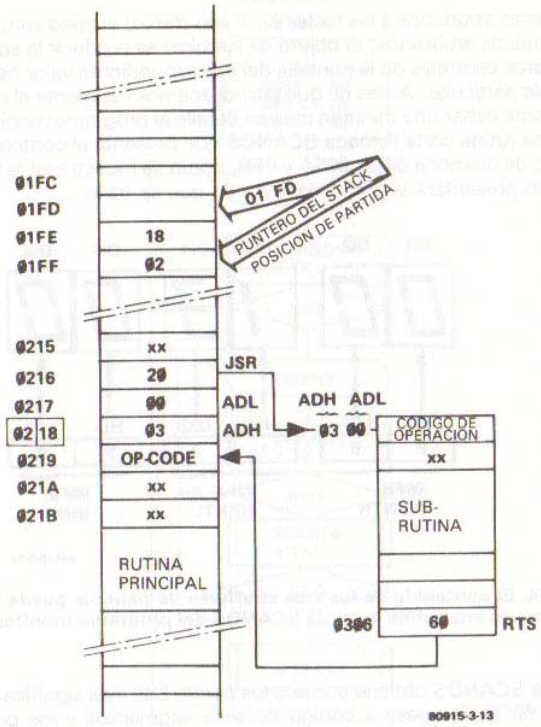


Figura 13. Sección de programa que ilustra el uso de subrutina y la función del satck y su puntero.

programa. Antes de volver al programa principal se incrementa en una unidad el contenido del PC. La dirección de retorno real, de este modo, resulta ser la dirección que estaba almacenada en el stack más uno.

Más ejercicios

Creemos que ya es hora de que comencemos a introducir cosas en el ordenador y viendo realmente que sucede con esas cosas. En el proceso vamos a aprender también algunas cosillas interesantes relativas al programa monitor. En primer lugar, vamos a desarrollar un programa que presente el valor de cada una de las teclas de nuestro teclado en forma hexadecimal. Asignemos a cada tecla los siguientes valores:

0 : 00	5 : 05	A : 0A	F : 0F	PC : 14
1 : 01	6 : 06	B : 0B	AD : 10	
2 : 02	7 : 07	C : 0C	DA : 11	
3 : 03	8 : 08	D : 0D	+ : 12	
4 : 04	9 : 09	E : 0E	GO : 13	

Los valores asignados a las teclas Ø...F son claros; el resto son, sin embargo, más o menos arbitrarios. El objeto de ejercicio es producir la aparición en los dos dígitos centrales de la pantalla del correspondiente valor hexadecimal de una tecla particular. Antes de que introduzcamos realmente el programa será interesante echar una miradita más en detalle al programa monitor. Este contiene una rutina corta llamada SCANDS que presenta el contenido de las posiciones de memoria ØØF9, ØØFA y ØFB, como se muestra en la figura 14. Cada dígito presentará la mitad del byte de que se trate.

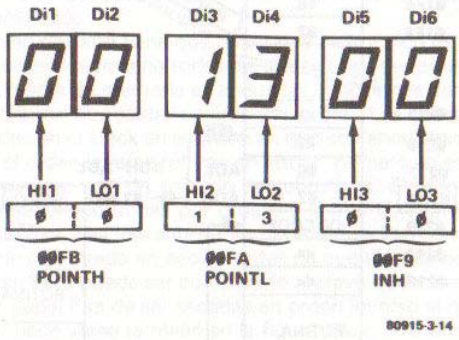


Figura 14. El contenido de los tres «buffers» de pantalla puede ser presentado utilizando una rutina llamada SCANDS del programa monitor.

La rutina SCANDS obtiene primero los cuatro bits más significativos de la dirección ØØFD, los pasa a código de siete segmentos y los presenta en el dígito más a la izquierda del visualizador. Los cuatro bits menos significativos del mismo byte son decodificados similarmente y esta información es introducida en el segundo dígito. El contenido de las direcciones ØFA y ØØF9 son presentados de la misma forma en los cuatro dígitos de siete segmentos restantes.

Veamos ahora cómo el programa monitor determina si una tecla ha sido o no pulsada. Esto se realiza con la rutina TK (TestKey), que es llamada por la rutina SCANDS. Tan pronto como se pulsa una tecla su valor debe de ser presentado en pantalla. El programa monitor contiene también una rutina llamada GETKEY, que hace exactamente eso.

El organigrama correspondiente puede verse, algo simplificado, en la figura 15. Puesto que los dos dígitos centrales deben presentar el número correspondiente a la tecla que se presione, los dígitos restantes deberán ser siempre ØØ. De esta forma, la primera operación a realizar es almacenar ØØ en las direcciones de memoria ØØFB y ØØF9. La sección siguiente del programa (SCAN1) contiene la subrutina SCANDS + TK, con la que el contenido de las direcciones ØØF9, ØØFA y ØØFB es presentado en la pantalla. Esta subrutina sondea también continuamente el teclado para «detectar» cuando una tecla cualquiera sea pulsada. Mediante una instrucción de ramificación condicional, BNE, el programa saltará siempre al principio de SCAN1 si no pulsamos alguna tecla. Tan pronto como pulsemos una tecla, sin embargo, el programa saltará a SCAN2.

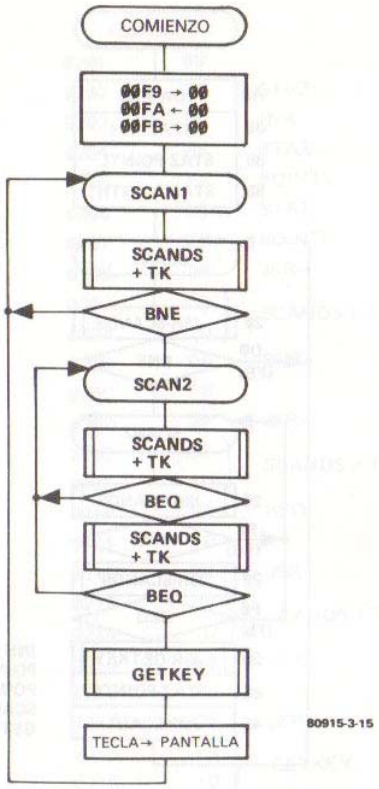


Figura 15. Organigrama «basto» del programa que detecta, identifica y presenta en pantalla cualquier tecla. Este programa utiliza subrutinas del programa monitor.

Esta sección del programa contiene también las subrutinas SCANDS + TK; pero esta vez el programa sondeará el teclado para ver si la tecla ha vuelto a su posición de reposo. Este sondeo se realiza en realidad dos veces, con el fin de prevenir por medio del software la posibilidad de rebote de contactos.
La última sección del programa contiene la subrutina GETKEY. Esta rutina asegura el que la tecla ha suministrado el valor hexadecimal correcto que es entonces almacenado en la dirección de memoria ØØFA. El programa entonces vuelve a SCAN1 y espera hasta que sea pulsada otra tecla. En la figura 16 puede verse un organigrama más detallado.
Como puede verse, no hay mucha diferencia entre éste y el de la figura 15. Las instrucciones y sus códigos de operación han sido incluidas junto a los símbolos. Las instrucciones de retorno de subrutina no se han incluido en el programa principal, puesto que son manejadas por el programa monitor. Co-

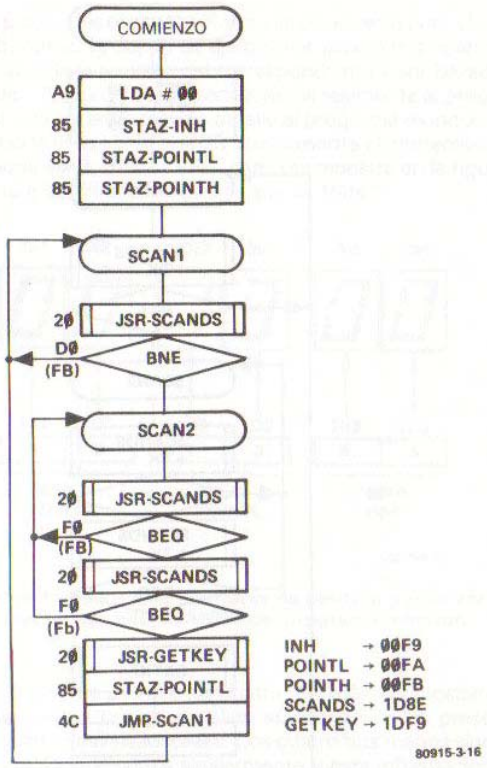


Figura 16. Organigrama detallado del programa de la figura 15.

mo siempre, deberemos calcular y apuntar los valores de desplazamiento antes de teclear el programa.

AD				xxxx	xx	
1	F	D	5	1FD5	D8	
GO				ØØØØ	ØØ	
Ø	B	Ø	8	ØBØ8	FB →	note el desplazamiento
1	Ø	Ø	D	1ØØD	FB →	note el desplazamiento
1	5	Ø	D	15ØD	F6 →	note el desplazamiento
RST						
AD						
Ø	2	Ø	Ø	Ø2ØØ	xx	
DA		A	9	Ø2ØØ	A9	LDA #

+		0	0	0201	00	
+		8	5	0202	85	STAZ–
+		F	9	0203	F9	INH
+		8	5	0204	85	STAZ–
+		F	A	0205	FA	POINTL
+		8	5	0206	85	STAZ–
+		F	B	0207	FB	POINTH
+		2	0	0208	20	JSR–
+		8	E	0209	8E	SCANDS + TK
+		1	D	020A	1D	
+		D	0	020B	D0	BNE
+		F	B	020C	FB	
+		2	0	020D	20	JSR–
+		8	E	020E	8E	SCANDS + TK
+		1	D	020F	1D	
+		F	0	0210	F0	BEQ
+		F	B	0211	FB	
+		2	0	0212	20	JSR–
+		8	E	0213	8E	SCANDS + TK
+		1	D	0214	1D	
+		F	0	0215	F0	BEQ
+		F	6	0216	F6	
+		2	0	0217	20	JSR–
+		F	9	0218	F9	GETKEY
+		1	D	0219	1D	
+		8	5	021A	85	STAZ–
+		F	A	021B	FA	POINTL
+		4	C	021C	4C	JMP–
+		0	8	021D	08	SCAN 1
+		0	2	021E	02	
AD				021E	02	
0	2	0	0	0200	A9	dirección de partida
GO				0000	00	el programa se ejecuta hasta que se pulse una tecla
GO				0013	00	valor tecla GO
6				0006	00	valor tecla 6
AD				0010	00	valor tecla AD
DA				0011	00	valor tecla DA
B				000B	00	valor tecla B

y así sucesivamente:

RST

Una vez que la dirección de partida (Ø2ØØ) ha sido introducida se presiona dos veces la tecla GO. La primera vez para ejecutar el programa, y la segunda vez como ejemplo del programa: el valor 13 aparecerá entonces en los dos dígitos centrales de la pantalla.

Direccionamiento implícito

El término «direccionamiento implícito» se utiliza para describir instrucciones que identifican uno de los registros programables. Estas son instrucciones de un solo byte y son utilizadas, por ejemplo, para transferir datos de un registro a otro o para activar/desactivar varias banderas. Hay 25 instrucciones de este tipo, algunas de las cuales conocemos ya:

BRK Código de operación ØØ	BReaK (Stop).
CLC Código de operación 18	CLear Carry (Borrado del dígito de acarreo).
CLD Código de operación D8	CLear Decimal mode (Borrado del modo decimal).
CLI Código de operación 58	CLear Interrupt flag (Desactivado de la bandera de Interrupción).
CLV Código de operación B8	CLear oVerflow flag (Desactivado de la bandera de exceso).
DEX Código de operación CA	DEcrement X register by one (Disminuir en 1 el registro X).
DEY Código de operación 88	DEcrement Y register by one (Disminuir en 1 el registro Y).
INX Código de operación E8	INcrement X register by one (Incrementar en 1 el registro X).
INY Código de operación C8	INcrement Y register by one (Incrementar en 1 en el registro).
NOP Código de operación EA	No OPeration (Espacio en blanco).
PHA Código de operación 48	PusH Accumulator onto stack (Almacena el acumulador en el stack).
PHP Código de operación Ø8	PusH Procesor status register onto stack (Almacena el registro de estado en el stack).
PLA Código de operación 68	PulL Accumulator from stack (Carga el acumulador desde el stack).
PLP Código de operación 28	PulL Processor status register from stack (Carga el registro de estado desde el stack).
RTI Código de operación 4Ø	ReTurn from Interrupt (Retorno desde Interrupción).
RTS Código de operación 6Ø	ReTurn from Subroutine (Retorno desde Subrutina).
SEC Código de operación 38	SEt Carry flag (Activa bandera de acarreo).
SED Código de operación F8	SEt Decimal flag (Activa bandera de modo decimal).
SEI Código de operación 78	SEt Interrupt flag (Activa bandera de Interrupción).

TAX Código de operación AA	Transfer Accumulator to X register (Transfiere el acumulador al registro X).
TAY Código de operación A8	Transfer Accumulator to Y register (Transfiere el acumulador al registro Y),
TSX Código de operación BA	Transfer Stack pointer to X register (Transfiere el puntero de stack al registro X).
TXA Código de operación 8A	Transfer X register to Accumulator (Transfiere el registro X al acumulador).
TXS Código de operación 9A	Transfer X register to Stack (Transfiere el registro X al stack).
TYA Código de operación 98	Transfer Y register to Accumulator (Transfiere el registro Y al acumulador).

Vamos a describir todo esto con más detalle.

SED y CLD

El conjunto de instrucciones del microprocesador 6502 hace posible calcular tanto en código decimal como en binario. Tras la instrucción SED (SEt Decimal flag = Activa la bandera decimal) el ordenador operará en modo decimal. Esto significa que la bandera de acarreo queda activada siempre que el resultado de un cálculo exceda 99 (10011001) y no FF (11111111). Cuando se desactiva la bandera decimal (con la instrucción CLD) el ordenador continuará operando en modo binario.

Un consejo útil: Si se necesita operar el computador en modo binario es aconsejable comenzar el programa o la sección de éste que se desee operar en binario con la instrucción CLD. Esto asegurará el funcionamiento correcto. La medida es más aconsejable de lo que parece, ya que se olvida fácilmente la situación de la bandera, que incluso ha podido ser activada en otra parte del programa *.

NOP (no operar)

Sin importar cuán eficaz pueda ser un programador de ordenadores hay siempre la posibilidad de olvidar una instrucción importante en la mitad (o más normalmente, al principio) de un programa. Esto significa, por supuesto, que el programa no funcionará y que será necesario sondear paso a paso su funcionamiento. El programador avisado, sin embargo, incluirá siempre un buen montón de instrucciones NOP dispuestas aleatoriamente a lo largo del programa antes de que éste pueda considerarse como una versión final y definitiva. Mediante este simple truco las instrucciones adicionales necesarias pueden ser intercaladas donde haga falta sin necesidad de reintroducir el programa completo. Si, por otra parte, se encuentran superfluas algunas instrucciones siempre podremos reemplazarlas con instrucciones NOP sin afectar el resto del programa. Por tanto, la instrucción NOP está muy lejos de ser una instrucción inútil como a primera vista podría parecer.

* El computador queda siempre en el modo binario tras pulsar la tecla RST.

Transferencia y manipulación

Todas las instrucciones implicadas cuyas siglas nemotécnicas comiencen con una P o una T nos indican que la transferencia de datos a realizar debe hacerse entre registros internos. Hay muchas razones por las que son necesarias estas instrucciones. Por ejemplo, imaginemos que estamos utilizando el registro X en el programa principal y deseamos entonces saltar a una subrutina que también utiliza el registro X. Antes de saltar a la subrutina en cuestión deberemos transferir el contenido actual del registro X a un sitio seguro para que podamos volverlo a colocar allí una vez que la subrutina haya sido ejecutada. El sitio más obvio para almacenar el registro X es, naturalmente, el stack. El procedimiento real podría ser como sigue:

TXA	Transferir el contenido del registro X al acumulador.
PHA	Meter el contenido del acumulador en el stack.
JSR-xxxx	Saltar a la subrutina (y hacer allí lo que se quiera con el registro X).
RTS	Retorno de la subrutina.
PLA	Volver a colocar el contenido del stack en el acumulador.
TAX	Colocar el contenido del acumulador en el registro X.

Podríamos también haber hecho lo siguiente:

STX-SAVX	Almacénese el contenido del registro X en la posición de memoria SAVX.
JSR-xxxx	Saltar a la subrutina (y hacer allí lo que se quiera con el registro X).
RTS	Retorno de la subrutina.
LDX-SAVX	Colocar el contenido de SAVX en el registro X.

A pesar de que el primer método parece más largo sobre el papel, utiliza en realidad menos bytes de programa y es, por tanto, preferible. Puede verse que si el contenido de los registros A, X y P hubiera tenido que ser también «rescatado», el segundo método hubiera sido todavía mucho más largo.

Para simplificar las cosas aún más, el programa monitor contiene subrutinas que realizan la operación anterior. La subrutina SAVE coloca el contenido de los diversos registros en el stack y la subrutina RESTO reenvía el contenido del stack a los registros en cuestión.

La figura 17 muestra las dos subrutinas. La rutina SAVE introduce primero el contenido del acumulador en el stack, continuando con el contenido de los registros X, Y y P (a través del acumulador). La rutina RESTO hace exactamente lo contrario en el sentido de que primero vuelve a colocar el contenido del registro P, después los de los registro Y y X, y, por último, el antiguo contenido del acumulador.

El puntero del stack (registro interno S) apunta siempre a la dirección del stack inmediatamente posterior. Tras haber llamado al programa monitor (pulsando RST) la última dirección de la página 1 (Ø1FF) es «apuntada» automáticamente por el puntero. Es también posible cambiar la dirección de partida del puntero tal como sigue:

LDX L 81	Cárguese el registro X con 81.
TXS	Transferir el contenido del registro X al stack.

El puntero del stack indicará ahora («apuntará hacia») la dirección de memoria Ø181. Debe tenerse cuidado al alterar la dirección de partida del stack, puesto que ello puede ocasionar muy bien que el programa deje de funcionar correctamente al equivocar las direcciones del stack. La disponibilidad de las direcciones del stack puede ser comprobada también con otra subrutina del

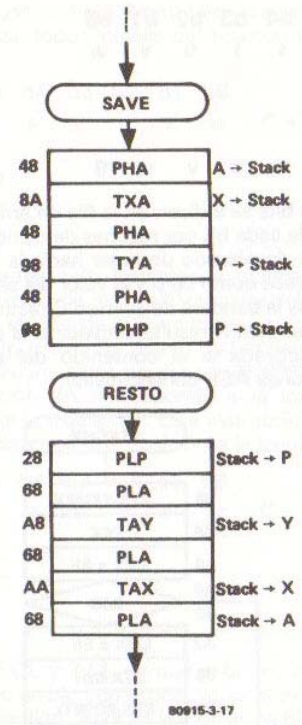


Figura 17. Estas dos subrutinas del programa monitor «rescatan« y «reinstauran» (es decir, almacenan en el stack y luego vuelven a disponer en cada registro respectivo) los contenidos de los diversos registros.

programa monitor denominada STKCHK. Esta rutina nos asegura que siempre que sea excedida la dirección del stack más alta posible (Ø1ØØ) se mostrará una señal de error en la pantalla (EEEEEE). Tras un error el JC entra en un lazo del que sólo saldrá (retornando al programa monitor) si la tecla RST es pulsada. Esta subrutina se muestra en la figura 18 y es específicamente útil para largos programas. Volveremos a ella un poco más adelante, pero, primero, veamos algo más sobre direccionamiento implicado.

Instrucciones de desplazamiento y rotación

Las cuatro instrucciones de un solo byte, que vamos a tatar ahora, son variaciones de las instrucciones implicadas y se utilizan para manipular el contenido del acumulador. Las instrucciones ASL, LSR, ROL y ROR se utilizan en toda clase de operaciones matemáticas. La primera, ASL (Arithmetic Shift Left = Desplazamiento aritmético a la izquierda) tiene el código de operación ØA y desplaza cada bit del byte, situado en el acumulador, una posición nacia la izquierda:

```
        b7  b6  b5  b4  b3  b2  b1  bØ
C = x   p   q   r   s   t   u   v   w
            ↓
           ASL
            ↓
C = p   q   r   s   t   u   v   w   Ø
```

Las posiciones de los bits se indican en la fila de arriba. Hemos asignado las letras p...w al valor de cada bit por razones de claridad. Como puede verse, todos los bits se han desplazado un lugar hacia la izquierda. El bit situado más a la derecha aparece como cero y el valor del situado más a la izquierda determina el estado de la bandera de acarreo C (activada si p era 1 y desactivada si p era Ø). La bandera N resulta activada si el séptimo bit (q) era 1 y la bandera Z resulta activada si el contenido del acumulador resulta ser ØØØØØØØØ (8 instrucciones ASL consecutivas).

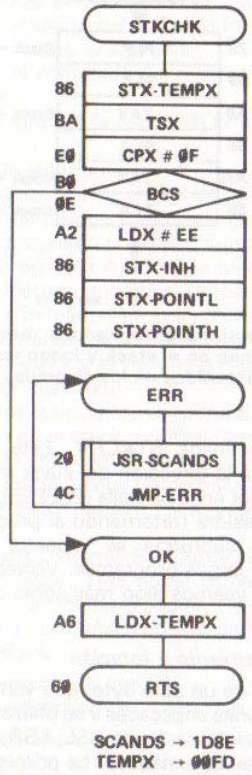
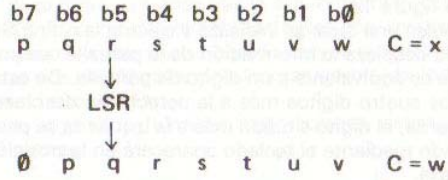
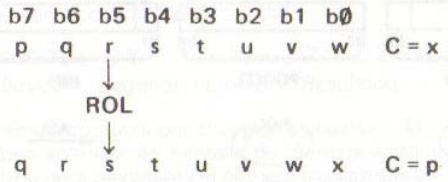


Figura 18. La rutina del programa monitor STKCHK que comprueba la disponibilidad de direcciones en el stack. En caso de no quedar sitio en el stack se da una indicación de error en pantalla (EEEEEE).

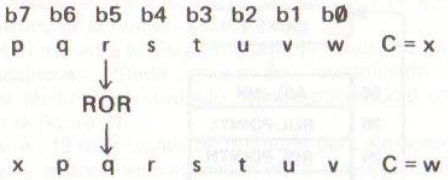
La instrucción LSR (Logical Shift Right) tiene el código de operación 4A. Esta instrucción desplaza todos los bits del acumulador una posición hacia la derecha.



Esta vez todos los bits han sido desplazados un lugar hacia la derecha. El bit situado más hacia la izquierda resulta Ø y el valor del bit situado más hacia la derecha determina el estado de la bandera de acarreo. La bandera N no es activada por esta instrucción. De nuevo, la bandera Z será activada únicamente si el valor del acumulador resulta ser ØØØØØØØØ.
El código de operación 2A corresponde a la instrucción ROL (ROtate Left = Rotación hacia la izquierda). Esta instrucción es similar a ASL y todos los bits son desplazados una posición a la izquierda:



La diferencia entre ROL y ASL es que esta vez el valor inicial del bit de acarreo es introducido en bØ, con lo que ninguno de los valores originales del acumulador resulta perdido. Para todos los otros bits el resultado es el mismo que con la instrucción ASL.
ROR es el código mnemotécnico de ROtate Right (rotación a la derecha). El código de operación correspondiente es 6A. Con esta instrucción se desplaza cada bit un lugar a la derecha.



La diferencia entre esta instrucción y la LSR es que el valor del bit de acarreo pasa a b7. Por lo demás, es exactamente igual que la instrucción LSR.
De esto se deduce que la diferencia entre el desplazamiento y la rotación estriba en que con las instrucciones de desplazamiento se pierde uno de los bits extremos y con las instrucciones de rotación no se pierde ninguno.

Un ejemplo de rotación y desplazamiento de registros
Como ya sabemos por la práctica, cada vez que pulsamos una tecla (es decir, una vez que introducimos un nuevo dato) del JC la información precedente es desplazada de derecha a izquierda a lo largo de la pantalla. También sabe-

mos (ver figura 14) que la información a presentar en la pantalla es almacenada y procesada por tres «buffers» de proceso correspondientes a las direcciones de memoria ØØF9..ØØFB. La información a presentar es desplazada y rotada como indica la figura 19.
Tan pronto como se pulsa una tecla se inicializa y ejecuta la rutina SHIFT del programa monitor. Ello desplaza la información de la pantalla cuatro bits hacia la izquierda, lo que es equivalente a un dígito de pantalla. De esta forma, tras la rutina SHIFT los cuatro dígitos más a la derecha se desplazarán una posición hacia la izquierda, el dígito situado más a la izquierda se perderá y el nuevo dígito introducido mediante el teclado aparecerá en la posición más a la derecha de la pantalla.
El funcionamiento real de la rutina es como sigue: Tras la instrucción ASL, el bit bØ en el byte INH se hace cero y el contenido del bit de acarreo es reemplazado por b7. La siguiente instrucción rota el contenido del byte POINTL hacia la izquierda. El bit de acarreo de INH es introducido en bØ de POINTL y b7 de POINTL es introducido en el bit de acarreo. Lo mismo suce-

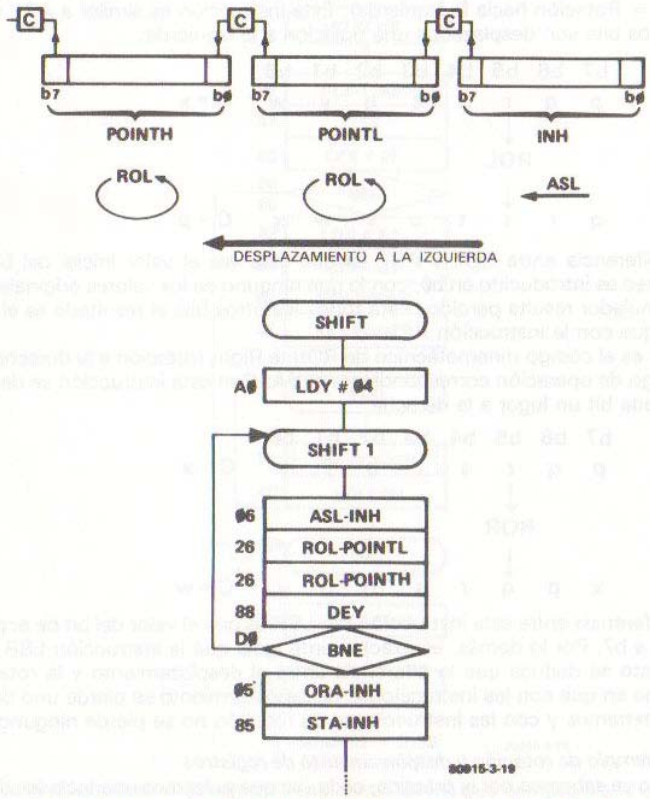


Figura 19. Operación de la rutina monitor SHIFT que desplaza la información presente en la pantalla un lugar hacia la izquierda.

de a POINTH durante la siguiente instrucción ROL. De esta forma, bØ del byte INH queda reemplazado por Ø y b7 de POINTH queda colocado en el bit de acarreo. El procedimiento completo es repetido cuatro veces, lo que significa que el bit de acarreo se pierde tras la siguiente instrucción ASL. El resultado final es que todos los bits de los tres bytes son desplazados cuatro lugares hacia la izquierda y los cuatro bits de mayor orden de POINTH se pierden. Los últimos dos rectángulos en el organigrama de la figura 19 son para el dígito recién introducido.
El programa siguiente utiliza las instrucciones vistas y algunas rutinas del programa monitor.

Adición decimal

Hay muchas maneras de sumar dos números decimales. Una de ellas es comparar una calculadora de bolsillo; otra, comprar lápiz y papel, y, otra, sin duda la más complicada, es diseñar una calculadora a base del «software» de nuestro JC.
Sin duda, ya habrá usted adivinado que nosotros vamos a elegir el camino difícil para aprender así a utilizar algunas de las nuevas instrucciones que hemos visto hasta ahora.
Queremos sumar dos números del tipo:

X X X X X X + Y Y Y Y Y Y = Z Z Z Z Z Z
Primer número Segundo número Resultado

Hay, sin embargo, unas pocas reglas a observar. Al introducir los números, éstos deben aparecer en pantalla de derecha a izquierda. El dígito situado más a la izquierda representa el número más significativo (centenas de millar) y el situado más a la derecha, lógicamente, es el menos significativo (unidades). La pantalla debe ser borrada antes de introducir cada número.
Las teclas Ø...9 se utilizan parea introducir los números. La tecla DA (11) realizará la operación «.= » y la tecla AD (10) realizará la función de borrado (clear). La suma se realizará, finalmente, al pulsar la tecla « + ». Si el resultado de la suma es mayor de 999999 será preciso presentar en la pantalla una señal de error. Finalmente, si un número es introducido de forma incorrecta la tecla «clear» debe ser capaz de borrarlo y, tras ello, el usuario podrá de nuevo introducir el número deseado.
La figura 20 muestra el diagrama de flujo requerido para la suma de dos números decimales. Puede parecer algo complicado a primera vista, pero puede ser fácilmente estudiado descomponiéndolo en las nueve subrutinas dadas en la figura 21.
Se requieren 12 direcciones de memoria para almacenar datos variados; sus «nombres y direcciones» aparecen en la lista siguiente:

POINTH ØØFB	POINTL ØØFA	INH ØØF9	buffers de pantalla
B12 ØØØ2	B11 ØØØ1	B1Ø ØØØØ	buffers para el primer número
B22 ØØØ5	B21 ØØØ4	B2Ø ØØØ3	buffers para el segundo número
R2 ØØØ8	R1 ØØØ7	RØ ØØØ6	buffers para el resultado

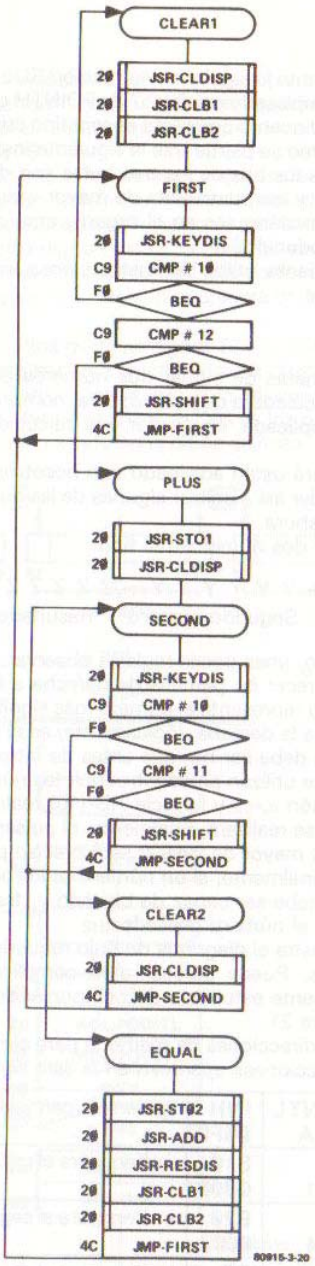
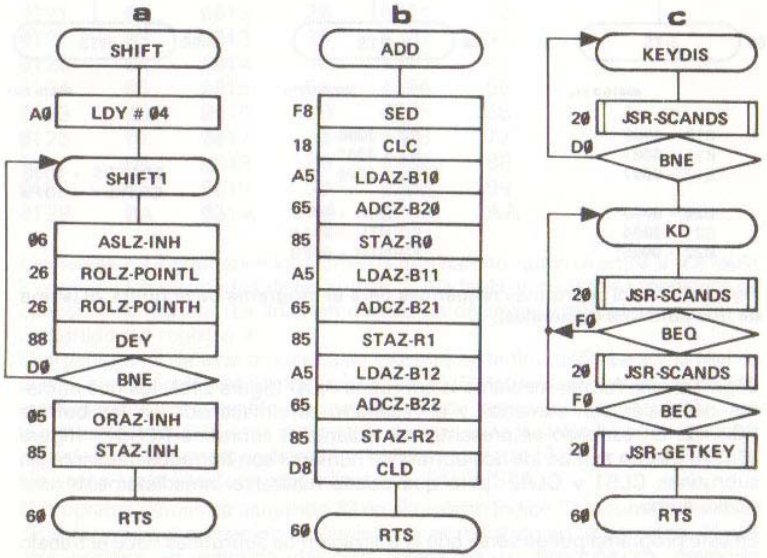


Figura 20. Programa principal para la suma de dos números decimales de 6 dígitos.

El programa (figura 20) comienza con CLEAR1. Esta parte del programa introduce simplemente 00000000 en la pantalla y en los «Buffers» de presentación de datos (ver también las figuras 21d, 21e y 21f). En la sección siguiente (FIRST) el programa salta a la subrutina KEYDIS (figura 21c), que utiliza las rutinas SCANDS y GETKEY contenidas en el programa monitor. Estas rutinas nos son ya familiares: son las encargadas del sondeo del teclado y de la supresión del rebote de contactos. Si se pulsa una tecla, la rutina GETKEY determinará cuál es y su valor, procedente de KEYDIS, será introducido en el acumulador. Se realiza entonces una prueba a través de la instrucción CMP # 10 para ver si la tecla AD (CLEAR) ha sido pulsada y, si es así, el programa saltará hasta CLEAR1. Si la tecla CLEAR no fue pulsada se realizará una segunda prueba (CMP # 12) para ver si fue la tecla « + ».
Si no es así supondremos que se trata de alguna de las teclas 0...9 en cuyo caso el programa saltará hasta la rutina SHIFT (figura 21a) para presentar en pantalla el dígito de que se trate.
Al pulsar la tecla « + » el programa se desplazará hasta la tercera sección (PLUS). Esta sección almacena el valor del primer número en su «Buffer» a través de la subrutina STO1 (figura 21h) y borra la pantalla, preparándola para el segundo número mediante la subrutina CLDISP (figura 21f). La siguiente sección del programa (SECOND) obtiene el valor del segundo número de la misma manera que la primera (FIRST). De nuevo se comprueba si la tecla pulsada fue la CLEAR, en cuyo caso el programa salta a la subrutina CLDISP (figura 21f). La única diferencia es que esta vez se sondea también la tecla « = » (CMP # 11) y si resulta pulsada el programa entrará en la sección final (EQUAL). El contenido de la pantalla es almacenado mediante STO2 (figura



80915-3-21a

80915-3-21b

80916-3-21c

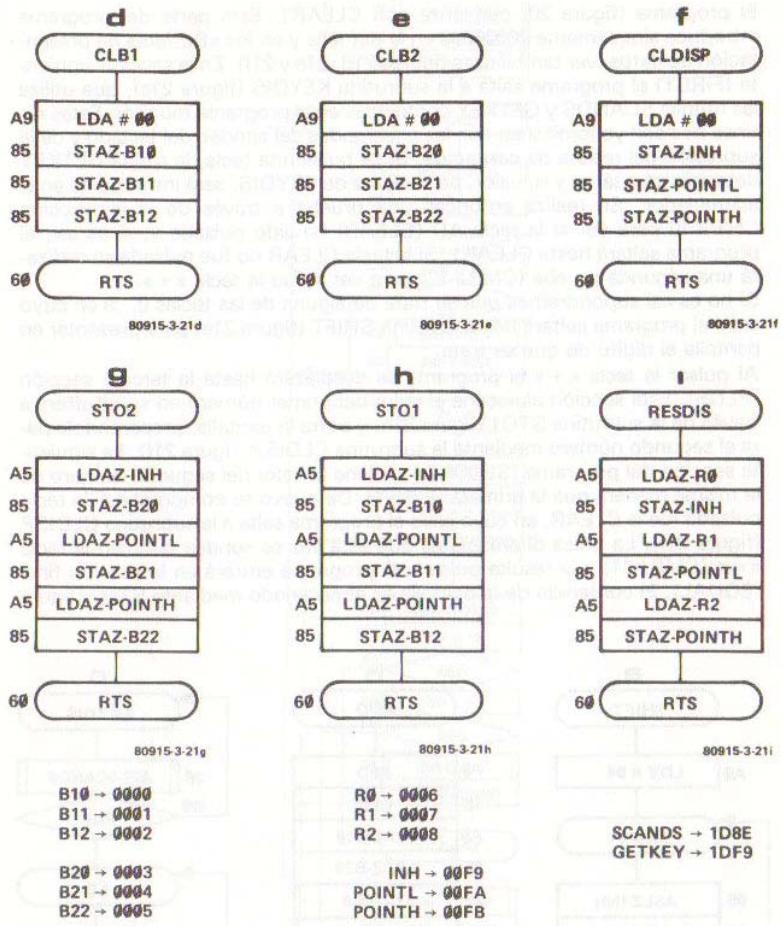


Figura 21. Las 9 subrutinas requeridas para el programa de la figura 20 (suma de dos números decimales).

21g). De esta forma, mediante la subrutina ADD (figura 21b), los dos números decimales son sumados y el resultado es almacenado en los buffers R0...R2. El resultado es presentado mediante la subrutina RESDIS (figura 21i). Al mismo tiempo los dos buffers de números son borrados mediante las subrutinas CLB1 y CLB2, para que pueda realizarse inmediatamente una nueva suma.

En este programa puede verse que la utilización de subrutinas hace el trabajo de programación mucho más sencillo y que, al mismo tiempo, éstas permiten que el programa sea «seguido» sin demasiada dificultad.

Direccionamiento absoluto indexado

En este modo de direccionamiento el contenido de los registros índice X e Y es sumado a la dirección de dieciséis bits suministrada por el segundo y el tercer byte de la instrucción. Lo anterior puede parecer más bien una forma de complicar el asunto, pero este modo de direccionamiento pronto nos demostrará que puede simplificar grandemente la programación.
Recapitulando, las instrucciones de direccionamiento absoluto consisten en tres bytes. El primero para el código de operación real de la instrucción y los dos siguientes (ADL y ADH) para la dirección de que se trate. El direccionamiento absoluto indexado es una variante del direccionamiento absoluto. De nuevo tenemos tres bytes. El primero, sigue siendo para el código de operación de la instrucción, pero esta vez el contenido del registro índice X ó Y es sumado a la dirección contenida en los dos bytes siguientes antes de que la instrucción sea ejecutada.
Si, por ejemplo, el contenido de los registros X ó Y fuera xx y el contenido de ADH y ADL fuera pp y qq, respectivamente, la dirección efectiva resultará ser ppqq + xx.
Para ilustrar este modo de direccionamiento vamos a examinar las tres tablas siguientes. Una tabla (TAB) es un conjunto de direcciones de memoria consecutivas, utilizadas para el almacenamiento de datos o resultados. Las instrucciones de direccionamiento absoluto indexado son ideales cuando tenemos que completar una tabla con los resultados de una operación realizada en dos o más tablas previas.

TAB 1:	0120	TAB 2:	0011	TAB 3:	0300
0120	01	0011	10	0300	11
0121	02	0012	20	0301	22
0122	03	0013	30	0302	33
0123	04	0014	40	0303	44
0124	05	0015	50	0304	55
0125	06	0016	60	0305	66
0126	07	0017	70	0306	77
0127	08	0018	80	0307	88
0128	09	0019	90	0308	99
0129	0A	001A	A0	0309	AA

Las tablas 1 y 2 contienen los números que han de sumarse entre sí. La tabla 3 contiene los resultados de las sumas. Cada tabla se denomina mediante su dirección de partida. La línea en que se ha de trabajar se determina por el contenido del registro X.
El organigrama de este programa se muestra en la figura 22. La suma real se realiza en la sección etiquetada como IND1. El acumulador se carga en primer lugar con el contenido de una posición de memoria dada en TAB1. Tras la instrucción CLC (Borrar el dígito de acarreo para la suma binaria) el contenido de la línea correspondiente en TAB2 es sumado (ADC) y el resultado se almacena en la línea correspondiente de TAB3 (STA).
El programa comienza cargando 09 en el registro índice X; la suma es realizada entonces y el contenido del registro X es reducido en una unidad. El primer número es extraído de esta forma de la dirección de memoria 0120 + 09 = 0129 de TAB1; el segundo lo es de la dirección

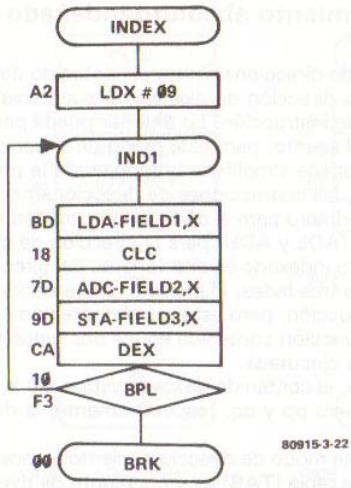


Figura 22. Programa que utiliza direccionamiento absoluto indexado para la suma de los números pertenecientes a dos tablas, almacenando los resultados en una tercera.

0011 + 09 = 001A de TAB2. El resultado de la suma es almacenado en la dirección 0300 + 09 = 0309 de TAB3. En otras palabras, el programa comienza por la línea final de las tablas y trabaja hacia arriba hasta llegar a la línea superior. La suma puede también ser realizada utilizando el método de direccionamiento absoluto:

Byte 1: LDA. Código de operación.
Byte 2: ADL. Dato a ser almacenado.
Byte 3: ADH. Dato a ser almacenado.
Byte 4: CLC. Para suma binaria.
Byte 5: ADC. Código de operación.
Byte 6: ADL. Dato a ser sumado.
Byte 7: ADH. Dato a ser sumado.
Byte 8: STA. Código de operación.
Byte 9: ADL. Dato a ser almacenado.
Byte 10: ADH. Dato a ser almacenado.

¡Y esto es solamente para una línea de cada tabla!
Utilizando lo anterior como ejemplo, esto significará que se requieren 10 × 10 = 100 direcciones de memoria para el programa completo. Comparándolo con el programa de la figura 22, que requiere solamente 16 direcciones de memoria de principio a fin, puede verse claramente la enorme ventaja de utilizar direccionamiento absoluto indexado. La figura 23 muestra el plano de memoria del programa cuando el contenido del registro X es 04.
Otro ejemplo en el que es indispensable el direccionamiento absoluto indexado se muestra en el programa (MOVE) de la figura 24. El objeto de este ejercicio es copiar los contenidos de un campo de direcciones (FROMAD, posi-

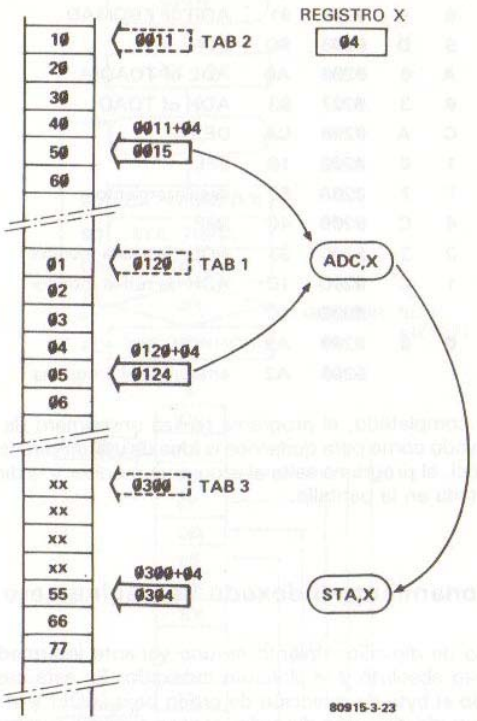


Figura 23. Plano de memoria del programa de la figura 22 cuando el valor del registro X es Ø4.

ción Ø172) en otro (TOAD, posición Ø3AØ). El programa, de hecho, transfiere un bloque de datos desde un área de direcciones a otra. Inicialmente, el registro X es cargado con Ø4. Una vez realizada una transferencia de datos el contenido del registro X es reducido en una unidad. El dato situado en la dirección Ø172 + Ø4 = Ø176 es cargado en el acumulador y después almacenado en la dirección Ø3AØ + Ø4 = Ø3A4. Esto continúa hasta que el registro X llega a ser negativo, en cuyo caso el programa salta y vuelve al monitor. El programa puede ser introducido en el JC tal como sigue:

```
AD                  xxxx  xx
  Ø  2  Ø  Ø        Ø2ØØ  xx
DA      A  2        Ø2ØØ  A2   LDX #
 +      Ø  4        Ø2Ø1  Ø4
 +      B  D        Ø2Ø2  BD   LDA-, X
 +      7  2        Ø2Ø3  72   ADL de FROMAD
```

```
+      0  1  0204  01  ADH de FROMAD
+      9  D  0205  9D  STA-, X
+      A  0  0206  A0  ADL of TOAD
+      0  3  0207  03  ADH of TOAD
+      C  A  0208  CA  DEX
+      1  0  0209  10  BPL
+      F  7  020A  F7  desplazamiento
+      4  C  020B  4C  JMP
+      3  3  020C  33  ADL de rutina monitor
+      1  C  020D  1C  ADH de rutina monitor
AD                020D  1C
 0  2  0  0      0200  A2
GO                0200  A2  arranque del programa
```

Hasta ser completado, el programa realiza un número de ciclos suficientemente elevado como para quitarnos la idea de utilizar direccionamiento absoluto. Al final, el programa salta al programa monitor y la dirección de partida es presentada en la pantalla.

Direccionamiento indexado de página cero

Este modo de direccionamiento es una variante intermedia entre el direccionamiento absoluto y el absoluto indexado. En este caso, sólo debe ser introducido el byte de dirección de orden bajo (ADL) y el código de operación. El byte de dirección de orden alto es siempre el mismo (ØØ). La dirección efectiva es calculada a partir del byte de datos inmediatamente siguiente al código de operación de la instrucción, sumándole el contenido de los registros índice X ó Y.
Debe notarse que el direccionamiento indexado de página cero es precisamente eso: de página cero. Esto significa que si la suma del registro índice y el byte de dirección de orden bajo excede 225 (FF) el bit de acarreo será descartado. En otras palabras, si el contenido del registro índice X ó Y fuera 8B y el byte de dirección de orden bajo fuera B1, la dirección efectiva resultaría ser ØØ3C y no Ø13C.

Direccionamiento indirecto

¡Por fin, podemos ver el final del túnel! Vamos a describir la última de las posibilidades de direccionamiento.
Las instrucciones que utilizan direccionamiento indirecto simple, consisten en tres bytes. El primero para el código de operación real; el segundo y tercero, dan una dirección de 16 bits en la que se encuentra la dirección real. Esto significa que la dirección indirecta puede estar localizada en cualquier lugar de la memoria. ¿Suena complicado? Creemos que no.
Como ejemplo, vamos a suponer que deseamos saltar a otra parte del

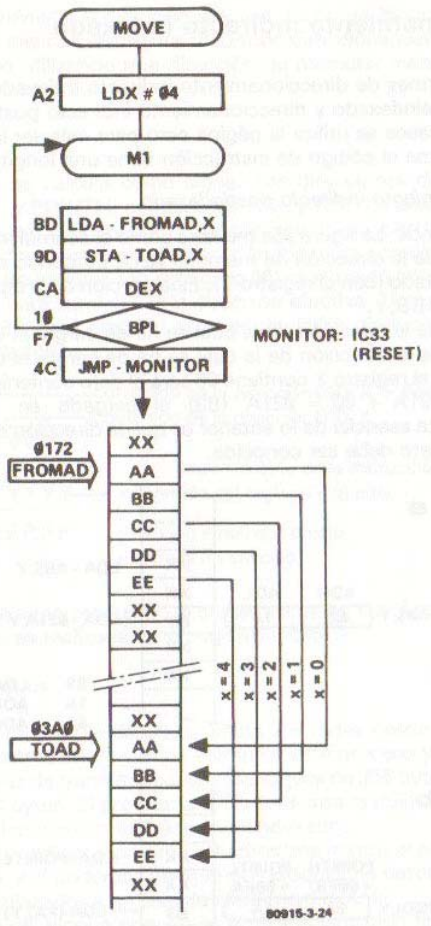


Figura 24. Un programa que copia un bloque de datos de un área de memoria a otra.

programa principal, pero no sabemos la dirección real a la que queremos saltar. Sabemos, sin embargo, que la dirección de salto está almacenada en la dirección 2B84. El código de operación para la instrucción de salto (JMP) utilizando direccionamiento indirecto, es 6C. de esta forma, la instrucción completa sería 6C842B. Si, por ejemplo, las direcciones de memoria 2B84 y 2B85 contuvieran Ø6 y 1A, respectivamente, el programa saltaría a la dirección 1AØ6.

Nota: Los registros X e Y son de ocho bits cada uno. Esto significa que el máximo número de bytes de datos que pueden ser trasladados de esta manera es 256.

Direccionamiento indirecto indexado

Hay dos formas de direccionamiento indirecto indexado: Direccionamiento indirecto preindexado y direccionamiento indirecto postindexado.
En ambos casos se utiliza la página cero para calcular la dirección efectiva. De esta forma el código de instrucción tiene una longitud de dos bytes.

Direccionamiento indirecto postindexado

Recapitulemos: La figura 25a muestra cómo el acumulador es cargado con el contenido de la dirección de memoria Ø21A utilizando direccionamiento absoluto indexado (con el registro Y). Esta porción de programa es como sigue:
B9 LDA-ABS,Y.
1A ADL de la dirección de la cual se ha de cargar el dato.
Ø2 ADH de la dirección de la cual se ha de cargar el dato.
Puesto que el registro Y contiene ØØ será el dato contenido en la dirección de memoria Ø21A + ØØ = Ø21A (B3) el cargado en el acumulador. La característica esencial de lo anterior es que la dirección en que hemos de encontrar el dato debe ser conocida.

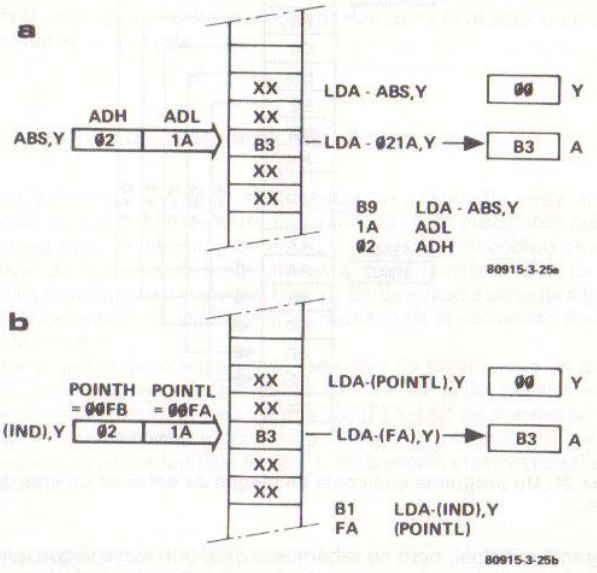


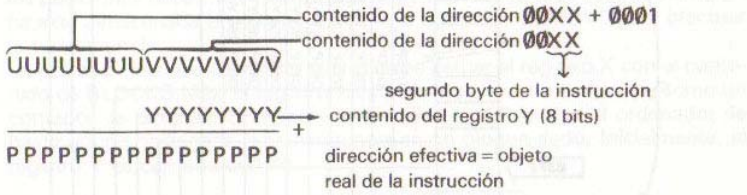
Figura 25. Comparación de las instrucciones de carga que utilizan direccionamiento absoluto indexado y direccionamiento indirecto postindexado. La última ocupa un byte menos.

El direccionamiento postindexado utiliza el registro índice Y para calcular la dirección efectiva. El segundo byte de la instrucción especifica una dirección en la primera página de la memoria en la que puede ser encontrada una dirección indirecta. El contenido del registro Y es sumado entonces a esta direc-

ción indirecta para suministar la dirección efectiva. La figura 25b muestra que pueden obtenerse los mismos resultados utilizando direccionamiento indirecto postindexado, pero utilizando una dirección de memoria menos. La instrucción real sería entonces:

B1 LDA-(IND),Y.
FA (POINTL).

La dirección efectiva se calcula como sigue. Las direcciones de memoria ØØFA y ØØFB (POINTL y POINTH) contienen el ADL y ADH de la dirección indirecta, respectivamente. Indicando al ordenador que busque en la dirección ØØFA se encuentra automáticamente la dirección indirecta completa (Ø21A). El contenido del registro índice Y (en este caso ØØ) es sumado entonces a esta dirección indirecta para suministrar la dirección efectiva. Una vez más, el contenido de Ø21A (B3) será cargado en el acumulador. Esto puede representarse también de la forma:



en donde cualquier acarreo producido al sumar los bytes V e Y es sumado al byte U. La suma real es realizada en el registro índice Y.

Transferencia de bloques

Hemos visto ya que es posible desplazar hasta 256 bytes desde un área de memoria a otra. Nosotros ahora vamos a ampliar este proceso y escribir un programa que es capaz de transferir hasta 255 bloques de 256 bytes. Esto hará un total de 65.280 bytes. El programa también ilustra la utilización de instrucciones de direccionamiento indirecto postindexado.
Antes de explicar en detalle el programa echemos una mirada al plano de memoria de la figura 26. Allí podemos ver que dos bloques de datos de 256 bytes cada uno son desplazados a un área de memoria diferente.
La primera dirección del bloque primero es Ø2ØØ y la dirección final es Ø2FF. Esta dirección se encuentra también en dos posiciones de memoria de la página cero; BEG = ØØ y BEG + 1 = Ø2. Este primer bloque de datos ha de desplazarse al área A8ØØ...A8FF. La primera dirección de este área se encuentra también en dos posiciones de memoria de la página cero: MOV = ØØ y MOV + 1 = A8. Las diversas direcciones de memoria de la página cero han sido etiquetadas como sigue:

BEG: Dirección ØØØØ. Su contenido se denomina FRADL y es igual a ØØ.
BEG + 1: Dirección ØØØ1. Su contenido se denomina FRADH y es igual a Ø2.
MOV: Dirección ØØØ2. Su contenido se denomina TOADL y es igual a ØØ.

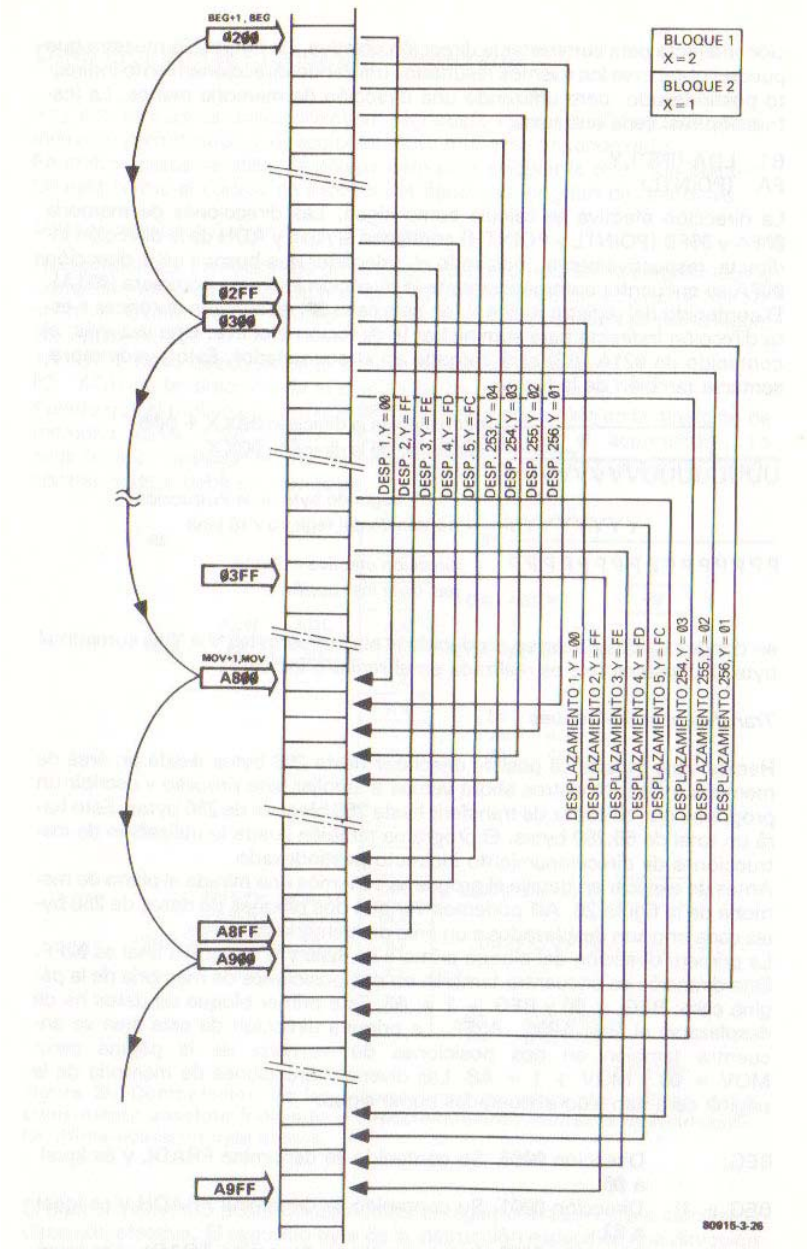


Figura 26. Dos bloques de datos de 256 bytes cada uno son desplazados a diferentes secciones de memoria. El programa real está en la figura 27.

MOV + 1: Dirección 0003. Su contenido se denomina TOADH y es igual a A8.

BLOCKS: Dirección 0004. Su contenido se denomina N y es igual a 02. (Corresponde al número de bloques que han de ser desplazados).

Las denominaciones son bastante lógicas; FR viene de FRom y TO, lógicamente, viene de TO. Para nuestros lectores poco familiarizados con la lengua inglesa será preciso explicar el significado de ambas palabras: From viene a significar *desde* y TO significa *hacia*. Las notaciones ADL (Byte de orden bajo) y ADH (Byte de orden alto) nos son ya bastante familiares.

El programa real para desplazamientos de bloques de datos no es tan complicado como podría esperarse. Consiste en dos subrutinas cuyos organigramas se muestran en la figura 27. El principio del programa (DEFMOV) almacena simplemente la información de las direcciones (de forma indirecta) en las posiciones 0000...0004 de la página cero. Una vez toda esta información ha sido almacenada el programa salta a la subrutina BLMOVE para efectuar la transferencia.

La primera cosa que hace esta subrutina es cargar el registro X con el contenido de BLOCKS (02). El registro X está siendo utilizado de hecho, como un contador de bloques. El registro Y se utiliza para informar al ordenador de hasta dónde ha llegado la transferencia en un bloque dado. Inicialmente, el registro Y es cargado con 00.

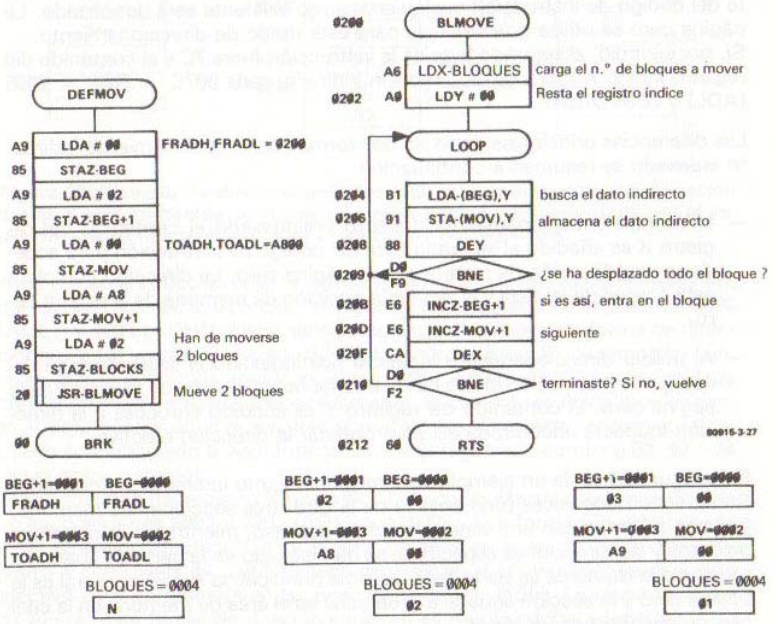


Figura 27. Programa completo para la transferencia de hasta 255 bloques de datos de 256 bytes cada uno. Se utiliza direccionamiento indirecto postindexado.

En la sección denominada LOOP el procesador carga el acumulador con el contenido de 0200 + 00 (Y = 0) y almacena este dato en la posición A800 + 00. El registro Y es entonces decrementado (= FF) y el dato de la posición 0200 + FF es almacenado en la posición A800 + FF. El registro Y es decrementado de nuevo (= FE) y el proceso continúa hasta que el registro Y contenga 00. Tan pronto como esto sucede el contenido de BEG + 1 y MOV + 1 son incrementados, quedando dispuestos para el siguiente bloque de datos, y el contenido del registro X es decrementado. El programa salta entonces a LOOP y el segundo bloque de datos es trasladado. Una vez ha sido trasladado el segundo bloque completo, los contenidos de BEG + 1 y MOV + 1 son incrementados una vez más, y el contenido del registro X decrementado. Cuando el registro X contenga 00 el procesador volverá al programa principal y se detendrá.

Direccionamiento indirecto preindexado

El direccionamiento indirecto preindexado utiliza el registro X. Esta vez, como su propio nombre sugiere, el segundo byte del código de instrucción es añadido al contenido del registro índice X antes de haber encontrado la dirección efectiva. Al utilizar direccionamiento indirecto preindexado, es preciso insistir en el problema en la suma, que ocasiona un ADH distinto de 00. En otras palabras, cuando el contenido del registro X es sumado al segundo byte del código de instrucción cualquier acarreo existente será descartado. La página cero se utiliza una vez más para este modo de direccionamiento.
Si, por ejemplo, el segundo byte de la instrucción fuera 7C y el contenido del registro índice X fuera 89, la dirección indirecta sería 007C + 0089 = 0005 (ADL) y 0006 (ADH).

Las diferencias principales entre las dos formas de direccionamiento indirecto indexado se resumen a continuación:

- Utilizando direccionamiento indirecto preindexado el contenido del registro X es añadido al segundo byte del código de instrucción para acceder a una dirección de memoria en la página cero. La dirección completa está contenida en ésta y la siguiente posición de memoria de la página cero.

- Al utilizar direccionamiento indirecto postindexado el segundo byte del código de instrucción indica directamente una dirección de memoria en la página cero. El contenido del registro Y es añadido entonces a la dirección indirecta encontrada allí para obtener la dirección efectiva.

En la figura 28 se da un ejemplo de direccionamiento indirecto preindexado. Se muestran dos veces (una encima de la otra) tres secciones de memoria. Las tres de arriba dan una vista general del proceso, mientras la información de datos y las direcciones específicas se han incluido en la parte de abajo. La sección a la izquierda es parte del programa principal, la sección central es la página cero y la sección situada a la derecha es el área de memoria en la cual han de encontrarse los datos.

En la dirección 032A del programa principal encontramos la instrucción A190 LDA-(INDX), que dice al computador que la dirección indirecta será en-

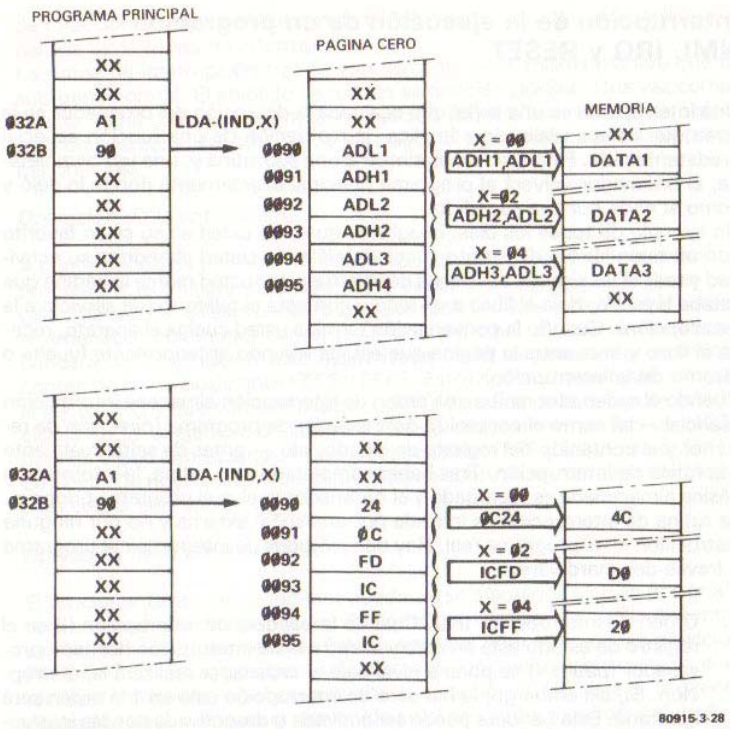


Figura 28. Ejemplo de direccionamiento indirecto preindexado. La dirección efectiva está contenida en la posición de la página cero, apuntada por el segundo byte de la instrucción más el contenido del registro X.

contrada en la dirección ØØ9Ø + (el contenido del registro X). Si el contenido del registro X es ØØ la dirección efectiva será encontrada en la posición ØØ9Ø (ADL1) y ØØ91 (ADH1). Debe tenerse cuidado al utilizar esta forma de direccionamiento, ya que si el contenido del registro X fuera Ø1, la dirección indirecta estaría contenida en las posiciones ØØ91 y ØØ92. Estas dos direcciones contienen ADH1 y ADL2, respectivamente, lo cual no resultaría muy correcto si tratamos de obtener resultados reacionales de nuestro programa. Esto puede ser remediado asegurándose de que el registro X contiene ØØ, Ø2 ó Ø4 (en nuestro ejemplo).

Si el contenido del registro X fuera Ø2, la dirección efectiva sería encontrada en las posiciones ØØ92 y ØØ93 (ADL2 y ADH2) y la información almacenada en la dirección efectiva (DATA2) sería cargada en el acumulador. Si el registro X contuviera Ø4, DATA3 sería cargada en el acumulador (la dirección efectiva sería encontrada en las posiciones ØØ94 y ØØ95). La razón principal de utilizar esta forma de direccionamiento es que la página cero puede ser usada como tabla de trabajo de varios punteros. Cada uno de los punteros indicará una dirección específica en la que los datos a procesar pueden ser encontrados.

Interrupción de la ejecución de un programa: NMI, IRQ y RESET

Una interrupción es una señal que ocasiona la detención del ordenador en la tarea que estaba realizado e implica la realización de una función especial predeterminada. Esta función es similar a una subrutina y, una vez completada, el ordenador volverá al programa principal exactamente donde lo dejó y como si nada hubiera sucedido.
Un ejemplo de todos los días: imagínese que está usted en su sillón favorito con un buen libro y de pronto suena el teléfono. Usted abandona su actividad y realiza las siguientes rutinas de interrupción: usted marca la página que estaba leyendo, deja el libro a un lado y contesta al teléfono (da servicio a la interrupción). Cuando la conversación termina usted cuelga el aparato, recoge el libro y encuentra la página que estaba leyendo anteriormente (vuelta o retorno de la interrupción).
Cuando el ordenador recibe una orden de interrupción almacena información esencial —tal como el contenido del contador de programa (dirección de retorno) y el contenido del registro de estado, etc.— antes de saltar realmente a la rutina de interrupción. Tras haber completado esta rutina, la información básica almacenada es recargada y el ordenador vuelve al programa principal.
La rutina de interrupción es iniciada por una señal externa y no por ninguna instrucción en el programa real. Hay dos métodos de interrumpir el programa a través del «hardware».

1. Orden de interrupción, IRQ. Cuando la bandera de interrupción (I) en el registro de estado está en cero y la entrada de interrupción del microprocesador (patilla 4) se pone a nivel bajo el ordenador realizará la interrupción. Si, sin embargo, la bandera de interrupción está en 1 la orden será ignorada. Esta bandera puede ser activada o desactivada con las instrucciones siguientes:
   CLI (Código de operación 58), con lo que I = Ø; interrupción factible.
   SEI (Código de operación 78), con lo que I = 1; interrupción no factible.
2. Interrupción prioritaria, NMI (Non-Maskable-Interrupt). Independientemente del estado de la bandera de interrupción se realizará la interrupción siempre que la entrada NMI (Patilla 6) del microprocesador esté en nivel bajo.

Las señales de interrupción IRQ y NMI, así como RES (todavía por describir) pueden ser denominadas «instrucciones de hardware» o, con más precisión, «instrucciones de salto en hardware». Como puede verse de lo anterior la instrucción IRQ es condicional y la instrucción NMI es incondicional. Todo esto, claro está, si es que pueden denominarse verdaderamente instrucciones.

Operación de la entrada NMI

Al poner en nivel bajo la entrada NMI el procesador investiga el contenido de la memoria en FFFA y FFFB. Estas posiciones de memoria contienen la dirección de partida de la rutina de interrupción. En el ejemplo de la figura 29 esta dirección de partida es Ø2ØØ. El byte de orden bajo está almacenado en FFFA y el byte de orden alto en FFFB. El vector NMI es muy similar a los punteros

de direcciones mencionados antes: simplemente «apunta» a la dirección de partida de la rutina de interrupción.
La rutina de interrupción trabaja exactamente de la misma manera que una subrutina normal. El anidado de rutinas es también posible. Una vez completada la rutina de interrupción, la instrucción RTI (ReTurn from Interrupt; código de operación 40) hará volver de nuevo al procesador dentro del programa principal.

Operación de la entrada IRQ

A continuación de un IRQ (Interrupt ReQuest = Solicitud de Interrupción) el procesador examinará el estado de la bandera de interrupción (I) en el registro de estado. Si esta bandera está activada (= 1) la orden será ignorada y el ordenador continuará con el programa principal. Si, por el contrario, la bandera de interrupción está desactivada (= 0) el procesador examina el contenido de las posiciones FFFE y FFFF. Estos son los bytes reservados para el vector IRQ y que contienen la dirección de partida de la rutina de interrupción IRQ. En el ejemplo de la figura 29 esta dirección de partida es 24C3. El byte de orden alto es almacenado en la posición FFFF y el byte de orden bajo en FFFE. De nuevo es posible el anidado de subrutinas y la rutina de interrupción con la instrucción RTI.

Operación de la entrada Reset

El funcionamiento del ordenador puede verse afectado también por una tercera «instrucción de hardware». Al poner en nivel bajo la patilla 40 (entrada de reset) del microprocesador éste examina el contenido de las direcciones FFFC y FFFD. Estas posiciones de memoria son, como cabía esperar, el *vector reset* y contiene, en el caso del JC, la dirección 1C1D. Esta es la dirección de partida de la rutina reset contenida en el programa monitor. Así, cada vez que pulsamos la tecla reset el ordenador salta realmente a la rutina reset que comienza en la dirección mencionada.
Es interesante observar que los tres «vectores» que hemos mencionado para las instrucciones de «hardware» se han situado en la página FF (la más alta posible). Esto se ha hecho así al no poder ser direccionada directamente esta página, debido al direccionamiento incompleto del JC. Sin embargo, las líneas de dirección necesarias para decodificar esta página son, de izquierda a derecha, A15...A8.
Tras examinar el circuito del Junior Computer (ver capítulo 1) se encontrará que las líneas de direcciones A13, A14 y A15 no son utilizadas. Esto quiere decir que estas líneas de direcciones no tienen influencia en las direcciones de memoria reales a direccionar. Al no utilizar estas tres líneas nos resulta el direccionamiento incompleto de que hablamos; la dirección más alta que podemos direccionar resulta ser 1FFF, al ser la página 1F la página direccionable más alta de que disponemos. El procesador de esta forma «enloquecería» al ponerse a buscar unas direcciones que no puede alcanzar y se dedicaría a indagar las direcciones 1FFA...1FFF en busca de unos vectores que no están allí. Estas direcciones son, por supuesto, parte del programa monitor grabado en EPROM y su contenido no puede ser alterado a través del teclado.
Un ejemplo más completo de lo que sucede cuando el ordenador recibe una orden de interrupción (NMI o IRQ) puede verse en el plano de memoria de la

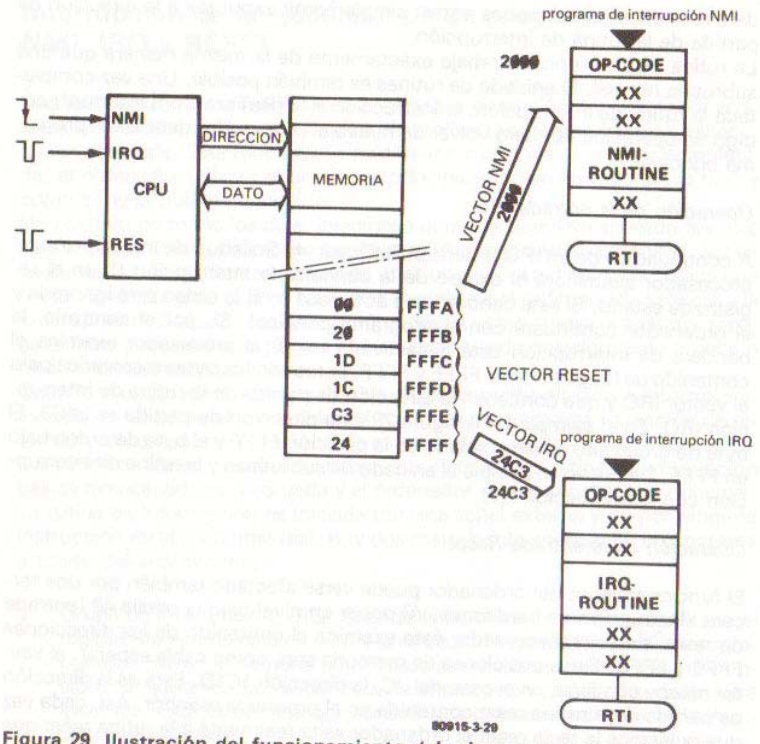


Figura 29. Ilustración del funcionamiento del microprocesador ante una señal de interrupción.

figura 30. Aquí la página Ø1 es utilizada como stack y la página Ø3 contiene el programa principal. El vector IRQ «apunta» a 24C3, dirección comienzo de la rutina IRQ, y el vector NMI «apunta» a 2000, principio de la rutina NMI. Imaginemos que el ordenador está realizando la instrucción contenida en Ø343 cuando la entrada NMI queda en nivel bajo. El ordenador no atenderá la orden de interrupción hasta que haya terminado la instrucción que realiza en esos momentos. Una vez dispuesto, el ordenador introducirá en el stack el byte de orden alto (PCH) de la dirección presente en el contador de programa. A continuación incrementará el puntero de stack. El byte de orden bajo del contador de programa (PCL) será cargado a continuación en el stack y el puntero de éste de nuevo incrementado. El puntero de stack quedará apuntando ahora a la dirección Ø1FC, que es donde se ha almacenado el contenido del registro de estado y se procederá a su introducción, etc. Tras todo el proceso el puntero de stack quedará apuntado a la dirección Ø1FB.
La siguiente fase de la operación es el salto al comienzo de la rutina NMI, que en nuestro caso es 2000. Al mismo tiempo es activada la bandera de interrupción, para que cualquier orden de interrupción (IRQ) subsiguiente sea ignorada. Si la interrupción fue causada por una instrucción IRQ el proceso es bas-

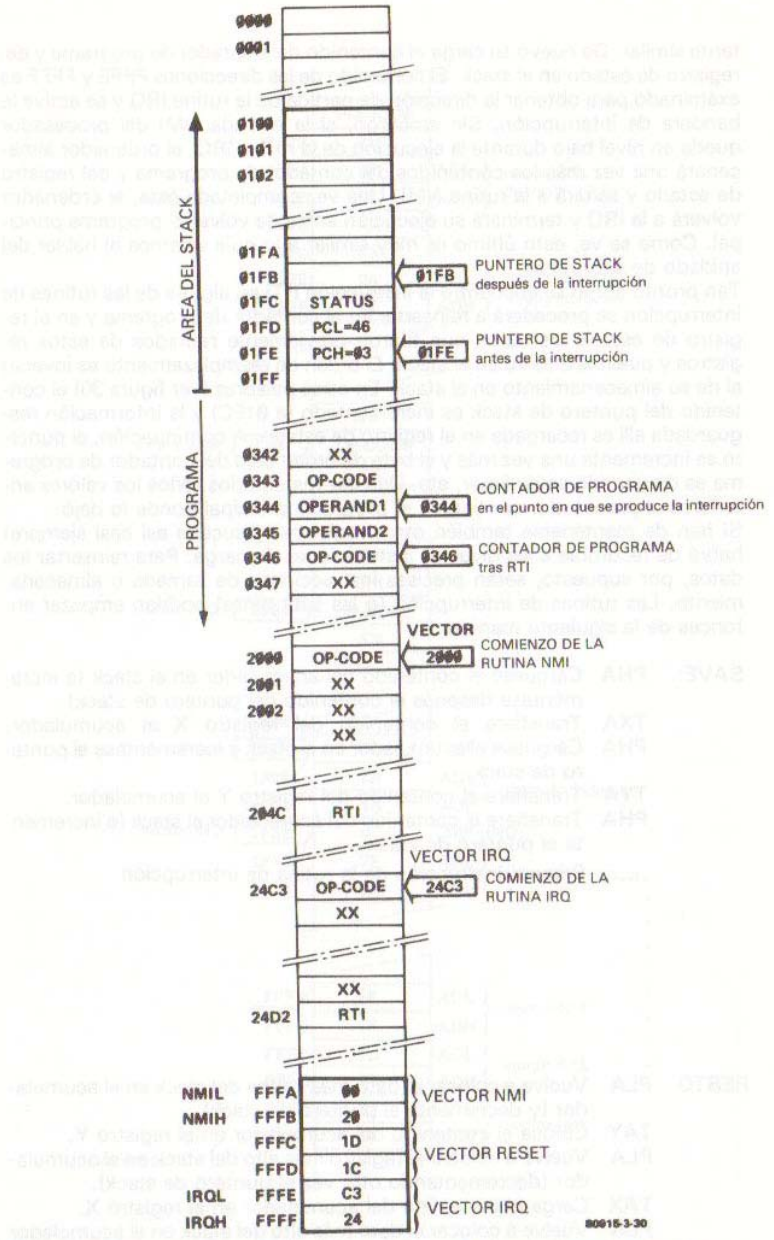


Figura 30. Plano de memoria del proceso que se lleva a cabo con las órdenes o solicitudes de interrupción.

tante similar. De nuevo se carga el contenido del contador de programa y del registro de estado en el stack. El contenido de las direcciones FFFE y FFFF es examinado para obtener la dirección de partida de la rutina IRQ y se activa la bandera de interrupción. Sin embargo, si la entrada NMI del procesador queda en nivel bajo durante la ejecución de la rutina IRQ, el ordenador almacenará una vez más los contenidos del contador de programa y del registro de estado y saltará a la rutina NMI. Una vez completada ésta, el ordenador volverá a la IRQ y terminará su ejecución antes de volver al programa principal. Como se ve, esto último es muy similar a lo quie veíamos al hablar del anidado de subrutinas.

Tan pronto como se encuentre la instrucción RTI en alguna de las rutinas de interrupción se procederá a reinsertar en el contador de programa y en el registro de estado los datos que fueron previamente retirados de estos registros y puestos a salvo en el stack. El orden de reemplazamiento es inverso al de su almacenamiento en el stack. En otras palabras (ver figura 30) el contenido del puntero de stack es incrementado (a Ø1FC) y la información resguardada allí es recargada en el registro de estado. A continuación, el puntero se incrementa una vez más y el byte de orden bajo del contador de programa es restaurado en su lugar, etc. Una vez restaurados todos los valores anteriores el procesador continuará el programa principal donde lo dejó.

Si han de mantenerse también otros registros (y sucede así casi siempre) habrá de recurrirse a las diversas instrucciones de carga. Para reinsertar los datos, por supuesto, serán precisas instrucciones de llamada o almacenamiento. Las rutinas de interrupción (o las subrutinas) podrían empezar entonces de la siguiente manera:

```
SAVE:   PHA   Cárguese el contenido del acumulador en el stack (e incremén-
              tese después el contenido del puntero de stack).
        TXA   Transfiere el contenido del registro X al acumulador.
        PHA   Cárguese el acumulador en el stack e increméntese el puntero de stack.
        TYA   Transfiere el contenido del registro Y al acumulador.
        PHA   Transfiere el contenido del acumulador al stack (e incrementa el puntero de stack).
        ..... Primera instrucción de la rutina de interrupción.
        .
        .
        .
        .
        .
        .
        .
        .
RESTO   PLA   Vuelve a colocar el dato más «alto» del stack en el acumulador (y decrementa el puntero de stack).
        TAY   Coloca el contenido del acumulador en el registro Y.
        PLA   Vuelve a colocar el registro más alto del stack en el acumulador (decrementando otra vez el puntero de stack).
        TAX   Carga el contenido del acumulador en el registro X.
        PLA   Vuelve a colocar el dato más alto del stack en el acumulador y decrementa una vez más el puntero.
FIN     RTI   Retorna al programa desde la rutina de interrupción.
```

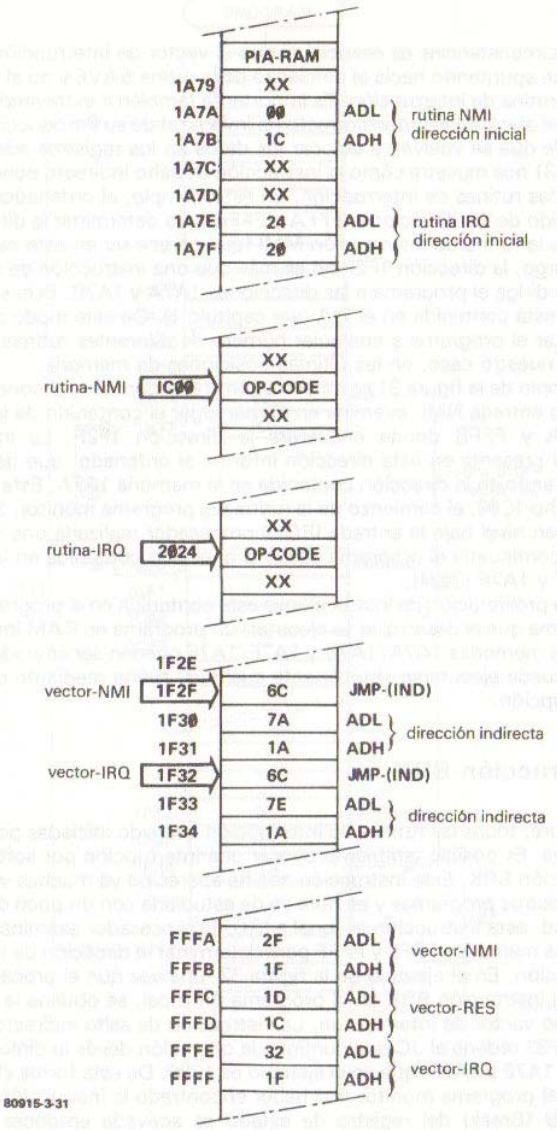


Figura 31. Atención a una interrupción mediante instrucción de salto indirecto. Los contenidos del punto de dirección indirecta y la dirección efectiva son programables.

En estas circunstancias es esencial el que el vector de interrupción que nos afecte esté apuntando hacia el comienzo de la rutina SAVE y no al comienzo real de la rutina de interrupción. Es importante también ir extrayendo la información del stack en el orden correcto (el inverso al de su introducción) y asegurarse de que se vuelven a colocar los datos en los registros adecuados.
La figura 31 nos muestra cómo la instrucción de salto indirecto puede ser utilizada en las rutinas de interrupción. En este ejemplo, el ordenador examina el contenido de las direcciones FFFA y FFFB para determinar la dirección de partida de la rutina de interrupción MMI (que parece ser en este caso 1F2F). Sin embargo, la dirección 1F2F no es más que una instrucción de salto indirecto que dirige el programa a las direcciones 1A7A y 1A7B. Esta sección de memoria está contenida en el PIA (ver capítulo 1). De este modo puede hacerse saltar el programa a cualquier número de diferentes rutinas, situadas como en nuestro caso, en las últimas posiciones de memoria.
En el ejemplo de la figura 31 se muestra cómo el ordenador, al ponerse en nivel bajo la entrada NMI, examina en primer lugar el contenido de las memorias FFFA y FFFB donde encuentra la dirección 1F2F. La instrucción (6C7A1A) presente en esta dirección informa al ordenador que debe continuar operando en la dirección contenida en la memoria 1A7A. Esta dirección es de hecho 1CØØ, el comienzo de la rutina del programa monitor. Si hubiera quedado en nivel bajo la entrada IRQ el procesador realizaría una operación similar y continuaría el programa desde la dirección contenida en las memorias 1A7E y 1A7F (2Ø24).
Toda esta proliferación de instrucciones está contenida en el programa monitor de forma que al desarrollar (o ejecutar) un programa en RAM los contenidos de las memorias 1A7A/1A7B y 1A7E/1A7F pueden ser alterados de forma que pueda ejecutarse virtualmente cualquier rutina mediante una orden de interrupción.

La instrucción BRK

Hasta ahora, todas las rutinas de interrupción han sido iniciadas por influencia externa. Es posible también provocar una interrupción por software con la instrucción BRK. Esta instrucción nos ha aparecido ya muchas veces al final de nuestros programas y es hora ya de estudiarla con un poco de detalle. En realidad, esta instrucción es igual a IRQ. El procesador examina el contenido de las memorias FFFE y FFFF para determinar la dirección de la siguiente instrucción. En el ejemplo de la figura 32, una vez que el procesador encuentra la instrucción BRK en el programa principal, se obtiene la dirección 1F32 como vector de interrupción. La instrucción de salto indirecto en la dirección 1F32 ordena al JC que continúe la operación desde la dirección contenida en 1A7E y 1A7F, que en el ejemplo es 1CØØ. De esta forma el procesador salta al programa monitor tras haber encontrado la instrucción BRK. La bandera B (Break) del registro de estado es activada entonces al haber ocurrido una interrupción vía software (esta bandera permanece en Ø durante las interrupciones normales-vía hardware).
Esto concluye el capítulo sobre las diversas posibilidades de direccionamiento. El capítulo quinto nos permitirá ampliar lo ya aprendido, analizando interesantes ejemplos de programación.

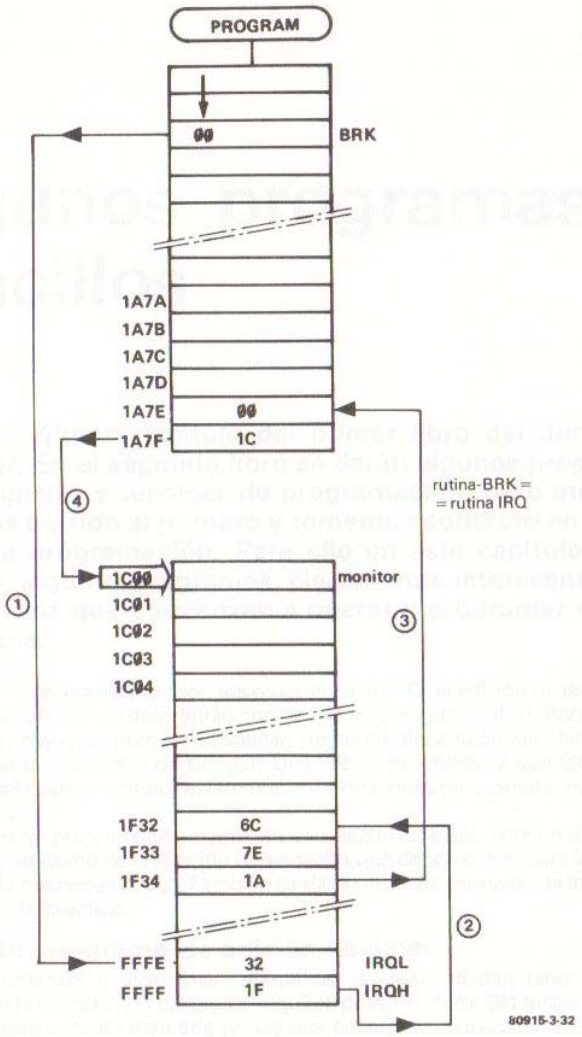


Figura 32. La instrucción BRK es una forma de interrupción realizada mediante software, es decir, sin recurrir al teclado en el momento en que se desea. Puede considerarse como una interrupción programable. Este ejemplo muestra los cuatro saltos efectuados antes de que la rutina BRK (= rutina IRQ) sea efectuada realmente.

5

Algunos programas sencillos

Este es el último capítulo del primer libro del Junior Computer. En el segundo libro se darán algunos programas complejos y técnicas de programación, pero antes saquemos partido al primero y tomemos contacto en serio con la programación. Para ello en este capítulo se describen algunos programas, ciertamente interesantes, para aquéllos que comienzan a operar y programar con su máquina.

Las rutinas del programa monitor, los programas de I/O, la edición hexadecimal y el ensamblador, se describirán con detalle en el segundo libro. Pero antes de correr, hay que aprender a caminar, ¡nadie empieza su primera lección de piano con un concierto de Chopin! Una vez comprendido y asentado el contenido del cuarto capítulo, estaremos en condiciones de acometer mayores empresas.
Los ejemplos de programación dados en este capítulo se acompañan de organigramas, así como de la relación de las teclas que deben pulsar para introducirlos en la memoria del JC. También se darán algunos ejemplos de índole eminentemente práctica.

Listado del programa de adición decimal

Antes de comenzar a programar, puede ser de gran utilidad tener este programa en la memoria. El programa requiere pulsar en total 594 teclas para llenar 196 posiciones de memoria ¡y hay que hacerlo sin equivocarse!
Una vez completada la operación de entrada en memoria, se podrá iniciar el programa que permite la adición de dos números decimales de seis cifras. Este programa ya se mencionó en el capítulo tercero: véase la figura 14, («nombre y dirección» de los tres buffer del visualizador), figura 20 (organigrama general del programa) y figura 21 (organigrama detallado de las nueve subrutinas).
La elaboración detallada de los organigramas es una de las etapas más importantes en el desarrollo de un programa. En efecto, de este modo los bloques del diagrama (decisiones, sentencias, condiciones) pueden transfor-

marse directamente en instrucciones, y éstas a su vez en códigos hexadecimales operativos, y se puede tener una idea bastante aproximada del número de BYTES necesarios para un programa específico. Lo cual, en algunos casos, puede ser de gran utilidad para «ajustar» un programa.
Para empezar echemos una mirada al programa de la figura 1 que consta de las siguientes secciones:

posiciones **Ø2ØØ** . . . **Ø248**: programa principal (figura 20)
posiciones **Ø249** . . . **Ø258**: subrutina SHIFT (figura 21a)
posiciones **Ø259** . . . **Ø26E**: subrutina ADD (figura 21b)
posiciones **Ø26F** . . . **Ø281**: subrutina KEYDIS (figura 21c)
posiciones **Ø282** . . . **Ø28A**: subrutina CLB1 (figura 21d)
posiciones **Ø28B** . . . **Ø293**: subrutina CLB2 (figura 21e)
posiciones **Ø294** . . . **Ø29C**: subrutina CLDISP (figura 21f)
posiciones **Ø29D** . . . **Ø2A9**: subrutina STO2 (figura 21g)
posiciones **Ø2AA** . . . **Ø2B6**: subrutina STO2 (figura 21h)
posiciones **Ø2B7** . . . **Ø2C3**: subrutina RESDIS (figura 21i)

Preparación del programa
La primera instrucción (situada en la dirección Ø2ØØ) conduce a la subrutina JSR. Sin embargo, la dirección de inicio de esta subrutina no siempre es conocida, debido a que todavía no se ha determinado la extensión del programa principal y de las subrutina implicadas. Lo que nos permite señalar dos reglas prácticas generales.
1.— Ahorraremos mucho tiempo si escribimos un listado del programa antes de introducirlo en la memoria del computador, y aún más si previamente desarrollamos un organigrama detallado.
2.— Deben reservarse suficientes posiciones de memoria para datos y desplazamientos. Estos espacios se podrán llenar más tarde según convenga. Esta última regla es muy importante, ya que:
3.— Todo programa debe estar contenido en una misma área de memoria. Esto se aplica igualmente a las subrutinas.
Al aplicar estas reglas debe tenerse presente que el contenido del contador de programa se incrementa después de cada instrucción, y que si la CPU encuentra un espacio «vacío» (dirección sin datos ni instrucciones) no sabrá que hacer con las instrucciones siguientes (o las malinterpretará), lo que normalmente resulta desastroso. En programas largos es normal saltarse algunos pasos de memoria, por accidente o al pulsar inadvertidamente la tecla « + », por lo cual debe tenerse mucho cuidado al teclear los programas. Para crear «huecos», se utilizan la instrucción NOP (no operación de código EA). La principal utilidad de esta instrucción es crear un espacio vacío en el programa, para más tarde rellenarlo, en caso de que se haya olvidado alguna instrucción (o hagan falta nuevas). Obviamente esto siempre es más corto que teclear de nuevo el programa completo...
Es muy normal cometer errores durante el desarrollo de las diferentes etapas de un programa, por lo que es conveniente comprobar la versión final del mismo antes de ponerlo en marcha. Esta es una tarea que puede realizar el mismo Junior Computer.

Verificación de los programas
Las teclas de control AD, DA y « + » pueden usarse para comprobar y (si fuera necesario) alterar el contenido de una posición específica de memoria. Para examinar un paso concreto de memoria se pulsa en el teclado:

AD XXXX

Donde XXXX es la dirección de la memoria que deseamos comprobar. Esta dirección aparecerá en la parte izquierda del visualizador (o pantalla) y el dato contenido en ella, a la derecha (dos dígitos). Para examinar la siguiente posición de memoria, bastará con pulsar la tecla « + » con lo cual la dirección que visualizará la pantalla será XXXX + 0001 y el dato mostrado será el correspondiente a la subsiguiente posición de memoria. De esta forma podremos examinar el programa completo. Si fuera preciso corregir el dato de una dirección en particular, es importante asegurarse de que se tiene en pantalla la dirección correcta. Una vez confirmado esto, pulsaremos la tecla «DA» y a continuación teclearemos el dato correcto. Si pulsamos la tecla « + », podremos igualmente modificar el dato contenido en la siguiente dirección, y en general de cualquiera que se necesite, sin más que repetir esta operación las veces que sea preciso.

Como puede verse, disponemos de las funciones necesarias para verificar o alterar los datos contenidos en cualquier posición de memoria, ¡pero es mucho mejor no tener que hacer uso de ellas!

Normalmente, las diferentes secciones y subrutinas del programa principal, se identifican mediante «etiquetas». En el programa de la figura 1, las etiquetas se presentan inscritas dentro de un rectángulo (EJ: CLEAR 1, FIRST, PLUS, etcétera) colocado a la derecha del listado. Cuando se escribe un programa es corriente (y muy útil) identificar sus distintas secciones con etiquetas en vez de utilizar su dirección, ya que como se dijo anteriormente, en esta etapa de elaboración del programa no se conocen las direcciones definitivas de las bifurcaciones y de los datos, ya que en la mayoría de los casos, cuando se escribe una instrucción de «salto» (bifurcación), se está pidiendo a la «máquina» que realice un salto a una sección del programa que todavía no se ha escrito, por tanto, no podemos saber con seguridad la dirección del paso de programa hacia el que saltamos, de aquí el poner una «etiqueta» en vez de la dirección final. Posteriormente, una vez que se conozca el número total de posiciones de memoria del programa completo (y a la vez las direcciones de cada instrucción) se podrán sustituir las etiquetas por la dirección definitiva. No sólo es importante que el programa ocupe un número fijo de pasos (o direcciones) consecutivos de memoria, sino que además, sus distintas secciones deben figurar una a continuación de otra sin «solaparse». El área reservada al programa monitor es inviolable, en ningún caso se utilizará esta sección de memoria para nuestros programas. Como regla general, los subprogramas se situarán a continuación de la última dirección del programa principal, como es el caso de la figura 1. Pero no siempre es así; es perfectamente posible dejar libres un cierto número de direcciones entre las subrutinas y el programa principal, sobre todo en programas cortos. Pero cuando se desarrollan programas largos es aconsejable economizar (en lo posible) posiciones de memoria.

¿Cuántos pasos de memoria están a disposición del usuario?

4. En la versión standard del Junior Computer se tiene acceso directo a 1 K de RAM (0000 a 03FF). En total son cuatro páginas de 256 BYTES cada una.

página 00: 0000 a 00FF
página 01: 0100 a 01FF
página 02: 0200 a 02FF
página 03: 0300 a 03FF

key				dirección	dato	comentarios
RST				xxxx	xx	
AD				xxxx	xx	
0	2	0	0	0200	xx	
DA		2	0	0200	20	JSR- CLEAR1
+		9	4	0201	94	ADL de CLDISP
+		0	2	0202	02	ADH de CLDISP
+		2	0	0203	20	JSR-
+		8	2	0204	82	ADL de CLB1
+		0	2	0205	02	ADH de CLB1
+		2	0	0206	20	JSR-
+		8	B	0207	8B	ADL de CLB2
+		0	2	0208	02	ADH de CLB2
+		2	0	0209	20	JSR- FIRST
+		6	F	020A	6F	ADL de KEYDIS
+		0	2	020B	02	ADH de KEYDIS
+		C	9	020C	C9	CMP #
+		1	0	020D	10	con 10
+		F	0	020E	F0	BEQ
+		F	0	020F	F0	ir a CLEAR1
+		C	9	0210	C9	CMP *
+		1	2	0211	12	con 12
+		F	0	0212	F0	BEQ
+		0	6	0213	06	ir a PLUS
+		2	0	0214	20	JSR-
+		4	9	0215	49	ADL de SHIFT
+		0	2	0216	02	ADH de SHIFT
+		4	C	0217	4C	JMP-
+		0	9	0218	09	ADL de FIRST
+		0	2	0219	02	ADH de FIRST
+		2	0	021A	20	JSR- PLUS
+		A	A	021B	AA	ADL de STO1
+		0	2	021C	02	ADH de STO1
+		2	0	021D	20	JSR-
+		9	4	021E	94	ADL de CLDISP
+		0	2	021F	02	ADH de CLDISP
+		2	0	0220	20	JSR- SECOND
+		6	F	0221	6F	ADL de KEYDIS
+		0	2	0222	02	ADH de KEYDIS
+		C	9	0223	C9	CMP #
+		1	0	0224	10	con 10
+		F	0	0225	F0	BEQ
+		0	A	0226	0A	ir a CLEAR2
+		C	9	0227	C9	CMP #
+		1	1	0228	11	con 11
+		F	0	0229	F0	BEQ
+		0	C	022A	0C	ir a EQUAL
+		2	0	022B	20	JSR-
+		4	9	022C	49	ADL de SHIFT
+		0	2	022D	02	ADH de SHIFT
+		4	C	022E	4C	JMP-
+		2	0	022F	20	ADL de SECOND
+		0	2	0230	02	ADH de SECOND

```
+   2  0  0231  20  JSR-                          CLEAR2
+   9  4  0232  94  ADI de CLDISP
+   0  2  0233  02  ADH de CLDISP
+   4  C  0234  4C  JMP-
+   2  0  0235  20  ADL de SECOND
+   0  2  0236  02  ADH de SECOND
+   2  0  0237  20  JSR-                          EQUAL
+   9  D  0238  9D  ADL de STO2
+   0  2  0239  02  ADH de STO2
+   2  0  023A  20  JSR-
+   5  9  023B  59  ADL de ADD
+   0  2  023C  02  ADH de ADD
+   2  0  023D  20  JSR-
+   B  7  023E  B7  ADL de RESDIS
+   0  2  023F  02  ADH de RESDIS
+   2  0  0240  20  JSR-
+   8  2  0241  82  ADL de CLB1
+   0  2  0242  02  ADH de CLB1
+   2  0  0243  20  JSR-
+   8  B  0244  8B  ADL de CLB2
+   0  2  0245  02  ADH de CLB2
+   4  C  0246  4C  JMP-
+   0  9  0247  09  ADL de FIRST
+   0  2  0248  02  ADH de FIRST
Fin del programa principal
+   A  0  0249  A0  LDY  #; subrutina            SHIFT
+   0  4  024A  04  04 en el registro índice Y
+   0  6  024B  06  ASLZ                          SHIFT1
+   F  9  024C  F9  INH (00F9)
+   2  6  024D  26  ROLZ
+   F  A  024E  FA  POINTL (00FA)
+   2  6  024F  26  ROLZ
+   F  B  0250  FB  POINTH (00FB)
+   8  8  0251  88  DEY
+   D  0  0252  D0  BNE
+   F  7  0253  F7  ir a SHIFT1
+   0  5  0254  05  ORAZ
+   F  9  0255  F9  OR IHN
+   8  5  0256  85  STAZ
+   F  9  0257  F9  IHN (00F9)
+   6  0  0258  60  RTS volver al programa principal
+   F  8  0259  F8  SED subrutina aritmética
                                                decimal   ADD
+   1  8  025A  18  CLC
+   A  5  025B  A5  LDAZ
+   0  0  025C  00  ADL de B10 (0000); B10 en el acumulador
+   6  5  025D  65  ADCZ
+   0  3  025E  03  ADL de B20 (0003);
                     acumulador = B10 + B20
+   8  5  025F  85  STAZ
+   0  6  0260  06  ADL de R0; acumulador R0
+   A  5  0261  A5  LDAZ
+   0  1  0262  01  ADL de B11 (0001); B11 en el acumulador
+   6  5  0263  65  ADCZ
```

```
+   0   4   0264  04   ADL de B21 (0004);
                       acumulador = B11 + B21
+   8   5   0265  85   STAZ
+   0   7   0266  07   ADL de R1 (0007);
                       acumulador R1
+   A   5   0267  A5   LDAZ
+   0   2   0268  02   ADL de B12 (0002);
                       B12 en el acumulador
+   6   5   0269  65   ADCZ
+   0   5   026A  05   ADL de B22 (0005);
                       acumulador = B12 + B22
+   8   5   026B  85   STAZ
+   0   8   026C  08   ADL de R2 (0008);
                       acumulador R2
+   D   8   026D  D8   CLD vuelve a binario
+   6   0   026E  60   RTS retrocede al programa principal
+   2   0   026F  20   JSR-subrutina  KEYDIS
+   8   E   0270  8E   ADL de SCANDS  } monitor
+   1   D   0271  1D   ADH de SCANDS  } (1D8E)
+   D   0   0272  D0   BNE
+   F   B   0273  FB   ir a
                       KEYDIS
+   2   0   0274  20   JSR-  KD
+   8   E   0275  8E   ADL de SCANDS  } monitor
+   1   D   0276  1D   ADH de SCANDS  } (1D8E)
+   F   0   0277  F0   BEQ
+   F   B   0278  FB   ir a KD
+   2   0   0279  20   JSR-
+   8   E   027A  8E   ADL de SCANDS  } monitor
+   1   D   027B  1D   ADH de SCANDS  } (1D8E)
+   F   0   027C  F0   BEQ
+   F   6   027D  F6   ir a KD
+   2   0   027E  20   JSR-
+   F   9   027F  F9   ADL de GETKEY  } monitor
+   1   D   0280  1D   ADH de GETKEY  } (1DF9)
+   6   0   0281  60   RTS retrocede al programa principal
+   A   9   0282  A9   LDA #; subrutina  CLB1
+   0   0   0283  00   00 acumulador
+   8   5   0284  85   STAZ
+   0   0   0285  00   acumulador B10 ( = 00)
+   8   5   0286  85   STAZ
+   0   1   0287  01   00 B11
+   8   5   0288  85   STAZ
+   0   2   0289  02   00 B12
+   6   0   028A  60   RTS retrocede al programa principal
+   A   9   028B  A9   LDA #; subrutina  CLB2
+   0   0   028C  00   00 acumulador
+   8   5   028D  85   STAZ
+   0   3   028E  03   00 B20
+   8   5   028F  85   STAZ
+   0   4   0290  04   00 B21
+   8   5   0291  85   STAZ
+   0   5   0292  05   00 B22
+   6   0   0293  60   RTS retrocede al programa principal
+   A   9   0294  A9   LDA #; subrutina  CLDISP
```

```
+  0  0  0295  00  00 → acumulador
+  8  5  0296  85  STAZ
+  F  9  0297  F9  00 → INH (00F9)
+  8  5  0298  85  STAZ
+  F  A  0299  FA  00 → POINTL (00FA)
+  8  5  029A  85  STAZ
+  F  B  029B  FB  00 → POINTH (00FB)
+  6  0  029C  60  RTS retrocede al programa principal
+  A  5  029D  A5  LDAZ; subrutina STO2
+  F  9  029E  F9  INH (00F9) → acumulador
+  8  5  029F  85  STAZ
+  0  3  02A0  03  acumulador (INH) → B20 (0003)
+  A  5  02A1  A5  LDAZ
+  F  A  02A2  FA  POINTL (00FA) → acumulador
+  8  5  02A3  85  STAZ
+  0  4  02A4  04  acumulador (POINTL) → B21 (0004)
+  A  5  02A5  A5  LDAZ
+  F  B  02A6  FB  POINTH (00FB) → acumulador
+  8  5  02A7  85  STAZ
+  0  5  02A8  05  acumulador (POINTH) → B22 (0005)
+  6  0  02A9  60  RTS retrocede al programa principal
+  A  5  02AA  A5  LDAZ; subrutina STO1
+  F  9  02AB  F9  INH (00F9) → acumulador
+  8  5  02AC  85  STAZ
+  0  0  02AD  00  acumulador (INH) → B10 (0000)
+  A  5  02AE  A5  LDAZ
+  F  A  02AF  FA  POINTL (00FA) → acumulador
+  8  5  02B0  85  STAZ
+  0  1  02B1  01  acumulador (POINTL) → B11 (0001)
+  A  5  02B2  A5  LDAZ
+  F  B  02B3  FB  POINTH (00FB) → acumulador
+  8  5  02B4  85  STAZ
+  0  2  02B5  02  acumulador (POINTH) → B12 (0002)
+  6  0  02B6  60  RTS retrocede al programa principal
+  A  5  02B7  A5  LDAZ; subrutina RESDIS
+  0  6  02B8  06  R0 (0006) → acumulador
+  8  5  02B9  85  STAZ
+  F  9  02BA  F9  acumulador (R0) → INH (00F9)
+  A  5  02BB  A5  LDAZ
+  0  7  02BC  07  R1 (0007) → acumulador
+  8  5  02BD  85  STAZ
+  F  A  02BE  FA  acumulador (R1) → POINTL (00FA)
+  A  5  02BF  A5  LDAZ
+  0  8  02C0  08  R2 (0008) → acumulador
+  8  5  02C1  85  STAZ
+  F  B  02C2  FB  acumulador (R2) → POINTH (00FB)
+  6  0  02C3  60  RTS retrocede al programa principal
```

Figura 1. Listado completo del programa de adición decimal. Requiere 196 posiciones de memoria y 594 pulsaciones de tecla.

Un cierto número de posiciones de memoria, situadas en la página cero están reservadas para la memorización de los datos utilizados por algunas rutinas del programa monitor. Concretamente son 31: de la dirección ØØE1 a ØØFF. En particular, algunas de ellas (POINTH = ØØFB; POINTL = ØØFA INH = ØØF9) se utilizan como buffers del visualizador.
Las direcciones ØØEF a ØØF5 están reservadas para los contenidos de todos los registros internos del microprocesador, mediante la subrutina SAVE. El resto del programa monitor se explicará detalladamente en el segundo libro.
La totalidad de la página Ø1 se utiliza como stack (posiciones de memoria para el almacenamiento temporal de datos) del microprocesador. Como es difícil que un programa requiera las 128 posiciones de memoria del stack, en algunos casos se podrá utilizar la zona libre de esta sección de memoria, para albergar el programa principal y/o subrutinas. Por razones obvias, en el caso de que el microprocesador vaya a utilizar algunas posiciones de la memoria STACK, es imperativo que ningún programa almacenado en la página Ø1, «entre» en esta zona de memoria.
Existen ciertas áreas de memoria que no pueden ser programadas por el usuario.

5. El programa monitor está contenido en 1 k de EPROM y utiliza las direcciones 1CØØ a 1FFF. Por tanto, no se podrán memorizar datos en las páginas 1C, 1D, 1E, 1F. En cambio, las subrutinas del programa monitor contenidas en estas direcciones, si podrán utilizarse como subprogramas del programa principal.
Una aplicación de esto último puede verse en la figura 1. La subrutina KEYDIS (Ø26F...Ø281) utiliza las subrutinas del programa monitor SCANDS y GETKEY.

6. Los 128 BYTs de RAM (1/8 K) contenidos en la PIA, pueden usarse para programar. Este área de RAM está situada en las direcciones 1AØØ...1A7F. Así el usuario dispone de media página para programar (concretamente la primera mitad de la 1A).
Al igual que en la página cero, en ésta también hay excepciones; las cuatro posiciones de memoria 1A7A...1A7F están reservadas para los vectores NMI y IRQ (véase capítulo tercero). Estas direcciones sólo pueden ser utilizadas como posiciones normales de memoria, siempre que el programa utilice rutinas de interrupción.
Algunas posiciones de memoria en la segunda mitad de la página 1A (1A8Ø...1AFB) se utilizan para el funcionamiento interno de la PIA. En el segundo libro se ampliarán éste y otros temas más detalladamente, sin embargo, y como anticipo diremos que (con algunas restricciones) pueden utilizarse para el programa normal.

Desplazamientos

Ya se ha hablado anteriormente en el capítulo 4, de cómo utilizar el JC para calcular desplazamientos de datos mediante la subrutina BRANCH del programa monitor (esta subrutina comienza en la dirección 1FD5). Para ello sólo es necesario introducir el byte de orden inferior de la dirección donde se encuentra la instrucción de transferencia (o bifurcación), seguido por el byte de orden inferior de la dirección a la que «salta» (transfiere) el programa.

Cualquier cálculo de desplazamientos* puede realizarse de la siguiente forma:

```
RST   AD          xxxx  xx
1     F   D   5   1FD5  D8
GO                0000  00
Ø     E   Ø   Ø   ØEØØ  FØ  FØ en la posición ØØØF
1     2   1   A   121A  Ø6  Ø6 en la posición Ø213
2     5   3   1   2531  ØA  ØA en la posición Ø226
2     9   3   7   2937  ØC  ØC en la posición Ø22A
5     2   4   B   524B  F7  F7 en la posición Ø253
7     2   6   F   726F  FB  FB en la posición Ø273
7     7   7   4   7774  FB  FB en la posición Ø278
7     C   7   4   7C74  F6  F6 en la posición Ø27D
RST
```

Como mencionamos en el capítulo cuarto, el programa monitor no se utilizará para el cálculo de desplazamiento. Recuérdese que el cálculo de desplazamientos complicados se realiza mediante el byte de orden inferior inmediatamente seguido por la instrucción de salto completa.

Inicialización

También se ha tocado brevemente este tema en el capítulo cuarto. El procedimiento de inicialización para el programa de la figura 1 es el siguiente:

```
RST       AD      xxxx  xx
1     A   7   A   1A7A  xx
DA        Ø   Ø   1A7A  ØØ
+         1   C   1A7B  1C
+                 1A7C  xx
+                 1A7D  xx
+         Ø   Ø   1A7E  ØØ
+         1   C   1A7F  1C
```

Las posiciones de memoria 1A7E y 1A7F se han cargado con ØØ y 1C respectivamente, indicando la dirección efectiva (1CØØ) dé comienzo el programa monitor, de manera que, si se recibe un orden de interrupción (IRQ) o una instrucción BRK (código operativo ØØ) el microprocesador retrocederá directamente al programa monitor. El mismo efecto producen los datos cargados en las posiciones 1A7A y 1A7B, es decir, si el microprocesador encuentra uns instrucción NMI (interrupción no evitable) o se pulsa la tecla ST, éste pasará el control al programa monitor. Naturalmente, si en un programa determinado no existen instrucciones BRK y no se prevee la aparición de interrupciones, no será necesario cargar estas posiciones de memoria.
Una vez completada la inicialización (suponiendo que el programa se ha cargado correctamente) podremos realizar algunos cálculos:

```
AD  Ø   2   Ø   Ø      (entrada de la dirección de comienzo)
GO                     (comienzo del programa)
2   4   5   6         ØØ2456
+                      ØØØØØØ
4   1   3   2         ØØ4132
```

Nota: Si pulsamos cualquier tecla de control mientras se realiza el cálculo de un desplazamiento, el visualizador indicará ØØØØØØ.

```
DA  (=  =)                  ØØ6588
AD  (=  CLEAR)              ØØØØØØ
 1   9   8   5   3   1      198531
 +                          ØØØØØØ
 8   3   2   7   Ø   2      832702
DA                          Ø31233
AD                          ØØØØØØ
```

En el último ejemplo el resultado es mayor que 999999 (sobrecarga), por tanto, sólo se visualizarán los seis dígitos menos significativos.

«Dado electrónico» con el Junior Computer

Desde luego el sistema más barato de jugar a los dados es utilizar el tradicional dado de madera (ahora de plástico) o bien la versión electrónica de los mismos (ya no tan baratos) que tan populares son últimamente. Sin embargo nada nos impide que utilicemos un sistema sofisticado para poder jugar al «parchis», como por ejemplo el Junior Computer. Véamos cómo.

El dado se hace «rodar» pulsando la tecla « + » y se detiene tan pronto como se libera la tecla. El visualizador presentará entonces el número de casillas que usted puede mover. El programa cuenta de 1 a 7, pero sólo presentará números del 1 al 6. Los dos visualizadores centrales indican el número que sale al «tirar» el dado, y los cuatro restantes presentan continuamente los dígitos FFFF.

En la figura 2 se muestra un sencillo organigrama de este programa. En este programa se ha utilizado las subrutinas del programa monitor SCANDS, TK, y GETKEY. El programa comianza dando entrada a la cifra FFFFFF en el visualizador. La siguiente sección (SCAN 1) registra si se ha pulsado alguna tecla, en particular la tecla « + ». En cuyo caso, el programa «saltará» la subrutina COUNT, donde se genera el número aleatorio que muestra el visualizador. Esta sección del programa se repite en tanto no se libere la tecla « + ».

En el organigrama de la figura 3, puede verse con detalle la subrutina COUNT. Primeramente se carga POINTL con ØØ (reposición del contador) y se incrementa (de uno en uno) hasta alcanzar el valor de Ø7 (COUNT 1). Tan pronto como esto sucede, el contador se pone nuevamente a cero, repitiéndose el proceso hasta que dejemos de pulsar la tecla « + ». Esta operación se produce con tal rapidez, que resulta imposible conocer el número que visualizará la pantalla, lo cual proporciona la necesaria aleatoriedad del juego real.

El listado completo del programa se muestra en la figura 4. La dirección de comienzo es Ø2ØØ. Los comentarios que figuran a la derecha del listado tienen como fin aclarar los diferentes pasos del programa. Una vez cargada la dirección de comienzo (AD Ø2ØØ GO), el dado está listo para ser «tirado».

Determinación de la longitud de instrucciones mediante programa

Como ya sabemos, las instrucciones pueden estar formadas por uno, dos, o tres bytes. El primero de ellos indica el código de la operación, el segundo y el tercero (si existen) son la información, es decir, datos, direcciones, etcéte-

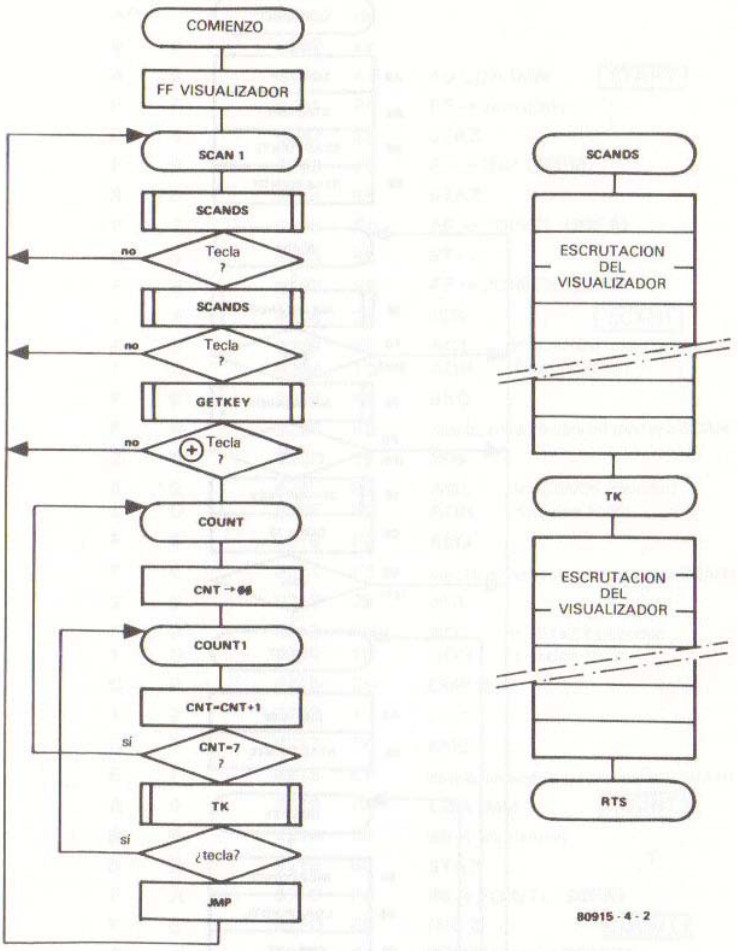


Figura 2. Organigrama simple del «dado electrónico». Para hacer más sencillo el programa se han utilizado subrutinas del programa monitor.

ra. El código de operación (o código operativo), se compone de dos dígitos hexadecimales. La tabla que figura en las páginas finales del libro contiene el cuadro completo de instrucciones del microprocesador 6502. En la tabla que figura en primer lugar, se encuentra las 256 combinaciones de dos números hexadecimales; como es lógico, algunos de ellos no pertenecen a ningún código hexadecimal, a la derecha de cada dígito figura la abreviatura de su función. Esta tabla (en forma condensada) se muestra también en la figura 5. Los bits más significativos figuran en la parte izquierda de la tabla (cuatro bits de cada byte) y los restantes (menos significativos) en la parte superior. La

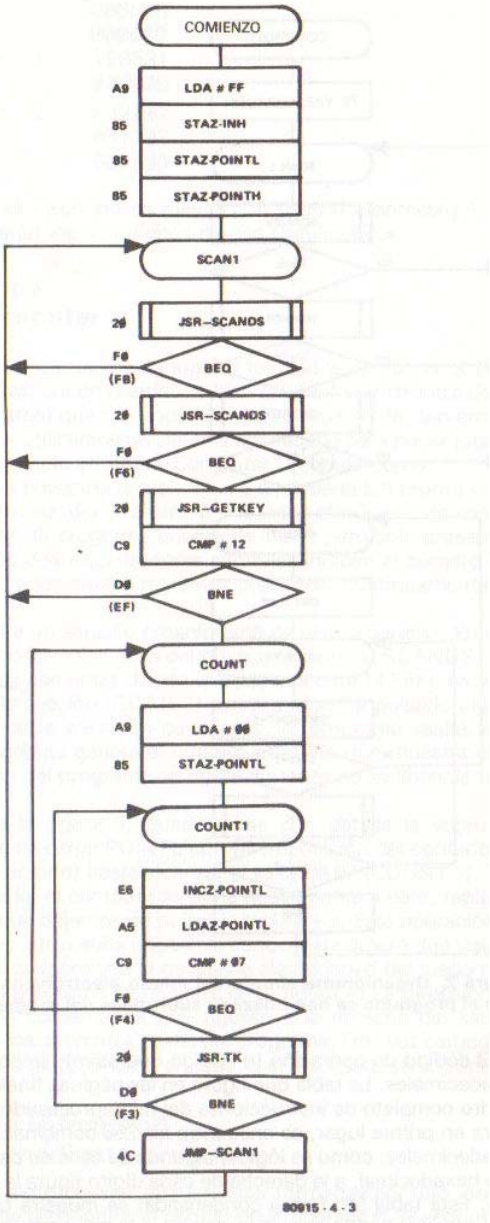


Figura 3. Organigrama detallado del programa del «dado electrónico».

				dirección	dato	comentario
RST		AD		xxxx	xx	
0	2	0	0	0200	xx	
DA		A	9	0200	A9	A9 LDA IMM **START**
+		F	F	0201	FF	FF → acumulador
+		8	5	0202	85	STAZ
+		F	9	0203	F9	FF → INH (00F9)
+		8	5	0204	85	STAZ
+		F	A	0205	FA	FF → POINTL (00FA)
+		8	5	0206	85	STAZ
+		F	B	0207	FB	FF → POINTH (00FB)
+		2	0	0208	20	JSR- **SCAN1**
+		8	E	0209	8E	ADL de SCANDS (monitor)
+		1	D	020A	1D	ADH (dirección 1D8E)
+		F	0	020B	F0	BEQ
+		F	B	020C	FB	desplazamiento para bifurcarse a SCAN1
+		2	0	020D	20	JSR-
+		8	E	020E	8E	ADL de SCANDS (monitor)
+		1	D	020F	1D	ADH (dirección 1D8E)
+		F	0	0210	F0	BEQ
+		F	6	0211	F6	desplazamiento para bifurcarse a SCAN1
+		2	0	0212	20	JSR-
+		F	9	0213	F9	ADL de GETKEY (monitor)
+		1	D	0214	1D	ADH (dirección 1DF9)
+		C	9	0215	C9	CMP IMM
+		1	2	0216	12	con 12
+		D	0	0217	D0	BNE
+		E	F	0218	EF	desplazamiento para bifurcarse a SCAN1
+		A	9	0219	A9	LDA IMM **COUNT**
+		0	0	021A	00	00 → acumulador
+		8	5	021B	85	STAZ
+		F	A	021C	FA	00 → POINTL (00FA)
+		E	6	021D	E6	INC Z **COUNT1**
+		F	A	021E	FA	POINTL + 1 → POINTL
+		A	5	021F	A5	LDA Z
+		F	A	0220	FA	POINTL → acumulador
+		C	9	0221	C9	CMP IMM
+		0	7	0222	07	con 07
+		F	0	0223	F0	BEQ
+		F	4	0224	F4	desplazamiento para bifurcarse a COUNT
+		2	0	0225	20	JSR-
+		B	1	0226	B1	ADL } de TK (monitor)
+		1	D	0227	1D	ADH } (dirección 1DB1)

```
                D    Ø        Ø228   DØ    BNE
                F    3        Ø229   F3    desplazamiento para bifurcación COUNT1
                4    C        Ø22A   4C    JMP-
                Ø    8        Ø22B   Ø8    ADL  } de SCAN 1
                Ø    2        Ø22C   Ø2    ADH  }

          2     Ø    Ø        ØØØØ   A9    dirección de comienzo
                                              ejecución del programa
                               FFØ4   FF    ¡el dado está en el aire!
                               FFØ1   FF
                               FFØ6   FF
                               FFØ2   FF
```

Figura 4. Listado completo del programa del «dado electrónico».

tabla se compone de: 29 instrucciones de un byte, 74 de dos bytes, 48 de tres bytes y los espacios vacíos.

Desarrollemos un programa para determinar si los bytes de un dato particular son

— El código operativo de una instrucción de un byte.
— El cógido operativo de una instrucción de dos bytes.
— El código operativo de una instrucción de tres bytes.
— No existen instrucciones para ese código.

Este programa es preferentemente ilustrativo. Prácticamente todos los ensambladores y editores poseen una subrutina que es muy similar a este programa (en el segundo libro se darán más detalles sobre este tema).

El programa para «medir instrucciones» utiliza una nueva subrutina llamada LENAC, cuyo organigrama se muestra en la figura 6, y que en realidad es el «corazón» del programa. Al entrar en funcionamiento esta subrutina, el acumulador contiene el byte de la instrucción sobre la que se va a trabajar. Este

Figura 5. Esta tabla es una versión abreviada de la tabla que se da al final del libro (pág. 146). La información contenida en las columnas (4 bits menos significativos) es de gran importancia en el programa que determina la longitud de las diversas instrucciones.

4 bits menos significativos

4 bits más significativos	0	1	2	3	4	5	6	7
0	BRK (1)	OPA (IND,X) (2)				ORA Z (2)	ASL Z (2)	
1	BPL (2)	ORA (IND),Y (2)				ORA Z,X (2)	ASL Z,X (2)	
2	JSR (3)	AND (IND,X) (2)			BIT Z (2)	AND Z (2)	ROL Z (2)	
3	BMI (2)	AND (IND),Y (2)				AND Z,X (2)	ROL Z,X (2)	
4	RTI (1)	EOR (IND,X) (2)				EOR Z (2)	LSR Z (2)	
5	BVC (2)	EOR (IND),Y (2)				EOR Z,X (2)	LSR Z,X (2)	
6	RTS (1)	ADC (IND,X) (2)				ADC Z (2)	ROR Z (2)	
7	BVS (2)	ADC (IND),Y (2)				ADC Z,X (2)	ROR Z,X (2)	
8		STA (IND,X) (2)			STY Z (2)	STA Z (2)	STX Z (2)	
9	BCC (2)	STA (IND),Y (2)			STY Z,X (2)	STA Z,X (2)	STX Z,Y (2)	
A	LDY # (2)	LDA (IND,X) (2)	LDX # (2)		LDY Z (2)	LDA Z (2)	LDX Z (2)	
B	BCS (2)	LDA (IND),Y (2)			LDY Z,X (2)	LDA Z,X (2)	LDX Z,Y (2)	
C	CPY # (2)	CMP (IND,X) (2)			CPY Z (2)	CMP Z (2)	DEC Z (2)	
D	BNE (2)	CMP (IND),Y (2)				CMP Z,X (2)	DEC Z,X (2)	
E	CPX # (2)	SBC (IND,X) (2)			CPX Z (2)	SBC Z (2)	INC Z (2)	
F	BEQ (2)	SBC (IND),Y (2)				SBC Z,X (2)	INC Z,X (2)	

subprograma termina cuando la extensión de la instrucción (en bytes) se almacena en la posición de memoria denominada «BYTES».
En el transcurso de la subrutina, el registro índice X, contiene la información concerniente a la longitud de la instrucción: X = Ø1 instrucción de un solo byte; X = Ø2 instrucción de dos bytes; X = Ø3 instrucción de tres bytes. En la última parte del programa, la subrutina LENEND se emplea simplemente para almacenar el contenido del registro índice X en la posición de memoria BYTES anteriormente citada. El registro índice Y se carga con los cuatro bits menos significativos del byte que deseamos examinar, para posteriormente determinar la columna de origen de la instrucción en la tabla de la figura 5. La función del registro índice Y es la de cargar el dato que figura en la parte izquierda de la tabla LENTBL en registro X. Esta tabla (LENTBL) contiene la longitud de las instrucciones (ordenadas por columnas), que aparecen en la figura 5.
Si observamos detenidamente la distribución de la tabla (figura 5), veremos que en las columnas 1, 3, 5, 6, 7, 8, B, D y F, la longitud de las instrucciones es la misma y no hay problema en utilizar la subrutina LENTBL, ¿pero que sucede con las siete columnas restantes? Aquí las instrucciones no tienen la misma longitud, y por tanto, la subrutina LENTBL no es válida, lo cual significa que habrá de perfeccionarse el programa —¡más trabajo!—.

Columnas con instrucciones
de diferente longitud

Si en la tabla de la figura 5, ignoramos los espacios vacíos, únicamente quedarán instrucciones de: dos bytes en las columnas 2 y 4, de un byte en la columna A, y de tres bytes en las columnas C y E. Estos espacios vacíos se pueden eliminar muy fácilmente. Véamos cómo.
Las únicas columnas «problemáticas» que restan son la cero y la nueve. La columna cero se compone principalmente de instrucciones de dos bytes, excepto: BRK, RTI y RTS (1 byte), y JSR (3 bytes). Por otra parte, la columna 9 contiene una mezcla de instrucciones de dos y tres bytes.
Para identificar las instrucciones de la columna cero (en el inicio de la subrutina LENACC), se carga el registro X con Ø1. Esto permite identificar (o filtrar) las instrucciones de un solo byte. La forma de hacerlo, es utilizando tres instrucciones consecutivas de comparación y bifurcación. Si el byte de entrada (en comprobación) corresponde a una de las instrucciones BRK, RTI o RST,

4 bits menos significativos

4 bits más significativos	8	9	A	B	C	D	E	F	
Ø	PHP (1)	ORA # (2)	ASL A (1)			ORA ABS (3)	ASL ABS (3)		Ø
1	CLC (1)	ORA ABS,Y (3)				ORA ABS,X (3)	ASL ABS,X (3)		1
2	PLP (1)	AND # (2)	ROL A (1)		BIT ABS (3)	AND ABS (3)	ROL ABS (3)		2
3	SEC (1)	AND ABS,Y (3)				AND ABS,X (3)	ROL ABS,X (3)		3
4	PHA (1)	EOR # (2)	LSR A (1)		JMP ABS (3)	EOR ABS (3)	LSR ABS (3)		4
5	CLI (1)	EOR ABS,Y (3)				EOR ABS,X (3)	LSR ABS,X (3)		5
6	PLA (1)	ADC # (2)	ROR A (1)		JMP IND (3)	ADC ABS (3)	ROR ABS (3)		6
7	SEI (1)	ADC ABS,Y (3)				ADC ABS,X (3)	ROR ABS,X (3)		7
8	DEY (1)		TXA (1)		STY ABS (3)	STA ABS (3)	STX ABS (3)		8
9	TYA (1)	STA ABS,Y (3)	TXS (1)			STA ABS,X (3)			9
A	TAY (1)	LDA # (2)	TAX (1)		LDY ABS (3)	LDA ABS (3)	LDX ABS (3)		A
B	CLV (1)	LDA ABS,Y (3)	TSX (1)		LDY ABS,X (3)	LDA ABS,X (3)	LDX ABS,Y (3)		B
C	INY (1)	CMP # (2)	DEX (1)		CPY ABS (3)	CMP ABS (3)	DEC ABS (3)		C
D	CLD (1)	CMP ABS,Y (3)				CMP ABS,X (3)	DEC ABS,X (3)		D
E	INX (1)	SBC # (2)	NOP (1)		CPX ABS (3)	SBC ABS (3)	INC ABS (3)		E
F	SED (1)	SBC ABS,Y (3)				SBC ABS,X (3)	INC ABS,X (3)		F

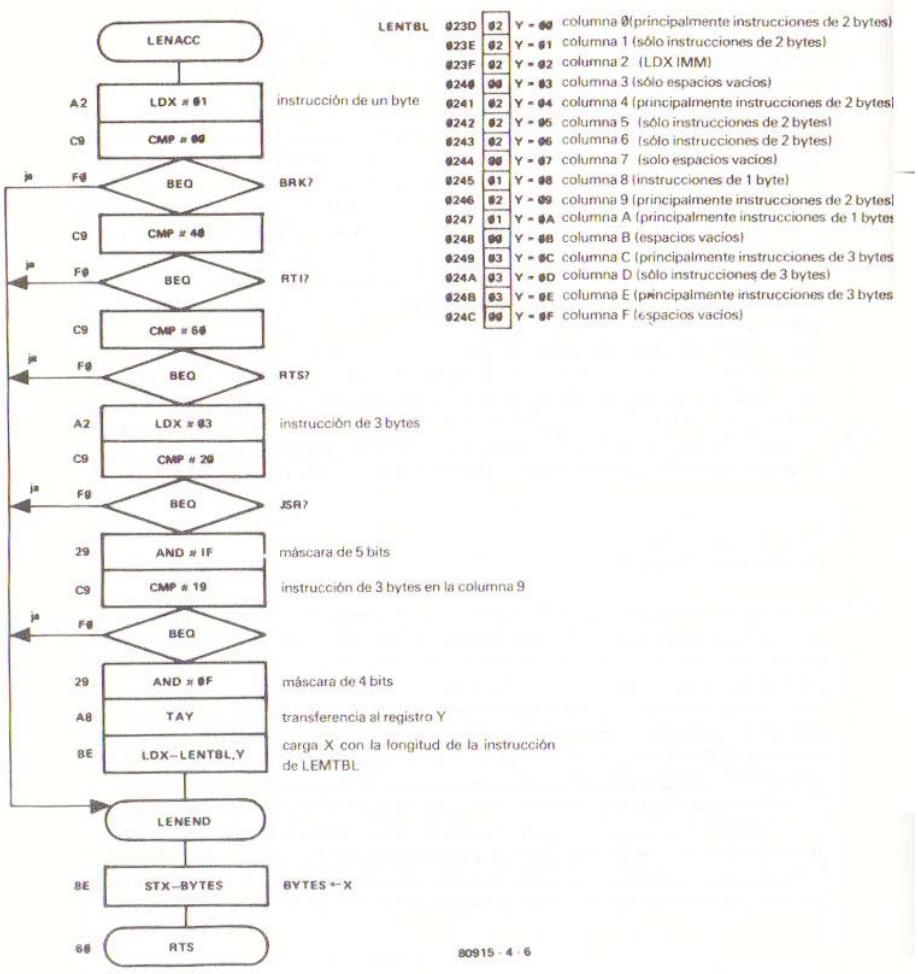


Figura 6. La subrutina LENACC asegura que la longitud de la instrucción se almacena en la posición BYTES.

el programa «saltará» a la subrutina LENEND, memorizando el valor del registro X (Ø1 en este caso) en la posición de memoria denominada BYTES, y contrariamente, si el byte de entrada no corresponde a una de estas instrucciones, el registro índice X se cargará con Ø2, continuándose la comprobación hasta verificar si se trata de la instrucción de bifurcación JSR. Si así fuera la posición BYTES se cargará con Ø3 (instrucción de tres BYTES).
La siguiente sección de la subrutina LENACC identifica las instrucciones de tres bytes en la columna 9. Esto se consigue aplicando la operación lógica

AND al contenido del acumulador y al dato 1F (ØØØ11111 hexadec.) que hace de máscara. La operación AND produce los siguientes resultados:

Ø AND X = Ø ; 1 AND X = X

Al aplicarlo al acumulador y a la máscara 1F resulta:

Contenido del acumulador: XXXXXXXX
Máscara (1F) : ØØØ11111
Resultado en el acumulador: ØØØXXXXX

Como puede verse, después de aplicar la operación AND al contenido del acumulador con la máscara 1F, los cinco bits de la derecha quedan igual (se llama máscara al byte 1F, ya que la disposición de sus bits indica los dígitos del acumulador que serán afectados, en este caso los tres primeros bits de la izquierda, dejando exactamente igual los demás).

Los cuatro bits menos significativos serán necesarios más adelante en el programa, para determinar la columna de procedencia del byte en cuestión. Como muestra la figura 9, las instrucciones de dos y tres bytes están colocadas en los lugares pares e impares, respectivamente.

Aplicando la «máscara» al byte de entrada como en el caso anterior, el quinto bit (empezando a contar por la derecha) será par («Ø») o impar («1»), dependiendo de la fila que se esté examinando. Los cuatro bits (menos significativos) de la derecha en la columna 9, será siempre igual a 9 (hexade.). Por tanto, el valor final del byte completo será: Ø9 ó 19, según sea de una fila par o impar respectivamente. La siguiente etapa del programa es una simple

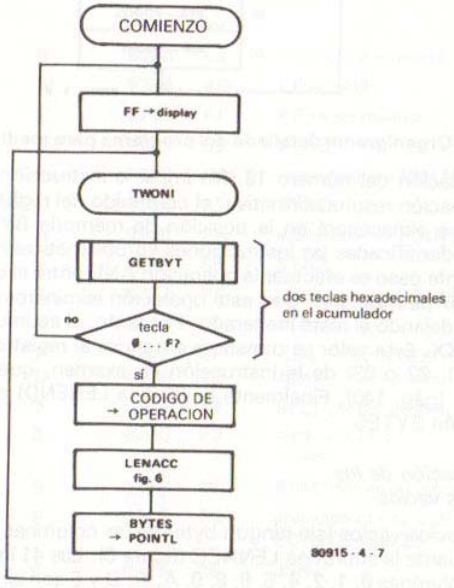


Figura 7. Organigrama simplificado del programa para medir instrucciones.

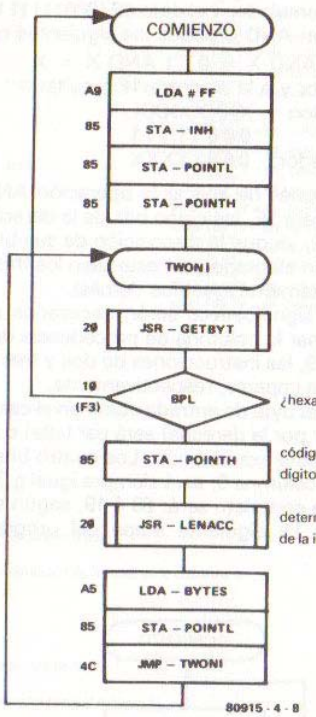


Figura 8. Organigrama detallado del programa para medir instrucciones.

comprobación del número 19 (fila impar o instrucción de tres bytes); si la comprobación resulta afirmativa, el contenido del registro X (Ø3 en este momento) se almacenará en la posición de memoria BYTES. De esta forma quedan identificadas las instrucciones «problemáticas».

El siguiente paso es efectuar la operación AND entre el contenido del acumulador y Ø7 (la máscara). Con esta operación eliminaremos el bit sobrante (el quinto), dejando el resto inalterado. Por tanto, el acumulador contiene ahora ØØØØXXXX. Este valor se transfiere entonces al registro índice X con la longitud (Ø1, Ø2 o Ø3) de la instrucción en examen, que figura en la «tabla» LENTBL (pág. 140). Finalmente (subrutina LENEND) este valor se carga en la posición BYTES.

Identificación de los espacios vacíos

Los espacios vacíos (sin ningún byte) de las columnas 3, 7, B y F, se detectan mediante la subrutina LENACC (figura 6). Las 41 instrucciones restantes de las columnas Ø, 1, 2, 4, 5, 6, 8, 9, A, C, D y E son simplemente ignoradas. la subrutina LENACC funciona de la siguiente forma: el registro índice X se carga con ØØ y a continuación 26 instrucciones consecutivas de comparación

inmediata y bifurcación, identifican si el byte dado pertenece a los espacios vacíos de las columnas Ø, 4, 9, A, C, o E. En caso afirmativo una instrucción de bifurcación lleva directamente al programa a la subrutina LENEND; si no, el registro índice X se cargará en Ø2 para comprobar si el byte pertenece a la única instrucción de la columna 2 (LDX IMM). En este caso el programa «ira» directamente a la instrucción TAY de la figura 6.

El organigrama (simplificado) del programa para medir instrucciones, se muestra en la figura 7, una versión más detallada se da en la 8, y el listado completo en la 9.

El programa comienza con la entrada de los dígitos FF en el visualizador. A continuación, éste realiza un test para comprobar si se ha pulsado alguna tecla, y en su caso determinar su origen (numérica o de control). Este trabajo lo realiza la subrutina GETBYT del programa monitor, que a su vez llama a la subrutina SCANDS (también del programa monitor).

Tan pronto como se hayan pulsado dos teclas hexadecimales (Ø...F) se transferirá el byte resultante a la subrutina POINTH 2 a través del acumulador. Seguidamente entra en funcionamiento la subrutina LENACC, y una vez cumplida su misión (determinar la longitud de la instrucción) se almacena el contenido de la posición BYTES en POINTL. El programa, entonces, retrocede a las subrutinas GETBYT y SCANDS para visualizar el primer resultado y prepararse para la siguiente entrada de datos.

Una vez más, el programa comienza en la posición Ø2ØØ. La rutina principal (figura 8) ocupa desde la posición Ø2ØØ hasta la Ø218 inclusive, y la subrutina

				dirección	dato	comentario
RTS	AD			xxxx	xx	
Ø	2	Ø	Ø	Ø2ØØ	xx	dirección de comienzo
DA		A	9	Ø2ØØ	A9	**LDA IMM**
+		F	F	Ø2Ø1	FF	**FF** → acumulador
+		8	5	Ø2Ø2	85	**STAZ**
+		F	9	Ø2Ø3	F9	acumulador → **INH (ØØF9)**
+		8	5	Ø2Ø4	85	**STAZ**
+		F	A	Ø2Ø5	FA	acumulador → **POINTL (ØØFA)**
+		8	5	Ø2Ø6	85	**STAZ**
+		F	B	Ø2Ø7	FB	acumulador → **POINTH (ØØFB)**
+		2	Ø	Ø2Ø8	2Ø	**JSR-**
+		6	F	Ø2Ø9	6F	**ADL** de GETBYT (dirección
+		1	D	Ø2ØA	1D	**ADH** en el monitor 1D6F)
+		1	Ø	Ø2ØB	1Ø	**BPL;** ¿se han pulsado dos teclas?
+		F	3	Ø2ØC	F3	si no, retrocer a START
+		8	5	Ø2ØD	85	STAZ (byte en el acumulador)
+		F	B	Ø2ØE	FB	acumulador (= op-code) → **POINTH (ØØFB)**
+		2	Ø	Ø2ØF	2Ø	**JSR-**
+		1	9	Ø21Ø	19	**ADL** } de LENACC
+		Ø	2	Ø211	Ø2	**ADH** }

```
+  A  5   0121  A5  LDAZ
+  E  0   0213  E0  BYTES 00E0 → acumulador
+  8  5   0214  85  STAZ
+  F  A   0215  FA  acumulador → POINTL (00FA)
+  4  C   0216  4C  JMP ABS
+  0  8   0217  08  ADL } de TWONI
+  0  2   0218  02  ADH }
+  A  2   0219  A2  LDX IMM;
                    subrutina  [LENACC]
+  0  1   021A  01  01 → X; instrucción de 1 byte
+  C  9   021B  C9  CMP IMM
+  0  0   021C  00  con 00
+  F  0   021D  F0  BEQ BRK?
+  1  A   021E  1A  si es así, ir a LENEND
+  C  9   021F  C9  CMP IMM
+  4  0   0220  40  con 40
+  F  0   0221  F0  BEQ RTI?
+  1  6   0222  16  si es así, ir a LENEND
+  C  9   0223  C9  CMP IMM
+  6  0   0224  60  con 60
+  F  0   0225  F0  BEQ RTS?
+  1  2   0226  12  si es así, ir a LENEND
+  A  2   0227  A2  LDX IMM
+  0  3   0228  03  03 → X; instrucción de 3 bytes
+  C  9   0229  C9  CMP IMM
+  2  0   022A  20  con 20
+  F  0   022B  F0  BEQ JSR?
+  0  C   022C  0C  si es así, ir a LENEND
+  2  9   022D  29  AND IMM
+  1  F   022E  1F  máscara de 5 bits
+  C  9   022F  C9  CMP IMM
+  1  9   0230  19  con 19
+  F  0   0231  F0  BEQ; ¿instrucción de 3 bytes en la columna
+  0  6   0232  06  si es así, ir a LENEND
+  2  9   0233  29  AND IMM
+  0  F   0234  0F  máscara de 4 bits
+  A  8   0235  A8  TAY; cuatro bits menos significativos → Y
+  B  E   0236  BE  LDX ABS,Y; (Y + 1) longitud
                    de la instrucción → X
+  3  D   0237  3D  ADL } de LENTBL
+  0  2   0238  02  ADH } (ver tabla)
+  8  E   0239  8E  STX ABS  [LENEND]
+  E  0   023A  E0  ADL BYTES
```

```
+          Ø    Ø    Ø23B  ØØ   ADH BYTES
+          6    Ø    Ø23C  6Ø   RTS retrocede al programa principal
+          Ø    2    Ø23D  Ø2   columna Ø ; Y = ØØ  LENTBL
+          Ø    2    Ø23E  Ø2   columna 1 ; Y = Ø1
+          Ø    2    Ø23F  Ø2   columna 2 ; Y = Ø2
+          Ø    Ø    Ø24Ø  ØØ   columna 3 ; Y = Ø3
+          Ø    2    Ø241  Ø2   columna 4 ; Y = Ø4
+          Ø    2    Ø242  Ø2   columna 5 ; Y = Ø5
+          Ø    2    Ø243  Ø2   columna 6 ; Y = Ø6
+          Ø    Ø    Ø244  ØØ   columna 7 ; Y = Ø7
+          Ø    1    Ø245  Ø1   columna 8 ; Y = Ø8
+          Ø    2    Ø246  Ø2   columna 9 ; Y = Ø9
+          Ø    1    Ø247  Ø1   columna A; Y = ØA
+          Ø    Ø    Ø248  ØØ   columna B ; Y = ØB
+          Ø    3    Ø249  Ø3   columna C; Y = ØC
+          Ø    3    Ø24A  Ø3   columna D; Y = ØD
+          Ø    3    Ø24B  Ø3   columna E ; Y = ØE
+          Ø    Ø    Ø24C  ØØ   columna F ; Y = ØF
AD
Ø    2    Ø    Ø                dirección de comienzo
GO                              ejecución
          A    9    A9Ø2  FF
          Ø    3    Ø3ØØ  FF
          D    2    D2Ø2  FF
          9    E    9EØ3  FF
          D    5    D5Ø2  FF
          etc.
```

Figura 9. Listado completo del programa para medir instrucciones.

LENACC desde Ø219 hasta Ø23C. Las 16 posiciones siguientes están reservadas para la tabla LENTBL. La posición de memoria BYTES se encuentra en la página cero (ØØF6). Esto es debido a que el programa monitor también utiliza una subrutina parecida para medir instrucciones (OPLEN), que comienza en la dirección 1E5C, e igualmente se sirve de esta posición de memoria para almacenar el resultado.

Trabajo para el lector

Como ya hemos mencionado, no se decodifican (o identifican) todos los espacios vacíos de la figura 5, pero se ha expuesto el método para efectuar la decodificación de los citados espacios. Como ejercicio y prueba de su experiencia en programación, puede realizar las oportunas modificaciones para completar el programa anteriormente descrito. No será necesario que nos envie los resultados, el propio Junior Computer (nadie más imparcial) se encargará de decirle si lo hizo bien o no.

1. Códigos de las instrucciones en orden numérico

Tabla completa del juego de instrucciones y códigos operacionales en orden numérico, ØØ... FF. También se han listado los códigos no utilizados.

ØØ	BRK	2Ø	JSR ABS	4Ø	RTI IMP	6Ø	RTS IMP
Ø1	ORA (IND,X)	21	AND (IND,X)	41	EOR (IND,X)	61	ADC (IND,X)
Ø2	–	22	–	42	–	62	–
Ø3	–	23	–	43	–	63	–
Ø4	–	24	BIT Z	44	–	64	–
Ø5	ORA Z	25	AND Z	45	EOR Z	65	ADC Z
Ø6	ASL Z	26	ROL Z	46	LSR Z	66	ROR Z
Ø7	–	27	–	47	–	67	–
Ø8	PHP IMP	28	PLP IMP	48	PHA IMP	68	PLA IMP
Ø9	ORA IMM	29	AND IMM	49	EOR IMM	69	ADC IMM
ØA	ASL A	2A	ROL A	4A	LSR A	6A	ROR A
ØB	–	2B	–	4B	–	6B	–
ØC	–	2C	BIT ABS	4C	JMP ABS	6C	JMP IND
ØD	ORA ABS	2D	AND ABS	4D	EOR ABS	6D	ADC ABS
ØE	ASL ABS	2E	ROL ABS	4E	LSR ABS	6E	ROR ABS
ØF	–	2F	–	4F	–	6F	–
10	BPL REL	3Ø	BMI REL	5Ø	BVC REL	7Ø	BVS REL
11	ORA (IND), Y	31	AND (IND), Y	51	EOR (IND), Y	71	ADC (IND),Y
12	–	32	–	52	–	72	–
13	–	33	–	53	–	73	–
14	–	34	–	54	–	74	–
15	ORA Z,X	35	AND Z,X	55	EOR Z,X	75	ADC Z,X
16	ASL Z,X	36	ROL Z,X	56	LSR Z,X	76	ROR Z,X
17	–	37	–	57	–	77	–
18	CLC IMP	38	SEC IMP	58	CLI IMP	78	SEI IMP
19	ORA ABS,Y	39	AND ABS,Y	59	EOR ABS,Y	79	ADC ABS,Y
1A	–	3A	–	5A	–	7A	–
1B	–	3B	–	5B	–	7B	–
1C	–	3C	–	5C	–	7C	–
1D	ORA ABS,X	3D	AND ABS,X	5D	EOR ABS,X	7D	ADC ABS,X
1E	ASL ABS,X	3E	ROL ABS,X	5E	LSR ABS,X	7E	ROR ABS,X
1F	–	3F	–	5F	–	7F	–

8Ø	–	AØ	LDY IMM	CØ	CPY IMM	EØ	CPX IMM
81	STA (IND,X)	A1	LDA (IND,X)	C1	CMP (IND,X)	E1	SBC (IND,X))
82	–	A2	LDX IMM	C2	–	E2	–
83	–	A3	–	C3	–	E3	–
84	STY Z	A4	LDY Z	C4	CPY Z	E4	CPX Z
85	STA Z	A5	LDA Z	C5	CMP Z	E5	SBC Z
86	STX Z	A6	LDX Z	C6	DEC Z	E6	INC Z
87	–	A7	–	C7	–	E7	–
88	DEY IMP	A8	TAY IMP	C8	INY IMP	E8	INX IMP
89	–	A9	LDA IMM	C9	CMP IMM	E9	SBC IMM
8A	TXA IMP	AA	TAX IMP	CA	DEX IMP	EA	NOP IMP
8B	–	AB	–	CB	–	EB	–
8C	STY ABS	AC	LDY ABS	CC	CPY ABS	EC	CPX ABS
8D	STA ABS	AD	LDA ABS	CD	CMP ABS	ED	SBC ABS
8E	STX ABS	AE	LDX ABS	CE	DEC ABS	EE	INC ABS
8F	–	AF	–	CF	–	EF	–
9Ø	BCC REL	BØ	BCS REL	DØ	BNE REL	FØ	BEQ REL
91	STA (IND),Y	B1	LDA (IND),Y	D1	CMP (IND),Y	F1	SBC (IND),Y·
92	–	B2	–	D2	–	F2	–
93	–	B3	–	D3	–	F3	–
94	STY Z,X	B4	LDY Z,X	D4	–	F4	–
95	STA Z,X	B5	LDA Z,X	D5	CMP Z,X	F5	SBC Z,X
96	STX Z,Y	B6	LDX Z,Y	D6	DEC Z,X	F6	INC Z,X
97	–	B7	–	D7	–	F7	–
98	TYA IMP	B8	CLV IMP	D8	CLD IMP	F8	SED IMP
99	STA ABS,Y	B9	LDA ABS,Y	D9	CMP ABS,Y	F9	SBC ABS,Y
9A	TXS IMP	BA	TSX IMP	DA	–	FA	–
9B	–	BB	–	DB	–	FB	–
9C	–	BC	LDY ABS,X	DC	–	FC	–
9D	STA ABS,X	BD	LDA ABS,X	DD	CMP ABS,X	FD	SBC ABS,X
9E	–	BE	LDX ABS,Y	DE	DEC ABS,X	FE	INC ABS,X
9F	–	BF	–	DF	–	FF	–

Nota: Las tres primeras letras después del código de operación es la abreviatura nemotécnica de la instrucción. El modo de direccionamiento (si hay), lo indican las letras siguientes:

IMM: direccionamiento inmediato (IMMediato addresing)
ABS: direccionamiento absoluto (ABSolute addresing)
Z: direccionamiento de página cero (Zero page addresing)
A: direccionamiento del acumulador (Accumulator addresing)
(IND, X): direccionamiento preindexado indirecto (pre-INDexed indirect addresing)
(IND, Y): direccionamiento post-indexado indirecto (post-INDexed indirect addresing)
Z,X: direccionamiento indexado de página cero (utiliza el registro X)
Z, Y: direccionamiento indexado de página cero (utiliza el registro Y)
ABS, X: direccionamiento indexado absoluto (utiliza el registro X)
ABS, Y: direccionamiento indexado de página cero (utiliza el registro Y)
IND: direccionamiento indirecto (INDirect addresing)
REL: direccionamiento relativo (RELative addresing)
IMP: direccionamiento implícito (IMPlied addresing)

2. Lista de instrucciones

Las 56 instrucciones del 6502 en orden alfabético. Cada instrucción puede utilizarse con varios modos de direccionamiento, por lo que se dispone en realidad 151 instrucciones.

abreviatura nemotécnica + explicación	modo de direccionamiento (5)	código hexadecimal	número de impulsos de reloj	número de bytes	banderas afectadas
ADC **Add memory to accumulator with carry** A + M + C → A (1) Sumar M y A con acarreo	IMM ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	69 6D 65 61 71 75 7D 79	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	NV - - - - ZC
AND **"AND" memory with accumulator** A ∧ M → A (1) Función AND entre M y A	IMM ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	29 2D 25 21 31 35 3D 39	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N - - - - - Z -
ASL Shift left one bit Desplazar un bit a la izquierda C ← [7 0] ← 0	ABS Z A Z,X ABS,X	0E 06 0A 16 1E	6 5 2 6 7	3 2 1 2 3	N - - - - - ZC
BCC **Branch on carry clear (2)** Ramificación si C = 0	REL	90	2	2	- - - - - - - -
BCS **Branch on carry set (2)** Ramificación si C = 1	REL	B0	2	2	- - - - - - - -
BEQ **Branch on result zero (2)** Ramificación si Z = 1	REL	F0	2	2	- - - - - - - -
BIT Test bits in memory Realizar: A ∧ M $M_7 \rightarrow N; M_6 \rightarrow V$	ABS Z	2C 24	4 3	3 2	M_7M_6 - - - - Z -
BMI **Branch on result minus (2)** Ramificación si N = 1	REL	30	2	2	- - - - - - - -

abreviatura nemotécnica + explicación (4)	modo de direcciona-niento (5)	código hexade-cimal	número de impulsos de reloj	número de bytes	banderas afectadas
BNE **Branch on result not zero (2)** Ramificación si Z = Ø	REL	DØ	2	2	- - - - - - - -
BPL **Branch on result plus (2)** Ramificación si N = Ø	REL	1Ø	2	2	- - - - - - - -
BRK **Force break** Interrupción obligada	IMP	ØØ	7	1	- - - 1 - 1 - - B I
BVC **Branch on overflow clear (2)** Ramificación si V = Ø	REL	5Ø	2	2	- - - - - - - -
BVS **Branch on overflow set (2)** Ramificación si V = 1	REL	7Ø	2	2	- - - - - - - -
CLC **Clear carry flag** Desactivar C; Ø → C	IMP	18	2	1	- - - - - - - Ø C
CLD Clear decimal mode Desactivar D; Ø → D	IMP	D8	2	1	- - - - Ø - - - D
CLI Clear interrupt flag Desactivar I: Ø → 1	IMP	58	2	1	- - - - - Ø - - I
CLV Clear overflow flag Desactivar V; Ø → V	IMP	B8	2	1	Ø - - - - - - - V
CMP **Compare memory and accumulator** **A-M** Comparar M y A	IMM ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	C9 CD C5 C1 D1 D5 DD D9	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N - - - - - ZC
CPX Compare memory and index X; Comparar M y X	IMM ABS Z	EØ EC E4	2 4 3	2 3 2	N - - - - - ZC
CPY Compare memory and index Y Comparar M e Y; M-Y	IMM ABS Z	CØ CC C4	2 4 3	2 3 2	N - - - - - ZC

abreviatura nemotécnica + explicación (4)	modo de direcciona-niento (5)	código hexade-cimal	número de impulsos de reloj	número de bytes	banderas afectadas
DEC **Decrement memory** Restar 1 de M M-1 → M	ABS Z Z,X ABS,X	CE C6 D6 DE	6 5 6 7	3 2 2 3	N - - - - - Z -
DEX Decrement index X by one Restar 1 de X	IMP	CA	2	1	N - - - - - Z -
DEY Decrement index Y by one Y-1 → Y Restar de 1 de Y	IMP	88	2	1	N - - - - - Z -
EOR **"Exclusive or" memory with accumulator** A ∀ M → A (1) Función EXOR entre M y A	IMM ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	49 4D 45 41 51 55 5D 59	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N - - - - - Z -
INC Increment memory by one X + 1 → M Sumar 1 a M	ABS Z Z,X ABS,X	EE E6 F6 FE	6 5 6 7	3 2 2 3	N - - - - - Z -
INX Increment index X by one X + 1 → X Sumar 1 a X	IMP	E8	2	1	N - - - - - Z -
INY Increment index Y by one Y + 1 → Y Sumar 1 a Y	IMP	C8	2	1	N - - - - - Z -
JMP Jump to new loc. Saltar (PC + 1) → PCL (PC + 2) → PCH	ABS IND	4C 6C	3 5	3 3	- - - - - - - -
JSR **Jump to new location saving return address** Saltar con retorno (PC + 1) → PCL (PC + 2) → PCH	ABS	20	6	3	- - - - - - - -

abreviatura nemotécnica + explicación (4)	modo de direcciona-niento (5)	código hexade-cimal	número de impulsos de reloj	número de bytes	banderas afectadas
LDA Load accumulator with memory M → A (1) Cargar A con M	IMM ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	A9 AD A5 A1 B1 B5 BD B9	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N - - - - - Z -
LDX Load index X with memory M → X (1) Cargar X con M	IMM ABS Z Z,Y ABS,Y	A2 AE A6 B6 BE	2 4 3 4 4	2 3 2 2 3	N - - - - - Z -
LDY Load index Y with memory M → Y (1) Cargar Y con M	IMM ABS Z Z,X ABS,X	A0 AC A4 B4 BC	2 4 3 4 4	2 3 2 2 3	N - - - - - Z -
LSR Shift right one bit Desplazar un bit a la izquierda 0 → [7 0] → C	ABS Z A Z,X ABS,X	4E 46 4A 56 5E	6 5 2 6 7	3 2 1 2 3	0 - - - - - ZC N
NOP No operation	IMP	EA	2	1	- - - - - - - -
ORA "OR" memory with accumulator AVM → A Función OR entre M y A	IMM ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	09 0D 05 01 11 15 1D 19	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N - - - - - Z -
PHA Push accumulator on stack A↓ Enviar A al stack	IMP	48	3	1	- - - - - - - -
PHP Push processor status on stack; P↓ Enviar P al stack	IMP	08	3	1	- - - - - - - -
PLA Pull accumulator from stack A↓ Recuperar A del stack	IMP	68	4	1	N - - - - - Z -

abreviatura nemotécnica + explicación (4)	modo de direcciona-niento (5)	código hexade-cimal	número de impulsos de reloj	número de bytes	banderas afectadas
PLP Pull procesor status from stack: P↑ Recuperar P del stack	IMP	28	4	1	(reset)
ROL Rotate one bit left Girar un bit a izquierdas 	ABS Z Z,X ABS,X A	2E 26 36 3E 2A	6 5 6 7 2	3 2 2 3 1	N - - - - - ZC
ROR Rotate one bit right Girar un bit a derechas 	ABS Z A Z,X ABS,X	6E 66 6A 76 7E	6 5 2 6 7	3 2 1 2 3	N - - - - - ZC
RTI Return from interrupt PC↑, P↑ Retorno de interrupción	IMP	40	6	1	(reset)
RTS Return from subrou-tine PC↑; PC + 1 → PC Retorno de subrutina	IMP	60	6	1	- - - - - - - -
SBC Subtract memory from accumulator with borrow (3) A-M-$\overline{C}$ → A Restar M de A «robando»	IMM ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	E9 ED E5 E1 F1 F5 FD F9	2 4 3 6 5 4 4 4	2 3 2 2 2 2 3 3	N - - - - - ZC
SEC Set carry flag 1 → C Activar C	IMP	38	2	1	- - - - - - - 1
SED Set D; activar D	IMP	F8	2	1	- - - - 1 - - - D
SEI Set interrupt; 1 → I Activar I	IMP	78	2	1	- - - - - 1 - - I
STA Store accumu-lator in memory A → M Almacenar A en M	ABS Z (IND,X) (IND),Y Z,X ABS,X ABS,Y	8D 85 81 91 95 9D 99	4 3 6 6 4 5 5	3 2 2 2 2 3 3	- - - - - - - -

abreviatura nemotécnica + explicación (4)	modo de direcciona-niento (5)	código hexade-cimal	número de impulsos de reloj	número de bytes	banderas afectadas
STX (X → M) Storex index X in M Almacenar X en M	ABS Z Z,Y	8E 86 96	4 3 4	3 2 2	- - - - - - - -
STY (Y → M) Store index Y in M Almacenar Y en M	ABS Z Z,X	8C 84 94	4 3 4	3 2 2	- - - - - - - -
TAX (A → X) Transfer accumu-lator to index X Transferir A a X	IMP	AA	2	1	N - - - - - Z -
TAY (A → Y) Transfer accumu-lator to index Y Transferir A a Y	IMP	A8	2	1	N - - - - - Z -
TSX (S → X) Transfer stack pointer to index X Transferir S a X	IMP	BA	2	1	N - - - - - Z -
TXA (X → A) Transfer index X to acomulator Transferir X a A	IMP	8A	2	1	N - - - - - Z -
TXS (X → S) Transfer index X to stack pointer Transferir X a S	IMP	9A	2	1	N - - - - - Z -
TYA (Y → A) Transfer index Y to accumulator Transferir Y a A	IMP	98	2	1	N - - - - - Z -

Notas:
(1) Sumar 1 a N si se excede el límite de página. (2) Sumar 1 a N si el salto se hace a la misma página. Sumar 2 a N, si el salto se hace a otra página. (3). Sin acarreo. (4) A = Acumulador; M = Memoria; C = Bandera de acarreo; Z = bandera de cero, V = bandera de exceso; N = bandera de signo negativo; D = bandera de número decimal; I = bandera de interrupción; X = índice X; Y = índice Y.

(5)

IMM: Direccionamiento inmediato
ABS: direccionamiento absoluto
Z: direccionamiento de página cero
A: direccionamiento del acumulador
IMP: direccionamiento implícito
(IND, X): direccionamiento preindexado
(IND, Y): direccionamiento preindexado
Z,X: direccionamiento indexado de página cero (utiliza registro
Z,Y: direccionamiento indexado de página cero (utiliza registro)
ABS,X: direccionamiento absoluto indexado (utiliza registro)
ABS,Y: direccionamiento absoltuo indexado (utiliza registro)
REL: direccionamiento relativo
IND: direccionamiento indirecto

3. Listado hexadecimal del programa monitor

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
1C00: 85 F3 68 85 F1 68 85 EF 85 FA 68 85 F0 85 FB 84
1C10: F4 86 F5 BA 86 F2 A2 01 86 FF 4C 33 1C A9 1E 8D
1C20: 83 1A A9 04 85 F1 A9 03 85 FF 85 F6 A2 FF 9A 86
1C30: F2 D8 78 20 88 1D D0 FB 20 88 1D F0 FB 20 88 1D
1C40: F0 F6 20 F9 1D C9 13 D0 13 A6 F2 9A A5 FB 48 A5
1C50: FA 48 A5 F1 48 A6 F5 A4 F4 A5 F3 40 C9 10 D0 06
1C60: A9 03 85 FF D0 14 C9 11 D0 06 A9 00 85 FF F0 0A
1C70: C9 12 D0 09 E6 FA D0 02 E6 FB 4C 33 1C C9 14 D0
1C80: 0B A5 EF 85 FA A5 F0 85 FB 4C 7A 1C C9 15 10 EA
1C90: 85 E1 A4 FF D0 0D B1 FA 0A 0A 0A 0A 05 E1 91 FA
1CA0: 4C 7A 1C A2 04 06 FA 26 FB CA D0 F9 A5 FA 05 E1
1CB0: 85 FA 4C 7A 1C 20 D3 1E A4 E3 A6 E2 E8 D0 01 C8
1CC0: 86 E8 84 E9 A9 77 A0 00 91 E6 20 4D 1D C9 14 D0
1CD0: 2A 20 6F 1D 10 F7 85 FB 20 6F 1D 10 F0 85 FA 20
1CE0: D3 1E A0 00 B1 E6 C5 FB D0 07 C8 B1 E6 C5 FA F0
1CF0: D9 20 5C 1E 20 F8 1E 30 E9 10 3E C9 10 D0 0A 20
1D00: 20 1E 10 C9 20 47 1E F0 C1 C9 13 D0 14 20 20 1E
1D10: 10 BB 20 5C 1E 20 F8 1E A5 FD 85 F6 20 47 1E F0
1D20: A9 C9 12 D0 07 20 F8 1E 30 A0 10 0D C9 11 D0 09
1D30: 20 83 1E 20 EA 1E 4C CA 1C A9 EE 85 FB 85 FA 85
1D40: F9 A9 03 85 F6 20 8E 1D D0 FB 4C CA 1C A2 02 A0
1D50: 00 B1 E6 95 F9 C8 CA 10 F8 20 5C 1E 20 8E 1D D0
1D60: FB 20 8E 1D F0 FB 20 8E 1D F0 F6 20 F9 1D 60 20
1D70: 5C 1D C9 10 10 11 0A 0A 0A 0A 85 FE 20 5C 1D C9
1D80: 10 10 04 05 FE A2 FF 60 A0 00 B1 FA 85 F9 A9 7F
1D90: 8D 81 1A A2 08 A4 F6 A5 FB 20 CC 1D 88 F0 0D A5
1DA0: FA 20 CC 1D 88 F0 05 A5 F9 20 CC 1D A9 00 8D 81
1DB0: 1A A0 03 A2 00 A9 FF 8E 82 1A E8 E8 2D 80 1A 88
1DC0: D0 F5 A0 06 8C 82 1A 09 80 49 FF 60 48 84 FC 4A
1DD0: 4A 4A 4A 20 DF 1D 68 29 0F 20 DF 1D A4 FC 60 A8
1DE0: B9 0F 1F 8D 80 1A 8E 82 1A A0 7F 88 10 FD 8C 80
1DF0: 1A A0 06 8C 82 1A E8 E8 60 A2 21 A0 01 20 B5 1D
1E00: D0 07 E0 27 D0 F5 A9 15 60 A0 FF 0A B0 03 C8 10
1E10: FA 8A 29 0F 4A AA 98 10 03 18 69 07 CA D0 FA 60
1E20: 20 6F 1D 10 21 85 FB 20 60 1E 84 F7 84 FD C6 F7
1E30: F0 12 20 6F 1D 10 0F 85 FA C6 F7 F0 07 20 6F 1D
1E40: 10 04 85 F9 A2 FF 60 20 A6 1E 20 DC 1E A2 02 A0
1E50: 00 B5 F9 91 E6 CA C8 C4 F6 D0 F6 60 A0 00 B1 E6
```

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
1E60: A0 01 C9 00 F0 1A C9 40 F0 16 C9 60 F0 12 A0 03
1E70: C9 20 F0 0C 29 1F C9 19 F0 06 29 0F AA BC 1F 1F
1E80: 84 F6 60 A5 E6 85 EA A5 E7 85 EB A4 F6 B1 EA A0
1E90: 00 91 EA E6 EA D0 02 E6 EB A5 EA C5 E8 D0 EC A5
1EA0: EB C5 E9 D0 E6 60 A5 E8 85 EA A5 E9 85 EB A0 00
1EB0: B1 EA A4 F6 91 EA A5 EA C5 E6 D0 06 A5 EB C5 E7
1EC0: F0 10 38 A5 EA E9 01 85 EA A5 EB E9 00 85 EB 4C
1ED0: AE 1E 60 A5 E2 85 E6 A5 E3 85 E7 60 18 A5 E8 65
1EE0: F6 85 E8 A5 E9 69 00 85 E9 60 38 A5 E8 E5 F6 85
1EF0: E8 A5 E9 E9 00 85 E9 60 18 A5 E6 65 F6 85 E6 A5
1F00: E7 69 00 85 E7 38 A5 E6 E5 E8 A5 E7 E5 E9 60 40
1F10: 79 24 30 19 12 02 78 00 10 08 03 46 21 06 0E 02
1F20: 02 02 01 02 02 02 01 01 02 01 01 03 03 03 03 6C
1F30: 7A 1A 6C 7E 1A B1 E6 A0 FF C4 EE F0 0D D1 EC D0
1F40: 0A 88 B1 EC AA 88 B1 EC A0 01 60 88 88 88 D0 E9
1F50: 60 38 A5 E4 E9 FF 85 EC A5 E5 E9 00 85 ED A9 FF
1F60: 85 EE 20 D3 1E 20 5C 1E A0 00 B1 E6 C9 FF D0 1D
1F70: C8 B1 E6 A4 EE 91 EC 88 A5 E7 91 EC 88 A5 E6 91
1F80: EC 88 84 EE 20 83 1E 20 EA 1E 4C 65 1F 20 F8 1E
1F90: 30 D3 20 D3 1E 20 5C 1E A0 00 B1 E6 C9 4C F0 16
1FA0: C9 20 F0 12 29 1F C9 10 F0 1A 20 F8 1E 30 E6 A9
1FB0: 03 85 F6 4C 33 1C C8 20 35 1F F0 EE 91 E6 8A C8
1FC0: 91 E6 D0 E6 C8 20 35 1F F0 E0 38 E5 E6 38 E9 02
1FD0: 91 E6 4C AA 1F D8 A9 00 85 FB 85 FA 85 F9 20 6F
1FE0: 1D 10 F2 85 FB 20 6F 1D 10 EB 85 FA 18 A5 FA E5
1FF0: FB 85 F9 C6 F9 4C DE 1F FF FF 2F 1F 1D 1C 32 1F
```

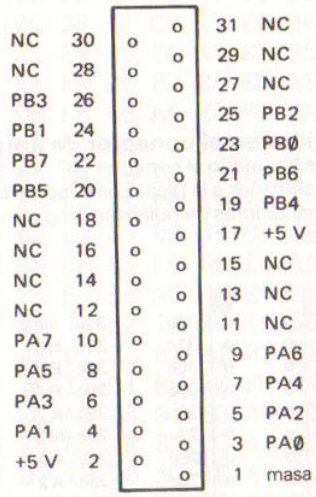
Esta es la lista completa del programa monitor contenido en la EPROM (IC2). Para ser exactos, sólo se ha listado el código de máquina. La primera columna de la tabla son las direcciones y el resto los datos de cada byte. Ejemplo: El dato 85 está en la dirección 1C00 y F3 en la siguiente posición (1C01).

4. Distribución de patillas

Distribución de patillas en el conector de ampliación
Por razones prácticas, se ha rotado el conector 90°. Las primeras conexiones son las más cercanas a la plca. Los cuadros en negro cerca de 1c y 32c, indican las hendiduras de polarización para situar (y conectar) correctamente el conector.

32a	masa	32a	o	o	32c	32c masa
31a	RAM-R/W	31a	o	o	31c	31c NC
30a	Φ1	30a	o	o	30c	30c EX
29a	K1	29a	o	o	29c	29c R/$\overline{W}$
28a	NC	28a	o	o	28c	28c K2
27a	Φ2	27a	o	o	27c	27c NC
26a	A1	26a	o	o	26c	26c AØ
25a	A3	25a	o	o	25c	25c A2
24a	A5	24a	o	o	24c	24c A4
23a	A7	23a	o	o	23c	23c A6
22a	A9	22a	o	o	22c	22c A8
21a	A11	21a	o	o	21c	21c A1Ø
20a	A13	20a	o	o	20c	20c A12
19a	A15	19a	o	o	19c	19c A14
18a	−5 V	18a	o	o	18c	18c K3
17a	K4	17a	o	o	17c	17c +12 V
16a	16c	16a	o	o	16c	16c
15a	K5	15a	o	o	15c	15c K6
14a	K7	14a	o	o	14c	14c S0
13a	NC	13a	o	o	13c	13c NC
12a	IRQ	12a	o	o	12c	12c NMI
11a	NC	11a	o	o	11c	11c NC
10a	D7	10a	o	o	10c	10c D6
9a	D5	9a	o	o	9c	9c D4
8a	D3	8a	o	o	8c	8c D2
7a	D1	7a	o	o	7c	7c DØ
6a	NC	6a	o	o	6c	6c NC
5a	RES	5a	o	o	5c	5c RDY
4a	masa	4a	o	o	4c	4c masa
3a	NC	3a	o	o	3c	3c NC
2a	NC	2a	o	o	2c	2c NC
1a	+5 V	1a	o	o	1c	1c +5 V

Distribución de patillas en el conector de entrada/salida



Para muchas personas la palabra ordenador es sinónimo de menos tiempo de trabajo/más tiempo de ocio. Sin embargo, lo correcto es decir que se realiza más trabajo en el mismo tiempo. En la actualidad el ordenador personal es cada vez más popular, de manera que cada vez es mayor el número de personas que utiliza su tiempo libre para trabajar con los ordenadores.

Ante la gran cantidad de libros y revistas diferentes, así como de microordenadores disponibles, es posible que el neófito se encuentre totalmente perdido. En respuesta a esta necesidad de orientación, Elektor ha diseñado el Junior Computer.

- Se trata de un ordenador monoplaca que puede construise uno mismo.
- Se describe en un lenguaje claro que cualquiera puede comprender, a través de cuatro libros, siendo este el primero.
- De concepción modular, el ordenador puede crecer a medida que aumentan los conocimientos de su propietario.

INGELEK, S. A.
Ginzo de Limia, 48
MADRID-29