ESCUELA POLITÉCNICA SUPERIOR

Ingeniería Técnica en Informática de Gestión

UNIVERSIDAD DE JAÉN
ESCUELA POLITÉCNICA SUPERIOR (JAÉN)

Proyecto Fin de Carrera

BIBLIOTECA DE PROBLEMAS DE PROGRAMACIÓN EN ENSAMBLADOR

Alumno: Francisco Charte Ojeda

Tutor: Prof. Dª. Catalina Rus Casas

Prof. D. Diego López Talavera

Dpto: Ingeniería Electrónica y Automática

JUNIO, 2008

I Introducción

Una parte fundamental de la asignatura *Estructura y Tecnología de Computadores* es la relativa a la programación de sistemas basados en microprocesador mediante lenguaje ensamblador, una actividad que permite a los alumnos apreciar las diferencias existentes entre los denominados lenguajes de alto nivel, como C o Pascal, y un lenguaje simbólico como es el ensamblador.

También es importante desde una perspectiva de metodología, ya que al programar en ensamblador se aprende a dividir el problema a resolver en pasos cada vez más pequeños y simples hasta llegar a la codificación de instrucciones que efectúan operaciones elementales: modificar el contenido de un registro, comprobar el estado de un bit, enviar o leer un byte por un puerto de E/S, etc. Esta experiencia le será útil al alumno en multitud de actividades futuras al crearle un hábito de análisis y división de los problemas aplicable en muchos otros contextos.

Actualmente es posible encontrar sistemas basados en microprocesador no solamente en los ordenadores, que es el tipo de dispositivo más obvio en este sentido, sino también en muchos otros aparatos de uso cotidiano. Los televisores y equipos de vídeo, los hornos microondas, las lavadoras y lavavajillas, los teléfonos móviles o los automóviles son ejemplos de sistemas complejos en los que pueden encontrarse un gran abanico de funciones controladas por microprocesador. De este hecho se deduce la gran importancia que tienen en el mundo actual estos pequeños pero sofisticados circuitos integrados, así como del gran campo de aplicación en que son útiles.

Los microprocesadores que incorporan los ordenadores modernos son extremadamente complejos y aúnan en el mismo circuito integrado varios núcleos de descodificación y procesamiento de instrucciones, unidades de cálculo en coma flotante, grandes cantidades de memoria caché de primer nivel, dispositivos de entrada/salida, etc. Por este motivo no resultan especialmente adecuados para iniciarse en la programación en lenguaje ensamblador, ya que antes de ser operativos precisan una complicada fase de inicialización que es la que llevan a cabo los sistemas operativos actuales.

Además cuentan con un conjunto de instrucciones muy extenso, en ocasiones bastante especializado, y modos de direccionamiento complejos que resultan adecuados para gestionar las grandes cantidades de memoria con que suelen contar estos equipos pero poco apropiados para el aprendizaje.

Existen microprocesadores mucho más sencillos y apropiados para la tarea que nos proponemos como objetivo, dispositivos que es posible programar en ensamblador mediante un conjunto de instrucciones reducido pero representativo de todos los tipos de operaciones que suelen encontrarse en los más completos. Estos microprocesadores se utilizaron en su día en ordenadores personales y aún hoy es posible encontrarlos en algunos electrodomésticos, allí donde los procesos a controlar no precisan la potencia de los micros modernos.

Uno de esos microprocesadores es el 8085, fabricado por la empresa Intel en los años 70 y que aún hoy, en diversas variantes, sigue siendo el corazón de algunos dispositivos. Precursor del 8086, primer microprocesador usado en los compatibles PC y del que derivan los más modernos Pentium o Core Duo, el 8085 es también el núcleo del sistema uP-2000 utilizado habitualmente en las prácticas de esta asignatura.

1 El microprocesador 8085

Antes de abordar la programación del 8085 es preciso conocer algunos detalles relativos a este microprocesador, cualidades tanto de tipo físico como lógico. En este primer punto se estudiarán los aspectos siguientes:

- La arquitectura del microprocesador 8085.
- El conjunto de registros con que cuenta.
- Características de la unidad de control, la unidad aritmético-lógica y los buses.
- El encapsulado y patillaje del microprocesador.

1.1 Arquitectura

El microprocesador 8085 fue creado por la empresa Intel para suceder al exitoso 8080, incorporando en un mismo circuito funciones para las que, hasta ese momento, era preciso agregar otros elementos externos, como la generación de pulsos de reloj, la disponibilidad de un bus de control del sistema o un gestor de interrupciones. Es un microprocesador de 8 bits de ancho de palabra y un bus de direcciones de 16 bits multiplexado parcialmente con el de datos. En la Figura 1 se han representado esquemáticamente los elementos del 8085 que van a describirse en los puntos siguientes, en los que también se refinará dicho esquema incluyendo más detalles.

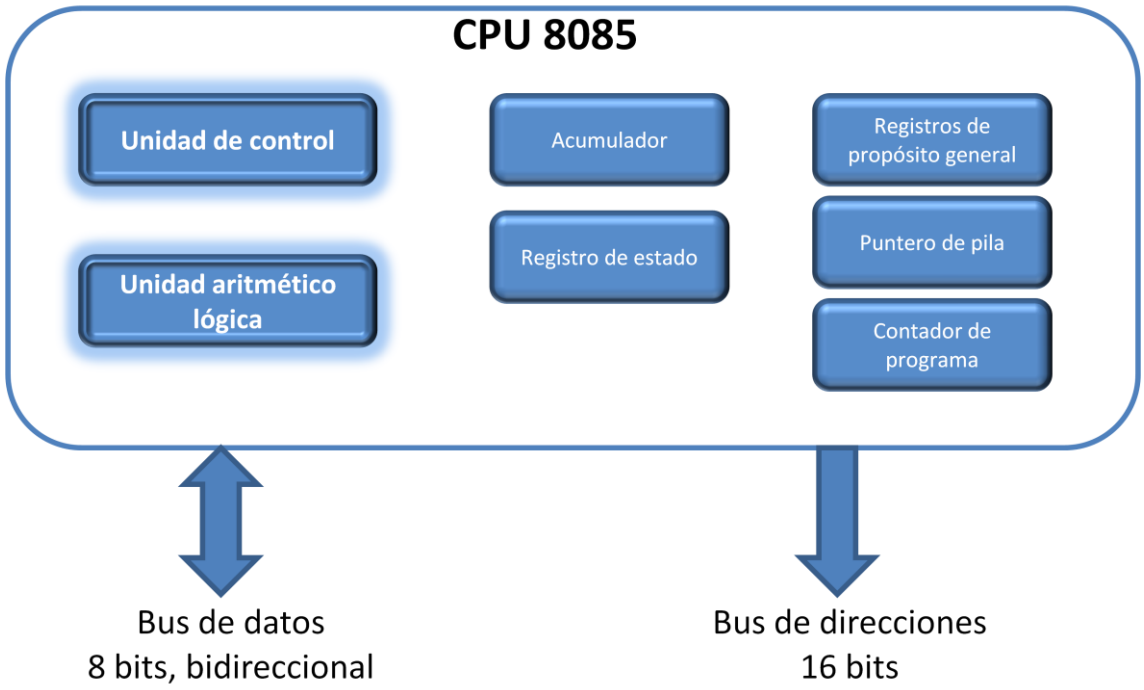


Figura 1. Esquema con los elementos básicos de la CPU 8085.

Al diseño original del 8085 se le denominó 8085A para distinguirlo de desarrollos posteriores que, manteniendo la misma arquitectura lógica y juego de instrucciones, conseguían una mayor integración, menor consumo y velocidades de ejecución superiores, tales como el 8085AH o el 8085AH-1. Las características principales de esta familia de microprocesadores son las siguientes:

- Precisa una única fuente de alimentación externa de +5V, lo cual simplifica el diseño de sistemas respecto a otros micros que necesitan múltiples alimentaciones de +5V, +12V, etc.

- Dispone de un generador interno de señales de reloj que se activa conectando un cristal externo.

- Cuenta con cuatro señales de interrupción hardware auto-vectorizada y una señal adicional de interrupción con vector por software.

CONCEPTOS

Una interrupción es una señal que altera el flujo normal de ejecución de un programa, provocando que el microprocesador pase a ejecutar otro código, que suele llamarse ISR (*Interrupt Service Rutine*) o rutina de servicio de la interrupción. Las líneas de interrupción suelen ser utilizadas por dispositivos externos conectados al microprocesador, lo cual les permite solicitar la atención de la CPU cuando se producen ciertos sucesos, como puede ser la pulsación de una tecla o la llegada de un dato a través de un dispositivo de comunicaciones.

La dirección en que se encuentra la ISR de una determinada interrupción puede ser conocida de antemano por el microprocesador, en cuyo caso se dice que es una interrupción auto-vectorizada. Cuando se produce una interrupción de dicho tipo la CPU siempre salta a la misma dirección y ejecuta la ISR que haya instalada en dicha dirección. También existen interrupciones no auto-vectorizadas, de forma que el dispositivo que la genera no solamente tiene que activar la línea de interrupción sino que, además, debe comunicar al microprocesador la dirección a la que debe saltar.

En un punto posterior se abordará con mayor detalle el estudio de las interrupciones en el 8085 y su funcionamiento.

- Capacidad para direccionar un máximo de 65.536 posiciones de memoria gracias a su bus de direcciones de 16 bits: 2^{16}=65536

- Cuenta con un bus de datos de 8 bits, siendo por tanto ésta la cantidad de información que puede trasladarse desde la CPU hacia el exterior, memoria o dispositivos, o viceversa.

- Integra un bus de control ampliado, lo cual simplifica también el diseño de sistemas. En microprocesadores anteriores, como el 8080, las señales de control del sistema dependían de un integrado externo a la CPU, conocido como 8228, que con el 8085 ya no es preciso.

- Dispone de un puerto serie de entrada/salida de datos e incorpora instrucciones específicas para programarlo.

- El encapsulado del microprocesador es el estándar de 40 pines o patillas, lo cual le hace compatible con sistemas diseñados para el 8080 e incluso el 8008.

El núcleo del 8085 está formado, como cualquier otro microprocesador, por lo que se denomina genéricamente CPU (*Central Process Unit*) o UCP (*Unidad Central de Proceso*), en la que se encuentran los tres componentes fundamentales de cualquier procesador:

- El conjunto de registros internos en los que se almacena temporalmente la información sobre la que se trabaja.

- La CU (*Control Unit*) o UC (*Unidad de control*), encargada de interpretar las instrucciones y de generar todas las señales de control que dirigen y coordinan al resto de los elementos.

- La ALU (*Aritmethic Logic Unit*) o UAL (*Unidad aritmético lógica*), en la que se efectúan todas las operaciones aritméticas y lógicas.

Estos tres componentes se comunican entre ellos a través de unos buses de control, de datos y de direcciones, y a través de las patillas del micro esos buses se extienden al resto de los elementos del sistema. En los siguientes apartados se estudian con detalle todos esos elementos: registros, UC, UAL, buses y patillaje.

1.2 Registros

Un registro es una pequeña porción de memoria que reside dentro del mismo circuito integrado que el resto de los elementos que forman el microprocesador, por lo que el acceso a su contenido, ya sea para recuperarlo o modificarlo, resulta muy rápido. Dependiendo del diseño del procesador se tendrán más o menos registros, su tamaño será de 8, 16, 32, 64 o más bits y cada uno de ellos tendrá asignada una función concreta o bien serán de uso general.

En el caso concreto del 8085 los registros son de 8 o de 16 bits y, aunque algunos pueden utilizarse como de uso general, en muchas operaciones tienen asignadas funciones predeterminadas. Cada registro se identifica habitualmente con una o dos letras, tal y como se indica en la Tabla 1.

Nombre	Bits	Función
A	8	Actúa como acumulador en todas las operaciones
B, C, D, E, H y L	8	Son de uso general. Puede combinarse en parejas como de 16 bits con los nombres BC, DE y HL.
SP	16	Puntero de pila
PC	16	Puntero de programa
F	8	Indicadores de estado

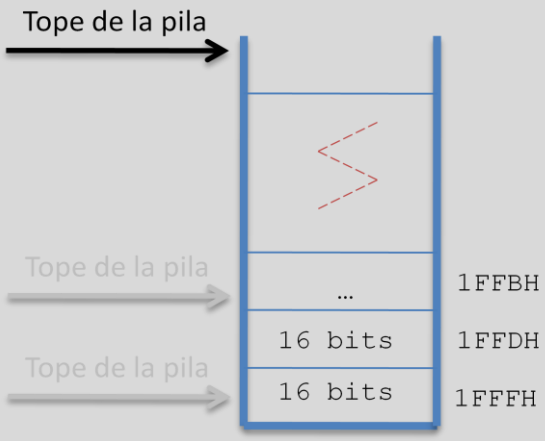
Tabla 1. Registros del 8085.

Los registros PC y SP siempre contienen direcciones, de ahí la denominación de *punteros*, indicando en qué posición de memoria está la siguiente instrucción a ejecutar o el tope de pila, respectivamente.

CONCEPTOS

Una pila es una estructura de datos básica que se caracteriza porque los datos introducidos se extraen en orden inverso, es decir, el último dato en apilarse es el primero que se recupera. Por ello se dice que tiene estructura LIFO (*Last In-First Out*).

Los microprocesadores usan una pila para distintas operaciones, principalmente para guardar direcciones de retorno a las que han de volver en algún momento. En el 8085 cada elemento de la pila tendrá un tamaño de 16 bits, ya que ése es el número de bits del bus de direcciones. El registro SP indica en cada momento la dirección donde está la parte superior de la pila, conocida también como *tope de la pila*, según se aprecia en el esquema siguiente:



Inicialmente el puntero de pila pudo contener la dirección `1FFFH`, al iniciar la ejecución de un programa. Al guardar un primer elemento se restan dos unidades a dicha dirección, de forma que el tope de pila se encontraría en `1FFDH`. Cuando se recuperan elementos, extrayéndolos de la pila, el `SP` incrementa la dirección que contiene, en lugar de reducirla.

El contenido del registro `PC` se actualiza automáticamente a medida que el procesador va ejecutando instrucciones, ya sea de forma secuencial o cambiando totalmente su valor al producirse un salto. El registro `SP` también se actualiza de manera automática, cada vez que se introduce o extrae un valor de la pila, si bien puede ser manipulado mediante ciertas instrucciones.

Aunque los registros `B`, `C`, `D`, `E`, `H` y `L` son de propósito general, al utilizarse como parejas de registros de 16 bits suele asignárseles alguna función como: `BC` actúa como puntero base o como contador, `DE` como puntero de destino y `HL` como puntero de datos y como acumulador en operaciones de 16 bits. En los ejercicios pueden verse algunos ejemplos de estos usos.

El registro `F`, también conocido como registro de *flags*, registro de estado o de indicadores, es de 8 bits pero únicamente 5 de ellos se utilizan en la práctica. Estos bits se modifican tras realizar ciertas operaciones según el resultado que generan, existiendo instrucciones que permiten, dependiendo de dicho estado, alterar el flujo de ejecución. La estructura de este registro es la indicada en la Figura 2.

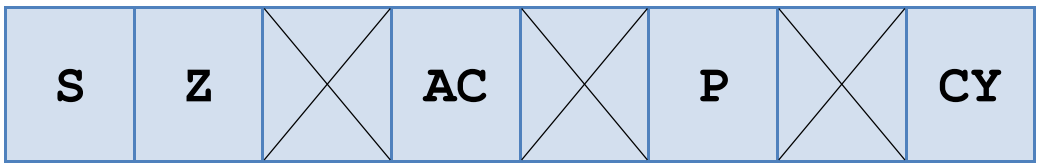


Figura 2. Bits del registro de estado.

Los bits del registro de estado intervienen y son modificados principalmente por la ALU (*Arithmetic Logic Unit*) o UAL (*Unidad Aritmético-Lógica*), al ejecutar operaciones aritméticas y lógicas. La estructura de bloques de la UAL es la mostrada en la Figura 3, en la que se destacan los elementos con que interacciona este componente del microprocesador: el acumulador, un registro temporal y el registro de estado.

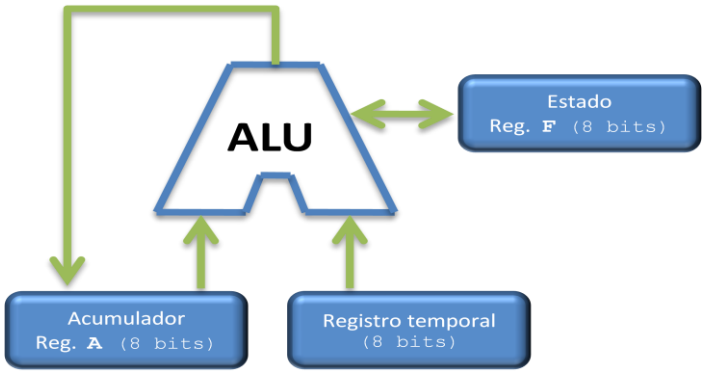


Figura 3. Esquema de bloques de la UAL.

En el apartado 1.4 se detallan algunos aspectos más sobre la UAL. Continuando con el registro de estado, el significado de cada uno de los indicadores señalados en la Figura 2 es el que se explica a continuación:

- `CY`: Es el conocido como *CarrY* o *arrastre*. Se activa cuando se produce un desbordamiento tras una operación aritmética, actuando como un noveno bit cuando los ocho del acumulador no resultan suficientes.

- `P`: Indica la paridad del dato que ha quedado en el acumulador (registro `A`), activándose cuando el número de bits a 1 es par y desactivándose cuando es impar.

- `AC`: Es el *Auxiliary Carry* o *arrastre auxiliar*. Se utilizado en operaciones aritméticas BCD (*Binary Coded Decimal*/*Decimal codificado en binario*), activándose cuando hay un desbordamiento del nibble bajo del acumulador hacia el nibble alto.

- `Z`: Se le conoce como *Zero* e indica cuándo el resultado de una operación ha sido cero, activándose ante tal situación.

- `S`: Cuando se utilizan datos con representación en complemento a dos el bit de mayor peso sirve para indicar el signo, un signo que se duplica en este indicador. Se activará, por tanto, cuando el resultado obtenido sea negativo.

A medida que se describan las distintas instrucciones del 8085 y su aplicación práctica, en la segunda parte de este manual, se irá indicando cómo su ejecución afecta a estos indicadores y la forma de comprobarlos.

Los registros mencionados son los que existen físicamente dentro del 8085, pero al programar en ensamblador se utilizarán ciertas denominaciones simbólicas como si fuesen registros. Así, en ciertas operaciones el nombre `B`, `D` o `H` hará referencia en realidad a las parejas de registros `BC`, `DE` y `HL`, respectivamente. La denominación `PSW` representa al acumulador y el registro de indicadores juntos, es decir, `AF`. Otro caso usual es el registro simbólico `M` que, en realidad, hace referencia al contenido de la dirección de memoria apuntada por `HL`.

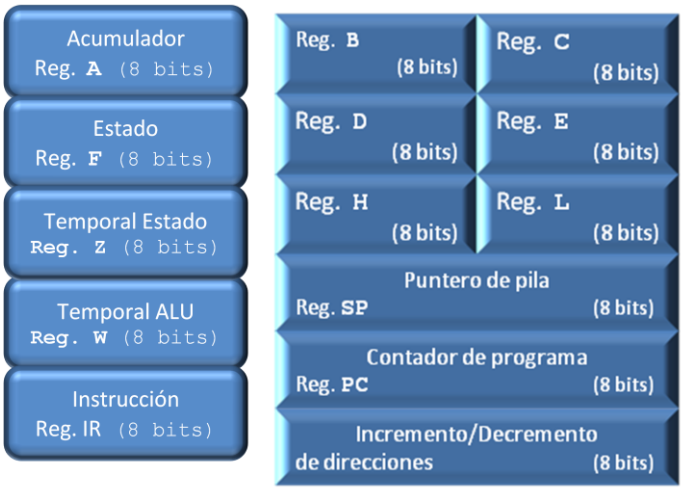


Figura 4. Conjunto de registros del 8085.

Además de los registros que resultan accesibles desde un programa en ensamblador, a través de los nombres físicos o simbólicos ya citados, el 8085 cuenta, como todos los microprocesadores, con algunos registros más que permanecen ocultos y que se utilizan internamente para, por ejemplo, conservar resultados parciales de una operación o almacenar el código de la instrucción a ejecutar. En el esquema de la Figura 4 puede verse el conjunto de todos los registros del 8085. Algunos de ellos no pueden usarse directamente, pero es preciso saber de su existencia para poder entender cómo tienen lugar algunas operaciones.

En algunos documentos de especificación del fabricante esos registros ocultos tienen asignados nombres como `IR` (*Instruction Register*), `W` o `Z`, pero son identificadores que no pueden ser utilizados a la hora de escribir un programa.

1.3 Unidad de control

Los registros son una parte del microprocesador relativamente simple, ya que se limitan a mantener una información durante el tiempo en que el sistema está en funcionamiento. Esa información, en el caso del 8085 datos de 8 o 16 bits, procede de la memoria del sistema o bien de operaciones efectuadas en la UAL, y es la UC la que ordena las transferencias por las cuales se lee el contenido de un registro o se escribe un dato en un registro.

Cuando el microprocesador lee una instrucción a ejecutar, incrementando a continuación el `PC`, es la UC la que se encarga de generar las señales necesarias para llevar a cabo esa ejecución, señales que desencadenan lecturas y escrituras en la memoria del sistema y los registros, realización de operaciones por parte de la UAL sobre el contenido de los registros o cambios en la propia configuración interna del microprocesador, por ejemplo alterando la manera en que tratará las interrupciones.

Podría decirse que el *cerebro* con el que se compara habitualmente un microprocesador está dividido claramente en dos hemisferios distintos: uno de control y otro de cálculo, al que se agrega un área de memoria. Es la parte de control la que se encarga de activar, desactivar, conectar y coordinar a las demás.

1.4 Unidad aritmético lógica

La UAL trabaja a las órdenes de la UC, llevando a cabo todas las operaciones aritméticas y lógicas. Entre las primeras se incluyen la suma y la resta y entre las segundas las de lógica booleana, como `AND` y `OR`.

El primer operando sobre el que actúa la UAL siempre se encuentra en el acumulador, mientras que el segundo puede proceder de un registro de propósito general o bien de la memoria. Efectuada la operación, el resultado queda depositado en el acumulador. Dependiendo de este resultado se activarán los bits adecuados en el registro de indicadores o estado.

Como se aprecia en la Figura 5, la comunicación de la UAL con el registro de estado es bidireccional. Esto se debe a que el estado de ciertos bits puede influir en el resultado de la operación a ejecutar. La instrucción `ADC`, por ejemplo, no se limita a sumar al acumulador un dato (contenido en el registro temporal) sino que, además, comprobará si está activo el bit de acarreo y de ser así sumará una unidad más al resultado.

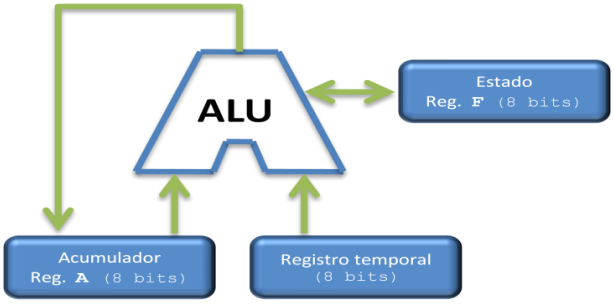


Figura 5. Esquema de bloques de la UAL.

Antes se indicaba que en el microprocesador existen registros ocultos, no accesibles para los programas en ensamblador. La UAL utiliza algunos de esos registros, por ejemplo a la hora de comparar el contenido del acumulador con otro valor. Lo que se realiza es una operación de resta, que afecta al registro de indicadores, pero el contenido del acumulador queda finalmente inalterado. Para ello la UAL mantiene el contenido original del acumulador en uno de esos registros ocultos, el registro temporal, de donde lo recupera tras realizar la operación de resta.

Las operaciones que puede llevar a cabo la UAL del 8085 son de los tipos siguientes:

- Sumas y restas de 8 bits en binario.
- Sumas y restas de 8 bits en BCD.
- Sumas de 16 bits en binario.
- Operaciones lógicas `OR`, `AND` y `XOR`.
- Rotaciones y desplazamientos de bits.

En todas estas operaciones siempre participa el acumulador como primer o único operando, dependiendo de los casos. Si la operación precisa dos operandos, el segundo será transportado desde donde se encuentre, la memoria u otro registro, hasta el registro temporal de la UAL. El resultado siempre quedará en el acumulador.

CONCEPTOS

La aritmética BCD (*Binary Coded Decimal*) se fundamenta, básicamente, en la representación de un dígito decimal (0 a 9) usando 4 bits, de forma que en un byte es posible introducir dos dígitos binarios. De esta forma la secuencia de bits 10010011 no se correspondería con el número 147, que sería el resultado al operar en binario, sino con el número 93 ($1001_2=9_{10}$ y $0011_s=3_{10}$).

La conversión entre BCD y decimal resulta muy rápida y cómoda, por eso la mayoría de los microprocesadores están preparados para utilizar este tipo de aritmética. El 8085 no es una excepción, contando con un bit en el registro de estado, el bit `AC`, que señala el acarreo en BCD. También hay instrucciones específicas parar operar con aritmética BCD.

1.5 Buses

Para diseñar un sistema útil basado en el microprocesador 8085 serán precisos, aparte del propio micro, una serie de elementos adicionales que, como mínimo, incluirá algún módulo de memoria en la que poder almacenar los programas y datos a ejecutar. Además pueden existir otros, como los controladores de teclado y pantalla, los módulos de interfaz con dispositivos paralelo, etc.

La comunicación entre el microprocesador y los elementos externos, así como entre las propias partes que componen el procesador, se lleva a cabo a través de unas vías denominadas *buses*. Lo habitual es que existan tres:

- **Bus de direcciones**: Es utilizado por la UC para indicar a la memoria la posición que se quiere leer o donde se va a escribir. Del tamaño de este bus dependerá la cantidad de memoria a la que pueda accederse.

- **Bus de datos**: Transporta los datos a través del sistema, por ejemplo desde los registros hacia la memoria o viceversa. Normalmente es el ancho de este bus el que se indica como *ancho de palabra* o *tamaño de palabra* del microprocesador, ya que establece el número de bits máximo que puede transferirse en una sola operación.

- **Bus de control**: Por él se transfieren todas las señales que transmite la UC a los diferentes elementos del sistema, por ejemplo indicando a la memoria si va a efectuarse una lectura o una escritura, comunicando a la UAL la operación que debe realizar, etc.

En el 8085 el bus de direcciones tiene 16 bits, por lo que es posible componer un total de 2^{16} direcciones distintas, desde la 0 a la 65535, lo que hace un total de 64 Kbytes. Esto es lo que se denomina habitualmente capacidad de direccionamiento. El bus de datos es de 8 bits, permitiendo por tanto la transferencia de 2^8 valores diferentes, codificados desde el 0 hasta el 255. En la Figura 6 puede verse un esquema del 8085 en el que se ha destacado la relación de los buses con los distintos elementos del microprocesador, así como el hecho de que existe un bus interno al integrado, que se utiliza para transportar información entre UAL, UC y registros.

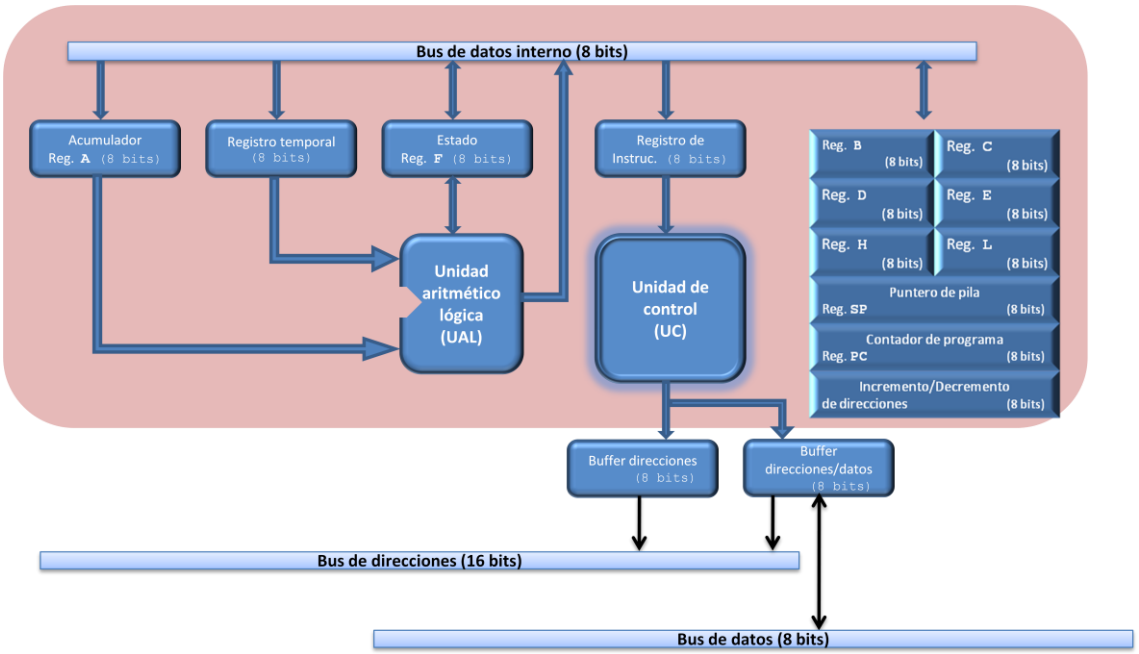


Figura 6. Esquema de los buses existentes en el 8085.

Desde un punto de vista físico los buses se reflejan como conectores (pines) del microprocesador que permiten interconectarlo con el resto del sistema, debiendo existir, teóricamente, un pin por cada bit de cada bus. Es posible, sin embargo, utilizar un mismo pin para transmitir señales alternadas en el tiempo mediante una técnica conocida como *multiplexación*, haciendo así posible la reducción del número total de pines necesarios con que debe contar el microprocesador. El 8085 emplea esta técnica y utiliza los 8 bits del bus de datos como parte de los 16 del bus de direcciones o, si se quiere pensar a la inversa, los 8 bits del bus de datos son en realidad parte de los 16 del bus de direcciones. En cualquier caso ambos buses utilizan solamente 16 pines del microprocesador, en lugar de los 24 (8 de datos y 16 de direcciones) que cabría esperar en un principio.

1.6 Esquema de bloques

Todos los componentes citados en los apartados previos están, como se ha ido apuntando, conectados entre sí y en ocasiones también conectados con el exterior, con el resto de los elementos que forman el sistema basado en microprocesador. Deducir cuál es la estructura resulta más fácil con un esquema de bloques como el de la Figura 7, en el que se han representado el banco de registros de memoria, la UAL, varios elementos de la UC y los buses con sus conexiones hacia el exterior. Para entender este esquema ha de tenerse en cuenta que:

Los bloques que aparecen sobre un fondo de color salmón componen el banco de registros del microprocesador, incluyendo, aparte de los accesibles desde un programa en ensamblador, algunos registros de uso interno como el registro de instrucción o el de incremento y decremento de direcciones. Hay que destacar la posición distintiva del acumulador, el registro A*, respecto al resto de registros de uso general. El acumulador está conectado directamente a*

la UAL, mientras que el resto de registros se comunican con esa parte del procesador a través del bus interno de datos, utilizando un registro temporal como intermediario en las operaciones.

Hay que poner atención a las puntas de flecha que comunican las distintas partes del microprocesador, ya que establecen el sentido en que se transfieren las señales, direcciones o datos. La línea de conexión entre la UAL y el registro de indicadores, por ejemplo, es de doble sentido, de forma que la UAL puede tanto modificar el registro como leerlo, de forma que su contenido puede influir en la operación que se esté llevando a cabo en ese momento. La unión entre el registro de instrucción y la unidad de decodificación de instrucciones, sin embargo, es de sentido único, de forma que dicho registro solamente puede ser leído desde esa unidad, pero nunca modificado, lo cual tiene sentido ya que los códigos de instrucción van leyéndose del bus de datos interno, descodificándose y generando las señales adecuadas para su ejecución, pero el propio procesador no genera códigos de instrucción que sea necesario descodificar.

Por último, en cuanto al esquema de bloques se refiere, las líneas de color negro con punta de flecha representan las conexiones hacia el exterior, es decir, las patillas o pines con que cuenta el microprocesador. El sentido de las flechas indica si las señales van desde el micro hacia afuera, desde el exterior hacia el micro o si la comunicación es bidireccional como ocurre, por ejemplo, con el bus de datos. Cada flecha representa un bit salvo en el caso del bus de direcciones y el bus de datos que, tal y como está indicado, son de 8 bits.

Si se hace un recuento de todas las señales que entran o salen del microprocesador: 14 de la unidad de temporización y control, 6 del control de interrupciones, 2 del control de E/S serie, 8 del bus de datos y 8 del bus de direcciones, se obtiene un total de 38 pines, a los que deben sumarse los dos pines de alimentación eléctrica del integrado, completando así las 40 patillas a las que antes se hacía referencia.

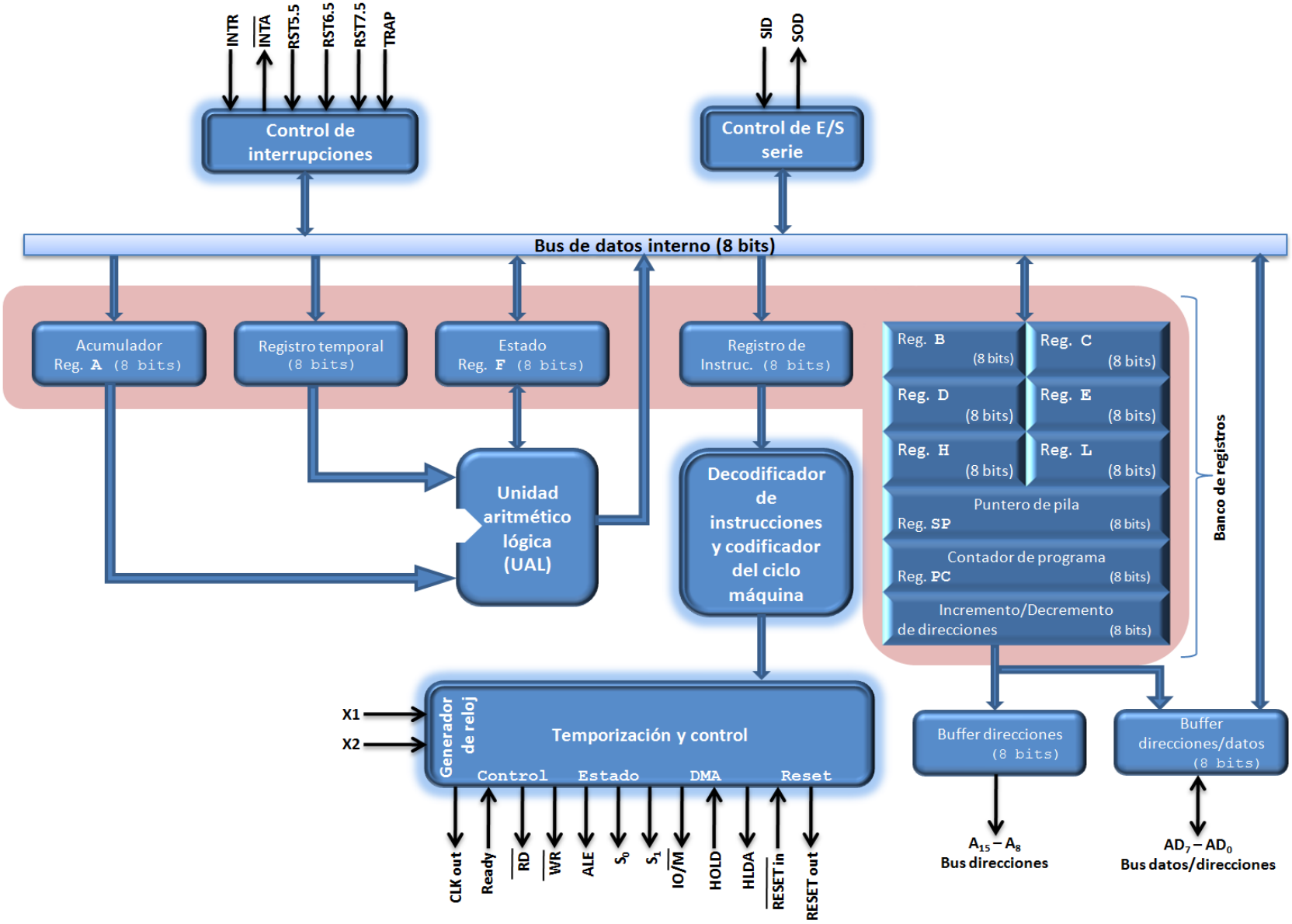


Figura 7. Esquema de bloques del microprocesador 8085.

1.7 Estados y ciclos de ejecución

En el momento en que se alimenta eléctricamente el 8085, asumiendo que éste forma parte de un sistema completo basado en microprocesador, el contador de programa (registro PC) toma un valor por defecto y a, partir de ese momento, se inicia el proceso de ejecución que estará repitiéndose constantemente hasta el momento en que se corte la alimentación o se reinicie el microprocesador.

El proceso de ejecución se divide en unos pasos a los que se denomina *ciclos de máquina*, los cuales se divides en partes más elementales llamadas *estados*. En el caso del 8085 existen siete tipos diferentes de ciclos de máquina que son los siguientes:

- **OF** (*Opcode Fetch*): Es el ciclo en el que se recupera un código de instrucción de la memoria del sistema.
- **MR** (*Memory Read*): Ciclo de lectura de datos de la memoria.
- **MW** (*Memory Write*): Ciclo de escritura de datos en la memoria.
- **IOR** (*IO Read*): Ciclo de lectura de entrada/salida.
- **IOW** (*IO Write*): Ciclo de escritura en entrada/salida.
- **INA** (*Acknowledge of interrupt*): Ciclo de reconocimiento de interrupción.
- **BI** (*Bus Idle*): Ciclo durante el que se libera el bus de datos.

Cada uno de estos ciclos corresponde a un tipo de operación que el microprocesador puede llevar a cabo, necesitando normalmente tres estados para completarse. Cada estado se corresponde a un pulso de reloj, de forma que la evolución de las señales a lo largo de los estados puede representarse mediante un cronograma como el de la Figura 8.

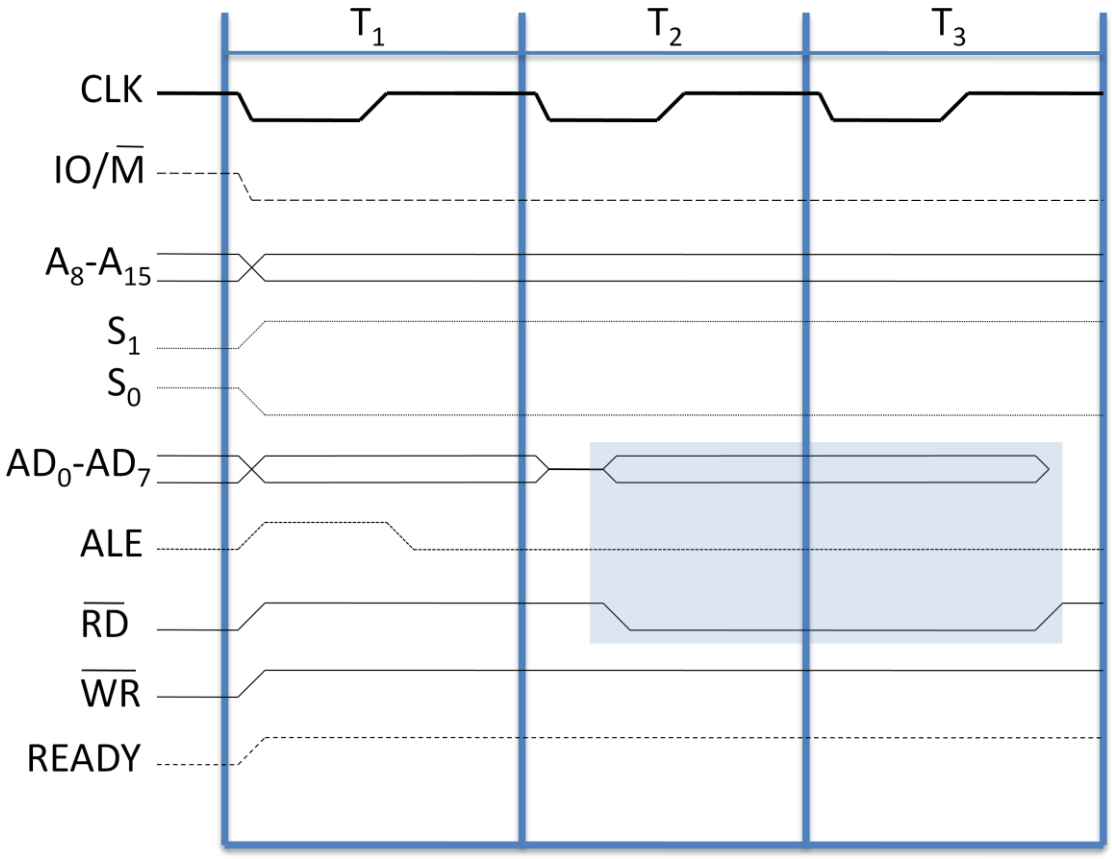


Figura 8. Cronograma de un ciclo máquina de lectura de memoria.

La señal de reloj (CLK) marca la cadencia con la que se va pasando de un estado al siguiente, de forma que la UC pueda ir activando las señales de control necesarias para transferir la dirección a leer al bus de direcciones, durante T_1, y leer el dato, entre T_2 y T_3.

En la Figura 8 se ha resaltado con un fondo de color el área que corresponde al momento en que las líneas AD_0–AD_7 pasan a actuar como bus de datos, al ponerse a nivel bajo la línea ALE. Es en ese instante cuando la patilla de lectura, activa a nivel bajo, desencadena la transferencia desde la memoria al microprocesador.

1.8 Patillaje

El diagrama de bloques de la Figura 7 ofrece una perspectiva lógica de los componentes que forman el microprocesador, un circuito integrado que físicamente tiene el aspecto que puede apreciarse en la Figura 9 y 10, correspondientes a la variante 8085AH2 que es la que incorpora el sistema uP-2000 utilizado en prácticas.

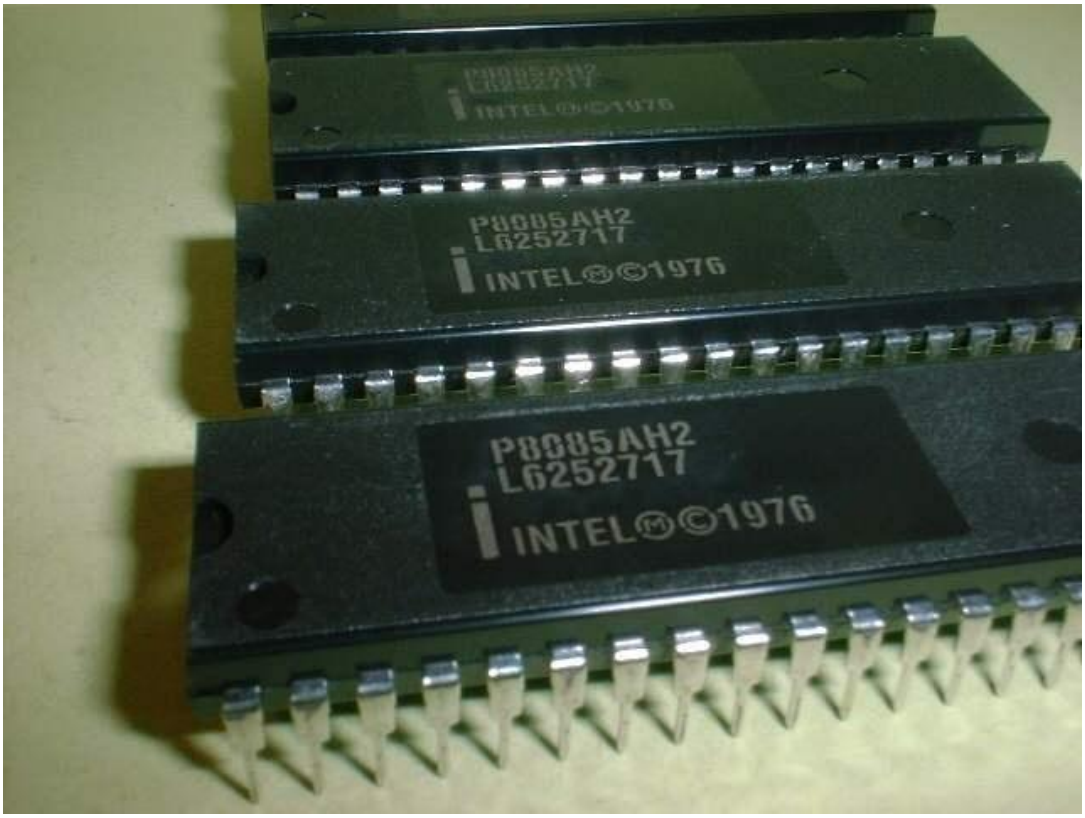


Figura 9. Colección de microprocesadores 8085-AH2-

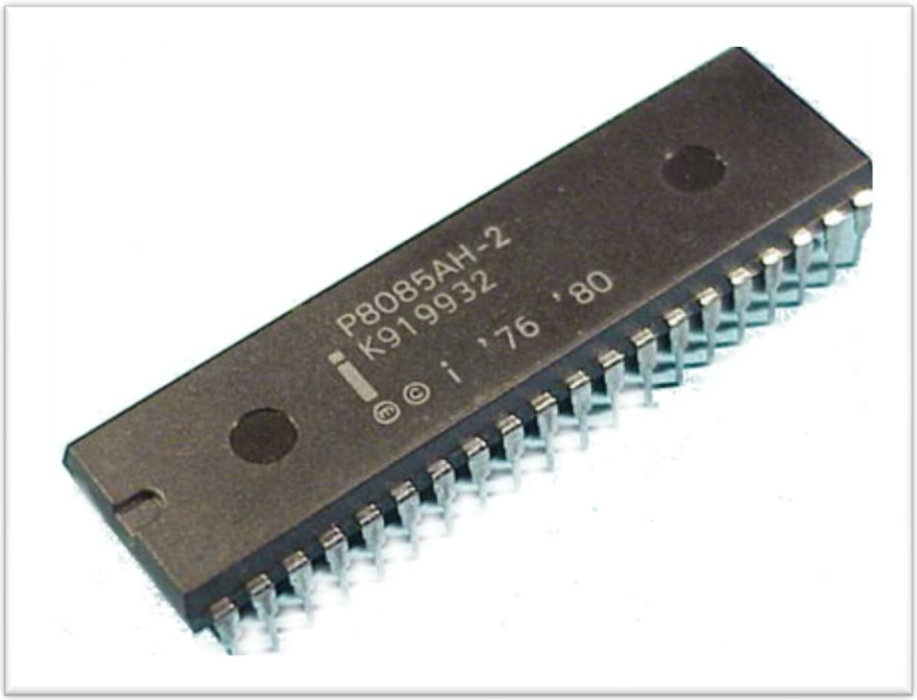


Figura 10. Detalle de uno de los microprocesadores.

A la izquierda de la muesca que puede verse claramente en las fotografías anteriores se encuentra el pin número 1, asignándose número secuencialmente a los inferiores del mismo lado y después, en sentido inverso, a los del lado derecho, como se muestra en el diagrama de la Figura 11 en el que también aparece el nombre de cada patilla y se han destacado las que forman parte del bus de datos y direcciones.

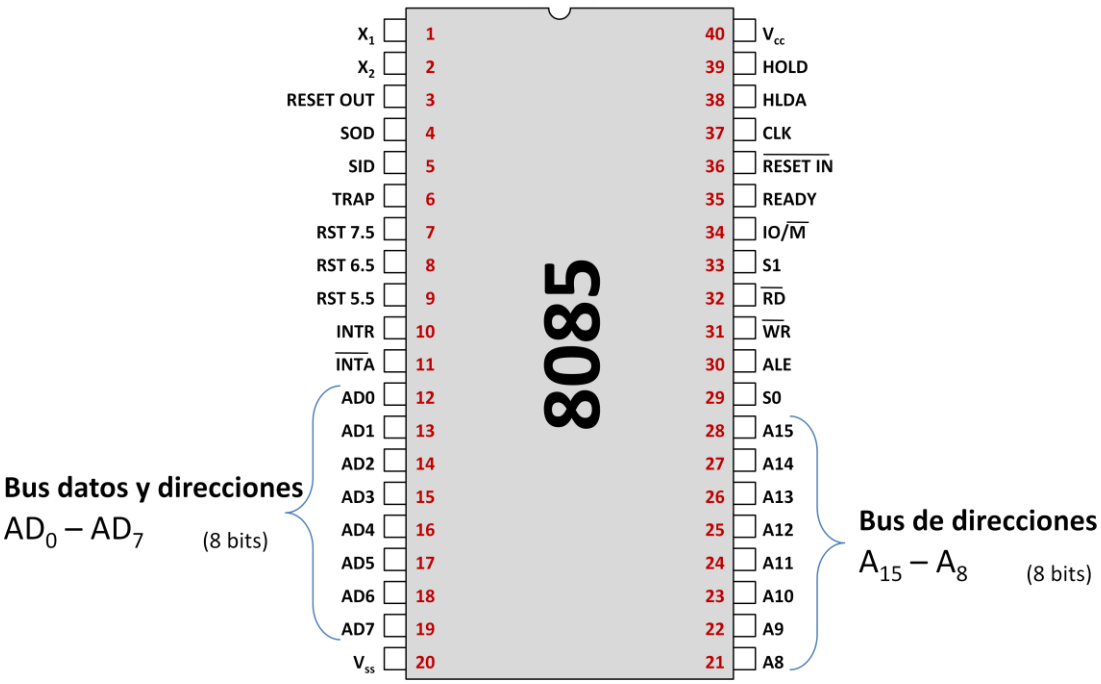


Figura 11. Patillaje del 8085.

En la Tabla 2 se describe con más detalle cuál es la función de cada uno de esos pines, así como el sentido que se transfieren los datos: E (entrada), S (salida) o E/S (entrada y salida).

Denominación	Sentido	Función
A_8-A_{15}	S	Forman parte del bus de direcciones, aportando los 8 bits más significativos o de mayor peso. En operaciones de E/S tienen la dirección del puerto.
AD_0-AD_7	E/S	Bus multiplexado de datos y direcciones. Durante el primer estado de cada ciclo de reloj estas líneas contienen los 8 bits menos significativos del bus de direcciones, o bien la dirección del puerto de E/S, pasando en los dos estados siguientes a actuar como bus de datos.
ALE	S	Esta señal es la encargada de activar, durante el primer estado de cada ciclo, la aparición de la dirección en las patillas AD_0-AD_7. El flanco de bajada de esta señal indica que el bus de direcciones ha recogido la

<table>
<tr><th>Denominación</th><th>Sentido</th><th>Función</th></tr>
<tr><td></td><td></td><td>información y, por tanto, las líneas AD_0-AD_7 pueden pasar a actuar como bus de datos en los dos estados siguientes.</td></tr>
<tr><td>IO/$\overline{M}$, S_0 y S_1</td><td>S</td><td>Mediante estos tres bits se establece el estado al inicio de cada ciclo máquina el estado según el patrón siguiente:

<table>
<tr><th>IO/M</th><th>S_0</th><th>S_1</th><th>Estado</th></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Escritura en memoria</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Lectura de memoria</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Escritura en puerto E/S</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Lectura de puerto E/S</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Recuperación de código de operación</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Reconocimiento de interrupción</td></tr>
<tr><td>*</td><td>0</td><td>0</td><td>Estado HALT</td></tr>
<tr><td>*</td><td>X</td><td>X</td><td>Estado HOLD o RESET</td></tr>
</table>

El símbolo * indica estado de alta impedancia en la patilla y el símbolo X que el estado no está determinado.</td></tr>
<tr><td>$\overline{RD}$</td><td>S</td><td>Control de lectura. Cuando esta patilla está a nivel bajo indica que se va a efectuar una lectura de memoria o un dispositivo de E/S y que el bus de datos está disponible para llevar a cabo la transferencia.</td></tr>
<tr><td>$\overline{WR}$</td><td>S</td><td>Control de escritura. Cuando esta patilla está a nivel bajo indica que el dato existente en el bus de datos ha de ser escrito en memoria o un dispositivo de E/S.</td></tr>
<tr><td>READY</td><td>E</td><td>Mediante esta señal los dispositivos de E/S notifican al microprocesador cuándo están preparados para recibir o enviar datos, poniéndola a nivel alto. Si durante un ciclo de lectura o escritura esta patilla está a nivel bajo, el micro se queda esperando durante un número de ciclos hasta que pase a nivel alto antes de completar la operación.</td></tr>
<tr><td>HOLD</td><td>E</td><td>Indica al procesador que un dispositivo externo quiere controlar los buses de datos y direcciones. Al recibir esta señal el micro finaliza el ciclo que estaba ejecutándose, libera los buses y puede continuar efec-</td></tr>
</table>

Denominación	Sentido	Función
		tuando operaciones internas, pero no podrá retomar el control de los buses de datos y direcciones hasta que se libere la señal HOLD. El caso más típico se da cuando un controlador de DMA quiere efectuar una transferencia entre un dispositivo y la memoria.
HLDA	S	Es la señal que el microprocesador envía para notificar que ha recibido la petición HOLD y que liberará los buses en el ciclo siguiente. En el momento en que la línea HOLD pasa a nivel bajo también lo hace HLDA, medio ciclo después el micro recupera el control de los buses.
INTR	E	Solicitud de interrupción de propósito general. Se comprueba tras el último ciclo de ejecución de cada instrucción. Si está activa se inhibe el incremento del registro PC (el contador de programa) y se activa la señal INTA. En ese momento se recoge del bus de datos la instrucción a ejecutar para atender la interrupción, que puede ser de tipo RST o CALL. La señal INTR es enmascarable.
$\overline{\text{INTA}}$	S	Con esta señal el microprocesador confirma el reconocimiento de la señal INTR, pudiendo utilizarse para activar un controlador de interrupciones externo como el 8259A.
RST 7.5 RST 6.5 RST 5.5	E	Señales de interrupción autovectorizadas. Estos pines se utilizan para conectar dispositivos que pueden precisar la atención inmediata del microprocesador, para lo cual activan la línea de interrupción provocando la ejecución del código alojado en la dirección establecida por el vector correspondiente. Estas señales de interrupción también son enmascarables y tienen un nivel de prioridad superior al de la señal INTR. En caso de presentarse más de una de estas señales, la prioridad de ejecución es primero la 7.5, después la 6.5, a continuación la 5.5 y, por último, la INTR.
TRAP	E	Señal de interrupción de más alta prioridad y no enmascarable. Es también de tipo autovectorizado, como las RST X.5.

Denominación	Sentido	Función
$\overline{\text{RESET IN}}$	E	Efectúa una inicialización del procesador, proceso durante el que ciertas señales, como HLDA, pasan a nivel bajo, poniéndose a cero el registro PC y reiniciándose la activación de interrupciones. El resto de registros, incluido el de indicadores, también podrían ver alterado su contenido si bien no es algo predecible.
RESET OUT	S	Es una señal que el procesador envía al resto del sistema para indicar que se ha reiniciado, pudiendo ser utilizada para reiniciar también otros elementos. Está sincronizada con la señal de reloj del sistema.
X_1 y X_2	E	Son las patillas a las que debe conectarse el cristal o red LC/RC que actuará como reloj externo. La frecuencia recibida se divide entre 2 para obtener la frecuencia de trabajo interna del microprocesador
CLK	S	Es la señal de reloj facilitada por el microprocesador al resto del sistema, haciendo posible la sincronización de los distintos elementos que lo compongan.
SID	E	Mediante este pin es posible enviar al procesador un flujo de datos en forma series de bits que, secuencialmente, irían llevándose al bit de mayor peso del acumulador.
SOD	S	Complementaria de la anterior, esta patilla actúa como salida en una comunicación serie llevando el bit más significativo del acumulador hacia el exterior.
Vcc	-	Fuente de alimentación +5V.
Rss	-	Fuente de alimentación 0V/tierra.

Tabla 2. Patillaje del 8085.

Lo interesante, en este momento, no es conocer todos los detalles sobre cada una de las patillas del procesador, sino tener una noción general sobre su función. Más adelante, a medida que se profundice en la programación en ensamblador del 8085, se conocerán los pormenores y la relación entre las instrucciones, las señales y el estado interno del micro.

1.9 Interrupciones

El mecanismo de interrupciones del 8085 facilita la comunicación con los dispositivos que tenga instalados el sistema, por ejemplo el teclado en el caso del uP2000. La secuencia lógica que se produce durante esa comunicación sería la siguiente:

- El dispositivo externo, por ejemplo el teclado, necesita que el micro le atienda, por lo que activa la señal `INTR` para generar una interrupción.

- El microprocesador examina la señal `INTR` tras cada instrucción. Al detectar la solicitud de interrupción detiene la ejecución actual y activa la señal `INTA`, indicando al dispositivo que puede tomar el control.

- A continuación el dispositivo envía por el bus de datos una instrucción `RST n`.

- El procesador guarda el valor del contador de programa e invoca al vector correspondiente, donde se encontrará presumiblemente el código para procesar la interrupción. Dicho vector se encuentra en la dirección de memoria `n x 8`. El propio número de interrupción sirve para obtener la posición donde está la ISR a ejecutar.

- Cuando el tratamiento ha terminado, se devuelve al control al punto anterior del programa.

El 8085 cuenta con seis patillas relacionadas con el tratamiento de interrupciones, patillas llamadas `INTR`, `INTA`, `TRAP`, `RST 5.5`, `RST 6.5` y `RST 7.5` según se aprecia en la Figura 11. La primera indica que se ha solicitado una interrupción, la segunda señala la aceptación de una interrupción, la tercera tiene lugar generalmente ante un fallo, las otras tres representan interrupciones con distintas prioridades. Las diferencias entre las cinco señales de petición de interrupción son las siguientes:

- `INTR`: Es una señal de interrupción sin máscara asociada pero que puede ser desactivada mediante la instrucción `DI`, es decir, para que se reconozca esta interrupción es preciso haber usado antes, en algún momento, la instrucción `EI`.

- `RST 5.5`, `RST 6.5` y `RST 7.5`: Son interrupciones enmascarables. Además de poder utilizar las instrucciones `DI/EI`, para activar o desactivar globalmente todas las interrupciones enmascarables, con estas interrupciones también es posible configurar una máscara que las activa o desactiva individualmente.

- `TRAP`: Es la interrupción no enmascarable.

Que una interrupción sea enmascarable significa que es posible desactivarla, ya sea de manera selectiva (individualmente) o de forma conjunta con otras. Como se ha indicado, las interrupciones `RST 5.5`, `RST 6.5` y `RST 7.5` son enmascarables de forma individual. Los

bits de la máscara de interrupción, que pueden obtenerse en el acumulador mediante la instrucción SIM, son los mostrados en la Figura 12.

--	--	--	R7.5	WR	M7.5	M6.5	M5.5

Figura 12. Máscara de establecimiento de interrupciones.

El bit WR debe ponerse a 1 para poder modificar la máscara, de lo contrario la instrucción SIM no tendrá efecto. Los bits M5.5, M6.5 y M7.5 determinan si estarán activas o no las interrupciones RST5.5, RST6.5 y RST7.5, respectivamente. Poniendo a 1 un bit se habilita la interrupción, mientras que con 0 se deshabilita. El bit R7.5 reinicia un biestable interno asociado a la interrupción RST7.5.

Cuando se lee la máscara, mediante la instrucción RIM, el significado de los primeros cuatro bits: M5.5, M6.5, M7.5 y WR, será el mismo indicado para SIM de forma que es posible determinar si cada señal de interrupción está habilitada o no, pero los demás bits cambian su significado según se indica en la Figura 13.

--	I7.5	I6.5	I5.5	WR	M7.5	M6.5	M5.5

Figura 13. Máscara de estado de interrupciones.

Los bits I7.5, I6.5 e I5.5 indican si hay pendientes interrupciones de cada uno de los tipos.

Hay que tener en cuenta que un reinicio del 8085, un RESET, pone a 0 los bits M5.5, M6.5 y M7.5, por lo que esas interrupciones estarán enmascaradas mientras no se modifique la máscara. Dicho de otra forma: en principio el 8085, tras un reinicio, no atenderá las interrupciones por las líneas RST 5.5, RST 6.5 y RST 7.5.

Las interrupciones tienen asociada una prioridad, de forma que cuando se producen simultáneamente el procesador siempre atiende en primer lugar a la de mayor prioridad. Las prioridades, de mayor a menor, son TRAP, RST 7.5, RST 6.5, RST 5.5 e INTR. Cada una de ellas tiene asociado un cierto vector en el mapa de memoria del sistema, de forma que al producirse, siempre que no estén deshabilitadas, el procesador transfiere el control a una dirección concreta. Las direcciones asociadas a TRAP, RST 5.5, RST 6.5 y RST 7.5 son 24H, 2CH, 34H y 3CH, respectivamente.

La interrupción INTR no está vectorizada, lo que significa que es necesario el uso de una instrucción RST que permita al procesador saber qué tiene que hacer. Cuando la interrupción recibida es del tipo RST X.5 o TRAP, por el contrario, se salta directamente al vector que corresponda y que se mencionaba antes.

Para determinar en qué dirección se encuentra el vector de una interrupción, tanto si ésta es del tipo `RST x.5` como si se trata de una instrucción `RST n`, basta con multiplicar el número que sigue a `RST` por 8. El vector de la `RST 6.5`, por ejemplo, se encuentra en la dirección `6.5*8=002CH`.

2 Programación del 8085

Para programar un microprocesador, en este caso el 8085, no es necesario saber cuántas patillas tiene, cuál es la función de cada una de ellas o si el bus de datos y direcciones está o no multiplexado. Toda esta información, sin embargo, hará que sea más fácil comprender cómo se ejecutan los programas y porqué el procesador hace lo que hace en cada momento.

Lo que sí resulta imprescindible es saber qué registros existen, cuál es su tamaño y función específica si existe; cuánta memoria puede direccionarse, cómo puede acceder a ella y cómo está estructurada en el sistema concreto en el que se trabaje y, obviamente, de qué instrucciones se dispone, junto con todos los detalles sobre su funcionamiento: qué operación efectúan, a qué registros afectan, cómo queda el procesador tras su ejecución, etc.

Aunque será en la segunda parte de este manual donde se describa instrucción por instrucción todo el repertorio del 8085, a medida que vayan poniéndose en práctica a través de los ejercicios propuestos, hay una serie de conceptos generales que es preciso conocer de antemano: el formato de las instrucciones, los modos de direccionar la memoria, las categorías en que se agrupan las instrucciones, los módulos de apoyo fundamentales y la configuración del sistema con el que se va a trabajar.

2.1 Formato de las instrucciones

Las instrucciones en ensamblador correspondientes a cualquier microprocesador representan siempre operaciones sencillas, elementales, cuando se les compara con las instrucciones de cualquier lenguaje de programación de alto nivel. Ésta es la razón de que el formato que siguen las instrucciones en ensamblador sea bastante homogéneo y simple. En el caso del 8085 existen tres posibilidades:

- `CodOp`: La instrucción se compone únicamente de un código de operación, sin precisar operandos, por lo que únicamente se precisa un byte.

- `CodOp Operando8`: Además del código de operación existe un operando de 8 bits preciso para que la instrucción pueda realizar su trabajo, por lo que ésta ocupará dos bytes.

- `CodOp Operando16`: Es una variante del caso anterior en el que el operando es de 16 bits y no de 8, de forma que la instrucción completa ocupará tres bytes.

El microprocesador, cuando ejecuta un programa, lee el byte apuntado por el `PC` e incrementa este último, tras lo cual se procede a la descodificación del código de operación. Dependiendo de dicho código la `UC` procederá a ordenar la lectura de uno o dos bytes adicionales, para recuperar el operando en caso de que fuese necesario, actualizando el registro `PC` como es lógico. Una vez que la instrucción al completo, incluyendo operandos, se encuentra dentro del micro se procede a su ejecución.

Desde un punto de vista exclusivamente sintáctico, las líneas de un programa en ensamblador se ajustarán al formato siguiente:

```
[etiqueta:]  Instrucción {Op1}{,Op2}  [; comentario]
```

Los elementos entre corchetes son opcionales, mientras que los dispuestos entre llaves aparecerán o no dependiendo de la instrucción que esté utilizándose. De izquierda a derecha su finalidad es la siguiente:

- Las etiquetas se colocan en los puntos del programa a los que se precise desviar la ejecución en algún momento, casos típicos son los bucles y las subrutinas. La etiqueta comenzará por una letra, seguida de una o más letras y números, y terminará con dos puntos. No es recomendable que la longitud de la etiqueta exceda de seis caracteres en total.

- Mediante la instrucción se indica la operación que quiere llevarse a cabo. La mayoría de las instrucciones del 8085 tienen 2, 3 o 4 caracteres de longitud y suelen ser abreviaciones o acrónimos de la operación asociada, por ejemplo `EI` (*Enable Interrupt*) para activar las interrupciones o `RET` (*Return*) para volver de una subrutina.

- Si hay un único operando éste aparecerá justo detrás de la instrucción, separado por uno o más espacios. De existir dos, se utilizará una coma para separar el primero del segundo.

- Tras la instrucción y los operandos, así como en cualquier línea vacía, es posible introducir comentarios que serán ignorados por el ensamblador. El comentario se inicia con el carácter `;` tras el cual puede escribirse cualquier texto hasta el final de la línea.

Los operandos, en caso de que aparezcan, pueden ser de los tipos siguientes:

- $\texttt{Reg}_8$: Un registro de 8 bits: `A`, `B`, `C`, `D`, `E`, `H`, `L` o `M`. Pueden existir limitaciones. La instrucción `MOV`, por ejemplo, precisa dos operandos y permite cualquier combinación de registros de 8 bits salvo en el caso de `M`, que como máximo puede aparecer una vez.

- $\texttt{Reg}_{16}$: Un registro de 16 bits: `B` (corresponde a `BC`), `D` (corresponde a DE), `H` (corresponde a `HL`) o `SP`.

- $\texttt{D}_8$: Un dato inmediato de 8 bits, como puede ser un número entre 0 y 255 o un carácter entrecomillado.

- D_{16}: Un dato inmediato de 16 bits, como puede ser un número entre 0 y 65535.

- P_8: Un número de puerto de E/S, que será un número entre 0 y 255.

- A_{16}: Una dirección de memoria de 16 bits.

Las combinaciones válidas de operandos dependerán, como ya se ha dicho, de la instrucción concreta a la que vayan asociados.

2.2 Modos de direccionamiento

Las distintas instrucciones con que cuenta el 8085 pueden clasificarse siguiendo distintos criterios. Uno de ellos las agrupa según la forma en que acceden a la información sobre la que van a operar, lo que se denomina habitualmente modo de direccionamiento, información que puede estar en el propio procesador o bien en la memoria del sistema.

Conocer los modos de direccionamiento que ofrece un microprocesador es importante a la hora de programar en ensamblador, ya que de ellos dependerá que para realizar una cierta tarea haya que trasladar primero la información a registros o bien almacenar en parejas de registros la dirección donde está esa información. Hay que tener en cuenta que, en general, cualquier operación que implique uno o más accesos a memoria resultará mucho más lenta que la equivalente operando sobre registros internos del microprocesador, por lo que la elección de los modos de direccionamiento influirá también de forma importante en el rendimiento obtenido. Los modos posibles son:

- **Por registro**: Hay una gran cantidad de instrucciones que toman un operando que puede ser un registro, de 8 o 16 bits según los casos, utilizando la información que almacena dicho registro y, en ocasiones, el de otro que se asume implícitamente. Un ejemplo es la instrucción `ADD R`, pudiendo ser `R` el registro `B`, `C`, `D`, `E`, `H` o `L`. Estas instrucciones suelen ocupar un único byte, ya que el código de operación cambia según el registro que se utilice como operando. Cuando el único operando de la instrucción es el acumulador éste se encuentra implícito, por ejemplo en instrucciones como `DAA` y `CMA`, por lo que no es preciso indicarlo.

En algunos textos sobre el microprocesador 8085, y otros de la misma familia, es posible encontrar referencias al *direccionamiento implícito* como un modo más, independiente del *direccionamiento por registro*.

Debe tenerse en cuenta que el direccionamiento implícito no es más que un direccionamiento por registro, la información sobre la que operará la instrucción se encuentra

en un registro del 8085, con la única diferencia de que la instrucción conoce de antemano de qué registro se trata, por lo que no es preciso indicarlo explícitamente. En este texto se ha seguido la clasificación que aparecen en diferentes libros del profesor J. M. Angulo.

- **Inmediato**: El direccionamiento inmediato corresponde a instrucciones que siempre toman uno o dos operandos y uno de ellos es un dato inmediato, lo que en otros lenguajes suele denominarse *constante literal*. Ese dato es incluido tras el código de operación, por lo que la instrucción pasa a ocupar 2 o 3 bytes, dependiendo del tamaño del operando. Un ejemplo sería `LXI SP,Dir`, instrucción en la que actúan como operando el registro `SP` y un dato inmediato de 16 bits que sería la dirección a cargar en dicho registro. Otro caso, más simple, sería la instrucción `ADI 5`, en la que uno de los operandos está implícito, es el acumulador, y el otro es un dato inmediato de 8 bits: el número 5.

- **Directo**: La instrucción incluye como operando la dirección de memoria donde se encuentra el dato que se utilizará en la operación, ocupando por tanto un total de tres bytes. Un ejemplo podría ser `STA 2010`, instrucción que almacena en el acumulador el contenido de la celdilla de memoria situada en la dirección `2010`.

- **Indirecto por pareja de registros**: Son instrucciones que actúan o utilizan un dato alojado en la memoria del sistema, obteniendo la dirección de una pareja de registros en lugar de directamente de la propia instrucción. El caso más típico es el de las instrucciones en las que aparece el registro `M`, como puede ser `MOV M,B`, ya que dicho registro en realidad hace referencia a la celdilla de memoria cuya dirección indica el par de registros `HL`. Otro ejemplo sería la instrucción `LDAX B`, en la que se utiliza el contenido de `BC` como puntero a para acceder a la memoria, recuperar el contenido de esa dirección y llevarlo hasta el acumulador. Son instrucciones que suelen ocupar solamente un byte, ya que no la dirección no se facilita como dato inmediato.

En ocasiones, la ejecución de una única instrucción de un programa puede conllevar múltiples operaciones más simples, en las que se recurrirá al uso de varios modos de direccionamiento sin que el programador, en principio, intervenga en ningún sentido. La Figura 14 representa esquemáticamente el proceso de ejecución de una subrutina desde un programa, con los pasos que desencadenaría la ejecución de la instrucción `CALL 2010H` y posteriormente de la instrucción `RET`.

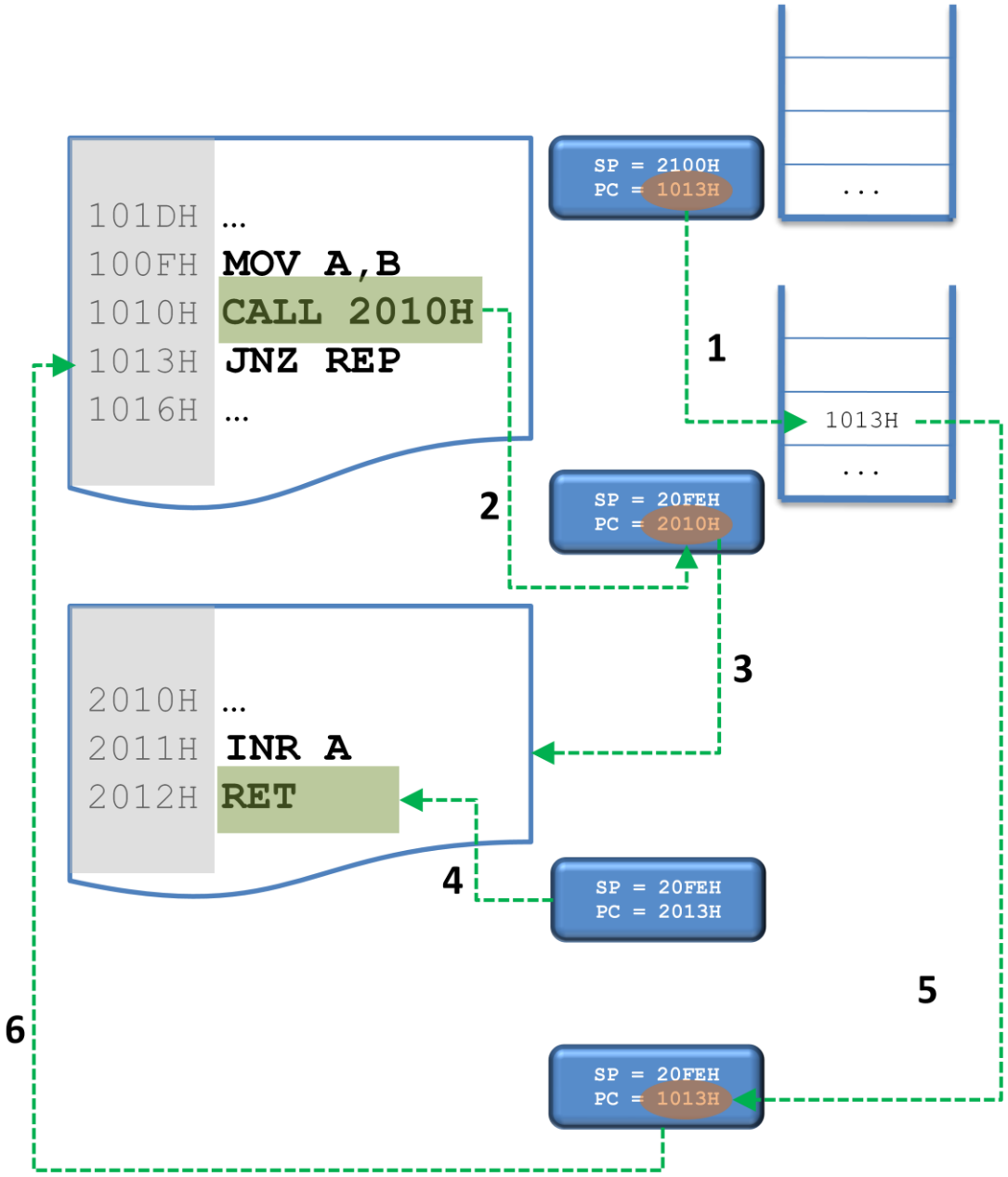


Figura 14. Esquema de la ejecución de una subrutina.

Las operaciones que se llevan a cabo para ejecutar la instrucción `CALL`, según la numeración del esquema, son las siguientes:

1. El primer paso consiste en almacenar el valor actual del `PC` en el tope de la pila, cuya dirección viene dada por el registro `SP`, por lo que se utiliza direccionamiento indirecto por registro para efectuar la operación y también implícito, ya que ni `SP` ni `PC` aparecen en la instrucción.

2. A continuación se toma el dato inmediato facilitado en la instrucción, que en este ejemplo sería `2010H`, y se lleva al registro `PC`, mediante direccionamiento inmediato.

3. La modificación del contador de programa desencadena la ejecución de la subrutina. En ese momento finaliza la ejecución de la instrucción `CALL`, que habrá alterado el flujo de ejecución del programa saltando a una subrutina tras guardar la dirección a la que de retornarse.

4. Cuando se llega a la última instrucción de la subrutina, la instrucción `RET`, el registro `SP` contiene la dirección desde la que debe recuperarse la dirección de retorno.

5. Se extrae de la pila la dirección de retorno y se lleva de nuevo hasta el `PC`, usando nuevamente direccionamiento indirecto por registro.

6. Finalmente el cambio de valor del contador de programa provoca que se vuelva al programa principal, a la instrucción siguiente a `CALL`.

Una peculiaridad del 8085 es el hecho de que ciertas instrucciones que desempeñan la misma función cambian su nombre según el modo de direccionamiento que se utilice, lo cual no ocurre en la mayoría de los microprocesadores. Las instrucciones `ADD` y `ADI`, por ejemplo, llevan a cabo una suma sobre el acumulador, con la única diferencia de que la primera se aplica a direccionamiento por registro o indirecto por registro (si se usa `M` como argumento) mientras que la segunda siempre se asocia al direccionamiento inmediato.

2.3 Tabla de instrucciones

Las instrucciones con que cuenta un microprocesador, en este caso el 8085, son construcciones por lo general muy sencillas y, además, el conjunto total de instrucciones es bastante reducido. La mejor manera de saber qué hace cada una de esas instrucciones, cómo lo hace y cuál es el resultado consiste en escribir programas en que se utilicen, como los propuestos en los ejercicios de la segunda parte.

Lo que se facilita en la Tabla 3 es una referencia completa de las instrucciones del 8085, información que resulta útil a la hora de escribir programas para saber qué operandos pueden utilizarse en una instrucción, a qué bits del registro de estados utilizará, etc. En la tabla existen cuatro columnas cuyo significado es el siguiente:

Instrucción: Es la primera columna e indica cuál es el formato de la instrucción, incluyendo los operandos que necesita, acompañándola de una breve explicación y, cuando procede, una representación simbólica de la operación que lleva a cabo. En la indicación del formato de la instrucción, que ocupa la primera línea, se utiliza la letra `R` para representar a un registro, la letra `D` para indicar un dato y la letra `n` para representar un número. Como subíndice aparece el número de bits del operando que, según los casos, será de 8 o 16 bits. Cuando el operando es una dirección ésta aparece como `Dir`. El símbolo ← indica que el contenido del operando que hay a la derecha se copia

en el operando de la izquierda, mientras que el símbolo ↔ se usa para representar un intercambio del contenido de los operandos. Cuando en la operación aparece un registro como origen o destino, por ejemplo A ←R, lo que se toma es, lógicamente, el contenido del registro o se asigna como contenido del registro. Si el operando va entre corchetes, por ejemplo R ← [HL], debe entenderse que el contenido del registro va a actuar como un puntero, leyéndose o escribiéndose en la posición de memoria cuya dirección indica el registro.

Cód. op.: La segunda columna facilita el código de operación (en hexadecimal) que corresponde a la instrucción. En caso de que ese código varíe según los operandos, algo bastante habitual, se indica en forma de lista el código que correspondería dependiendo de los operandos.

Estado: Cuando se ejecuta una instrucción suele interesar saber qué bits del registro de estado quedan afectados, siendo ésta la información de la tercera columna. Las letras SZAPC representan a los bits S, Z, AC, P y CY, apareciendo debajo de cada una de ellas un − si el bit no resulta afectado, una X si la instrucción lo altera, un 1 si la instrucción lo pone a 1 y un 0 si queda a 0.

Direcc.: La cuarta y última columna indica qué tipo de direccionamiento usa la instrucción, recurriendo para ello las abreviaturas siguientes: R=Registro, D=Directo, PRI=Par de registros indirecto, I=Inmediato, RI=Registro indirecto e IR=Registro implícito. Esta columna puede quedar vacía en caso de que la instrucción no implique ningún acceso a la memoria ni movimiento de datos, por ejemplo en los casos en que únicamente se altera el contenido de un bit del registro de estado o se modifica el estado interno del microprocesador.

Instrucción	Cód. op.	Estado	Direcc.
Transferencia de datos			
MOV $R1_8$,$R2_8$ **R1 ←R2** **El contenido del registro R2 se copia en el registro R1.**	R_1 R_2 Cód A A 7FH A B 78H A C 79H A D 7AH A E 7BH A H 7CH A L 7DH B A 47H B B 40H B C 41H B D 42H B E 43H B H 44H B L 45H C A 4FH C B 48H C C 49H C D 4AH C E 4BH R_1 R_2 Cód D A 57H D B 50H D C 51H D D 52H D E 53H D H 54H D L 55H E A 5FH E B 58H E C 59H E D 5AH E E 5BH E H 5CH E L 5DH H A 67H H B 60H H C 61H H D 62H H E 63H	**SZAPC** -----	**R**

Instrucción	Cód. op.	Estado	Direcc.
	C H 4CH H H 64H C L 4DH H L 65H L A 6FH L B 68H L C 69H L D 6AH L E 6BH L H 6CH L L 6DH		
MOV R_8,M **R ←[HL]** El contenido de la posición de memoria apuntada por **HL** se copia en el registro R	**R Cód** / A 7EH / C 4EH / E 5EH / L 6EH **R Cód** / B 46H / D 56H / H 66H	**SZAPC** -----	PRI
MOV M,R_8 **[HL] ←R** El contenido del registro R se copia a la posición de memoria apuntada por HL	**R Cód** / A 77H / C 71H / E 73H / L 75H **R Cód** / B 70H / D 72H / H 74H	**SZAPC** -----	PRI
MVI R_8,D_8 **R ←D_8** El dato de 8 bits se introduce en el registro R	**R Cód** / A 3EH / C 0EH / E 1EH / L 2EH **R Cód** / B 06H / D 16H / H 26H	**SZAPC** -----	I
MVI M,D_8 **[HL] ←D_8** El dato de 8 bits se introduce en la posición de memoria apuntada por HL	36H	**SZAPC** -----	I/PRI
LXI R_{16},D_{16} **R ←D_{16}** El dato de 16 bits se introduce en la paraja de registros R	**R Cód** / B 01H / H 21H **R Cód** / D 11H / SP 31H	**SZAPC** -----	I
LDA Dir **A ←[Dir]** El contenido de la dirección de memoria indicada se copia en el acumulador	3AH	**SZAPC** -----	D

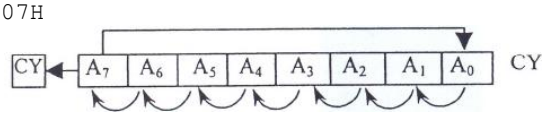
Instrucción	Cód. op.	Estado	Direcc.
STA Dir **[Dir] ←A** Almacena en la dirección de memoria indicada el contenido del acumulador	32H	**SZAPC** -----	D
LHLD Dir **L ←[Dir]** **H←[Dir+1]** El contenido de la dirección de memoria indicada se lleva al registro L y el de la posición siguiente al registro H	2AH	**SZAPC** -----	D
SHLD Dir **[Dir] ←L** **[Dir+1]←H** El contenido de L se almacena en la dirección de memoria indicada y el registro H en la posición siguiente	22H	**SZAPC** -----	D
LDAX R_{16} **A ←[R]** El contenido de la dirección de memoria indicada por la pareja de registros R se lleva al acumulador	**R** **Cód** B 0AH D 1AH	**SZAPC** -----	PRI
STAX R_{16} **[R] ←A** El contenido del acumulador se almacena en la dirección de memoria indicada por la pareja de registros R	**R** **Cód** B 02H D 12H	**SZAPC** -----	PRI
XCHG **H ↔D** **L ↔ E** Se intercambia el contenido de las parejas de registros **HL** y **DE**	EBH	**SZAPC** -----	R
Cálculos aritméticos			

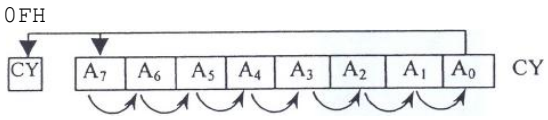
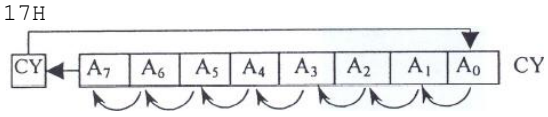
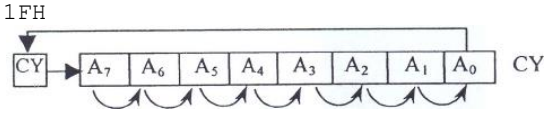
Instrucción	Cód. op.	Estado	Direcc.
ADD R_8 **A ←A + R** Se suma al contenido del acumulador el contenido del registro **R**, el resultado queda en A	**R** Cód A 87H C 81H E 83H L 85H **R** Cód B 80H D 82H H 84H	**SZAPC** XXXXX	R
ADD M **A ←A + [HL]** Se suma al contenido del acumulador el contenido de la posición de memoria apuntada por HL, el resultado queda en A	86H	**SZAPC** XXXXX	PRI
ADI D_8 **A ←A + D_8** Se suma al contenido del acumulador el dato inmediato y el resultado queda en A	C6H	**SZAPC** XXXXX	I
ADC R_8 **A ←A + R + CY** Se suma al contenido del acumulador el contenido del registro R y el indicador de arrastre, el resultado queda en A	**R** Cód A 8FH C 89H E 8BH L 8DH **R** Cód B 88H D 8AH H 8CH	**SZAPC** XXXXX	R
ADC M **A ←A + [HL] + CY** Se suma al contenido del acumulador el contenido de la posición de memoria apuntada por HL y el indicador de arrastre, el resultado queda en A	8EH	**SZAPC** XXXXX	PRI
ACI D_8 **A ←A + D_8 + CY** Se suma al contenido del acumulador el dato inmediato y el indicador de arrastre, el resultado queda en A	CEH	**SZAPC** XXXXX	I

Instrucción	Cód. op.	Estado	Direcc.
SUB R_8 **A ←A - R** Se resta del contenido del acumulador el contenido del registro R, el resultado queda en A	**R** **Cód** A 97H C 91H E 93H L 95H **R** **Cód** B 90H D 92H H 94H	**SZAPC** XXXXX	R
SUB M **A ←A - [HL]** Se resta del contenido del acumulador el contenido de la posición de memoria apuntada por HL, el resultado queda en A	96H	**SZAPC** XXXXX	PRI
SUI D_8 **A ←A - D_8** Se resta del contenido del acumulador el dato inmediato y el resultado queda en A	D6H	**SZAPC** XXXXX	I
SBB R_8 **A ←A - R - CY** Se resta del contenido del acumulador el contenido del registro R y el indicador de arrastre, el resultado queda en A	**R** **Cód** A 9FH C 99H E 9BH L 9DH **R** **Cód** B 98H D 9AH H 9CH	**SZAPC** XXXXX	R
SBB M **A ←A - [HL] - CY** Se resta del contenido del acumulador el contenido de la posición de memoria apuntada por HL y el indicador de arrastre, el resultado queda en A	9EH	**SZAPC** XXXXX	PRI
SBI D_8 **A ←A - D_8 - CY** Se resta del contenido del acumulador el dato inmediato y el indicador de arrastre, el resultado queda en A	DEH	**SZAPC** XXXXX	I

Instrucción	Cód. op.	Estado	Direcc.
INR R_8 **R ← R + 1** Se incrementa en una unidad el contenido del registro R	**R Cód** / A 3CH / C 0CH / E 1CH / L 2CH / B 01H / D 14H / H 24H	**SZAPC** XXXX-	R
INR M **[HL] ←[HL] + 1** Se incrementa en una unidad el contenido de la posición de memoria apuntada por HL	34H	**SZAPC** XXXX-	PRI
INX R_{16} **R ← R + 1** Se incrementa en una unidad el contenido de la pareja de registros R_{16}	**R Cód** / B 03H / H 23H / D 13H / SP 33H	**SZAPC** -----	R
DCR R_8 **R ← R - 1** Se reduce en una unidad el contenido del registro R	**R Cód** / A 3DH / C 0DH / E 1DH / L 2DH / B 05H / D 15H / H 25H	**SZAPC** XXXX-	R
DCR M **[HL] ←[HL] - 1** Se reduce en una unidad el contenido de la posición de memoria apuntada por HL	35H	**SZAPC** XXXX-	PRI
DCX R_{16} **R ← R -1 1** Se reduce en una unidad el contenido de la pareja de registros R_{16}	**R Cód** / B 08H / H 28H / D 18H / SP 38H	**SZAPC** -----	R
DAD R_{16} **HL ←HL + R** Se suma a HL el contenido de la pareja de registros R, el resultado queda en HL	**R Cód** / B 09H / H 29H / D 19H / SP 39H	**SZAPC** ----X	R
DAA	27H	**SZAPC**	IR

Instrucción	Cód. op.	Estado	Direcc.
A ←Ajuste decimal A Efectúa un ajuste decimal del contenido de **A**, formando dos dígitos BCD		XXXXX	
Operaciones lógicas			
ANA R_8 **A ← A & R** Se hace el **AND** lógico entre el acumulador y **R**, el resultado queda en **A**	**R** **Cód** A A7H C A1H E A3H L A5H **R** **Cód** B A0H D A2H H A4H	**SZAPC** XXXX0	R
ANA M **A ←A & [HL]** Se hace el **AND** lógico entre el acumulador y el contenido de la posición de memoria apuntada por HL	A6H	**SZAPC** XXXX0	PRI
ANI D_8 **A ←A & D** Se hace el AND lógico entre el acumulador y el dato inmediato	E6H	**SZAPC** XX0X0	I
XRA R_8 **A ← A ⊕ R** Se hace el XOR lógico entre el acumulador y R, el resultado queda en A	**R** **Cód** A AFH C A9H E ABH L ADH **R** **Cód** B A8H D AAH H ACH	**SZAPC** XX0X0	R
XRA M **A ←A ⊕ [HL]** Se hace el XOR lógico entre el acumulador y el contenido de la posición de memoria apuntada por HL	AEH	**SZAPC** XX0X0	PRI
XRI D_8 **A ←A ⊕ D** Se hace el XOR lógico entre el acumulador y el dato inmediato	EEH	**SZAPC** XX0X0	I

Instrucción	Cód. op.	Estado	Direcc.
ORA R_8 **A ← A ^ R** Se hace el OR lógico entre el acumulador y R, el resultado queda en A	**R** **Cód** A B7H C B1H E B3H L B5H **R** **Cód** B B0H D B2H H B4H	**SZAPC** XX0X0	R
ORA M **A ←A ^ [HL]** Se hace el OR lógico entre el acumulador y el contenido de la posición de memoria apuntada por HL	B6H	**SZAPC** XX0X0	PRI
ORI D_8 **A ←A ^ D** Se hace el OR lógico entre el acumulador y el dato inmediato	F6H	**SZAPC** XX0X0	I
CMP R_8 Compara el contenido del acumulador, sin llegar a modificarlo, con el del registro R	**R** **Cód** A BFH C B9H E BBH L BDH **R** **Cód** B B8H D BAH H BCH	**SZAPC** XXXXX A=R⇒Z=1 A<R⇒CY=1	R
CMP M Compara el contenido del acumulador, sin llegar a modificarlo, con el contenido de la posición de memoria apuntada por HL	BEH	**SZAPC** XXXXX A=[HL]⇒ Z=1 A<[HL]⇒ CY=1	PRI
CPI D_8 Compara el contenido del acumulador, sin modificarlo, con el dato inmediato	FEH	**SZAPC** XXXXX A=D⇒Z=1 A<D⇒CY=1	I
RLC **A_{n+1} ←A_n** **A_0 ← A_7** **CY ← A_7** **Rotación a** la izquierda de los	07H  	**SZAPC** ----X	IR

Instrucción	Cód. op.	Estado	Direcc.
bits del acumulador			
RRC **$A_n \leftarrow A_{n+1}$** **$A_7 \leftarrow A_0$** **CY ← A_0** Rotación a la derecha de los bits del acumulador	0FH 	**SZAPC** ----X	IR
RAL **$A_{n+1} \leftarrow A_n$** **CY ← A_7** **A_0 ← CY** Rotación a la izquierda a través del ***carry*** de los bits del acumulador	17H 	**SZAPC** ----X	IR
RAR **$A_n \leftarrow A_{n+1}$** **CY ← A_0** **A_7 ← CY** Rotación a la derecha a través del *carry* de los bits del acumulador	1FH 	**SZAPC** ----X	IR
CMA **A ←~A** Complemento del contenido del acumulador	2FH	**SZAPC** -----	IR
CMC **CY ←~CY** Complemento del bit **CY** del registro de estado	3FH	**SZAPC** ----X	
STC **CY ←1** Pone a 1 el bit CY del registro de estado	37H	**SZAPC** ----1	
Control de flujo del programa			
JMP Dir	C3H	**SZAPC** -----	I

Instrucción	Cód. op.	Estado	Direcc.
PC ←Dir Carga en el contador de programa la dirección (salto incondicional)			
J_{cond} Dir **PC ←Dir** Si se cumple la condición, que un bit del registro de estado esté a 0 ó 1, carga en el contador de programa la dirección indicada	**Condición** / **Instrucción** / **Cód** CY=1 / JC / DAH S=1 / JM / FAH CY=0 / JNC / D2H Z=0 / JNZ / C2H S=0 / JP / F2H P=1 / JPE / EAH P=0 / JPO / E2H Z=1 / JZ / CAH	**SZAPC** -----	I
CALL Dir **[SP-1] ←PCH** **[SP-2] ←PCL** **SP ← SP - 2** **PC ←Dir** Guarda en la pila el actual valor del PC y carga en éste la dirección de una subrutina	CDH	**SZAPC** -----	I/RI
C_{cond} Dir **[SP-1] ←PCH** **[SP-2] ←PCL** **SP ← SP - 2** **PC ←Dir** Si se cumple la condición, que un bit del registro de estado esté a 0 ó 1, guarda en la pila el valor del PC y carga en éste la dirección	**Condición** / **Instrucción** / **Cód** CY=1 / CC / DCH S=1 / CM / FCH CY=0 / CNC / D4H Z=0 / CNZ / C4H S=0 / CP / F4H P=1 / CPE / ECH P=0 / CPO / E4H Z=1 / CZ / CCH	**SZAPC** -----	I/RI
RET **PCL ←[SP]** **PCH ←[SP+1]** **SP ← SP + 2** Recupera la última dirección guardada en la pila y la carga en el PC	C9H	**SZAPC** -----	RI
R_{cond} **PCL ←[SP]** **PCH ←[SP+1]** **SP ← SP + 2** Si se cumple la condición, que un bit del registro de estado esté a 0 ó 1, actúa	**Condición** / **Instrucción** / **Cód** CY=1 / RC / D8H S=1 / RM / F8H CY=0 / RNC / D0H Z=0 / RNZ / C0H S=0 / RP / F0H P=1 / RPE / E8H P=0 / RPO / E0H	**SZAPC** -----	RI

Instrucción	Cód. op.	Estado	Direcc.
como la instrucción `RET`	`Z=1` `RZ` `C8H`		
`RST n` **[SP-1] ←PCH** **[SP-2] ←PCL** **SP ← SP - 2** **PC ←8 * n** Guarda en la pila el actual valor del PC y carga en éste la dirección de un vector de interrupción software	**n** / **Cód** 0 C7H 1 CFH 2 D7H 3 DFH 4 E7H 5 EFH 6 F7H 7 FFH	**SZAPC** -----	RI
Pila			
`PUSH` R_{16} **[SP-1] ←RH** **[SP-2] ←RL** **SP ← SP - 2** Guarda en la pila el contenido de la pareja de registros indicada	**R** **Cód**: B C5H, H E5H; **R** **Cód**: D D5H	**SZAPC** -----	RI
`POP` R_{16} **RL ←[SP]** **RH ←[SP+1]** **SP ← SP + 2** Se recupera el último dato introducido en la pila y se almacena en la pareja de registros indicada	**R** **Cód**: B C1H, H E1H; **R** **Cód**: D D1H	**SZAPC** -----	RI
`PUSH PSW` **[SP-1] ←A** **[SP-2] ←F** **SP ← SP - 2** Guarda en la pila el contenido del acumulador y el registro de estado	`F5H`	**SZAPC** -----	RI
`POP PSW` **F ←[SP]** **A ←[SP+1]** **SP ← SP + 2** Se recupera el último dato introducido en la pila y se almacena en el acumulador y el registro de stado	`F1H`	**SZAPC** -----	RI

Instrucción	Cód. op.	Estado	Direcc.
XTHL **L ↔ [SP]** **H ↔ [SP+1]** Se intercambia el contenido de la pareja de registros **HL** con las dos posiciones superiores de la pila	E3H	**SZAPC** -----	RI
SPHL **SP ← HL** Copia el contenido de la pareja de registros **HL** en el puntero de pila	F9H	**SZAPC** -----	RD
E/S y control			
PCHL **PC ← HL** Copia el contenido de la pareja de registros HL en el contador de programa, lo que equivale a saltar a la dirección indicada por **HL**	E9H	**SZAPC** -----	R
IN n_8 **A ← [n]** Lee del puerto de E/S **n** un byte y lo deposita en el acumulador	DBH	**SZAPC** -----	R
OUT n_8 **[n] ← A** Se envía por el puerto de E/S n el valor contenido en el acumulador	D3H	**SZAPC** -----	R
EI Habilita las interrupciones, de forma que la CPU responda a solicitudes de este tipo	FBH	**SZAPC** -----	
DI Deshabilita las interrupciones, de forma que la CPU no	F3H	**SZAPC** -----	

Instrucción	Cód. op.	Estado	Direcc.
responda a solicitudes de este tipo			
HLT Detiene la CPU, poniéndola en un estado del que saldrá a producirse una interrupción	76H	**SZAPC** _____	
NOP Emplea cuatro ciclos de CPU sin hacer nada	00H	**SZAPC** _____	
RIM **A ← MI** Lee la máscara de interrup-ciones y la almacena en el acumulador	20H	**SZAPC** _____	
SIM **MI ← A** Escribe el contenido del acumulador en la máscara de interrupciones	30H	**SZAPC** _____	

Tabla 3. Juego de instrucciones del 8085.

Conociendo el formato de las instrucciones del 8085, los modos de direccionamiento y teniendo una idea general sobre las operaciones que pueden llevarse a cabo, con la breve descripción aportada en esta tabla, se tiene prácticamente todo lo necesario para comenzar a trabajar con un sistema basado en este microprocesador. En dicho sistema, que es el uP-2000, existen una serie de módulos hardware adicionales, de los cuales dos son fundamentales: el PPI 8255 y el controlador de teclado y pantalla 8279.

3 El módulo de E/S PPI 8255

Este integrado, conocido genéricamente como PPI (*Programmable Peripherial Interface*), permite al microprocesador comunicarse con dispositivos externos a través de 24 líneas en paralelo que pueden configurarse de distintas formas. El encapsulado, como se aprecia en la Figura 15, es el habitual de 40 patillas.

Figura 15. El integrado 8255A de Intel.

En la Figura 16 se indica el nombre de cada una de las patillas, nombres a partir de los cuales es fácil deducir la forma en que se agrupan.

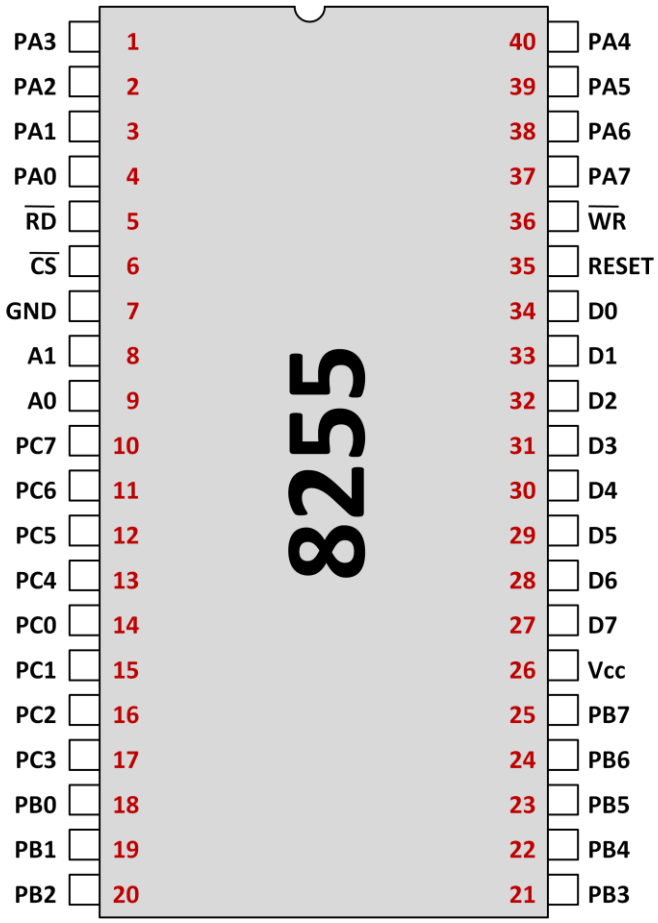


Figura 16. Patillaje del integrado 8255.

Desde un punto de vista lógico, 24 de las 40 líneas se unen en tres grupos de 8 a los que se denomina puertos A, B y C. Son las patillas PA_n, PB_n y PC_n. En el caso del puerto C, los ocho bits pueden tratarse como dos partes independientes de 4 bits cada una. El 8255 contempla tres modos de funcionamiento diferentes que, de manera simplificada, serían los descritos a continuación:

- **Modo 0**: Los tres puertos se utilizan como entrada o salida de datos, sin ningún control sobre la comunicación con los dispositivos. Es el modo más simple de funcionamiento del PPI.

- **Modo 1**: Los puertos A y B se pueden usar como entrada o salida, repartiéndose el puerto C en 4 bits para control de la comunicación por el puerto A y otros 4 bits para el control por el puerto B.

- **Modo 2**: Este modo permite que el puerto A funcione de manera bidireccional, es decir, como entrada y salida simultáneamente, apoyándose para ello en ciertos bits del puerto C que se usan para controlar el diálogo con los dispositivos externos. El puerto B se puede configurar para operar en modo 0 ó 1, con independencia del puerto A.

Estas 24 patillas se conectan desde el 8255 hacia los dispositivos externos. En un sistema completo podrían utilizarse para conectar al ordenador una impresora, un teclado u otros periféricos de comunicación en paralelo.

En el sistema uP-2000 el 8255 tiene sus tres puertos de E/S acoplados a tres conectores externos, denominados J2, J3 y J4, a los que pueden conectarse dispositivos como la tarjeta de pulsadores, microinterruptores y leds empleados en las prácticas más avanzadas. Por otra parte el integrado se conecta con el resto del sistema a través del bus de datos, que el 8085 usará para enviar o leer datos al 8255; el bus de direcciones, usado para seleccionar el puerto a leer o escribir, y el bus de control, que será el que active las líneas de lectura o escritura.

La Figura 17 muestra gráficamente la conexión del integrado 8255 en el sistema uP-2000, destacando los conectores externos y las patillas que van a los buses de datos y direcciones.

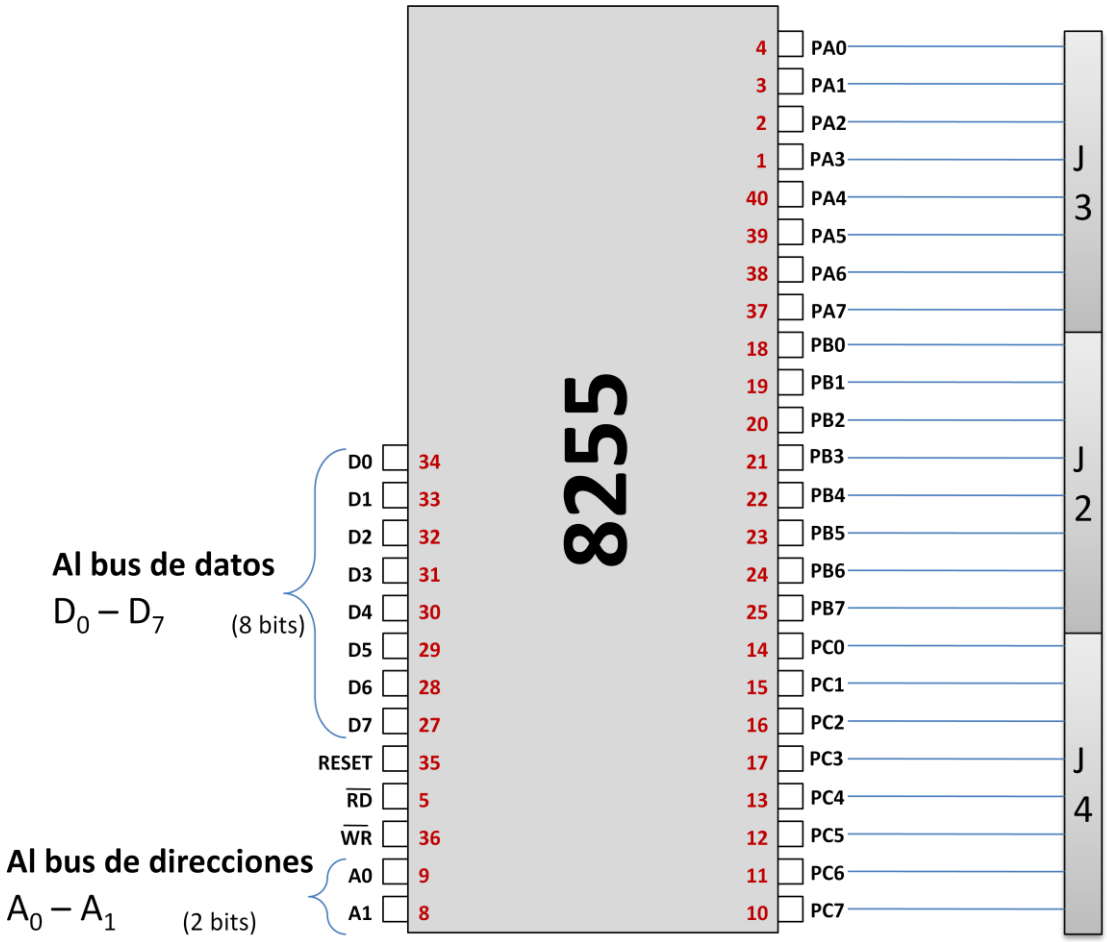


Figura 17. Conexión del 8255 en el sistema uP-2000.

De estas patillas las del margen izquierdo son las que permiten al microprocesador comunicarse con el PPI para programarlo, enviar y recibir datos. Por una parte están los pines A_0 y A_1, dos bits mediante los que el microprocesador selecciona el puerto donde se escribirá o del que se leerá: 00 - Puerto A, 01 - Puerto B, 10 - Puerto C y 11 - Registro de control. Hecha la selección, se usarán los pines D_0 a D_7 para enviar o recuperar los 8 bits de datos o configuración, según el estado de las patillas RD y WR que son las que establecen si se va a leer o a escribir.

Para establecer la configuración del PPI se recurre a un registro de control interno, cuya estructura es la indicada en la Figura 18.

C7	C6	C5	C4	C3	C2	C1	C0
CFG	Modo A	Modo A	PA	PCH	Modo B	PB	PCL

Figura 18. Registro de control del 8255.

La finalidad de cada bit o conjunto de bits es la siguiente:

- **C7**: Determina el tipo de configuración que va a efectuarse. Puesto a 1 permite establecer el modo y dirección de comunicación de cada puerto, con el patrón

de bits anterior. Si se pone a 0 facilita la configuración individual de cada uno de los bits del puerto C.

- **C6** y **C5**: Estos dos bits establecen el modo de funcionamiento para el puerto A: 00 - Modo 0, 01 - Modo 1 y 1X - Modo 2.

- **C4**: Configura el puerto A para: 0 - Salida, 1 - Entrada.

- **C3**: Configura los cuatros bits de mayor peso del puerto C (PC7-PC4) para: 0 - Salida, 1 - Entrada.

- **C2**: Este bit establece el modo de funcionamiento para el puerto B más la parte baja del puerto C: 0 - Modo 0, 1 - Modo 1.

- **C1**: Configura el puerto B para: 0 - Salida, 1 - Entrada.

- **C0**: Configura los cuatro bits de menor peso del puerto C (PC3-PC0) para: 0 - Salida, 1 - Entrada.

Con toda esta información, la secuencia habitual de trabajo con el PPI desde el punto de vista del programador sería la siguiente:

1. Configuración del PPI mediante el registro de control, estableciendo el modo de funcionamiento y sentido de la comunicación. Para ello se pondría un 11 en las patillas A0-A1 y se enviaría por D0-D7 el byte con la configuración, según los bits descritos antes.

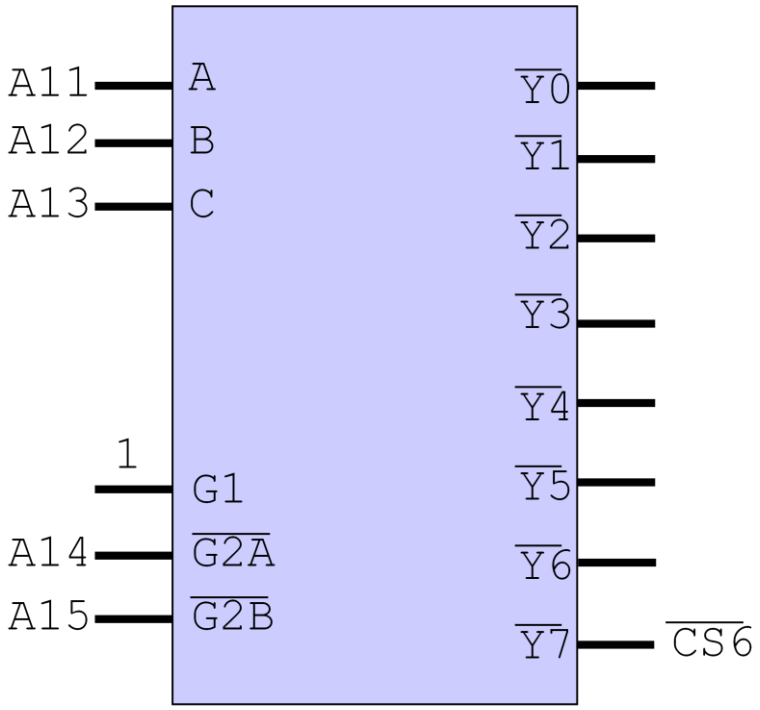
2. Selección mediante A0-A1 del registro del que se va a leer o donde se va a escribir.

3. Envío o lectura por D0-D7 del byte de datos.

En realidad para el programador la selección de registro mediante las patillas A0-A1 se hará automáticamente según la dirección a la que acceda y la configuración del sistema.

3.1 Selección del 8255 en el sistema uP-2000

En el uP-2000 el PPI tiene asignado el rango de direcciones que va de 3800h a 3FFFh. El patrón que activará la señal CS del 8255 se encuentra en los 5 bits de mayor peso, A15-A11, que será 00111. Los dos bits de menor peso, A1-A0, permiten elegir uno de los puertos o el registro de control. El resto de los bits no importan, por lo que es posible acceder a cada uno de los puertos y el registro de control con multitud de direcciones en el rango citado. La Figura 19 representa esquemáticamente la lógica de selección del 8255.



A15-A11: 00111 → Y7

Figura 19. Lógica de selección del 8255 en el sistema uP-2000.

En el rango de direcciones de E/S, a utilizar con las instrucciones `IN` y `OUT` del 8085, la situación de los puertos y registro de control del PPI serían los siguientes:

- **38H**: Puerto A.
- **39H**: Puerto B.
- **3AH**: Puerto C.
- **3BH**: Registro de control.

4 El módulo 8279

El 8279 es un integrado de Intel que actúa como interfaz programable de teclado y pantalla para microprocesadores de 8 bits, especialmente los de Intel como el 8085 y 8080. Este integrado se conecta directamente al bus del sistema y libera al microprocesador de las tareas de inspección del teclado y actualización de la pantalla, limitándose a enviar al 8279 los comandos y datos apropiados cuando se quiere efectuar algún cambio.

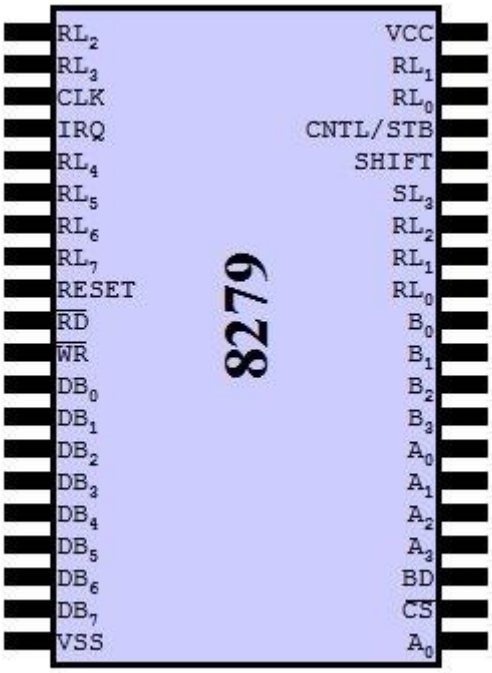


Figura 20. Patillaje del 8279.

El 8279 viene encapsulado en una pastilla de 40 pines, según se aprecia en la Figura 20. La patilla CS, como es habitual, selecciona el integrado cuando se pone a nivel bajo. En ese estado, con el pin A_0 (patilla 21) se indica si lo que hay en los pines DB_0–DB_7 es un comando o bien un dato. Estos ocho pines son el bus bidireccional de 8 bits que permite al 8279 comunicare con el microprocesador. En el proceso de lectura/escritura están implicados, aparte del pin A_0 que establece si es un dato o un comando, los pines RD y WR, que con nivel bajo determinan si los datos entran del bus externo a los buffers internos del 8279 o viceversa.

Al 8279 puede conectarse un teclado con hasta 64 teclas, habitualmente dispuestas en forma de matriz con una serie de filas y columnas (véase la Figura 21), y cuenta con un buffer interno que permite guardar hasta 8 caracteres. La matriz puede ser desde 2x2 hasta 8x8. Los pines RL_0–RL_7 (patillas 1, 2, 5, 6, 7, 8, 38 y 39) indican al 8279 qué teclas están activas, mientras que los pines SL_0 a SL_3 (patillas 32-35) seleccionan la fila a inspeccionar en cada caso. Los pines CNTL y SHIFT (patillas 36 y 37) permiten almacenar junto a cada pulsación de tecla el estado de dos teclas especiales, que suelen ser **Control** y **Mayúsculas**. Cuando el 8279 detecta un cambio, según la programación que se le haya hecho previamente, genera una interrupción para comunicar al microprocesador el evento y que éste pueda solicitar el código correspondiente.

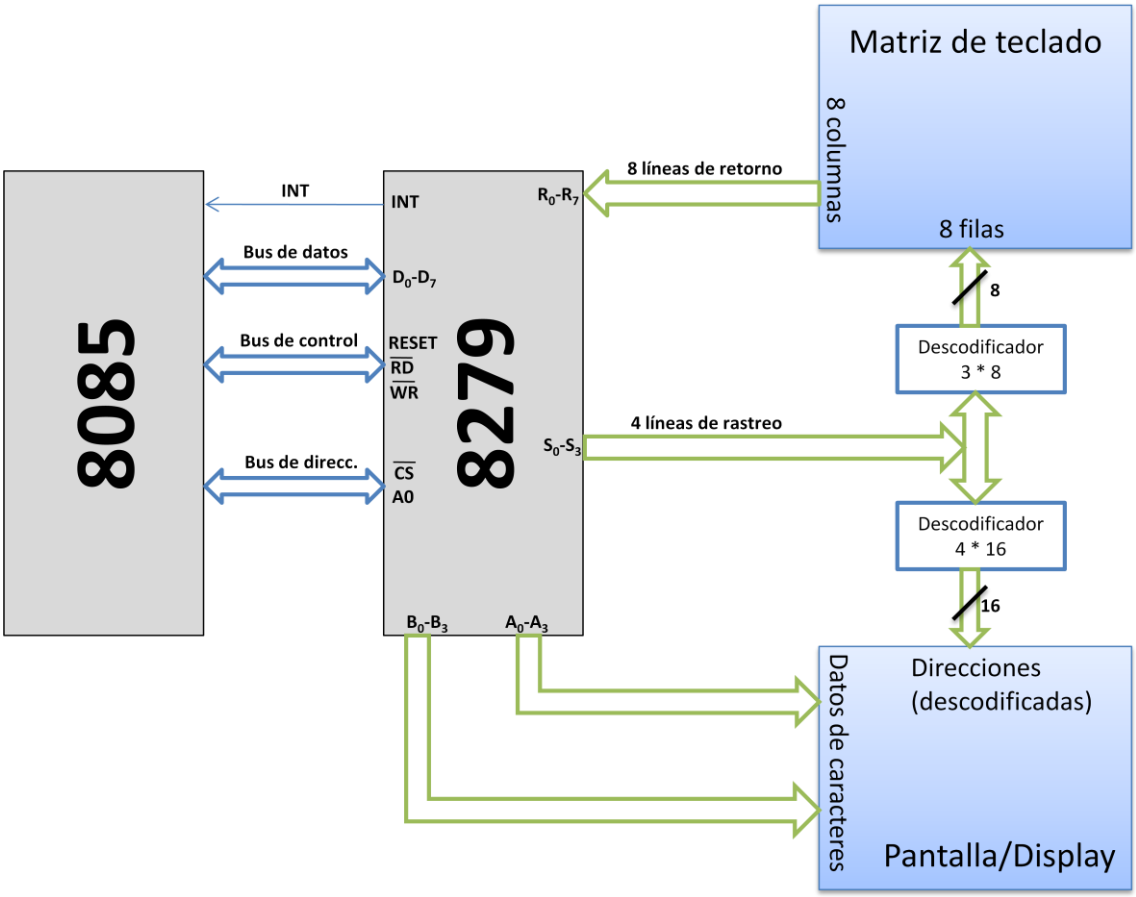


Figura 21. Esquema de bloques de las conexiones del 8279.

La pantalla que puede controlar el 8279 tendrá hasta 16 celdas de 8 segmentos, que pueden configurarse también como un panel doble de 16 celdas por 4 segmentos cada uno. Los datos mostrados en esa pantalla o display se almacenan internamente en 16 bytes de memoria, que el microprocesador puede ir estableciendo enviando los comandos adecuados al 8279. Los pines A_0–A_3 (patillas 24-27) y B_0–B_3 (patillas 28-31) se conectan al display y serán los encargados de activar o desactivar los segmentos adecuados. El microprocesador solamente tendrá que ir enviando los datos a mostrar, ocupándose el 8279 de ir efectuando el desplazamiento y la visualización.

Además el 8279 tiene la habitual entrada de reloj, la patilla RESET, un pin BD que borra el display y los pines de alimentación y masa. Es un integrado que se conecta directamente al bus de datos del sistema y que el microprocesador puede programar de diferentes modos.

4.1 Estructura del 8279 en la conexión con el uP-2000

En el sistema uP2000, el 8279 se encuentra conectado al 8085 a través de los buses de datos, control y direcciones, como se ve en el mismo esquema de la Figura 21. Las patillas de reloj, escritura, lectura e interrupción están conectados al bus de control, de forma que el 8085 puede facilitar la señal de reloj al 8279, indicar si se va a efectuar una operación de lectura o escritura y recibir las señales de interrupción del 8279. Según la figura P4.12, correspondiente al esquema del sistema uP2000, la señal A_0 del 8279 estaría conectada al bus de direcciones. No he encontrado más documentación al respecto, supongo que dependiendo del es-

tado de un cierto hilo del bus de direcciones se interpretará que el bus de datos lleva un comando o bien un dato para el 8279.

El teclado del uP2000 cuenta en total con 22 teclas distribuidas, desde un punto de vista lógico, en una matriz de 3 filas por 8 columnas, si bien las dos últimas columnas de la última fila no están usadas. Los pines RL_0-RL_7 del 8279 están conectados como columnas de esta matriz, mientras que los pines SL_0-SL_2 seleccionan la fila a inspeccionar en cada ciclo.

La pantalla del uP2000 está formada por un display de 4 dígitos + 2 dígitos, usados para mostrar direcciones y datos. Los pines A_0-A_3 y B_0-B_3 del 8279 controlan los segmentos que se activarán en la siguiente entrada al display, mientras que las señales SL_0-SL_2, a través de un decodificador/multiplexor, selecciona la posición del display a la que se envía el dato. Las señales SL_3, CNTL, SHIFT no son utilizadas en este diseño.

En cuanto a su disposición en el mapa de memoria del sistema microprocesador uP-2000, el 8279 ocupa las direcciones 3000H-37FFH. Ésta sería la disposición lógica, puesto que este integrado solamente tiene dos posiciones de memoria accesibles por parte del microprocesador.

4.2 ¿Cómo se selecciona el 8279?

En el esquema del uP-2000 puede verse que el pin CS del 8279 se encuentra conectado a la señal CS_5, que tendrá asociada una lógica de selección que pondrá a nivel bajo esta señal, y a nivel alto el resto de pines CS, cuando en el bus de direcciones se detecte el patrón 0011 0XXS XXXX XXXX en binario. Los cinco bits de mayor peso son los que activarán CS_5, mientras que el bit marcado como S, con sus dos posibles estados, seleccionaría entre la escritura en el display y la lectura del teclado.

Puesto que en el uP-2000 existen muchos otros dispositivos, a cada uno de los cuales le corresponde un cierto patrón de los bits A_{15}-A_{11} del bus de direcciones, se utiliza un multiplexor para implementar la lógica de selección, activando (poniendo a nivel bajo) únicamente la señal CS adecuada. Esta lógica podría ser la de la figura 22.

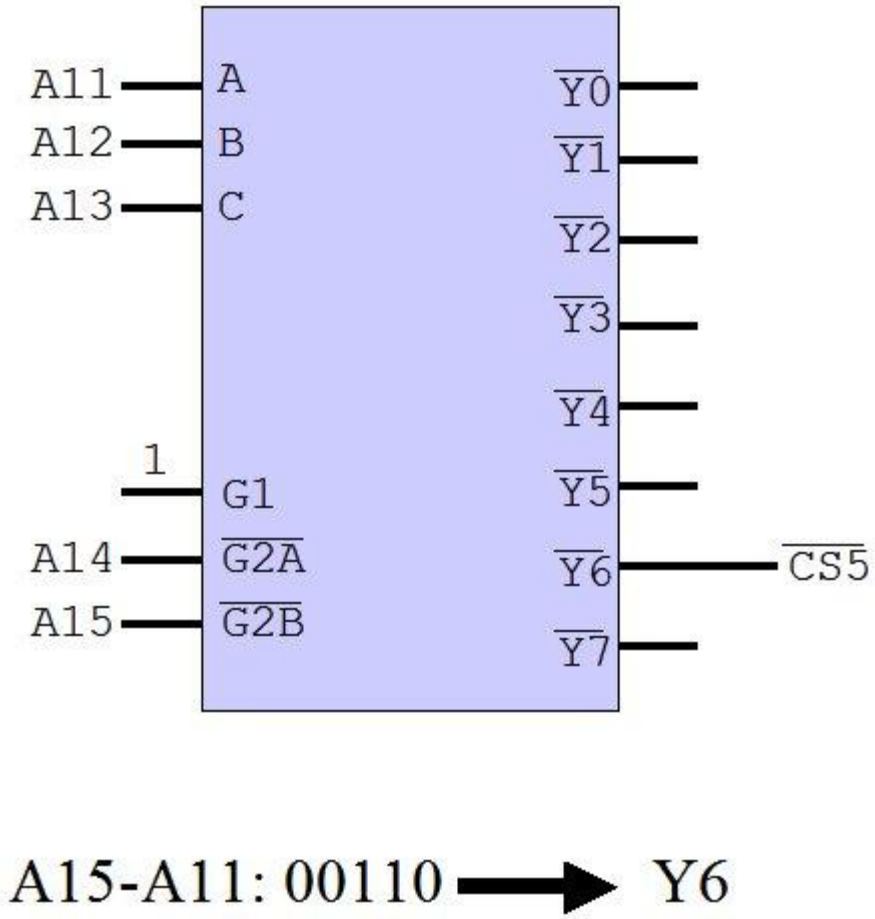


Figura 22. Lógica de selección del 8279 en el sistema uP-2000.

Para seleccionar el 8279 en un sistema uP-2000, por tanto, bastaría con poner en el bus de direcciones una dirección con el patrón adecuado.

5 El sistema uP-2000

Un microprocesador aislado no tiene utilidad por sí mismo, para poder aprovechar sus posibilidades es necesario facilitarle alimentación eléctrica, una señal de reloj externa, memoria en la que almacenar datos y de la que leer el programa que debe ejecutar y, sobre todo, algún tipo de conexión que facilite la comunicación con los usuarios, ya que sin esto el resto no tendría sentido.

El sistema uP-2000 (véase la Figura 23) es lo que se conoce como un *sistema basado en microprocesador*, concretamente basado en el 8085. Desarrollado por la firma Alecop, este sistema ofrece todos los elementos que se han mencionado antes y algunos más, como interfaces de comunicación serie y paralelo (usando módulos 8255 y USART), una pequeña pantalla numérica y un teclado y software monitor alojado en memoria EPROM.

Figura 23. Imagen del sistema uP-2000.

En conjunto, el sistema uP-2000 nos permitirá trabajar con el microprocesador 8085 a distintos niveles, lo cual nos facilitará el aprendizaje relativo a la programación de este micro y los dispositivos a él conectados.

El esquema de la Figura 24 (página siguiente) es un diagrama de bloques en el que aparecen los elementos principales del uP-2000: el microprocesador 8085, el conjunto de buses de direcciones, datos y control, los integrados auxiliares y la lógica de selección que permite activar uno u otro según las líneas que se activen en el bus de direcciones.

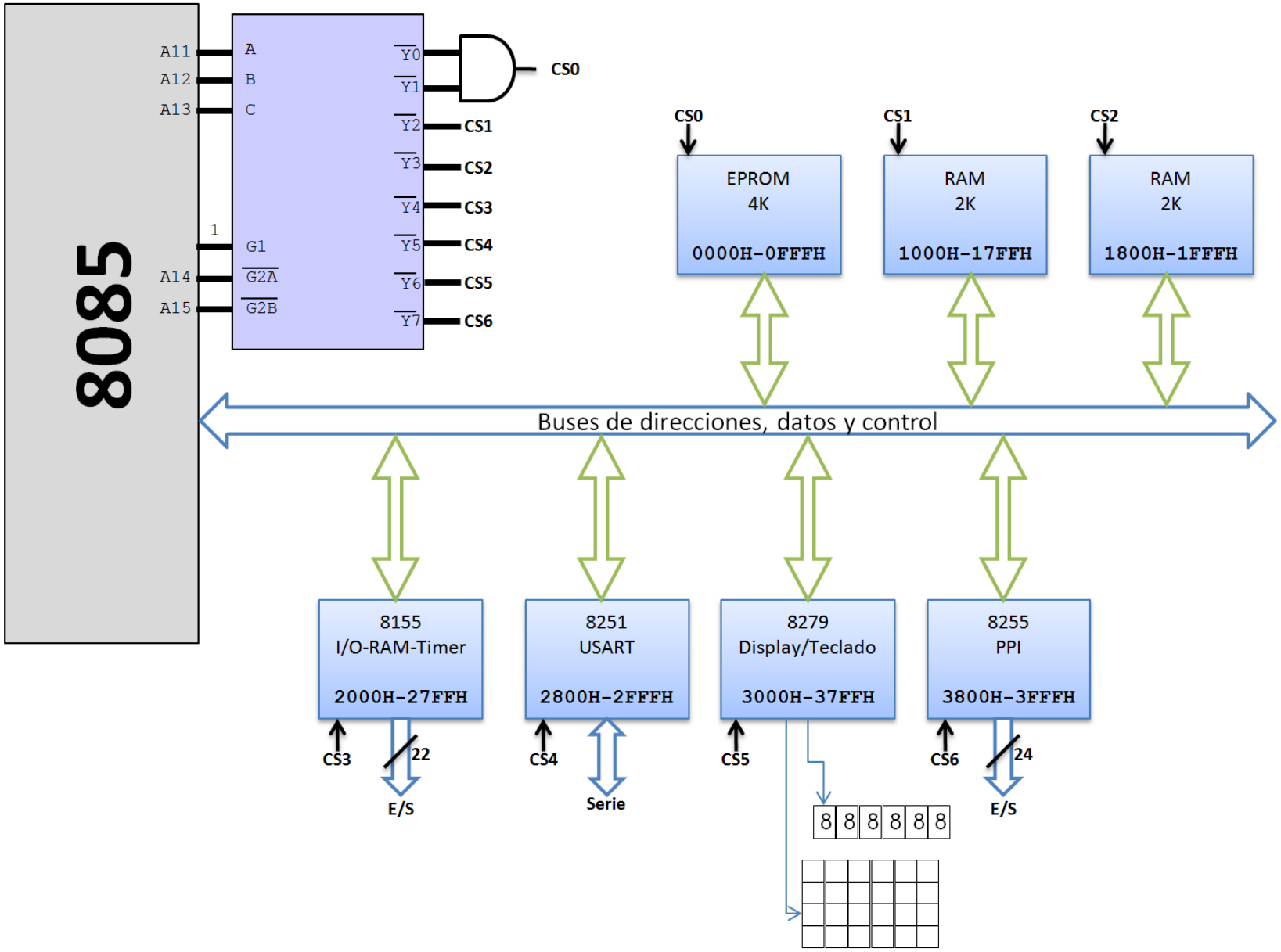


Figura 24. Esquema de bloques del sistema uP-2000.

5.1 Descripción de los componentes que forman el sistema

El diseño físico del uP-2000, con una tapa de metacrilato transparente y serigrafiada, facilita la identificación visual de los componentes que forman este sistema, comenzando por el propio microprocesador que es claramente visible, como se aprecia en la Figura 25, en la parte superior izquierda cuando se tiene el equipo orientado para operar sobre él.

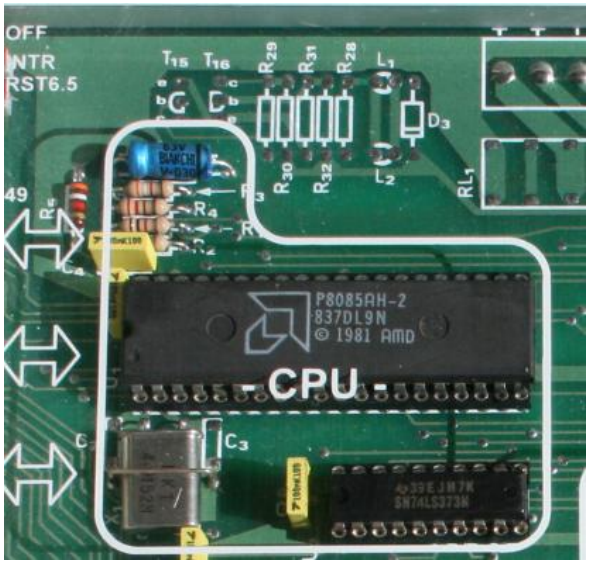


Figura 25. Situación del 8085 en el uP-2000.

La parte inferior derecha está ocupada por el teclado (véase la Figura 26), dividido en dos áreas diferentes: 16 teclas con dígitos hexadecimales y, a su izquierda, ocho teclas más con comandos. Sobre el teclado se encuentra el display numérico, una pequeña pantalla capaz de mostrar un máximo de seis dígitos en dos grupos: uno a la izquierda, conocido como campo de direcciones, y otro a la derecha denominado campo de datos. La visualización está basada en celdas de 7 segmentos, similares a las utilizadas en las calculadoras. Tanto el teclado como la pantalla están controlados por un 8279, integrado descrito anteriormente.

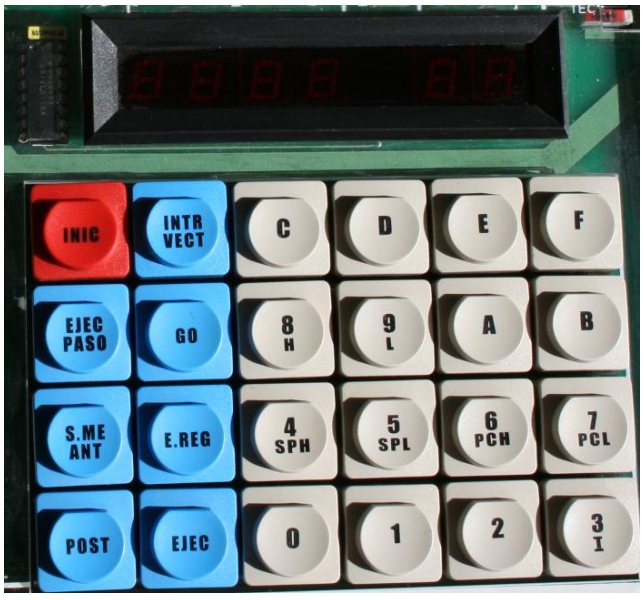


Figura 26. Teclado y display del uP-2000.

El uP-2000 cuenta con tres módulos de memoria, son los integrados que pueden verse en la Figura 27, físicamente dispuestos verticalmente debajo del microprocesador. El primero de ellos es de tipo EPROM y contiene un software programado por el fabricante y que se denomina *programa monitor*. La finalidad de éste es indicar al 8085 qué debe hacer al iniciar el sistema, así como facilitar la comunicación a través del teclado, la pantalla y el resto de dispositivos de E/S que componen el sistema. También facilita una serie de rutinas de utilidad general que se describen más adelante.

Los otros dos integrados corresponden a la memoria RAM y serán, por tanto, los que faciliten el almacenamiento de los programas que se vayan codificando y los datos asociados. Cada uno de ellos tiene una capacidad de 2 Kbytes, por lo que en total el sistema cuenta con 4 Kbytes de RAM. Éstos ocupan las direcciones comprendidas entre 1000H y 1FFFH (en hexadecimal), mientras que la EPROM ocupa las posiciones 0000H a 0FFFH.

Figura 27. Módulos de memoria.

Bajo los módulos de memoria se encuentra el 8255 (véase la Figura 28), componente encargado de controlar la entrada/salida de datos en paralelo a través de sus tres puertos, correspondientes a los conectores J2, J3 y J4, situados en el margen inferior izquierdo del uP-2000. Es posible conectar ahí un módulo externo con pulsadores, microinterruptores y leds, accediendo a estos elementos a través del PPI.

Figura 28. Módulos 8255 (PPI) y 8279.

Además de los ya citados, en el interior del uP-2000 también se encuentran los siguientes componentes:

- 8155: Es un módulo que cuenta con un reloj, puertos de entrada/salida y también una pequeña porción de memoria.

- 8251: Conocido como USART, se encarga de facilitar la transmisión de datos en serie que, por ejemplo, hace posible la comunicación con un PC para la carga de programas.

- Decodificador: Son dos circuitos integrados que facilitan la decodificación del mapa de direcciones del sistema uP-2000, según el esquema de la Figura 24.

- Microinterruptores: La tapa transparente del uP-2000 tiene algunas aberturas que permiten acceder a microinterruptores que modifican su funcionamiento. De éstos el más interesante es el que puede verse en la Figura 28, situado en la esquina superior derecha, ya que permite alternar el control entre el teclado y pantalla propios y el PC a través del conector serie.

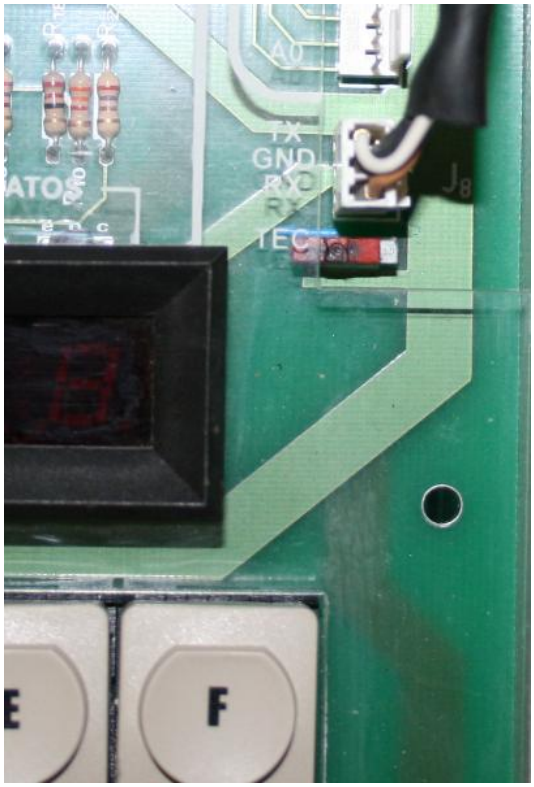


Figura 29. Microinterruptor que alterna entre teclado y PC.

Al alimentar el sistema uP-2000 en la pantalla debe aparecer la indicación - 8085, tal y como se muestra en la Figura 30. De no ser así puede pulsarse la tecla **INIC** para inicializar el sistema. Si aún así no aparece ese mensaje posiblemente haya algún fallo.

Figura 30. Aspecto del display del uP-2000 al conectarlo.

Como se indica en la propia Figura 30, la pantalla se encuentra dividida en dos zonas. La que ocupa el lado izquierdo es conocida como *campo de direcciones* y está formada por cuatro dígitos. El denominado *campo de datos*, en el margen derecho, se compone de dos dígitos. En ambos casos se usa siempre numeración hexadecimal, por lo que el campo de datos puede mostrar un byte y el de direcciones dos.

5.2 Uso del teclado del uP-2000

Para comunicarse con el uP-2000 es indispensable conocer la función de cada una de las teclas de comando, así como el procedimiento a seguir para introducir direcciones o datos, lanzar la ejecución de un programa almacenado en memoria o comprobar el estado de los registros del procesador. En la Figura 31 puede verse un detalle del teclado del uP-2000.

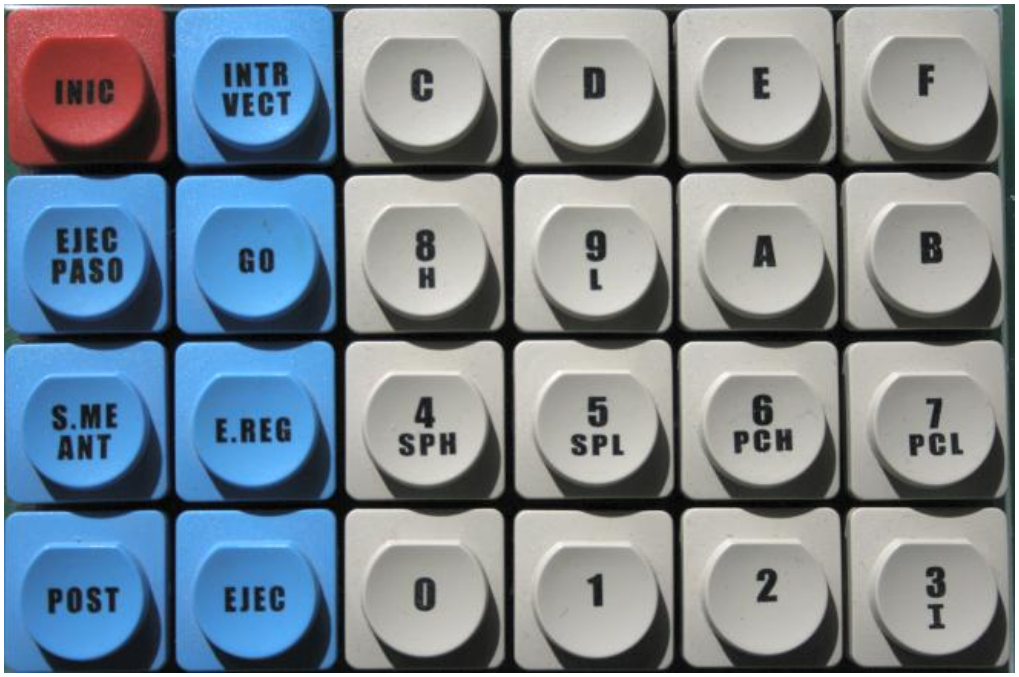


Figura 31. Detalle del teclado del uP-2000.

Los procedimientos completos para efectuar cada operación se plantean como ejercicios, más adelante, por lo que de momento será suficiente con tener una idea sobre cuál es la finalidad de cada tecla.

Las ocho teclas de comando del uP-2000 tienen el siguiente cometido:

- 

INIC: Se encuentra conectada a la patilla RESET del 8085, por lo que en cuento es pulsada reinicia el microprocesador y, en consecuencia, el sistema. Puede ser utilizada para sacar al sistema de una situación de error o bloqueo.

- 

INTR VECT: Produce una interrupción 7.5 en el 8085, activando la ejecución del correspondiente vector situado en la memoria EPROM del uP-2000.

- 

EJEC PASO: Hace posible la ejecución de programas paso a paso, instrucción a instrucción, de manera que se permite examinar tanto los registros del procesador como la memoria a medida que se ejecuta un programa.

- 

GO: Usando el teclado del uP-2000 es posible introducir en la memoria del sistema un programa completo. Mediante esta tecla se lanzaría la ejecución de dicho programa, para lo cual habría que pulsarla, introducir la dirección de memoria donde comienza el programa y finalmente ejecutarlo con la tecla **EJEC**.

- 

S.ME ANT: Facilita el acceso a la memoria del sistema, bien sea para leerla o para modificarla. Al pulsar esta tecla la pantalla mostrará el valor 0000 en el campo de direcciones, momento en el que debe introducirse la dirección a examinar o modificar. Usando las teclas hexadecimales hay que escribir los cuatro dígitos de la dirección, que irán apareciendo en la pantalla.

También puede utilizarse esta tecla, mientras está examinándose la memoria, para retroceder a la dirección anterior.

- 

E.REG: Examinar el contenido de los registros del procesador es una posibilidad interesante a medida que se ejecuta un programa, siendo ésa la finalidad de esta tecla. Al pulsarla el sistema quedará a la espera de que, usando el resto del teclado, se indique qué registro quiere examinarse.

- 

POST: Mediante esta tecla se confirma una acción, por ejemplo la introducción de una dirección de memoria cuyo contenido quiere leerse o modificarse, o bien se avanza a la dirección siguiente, dependiendo del contexto.

- 

EJEC: La finalidad de esta tecla es interrumpir el comando que esté en proceso actualmente, comando que puede ser una lectura/escritura en la memoria o la ejecución paso a paso de un programa.

Dependiendo del comando que se utilice el uP-2000 quedará a la espera de que se facilite una dirección de memoria o se elija un registro, según los casos, tareas para las que se recurrirá al área numérica del teclado.

En el apartado *Familiarizarse con el entorno,* unas páginas más adelante, se proponen diversos ejercicios en los que se describen las tareas habituales de uso de teclado y display: introducción de direcciones, obtención y modificación de posiciones de memoria y registros del 8085, etc.

5.3 Los servicios software del uP-2000

Los distintos dispositivos hardware que componen el sistema uP-2000 pueden programarse de manera directa o indirecta, según los casos, pero para ello es preciso contar con un conocimiento avanzado de su funcionamiento. Para mostrar una letra o un número en la pantalla, por ejemplo, se necesita saber cómo iluminar los segmentos adecuados y cómo hacer llegar dicha información hasta ese dispositivo, a través del integrado 8279. Lo mismo sería aplicable a la lectura de información a través del teclado o cualquier otra operación.

Una parte importante del uP-2000 es el software que integra en su memoria EPROM, compuesto de una serie de pequeños programas o rutinas que se ocupan de tareas como las que acaban de citarse y de muchas otras. Al escribir un programa propio, esas rutinas actúan como servicios a los que se puede llamar cuando se necesite, fundamentalmente a través de la instrucción **CALL** del 8085. Suponiendo que se necesite mostrar un valor en el campo de datos del uP-2000, los pasos a seguir serían los indicados en el esquema de la Figura 32.

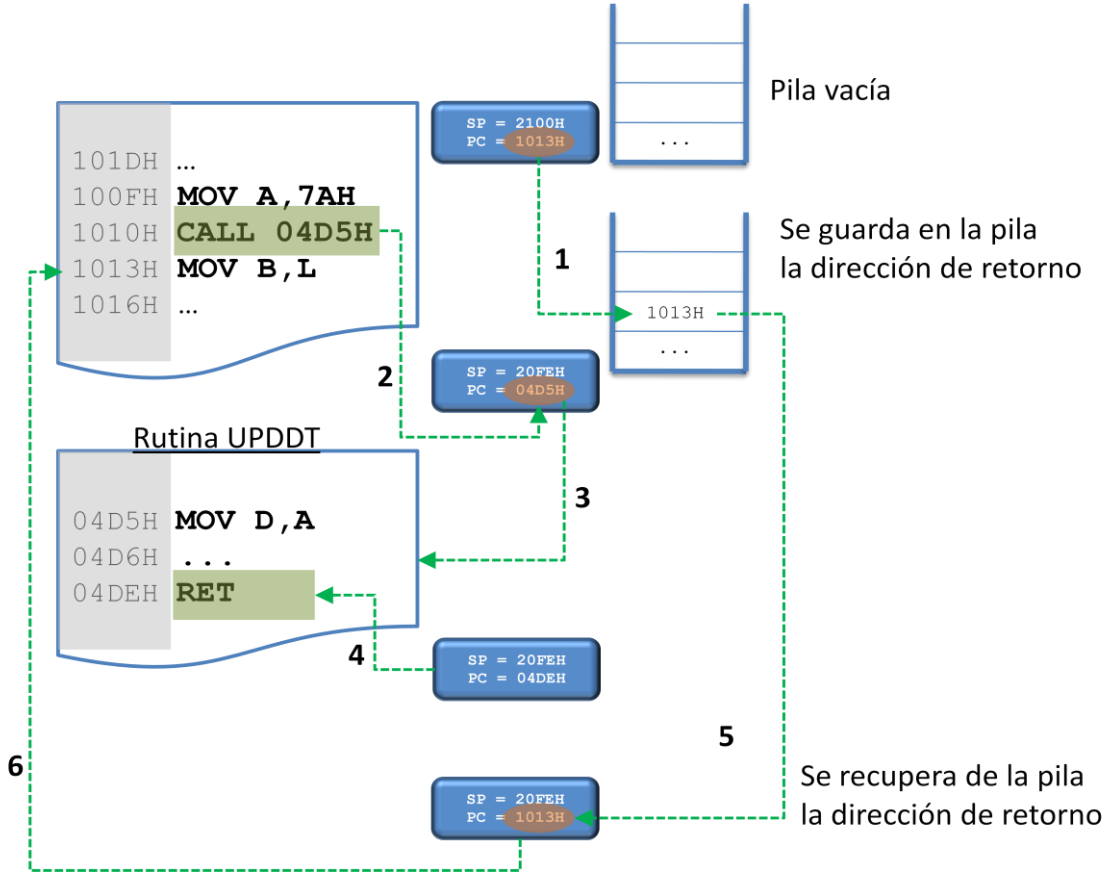


Figura 32. Ejecución de la rutina que muestra un valor en el campo de datos.

La rutina `UPDDT` (cada rutina tiene asignado un nombre simbólico) se encuentra alojada en la dirección `04D5H` y espera encontrar en el acumulador el valor a mostrar. Cuando ha terminado su trabajo devuelve el control al programa, que continuará su ejecución.

Como en el caso de los comandos accesibles desde el teclado del uP-2000, en la sección de problemas se propondrán multitud de ejercicios en los que habrá que usar los servicios que en la Tabla 4 simplemente se recogen a modo de referencia.

Dirección	Finalidad	Parámetros entrada	Valores salida	Modifica
04C9H	Mostrar un valor en el campo de direcciones del display.	`HL` contendrá el valor a mostrar.	Ninguno.	Todos los registros.
04D5H	*Mostrar un valor en el campo de datos.*	*`A` contendrá el valor a mostrar.*	*Ninguno.*	*Todos los registros.*
03C3H	Mostrar un valor en hexadecimal en el presentador.	`DE` contendrá el valor a mostrar.	Ninguno.	Los registros `A`, `H`, `L` y `F`.
0B74H	*Produce un retardo de*	*Ninguno.*	*Ninguno.*	*No se indica.*

Dirección	Finalidad	Parámetros entrada	Valores salida	Modifica
	1ms.			
044EH	Recoge una pulsación del teclado.	Ninguno.	El código de la tecla en A.	Los registros A, H, L y F.
041DH	*Envía caracteres al presentador visual.*	*A=0 -> Se muestra en el campo de direcciones.* *B=0 -> Se muestra en el campo de datos.* *B=1 -> Se ilumina el punto decimal.* *HL=Dirección donde están las posiciones con los caracteres a mostrar.*	*Ninguno.*	*No se indica.*
037DH	Leer una dirección de 16 bits desde el teclado.	Ninguno.	DE=Dirección leída. A=Código de la última tecla usada. Acarreo=Indica que se ha leído al menos un carácter.	No se indica.
02BFH	*Leer una dirección de 16 bits desde el teclado.*	*Ninguno.*	*DE=Dirección leída.*	*Todos los registros, incluido el de indicadores.*
02CFH	Muestra el valor del acumulador en el campo de datos.	A=Valor a mostrar.	Ninguno.	Todos los registros, incluido el de indicadores.
031FH	*Borra el campo de direcciones y de datos*	*B=Indica si hay que mostrar el punto (1) o no (0)*	*Ninguno*	*Todos los registros, incluido el de indicadores*

Tabla 4. Rutinas de servicio del uP-2000.

El significado de las columnas de esta tabla es el siguiente:

- **Dirección**: Indica la dirección a la que hay que efectuar la llamada. Se dispondría, por tanto, a continuación de la instrucción CALL.

- **Finalidad**: Breve descripción de lo que hace la rutina cuyo código se encuentra en esa dirección.

- **Parámetros de entrada**: Para hacer su trabajo, algunas de estas rutinas precisan unos valores de entrada que hay que asignar a ciertos registros. En esta columna se indica cuáles son esos parámetros y en qué registros hay que almacenarlos.

- **Valores de salida**: En caso de que la rutina genere alguna información para devolver al programa que le ha llamado, en esta columna se indica el registro usado para devolverla y su significado.

- **Modifica**: Indica los registros que se verán modificados por la rutina y que, por tanto, sería necesario preservar antes de hacer la llamada en caso de que no quieran perderse los valores que contienen.

5.4 Mapa de memoria del uP-2000

El microprocesador 8085 tiene un bus de direcciones de 16 bits y, en consecuencia, una capacidad de direccionamiento de 65536 posiciones de memoria. En realidad no todas las direcciones corresponden a memoria porque, como se deduce del esquema de la Figura 24, parte de las líneas de direcciones se emplean para seleccionar el componente del sistema sobre el que va a trabajarse.

Una porción de la memoria está ya ocupada por programas escritos en una memoria EPROM: el programa monitor del uP-2000 y las rutinas de servicio descritas en el punto anterior. Otra zona está ocupada por memoria RAM que el usuario final puede usar para introducir sus programas y datos, etc. Conocer la distribución de direcciones que se ha efectuado al diseñar el uP-2000, lo que se conoce habitualmente como *mapa de memoria del sistema*, es un importante, ya que es la única forma de saber qué direcciones hay que utilizar para cada propósito.

La Figura 33 representa ese mapa de memoria de forma lineal, comenzando con la dirección `0000H` en la parte superior, indicando a qué componente corresponde cada rango. El mapa finaliza en la dirección `3FFFH` porque ésta es la última empleada en el uP-2000, pero en otros sistemas podría ampliarse hasta la dirección `FFFFH` que sería la máxima direccionable con un bus de 16 bits.

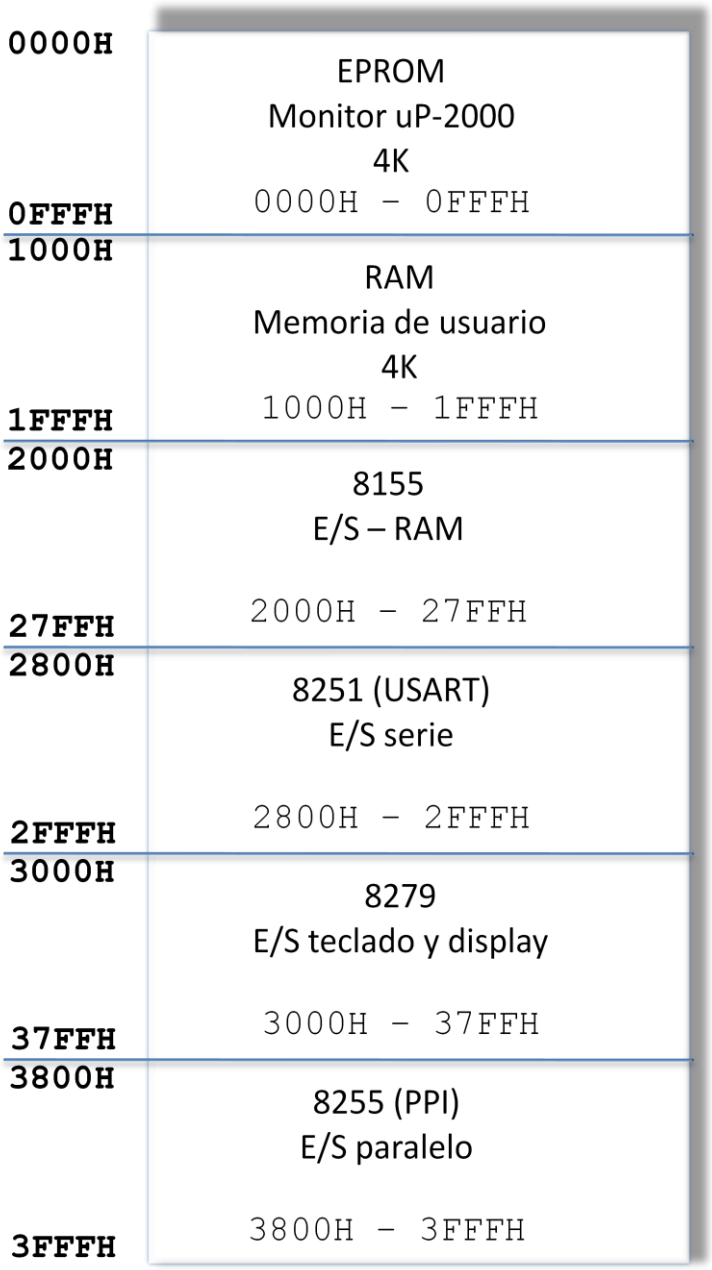


Figura 33. Mapa de memoria del uP-2000.

II Problemas por niveles

6 Familiarizarse con el entorno

El sistema uP-2000 va a ser la herramienta de trabajo que se emplee durante el estudio de esta asignatura, una herramienta que permitirá comprobar cuál es el resultado de la ejecución de diversos ejercicios, escritos en lenguaje ensamblador 8085, facilitando así el proceso aprendizaje.

Como cualquier otra herramienta, antes de poder aprovechar el uP-2000 hay que aprender a utilizarlo, siendo éste el propósito de los ejercicios que se plantean en esta sección. Los ejercicios y su objetivo son:

- Aprender a utilizar el teclado para leer el contenido de la memoria identificándolo en la pantalla del uP-2000.
- Saber cuáles son los pasos a seguir para cambiar el contenido de una posición de memoria.
- Practicar la lectura de los datos contenidos en la memoria y el uso de la referencia de instrucciones del 8085 para desensamblar un programa.
- Aprender a leer el contenido de los registros del 8085 sirviéndose del teclado y la pantalla del uP-2000.
- Usar teclado y display para modificar el contenido de los registros del 8085.
- Aprender a consultar e interpretar el contenido del registro de estado del 8085.
- Reúne en un mismo ejercicio distintas operaciones de manipulación de la memoria y los registros.
- Aprender a lanzar la ejecución de un programa almacenado en un punto concreto de la memoria del uP-2000.
- Finalmente, usando lo aprendido en la sección, se proponen ejercicios que recorren todo el proceso de escritura de un programa, almacenamiento en la memoria del uP-2000, ejecución y comprobación del resultado que genera.

6.1 Acceso a la memoria del uP-2000

Para completar los ejercicios siguientes no se precisa el PC, únicamente el sistema uP-2000 debidamente alimentado y con el interruptor de puesta en marcha en posición **ON**. En la pantalla debe mostrarse la indicación **- 8085**.

Ejercicio 6.1.1

Examine el contenido de la posición de memoria **0B74H** y las tres siguientes, anotando los cuatro bytes.

Indicaciones

En principio resultará útil tener a mano la referencia del teclado del uP-2000, a fin de consultar la finalidad de cada tecla.

Este ejercicio consta básicamente de dos pasos:

1. Establecer la dirección de memoria inicial que se quiere examinar.

2. Avanzar a las posiciones siguientes para leer su contenido.

Cuestiones

1. ¿Qué tecla habría que utilizar para retroceder una posición de memoria, en lugar de avanzarla, mientras se examina su contenido?

2. ¿Cómo puede cancelarse el comando de lectura de memoria para poder introducir otro distinto?

Solución

Para completar el ejercicio hay que reproducir los pasos indicados a continuación:

1. Comprobar que el uP-2000 está esperando la introducción de un comando, mediante la aparición del mensaje – `8085` en la pantalla. Si es necesario, pulsar la tecla `INIC` para inicializarlo.

2. Pulsar la tecla **`S.ME ANT`** para indicar al sistema que vamos a introducir una dirección de memoria.

3. Escribir la dirección de memoria, pulsando sucesivamente las teclas **`0`**, **`B`**, **`7`** y **`4`**. Los dígitos irán apareciendo en el campo de direcciones, desplazándose de izquierda a derecha.

4. Pulsar la tecla **`POST`** para confirmar que hemos terminado de introducir la dirección. En ese momento aparecerá en el campo de datos el contenido de la dirección de memoria anterior.

5. Utilizando la tecla **`POST`** iremos incrementando el valor del campo de direcciones y leyendo el contenido de esa celdilla de memoria, cuyo valor aparecerá en el campo de datos.

La secuencia de valores obtenida debe ser `C5H`, `06H`, `86H`, `05H`.

Ejercicio 6.1.2

Escriba en las posiciones de memoria **1000H**, **1001H** y **1002H** los valores **CDH**, **74H** y **0BH**, respectivamente. Verifique después la operación, leyendo esas mismas posiciones de memoria.

Indicaciones

Los pasos básicos en los que se descompone este ejercicio son:

1. Establecer la dirección de memoria inicial que va a modificarse.
2. Modificar el contenido de esa posición de memoria.
3. Avanzar a la dirección siguiente y volver al paso 2.

Cuestiones

1. ¿Cuál es la razón de que el campo de datos del uP-2000 solamente cuente con 2 dígitos?

2. Tras completar el ejercicio si se pulsa la tecla **INIC** ¿se pierden los datos introducidos en la memoria? ¿Por qué?

Solución

Para completar el ejercicio hay que reproducir los pasos indicados a continuación:

1. Comprobar que el uP-2000 está esperando la introducción de un comando.

2. Pulsar la tecla **S.ME ANT** para indicar al sistema que vamos a introducir una dirección de memoria.

3. Escribir la dirección de memoria, como en el ejercicio 1, hasta que en el campo de direcciones aparezca **1000H**.

4. Pulsar la tecla **POST**. En el campo de datos aparecerá el contenido de la celdilla de memoria que ocupa la posición **1000H**.

5. Escribir el nuevo valor a almacenar en la celdilla: **C5H**. Éste aparecerá en el campo de datos.

6. Pulsar la tecla **POST** para confirmar la escritura del nuevo valor y avanzar a la dirección siguiente.

7. Repetir los pasos 5 y 6 hasta haber introducido los dos valores restantes.

8. Pulsar la tecla **EJEC** para finalizar el comando en curso.

9. Seguir los pasos del ejercicio 1 para examinar el contenido de las celdillas modificadas, confirmando que contienen los valores indicados.

La secuencia de valores obtenida debe ser `C5H`, `06H`, `86H`, `05H`.

Ejercicio 6.1.3

Escriba en las posiciones de memoria **100H**, **101H** y **102H** los valores **CDH**, **74H** y **0BH**, respectivamente. Verifique después la operación, leyendo esas mismas posiciones de memoria.

Indicaciones

Ver las indicaciones del ejercicio 2.

Cuestiones

1. ¿Por qué al verificar la operación nunca aparecen los valores que teóricamente se han escrito en la memoria?

Solución

Seguir el procedimiento del ejercicio 2.

6.1.1 Ejercicios propuestos

Ejercicio 6.1.1.1

Determine qué instrucciones ejecuta el uP-2000 cada vez que se pulsa la tecla **INIC**.

Indicaciones

Cuando se activa la patilla **RESET** del 8085, al pulsar la tecla **INIC**, el microprocesador pasa por un proceso de inicialización que le lleva a poner la dirección **0000** en el puntero de instrucción, pasando por tanto a ejecutar las instrucciones que se encuentran a partir de esas dirección.

Sabiendo esto, este ejercicio se dividiría en las siguientes acciones:

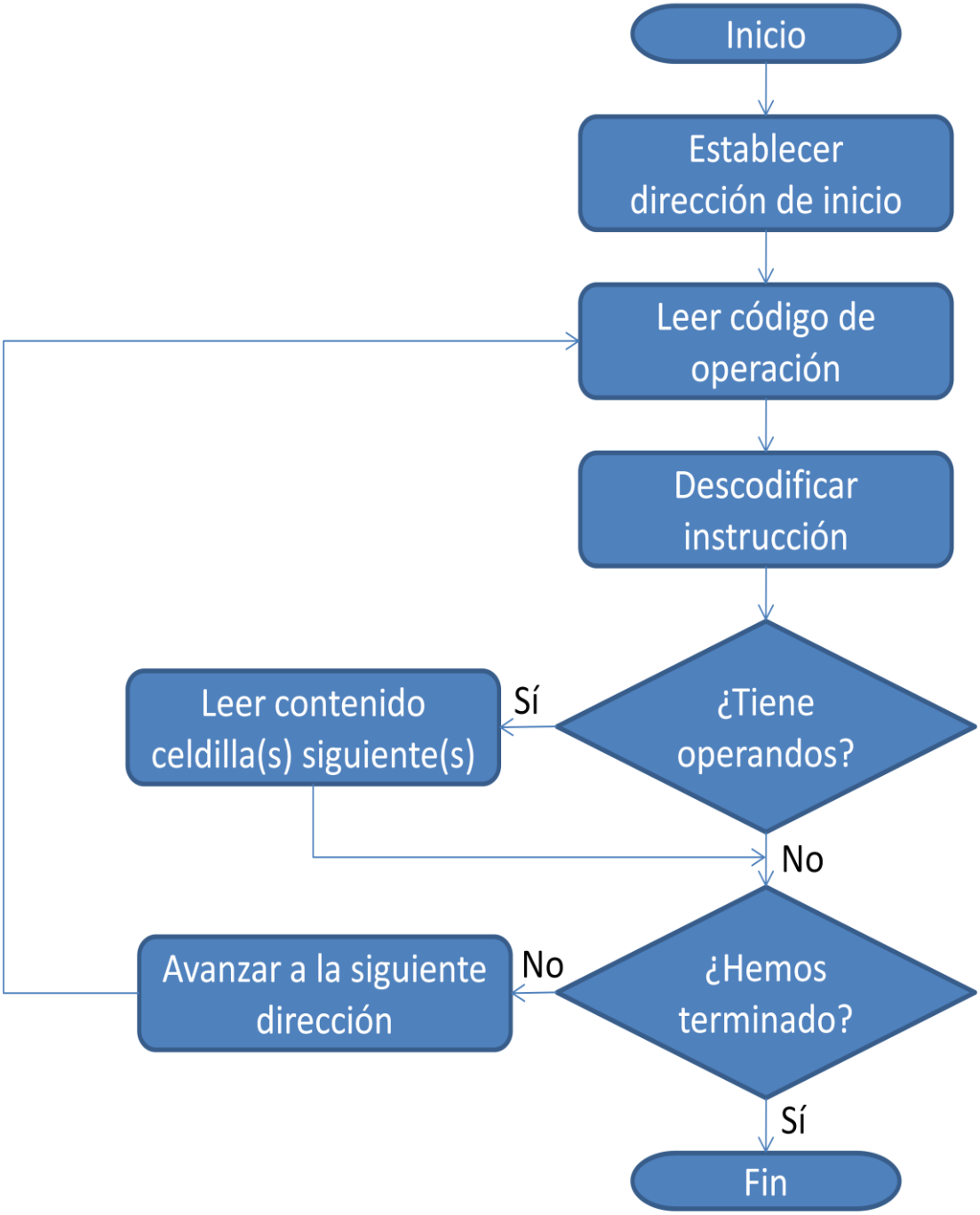
1. Leer el contenido de la celdilla de memoria cuya dirección es **0000H**, obteniendo un valor que será el código de operación de la primera instrucción.

2. Buscar el código de operación en la referencia de instrucciones del 8085, para saber de qué instrucción se trata y anotarla.

3. Si la instrucción tiene operando, determinar si éste es de 8 o 16 bits para leer el contenido de una o dos celdillas, las que ocupan las direcciones siguientes.

4. Repetir los tres pasos anteriores hasta conocer las cuatro primeras instrucciones y sus operandos.

El siguiente diagrama de flujo representa el proceso.



Ordinograma del ejercicio 6.1.1.1.

Cuestiones

1. Describa brevemente la finalidad de las cuatro instrucciones que haya obtenido.

2. ¿Es capaz de imaginar qué hacen esas cuatro instrucciones de forma conjunta?

6.2 Acceso a los registros del 8085

Ejercicio 6.2.1

Examine el contenido de los registros de uso general del 8085.

Indicaciones

Busque en la referencia de teclas del uP-2000 para saber qué tecla debe utilizar para obtener el contenido de un registro.

Observe que en el teclado del uP-2000 las teclas numéricas 3 a 9 contienen un segundo símbolo, debajo del dígito numérico, que representan diferentes registros.

Cuestiones

1. ¿Contienen los registros siempre el mismo valor tras reiniciar el sistema con la tecla **INIC**? ¿Por qué?

2. ¿Qué ocurre si examina los registros justo después de alimentar el sistema, sin llegar a pulsar la tecla **INIC**?

Solución

Para completar este ejercicio es posible seguir dos procedimientos distintos. Los pasos del primero serían los siguientes:

1. Comprobar que el uP-2000 está esperando la introducción de un comando. Si es necesario, usar la tecla **EJEC** para finalizar un comando previo o la tecla **INIC** para inicializar el sistema.

2. Pulsar la tecla **E.REG** para indicar al sistema que queremos examinar el contenido de un registro.

3. El sistema queda a la espera de que se indique qué registro se quiere leer. Usar las teclas **A** a **F** y **3** a **9** para elegir el registro.

4. En el campo de datos aparece el contenido del registro indicado.

5. Pulsar la tecla **EJEC** para finalizar el comando y volver al paso 2 si se quiere examinar otro registro.

El segundo procedimiento coincide en los cuatro primeros pasos con el anterior, pero en el quinto en lugar de pulsar **EJEC** se utilizan las teclas **POST** y **S.ME ANT** para avanzar al registro siguiente o retroceder al anterior. De esta forma es posible examinar varios registros de forma secuencial.

Ejercicio 6.2.2

Inicialice el uP-2000 y compruebe cuál es la dirección a la que apunta el puntero de pila.

Indicaciones

El puntero de pila es un registro de 16 bits, por lo que no es posible mostrar completo su contenido en el campo de datos del uP-2000. Por ello se divide en dos partes: 8 bits altos (*high*) y 8 bits bajos (*low*) o menos significativos. Estas dos partes se identifican como **SPH** y **SPL**, respectivamente.

Las teclas **4** y **5** del uP-2000, que muestran debajo las leyendas **SPH** y **SPL**, son las que deben utilizarse para obtener el valor del puntero de pila.

Cuestiones

1. ¿Cuál es la dirección de la cabecera de pila tras inicializar el sistema?

2. ¿Puede determinar, examinando la serigrafía de la tapa del uP-2000, a qué elemento corresponde dicha dirección?

Solución

Los pasos a seguir para completar el ejercicio son los siguientes:

1. Pulsar la tecla **INIC** para que el uP-2000 ejecute la rutina de inicialización del sistema.

2. Pulsar la tecla **E.REG** para indicar que se quiere examinar un registro del procesador.

3. Pulsar la tecla **4/SPH** para obtener la parte alta del puntero de pila en el campo de datos.

4. Pulsar la tecla **POST** para avanzar y obtener la parte baja del puntero de pila.

5. Componer la dirección a partir de los datos previos.

Ejercicio 6.2.3

Modifique el contenido del acumulador del 8085, asignándole el valor **7FH**. Verifique a continuación que contiene dicho valor.

Indicaciones

Tras obtener el contenido de un registro, según el procedimiento descrito en el ejercicio 5, el valor que aparece en el campo de datos del uP-2000 puede ser modificado simplemente escribiendo otro nuevo y pulsando la tecla **POST**.

Cuestiones

1. ¿Pierde su nuevo valor el acumulador si se efectúan otras operaciones en el uP-2000, como examinar otros registros o la memoria?

2. ¿Se pierde el valor del acumulador al pulsar la tecla **INIC**?

Solución

Para completar este ejercicio hay que reproducir los pasos indicados a continuación:

1. Comprobar que el uP-2000 está preparado para la introducción de un nuevo comando.

2. Pulsar la tecla **E.REG** para indicar que queremos examinar un registro.

3. Pulsar la tecla **A** para leer el contenido del acumulador.

4. Escribir el nuevo valor: **7FH**.

5. Pulsar la tecla **POST** para confirmar el cambio.

6. Pulsar la tecla **EJEC** para finalizar.

7. Repetir los pasos 2 y 3 para verificar el valor contenido en el acumulador.

6.2.1 Ejercicios propuestos

Ejercicio 6.2.1.1

Determine cuál es el estado inicial del registro de indicadores del 8085 en el sistema uP−2000.

Indicaciones

En el teclado y la pantalla del uP-2000 el registro de indicadores o *flags* está identificado como **F**.

El apartado 1.1 de la sección de teoría, en la que se describen los registros del 8085, se encuentra el significado de cada byte del registro de indicadores.

Cuestiones

1. ¿Qué indicadores están activos?

2. ¿Cómo podría modificarse uno cualquiera de los indicadores del registro, por ejemplo poniendo a 1 el indicador de cero (**Z**)?

Ejercicio 6.2.1.2

Tras inicializar el sistema compruebe cuál es el contenido del acumulador y de la posición de memoria **20EEH**. Modifique el valor del acumulador y examine de nuevo esa misma posición de memoria. A continuación modifique el contenido de esa celdilla de memoria y examine el contenido del acumulador.

Indicaciones

Para completar este ejercicio es necesario haber realizado los anteriores, en los que se pone en práctica la lectura y modificación tanto de posiciones de memoria como de los registros del procesador.

Cuestiones

1. ¿Qué ocurre al modificar el valor del acumulador o de la posición de memoria **20EEH**?

2. ¿Sucede lo mismo con el registro de indicadores y la dirección de memoria **20EDH**?

6.3 Ejecución de programas

Ejercicio 6.3.1

Lance la ejecución del proceso de inicialización del 8085 sin hacer uso de la tecla **INIC**.

Indicaciones

En la referencia de teclas del uP-2000 se encuentra la tecla que hay que usar para iniciar la ejecución de un programa.

La dirección que toma el puntero de instrucción al inicializar el procesador ha sido indicada en varios de los ejercicios anteriores.

Cuestiones

1. ¿Cuál es el efecto que tiene la ejecución del proceso de inicialización?

Solución

Para completar este ejercicio los pasos a reproducir son los siguientes:

1. Comprobar que el uP-2000 está preparado para aceptar un nuevo comando.
2. Pulsar la tecla **GO**. En el campo de direcciones aparece el valor actual del puntero de programa.
3. Introducir la dirección donde se encuentra el programa a ejecutar, en este caso la **0000H**.
4. Pulsar la tecla **EJEC** para iniciar la ejecución.

Ejercicio 6.3.2

Escriba un programa para almacenar el valor **A5H** en el acumulador, alójelo al inicio de la memoria RAM del uP-2000 y ejecútelo. Compruebe a continuación el contenido del acumulador para verificar el funcionamiento.

Indicaciones

Antes de iniciar este ejercicio será preciso tener a mano la referencia del teclado del uP-2000, si aún no se conocen de memoria todas sus funciones, y la referencia de instrucciones del 8085.

El ejercicio puede dividirse en las siguientes tareas individuales:

1. Escribir en papel el programa indicado y, sirviéndose de la referencia de instrucciones del 8085, traducirlo a código máquina, obteniendo la secuencia de bytes que habrá que introducir en el uP-2000.

2. Determinar cuál es la dirección de inicio de la RAM en el uP-2000. Está indicado en la serigrafía de la propia tapa transparente del equipo.

3. Usando el teclado y la pantalla del uP-2000 introducir el programa en memoria, ejecutarlo y comprobar el valor del acumulador cuando termine.

Debe tenerse en cuenta que la última instrucción de cualquier programa que vaya a ejecutarse en el uP-2000, aquella que devolverá el control al programa monitor del sistema, debe ser **RST 1**.

Cuestiones

1. ¿Qué efecto tiene la ejecución de la instrucción **RST 1**? ¿Qué consecuencia tendría su omisión?

2. ¿Habría que modificar el programa si se cambiase de posición en la memoria, alojándolo en la dirección **1400H** en lugar de **1000H**?

Solución

1. Escribir el programa que, en este caso, se compondría de dos instrucciones:

```
MVI A,0A5H
RST 1
```

2. Buscar en la hoja de instrucciones del 8085 **MVI A,n** para hallar su código de operación: **3EH**. Tras éste se colocaría el operando que acompaña a la instrucción: **A5H**.

3. De igual forma hallar el código de operación de la instrucción **RST 1**: **CFH**.

4. La secuencia de bytes que componen el programa es, por tanto, la siguiente: **3EH A5H CFH**.

5. Examinar la serigrafía de la tapa del uP-2000. Hay dos módulos de memoria RAM, siendo la dirección inicial la **1000H**.

6. Pulsar la tecla **INIC** para inicializar el uP-2000.

7. Pulsar la tecla **S.ME ANT** e introducir la dirección **1000H**, terminando con la tecla **POST**.

8. Introducir el primer byte del programa: **3EH**.

9. Pulsar la tecla **POST** para confirmar el cambio y avanzar a la posición de memoria siguiente.

10. Repetir los pasos 8 y 9 para los dos bytes siguientes.

11. Pulsar la tecla **EJEC** para salir de la función. En este momento el programa ya está en memoria.

12. Pulsar la tecla **GO** e introducir la dirección donde se encuentra el programa: **1000**H.

13. Pulsar la tecla **EJEC** para iniciar la ejecución. Al terminar debe aparecer el habitual mensaje **- 8085** en la pantalla del uP-2000.

14. Pulsar la tecla **E.REG** y a continuación la tecla **A** para examinar el contenido del acumulador, que debe ser **A5H**.

6.3.1 Ejercicios propuestos

Ejercicio 6.3.1.1

Lance la ejecución del proceso de inicialización del 8085 como lo haría la tecla **INIC** en caso de que el microinterruptor **TEC**, que otorga el control al teclado del uP-2000 o al PC conectado vía transmisión serie, se hubiese cambiado para dar el control al PC, pero sin manipular dicho microinterruptor.

Indicaciones

En el ejercicio 4 se obtenían, tras examinar las primeras posiciones de memoria a partir de la dirección **0000**, una serie de instrucciones, entre ellas un salto condicional en caso de que el microinterruptor **TEC** se encontrase en la posición que otorga el control al PC.

Cuestiones

1. ¿En qué dirección se encuentra la rutina de inicialización del uP-2000 cuando se otorga el control al PC?

2. ¿Qué ocurre al ejecutar dicho proceso de inicialización, a pesar de no haber modificado la posición del microinterruptor **TEC**?

Ejercicio 6.3.1.2

Escriba un programa que copie en el registro **C** el contenido del acumulador y a continuación incremente el valor del registro **C**. Ejecútelo varias veces asignando desde el teclado del uP-2000 distintos valores al acumulador y observe los resultados.

Indicaciones

Siga las mismas indicaciones dadas para el ejercicio previo.

Cuestiones

1. ¿Qué indicadores se activan cuando se asigna al acumulador el valor **1FH** y después se ejecuta el programa?

2. ¿Qué ocurre si se asigna el valor **FFH** al acumulador? ¿Qué valor queda en el registro **C** y qué señalan los indicadores del registro de estado?

3. ¿Sería posible tener más de un programa simultáneamente en la memoria del uP-2000?

Ejercicio 6.3.1.3

Modifique el programa del ejercicio previo para que evite el desbordamiento del registro **C**.

Indicaciones

Este ejercicio, al igual que muchos otros, tiene distintas soluciones. Las dos más obvias serían las siguientes:

1. Comprobar si el valor del acumulador es ya **FFH**, no incrementando el registro **C** en ese caso.

2. Comprobar si se ha activado el indicador **Z** tras haber incrementado **C** y, en caso afirmativo, devolver a este registro el valor **FFH.**

En cualquiera de los dos casos deberá efectuarse un salto condicional, por lo que es preciso calcular correctamente la dirección de destino.

Cuestiones

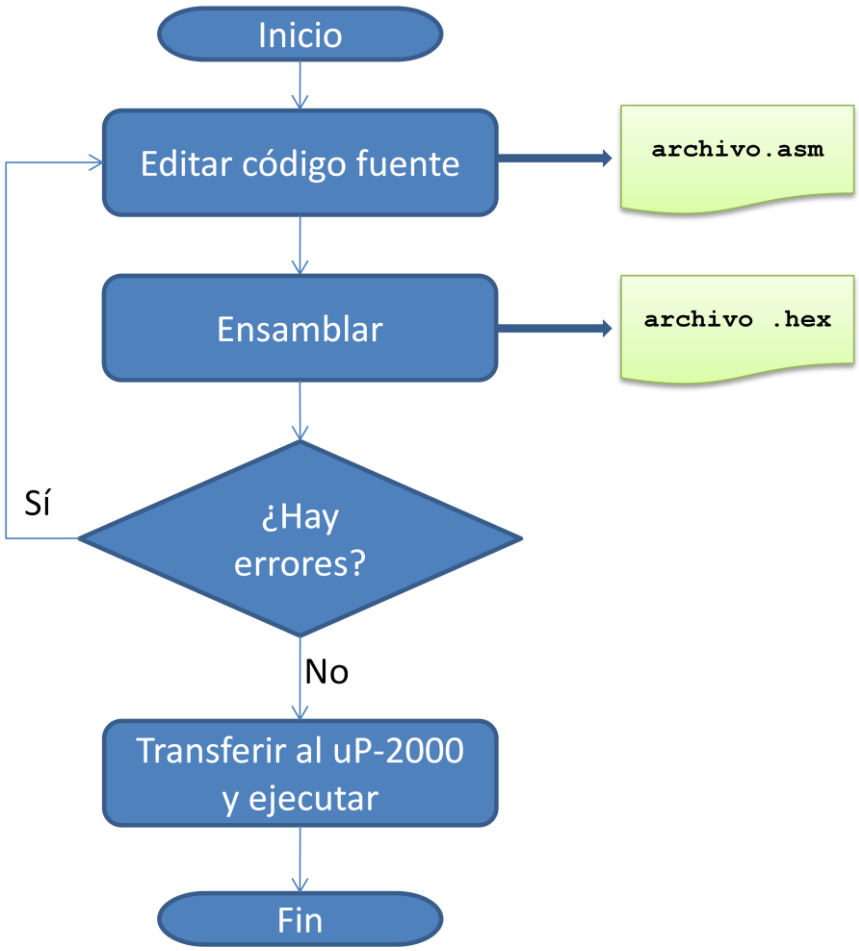
1. ¿Qué cambios habría que efectuar para situar este programa a partir de la dirección **1500H** en lugar de la **1000H**?

7 Ensamblado y comunicación con el uP-2000

La traducción a mano de los programas en ensamblador 8085 a código máquina, y su posterior introducción en la memoria del uP-2000 escribiendo byte a byte dicho código, es un proceso tedioso y que requiere mucho tiempo. Es fundamental conocerlo, pero no resulta práctico cuando los programas comienzan a tener cierta complejidad y lo que importa, en realidad, es conocer el conjunto de instrucciones que ofrece el microprocesador y saber cómo utilizarlas para resolver los problemas que sea preciso resolver.

El proceso de traducción de un código fuente a código objeto se denomina *compilación* y lo lleva a cabo un programa llamado *compilador*. Sin embargo, cuando el código fuente es el lenguaje ensamblador de un cierto microprocesador al proceso de traducción se le llama *ensamblado* y la propia herramienta, el programa que interpreta el código fuente y genera el código objeto, se denomina *ensamblador*.

El objetivo de esta sección, dividida en dos apartados, es adquirir los conocimientos necesarios para poder escribir programas en ensamblador 8085, ensamblarlos para obtener un archivo con el código objeto, transferir el contenido de dichos archivos al sistema uP-2000 y, finalmente, ejecutar esos programas para comprobar el resultado. El proceso se resume en el ordinograma siguiente:



Ordinograma del proceso de ensamblado y ejecución.

7.1 Estructura de un programa 8085

Un programa en ensamblador se escribe usando las instrucciones de un microprocesador determinado, en este caso el 8085. Además las instrucciones van en muchos casos acompañadas de datos inmediatos, valores como pueden ser números o caracteres, y direcciones de memoria correspondientes a subrutinas, posiciones en las que se quiere escribir o leer un dato, etc.

Ciertas instrucciones, como pueden ser las de salto o bifurcación (tipo `JMP`), desvían la ejecución a un punto del programa cuya dirección es necesario calcular de manera relativa a la posición de memoria en la que vaya a alojarse dicho programa. Tomando como punto de partida el código mostrado a continuación, en el que se mueve un dato al acumulador, a continuación se comprueba si es el valor `0FFH` y, en caso afirmativo, se desvía la ejecución a la dirección `1007H` para evitar el incremento, cabe hacerse la siguiente pregunta: **¿en qué dirección habría que cargar este programa para que funcionase correctamente?**

```
1. MOV A,C
2. CPI 0FFH
3. JZ 1007H
4. INR C
5. RST 1
```

Puesto que la instrucción que debe quedar en la dirección `1007H` es la de la línea 5, habría que ir restando a dicha dirección lo que ocupan las instrucciones previas hasta llegar a la primera, obteniendo la dirección `1000H`. **¿Qué ocurriría si se realojase el programa en otra posición de memoria?** Sería preciso recalcular dónde queda la instrucción de la línea 5 y modificar el salto de la línea 3.

7.1.1. La directiva ORG y el uso de etiquetas

El programa ensamblador, que se encarga de traducir cada instrucción a su correspondiente secuencia de códigos de operación, puede encargarse de calcular automáticamente las direcciones de memoria en las que se alojan las instrucciones, haciendo innecesarias los ajustes antes citados cada vez que se introduzcan modificaciones en el programa. Para ello es necesario cumplir las dos condiciones siguientes:

1. El programa debe contener una línea con la palabra `ORG` seguida de la dirección en la que se pretende alojar el programa en memoria, por ejemplo `ORG 1000H`. De esta forma el ensamblador sabrá cuál es la dirección base para realizar los cálculos.

2. Las instrucciones a las que sea necesario saltar en algún momento tendrá que ir precedidas de una *etiqueta*, un identificador de seis caracteres como máximo terminado con dos puntos. Esa etiqueta será la que se emplee en las instrucciones de salto en lugar de las direcciones.

Teniendo en cuenta estas premisas, el programa anterior podría rescribirse de la siguiente forma:

```
1. ORG 1000H
2. MOV A,C
3. CPI 0FFH
4. JZ FIN
5. INR C
6. FIN: RST 1
```

La etiqueta `FIN:` indica la posición del programa cuya dirección hay que calcular, en este caso la de la línea 6, dirección que se colocará en todas las apariciones de la etiqueta sin los dos puntos. No importa qué dirección sea la usada para alojar el programa, ni que entre la instrucción de salto y el destino se agreguen o eliminen instrucciones, incidencias ambas que sin disponer de un programa ensamblador forzaría al recálculo manual de la dirección.

CONCEPTOS

Se denominan *directivas* aquellas órdenes que se introducen en un programa ensamblador pero que no son instrucciones dirigidas al microprocesador, sino indicaciones para el programa encargado de ensamblar. `ORG` es la directiva que establece la dirección en la que se alojarán las instrucciones escritas a continuación. Nada impide emplear varias directivas de este tipo en un mismo programa, de forma que distintas partes del programa se coloquen en diferentes áreas de memoria.

7.1.4 Inicio y fin de un programa ensamblador

Todo programa que vaya a ser ensamblado con el programa `C16`, que es el ensamblador utilizado en las prácticas de la asignatura, y a ejecutarse en el sistema uP-2000 debe ajustarse a una estructura concreta que se resume en el siguiente fragmento de código:

```
CPU "8085.TBL"
HOF "INT8"
...
ORG inicio
...
RST 1
END
```

El significado de cada una de estas líneas es el explicado a continuación:

- `CPU "8085.TBL"`: El programa `C16` es en realidad un ensamblador cruzado o *cross-assembler*, lo que significa que puede generar código para distintos procesadores. Con la directiva `CPU` se le indica el archivo en el que se encuentra la tabla

de instrucciones y códigos de operación del microprocesador a emplear, en este caso `8085.TBL`.

- `HOF "INT8"`: Por la misma razón anterior, el hecho de que `C16` sea un ensamblador cruzado, es necesario indicar el formato de salida del código objeto. Para poder transferir los programas al uP-2000 dicho formato es `INT8` y se especifica con la directiva `HOF`.

- `ORG inicio`: La directiva explicada anteriormente para indicar la dirección en la que se alojarán las instrucciones que le sigan en el programa.

- `RST 1`: Ésta no es una directiva sino una instrucción del 8085, encargada de devolver el control al programa monitor del uP-2000 cuando se llega al final de la ejecución del programa. Este modo de finalización garantiza que no se modifiquen los valores de los registros del microprocesador, lo cual permitirá su inspección posterior.

- `END`: Es una simple directiva que indica al programa ensamblador que el programa concluye en ese punto.

Retomando el ejemplo anterior, en el que se introdujeron la directiva `ORG` y una etiqueta, el programa completo, listo para ser ensamblado, quedaría de la siguiente forma:

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

MOV A,C    ; Vamos a examinar el valor de C
CPI 0FFH
JZ FIN     ; Terminar si ha llegado a 255
INR C
FIN: RST 1
END
```

En esta última versión se han agregado también comentarios. Éstos se inician con el carácter "`;`" y continúan hasta el final de la línea, siendo recomendable su uso siempre que se considere necesario aclarar algún paso del programa.

Para escribir los programas puede recurrirse a cualquier editor de texto simple, debiendo guardarse en archivos con extensión **`.asm`**.

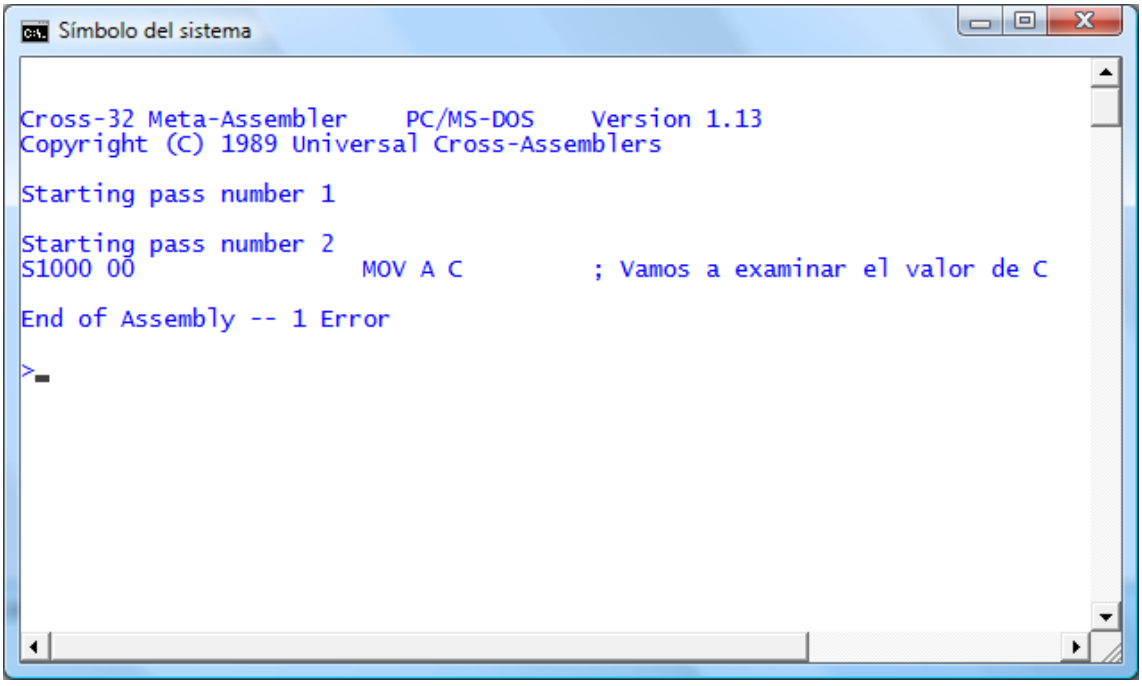
7.1.3 Ensamblado del programa

Una vez que se tenga escrito el programa en lenguaje ensamblador 8085, según la sintaxis del ensamblador `C16` descrita en el punto previo, llega el momento de ensamblarlo a fin de obtener el mismo programa en un formato que pueda ser transferido y ejecutado en el uP-2000. Para ello habrá que reproducir los pasos siguientes:

- Abrir una consola "Símbolo del sistema" y cambiar al directorio en el que se encuentra el software del uP-2000 que, habitualmente, será `C:\UP2000`.

- Si es necesario, crear el archivo `.asm` con el código fuente. Para ello puede utilizarse desde la propia consola el programa `EDIT`, sencillamente escribiendo `EDIT nombre.asm`.

- Ensamblar el programa, escribiendo en la línea de comandos `ensambla nombre`, sin incluir la extensión `.asm`. Si se encuentra algún fallo durante el proceso de ensamblado, aparecerán en pantalla las líneas donde se han encontrado los errores y un resumen al final, como ocurre a continuación. En este caso el fallo es que falta la coma de separación en la instrucción `MOV A C`.



Ensamblado del programa e indicación de errores.

Si el programa se ensambla sin problemas, se habrá creado un archivo con el mismo nombre que el facilitado a `ensambla` pero con extensión `.hex`. Dicho archivo es el que contiene el código a enviar al sistema uP-2000, la secuencia de bytes que representan los códigos de operación y datos utilizados en el programa.

7.2 Comunicación entre PC y uP-2000

Para transferir el código objeto desde el archivo generado por el ensamblador, y que se encuentra almacenado en el PC, hasta el sistema uP-2000, que es donde se ejecutará, será preciso comunicar ambos dispositivos. Para ello lo primero que hay que verificar es que el cable serie diseñado para esa comunicación se encuentra conectado en ambos extremos, en el lado del PC mediante un conector estándar DB-9 (serie, 9 pines) y en el lado del uP-2000 con un conector de tres hilos en la entrada **J8**, situado en el margen derecho del equipo.

Lo siguiente será configurar el uP-2000 para que atienda al puerto de comunicación con el PC en lugar de al teclado, desplazando a la derecha el microinterruptor `TEC` (véase la figura 29, en la descripción del sistema uP-2000).

Asumiendo que los elementos físicos están preparados como acaba de describirse, la transmisión del programa desde el PC al uP-2000 se llevará a cabo con el siguiente procedimiento:

- Desde la consola "Símbolo del sistema", y estando en el directorio que contiene el archivo con el código fuente, se ejecuta el programa `ddt85`.

- El programa preguntará qué línea serie va a utilizarse para establecer la comunicación con el uP-2000. Dicha línea generalmente será `COM1`, por lo que habrá que pulsar la tecla `0`.

- A continuación hay que pulsar la tecla **`INIC`** del uP-2000. Si la comunicación entre los dos dispositivos se establece satisfactoriamente aparecerá en la pantalla del PC la indicación "MONITOR 8085".

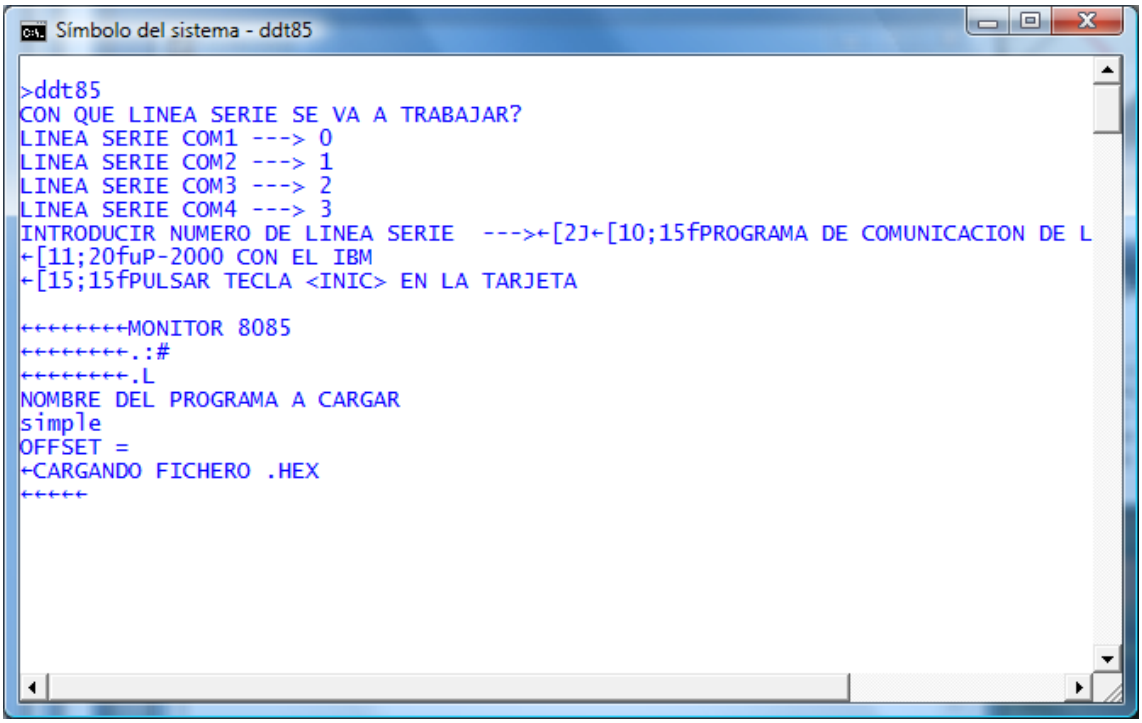
- Pulsando la tecla `L` se indica al programa monitor que se desea transferir un programa.

- El programa pide el nombre del programa que, como al ensamblar, debe ser introducido sin especificar la extensión, ya que se asume que ésta es `.hex`.

- Cuando aparezca la indicación "OFFSET=" pulsar la tecla **Intro** para continuar.

- Finalmente pulsar la tecla `H` para finalizar. El programa deberá encontrarse ya en la memoria del sistema uP-2000.

La pantalla siguiente muestra el proceso que acaba de describirse, mediante el que se ha cargado en el uP-2000 el programa contenido en el archivo `simple.hex`.



Proceso de comunicación entre PC y uP-2000.

Ahora ya puede invertirse la posición del microinterruptor `TEC`, para devolver el control al teclado, pulsar la tecla **`INIC`** del uP-2000 y ejecutar el programa.

El procedimiento para poner en práctica los ejercicios de la siguiente sección será siempre el mismo, el que acaba de describirse, editando y ensamblando desde el PC, transfiriendo el código al sistema uP-2000 y ejecutando en éste los programas. Puesto que ya se han descrito todos los pasos necesarios para trabajar con el software en el PC y manejar el teclado del uP-2000, éstos son dos aspectos sobre los que no se volverá a entrar en detalles.

8 Ejercicios de programación

Conociendo ya todos los aspectos teóricos relativos a la estructura del microprocesador 8085, así como algunos de los módulos que le acompañan en el sistema uP-2000, y disponiendo de todas las herramientas precisas para escribir programas, ensamblarlos y transferirlos a dicho sistema para ser ejecutados, llega el momento de iniciar el trabajo práctico. Para ello se proponen en esta sección un extenso conjunto de ejercicios, algunos resueltos y otros propuestos para resolver por el alumno. Estos ejercicios se han estructurado en los conjuntos siguientes:

1. **Acceso a memoria**: Son ejercicios sencillos cuya finalidad es la de familiarizarse con las distintas instrucciones que permiten recuperar datos y almacenar datos en la memoria del sistema.

2. **Implementación de bucles**: Salvo en los casos más simples, prácticamente cualquier programa de una determinada aplicación necesitará repetir la ejecución de conjuntos de instrucciones, usando para ello saltos condicionales. Es el tema sobre el que versan los ejercicios de este bloque.

3. **Operaciones aritméticas**: En el tercer bloque se proponen ejercicios que permitirán aprender a realizar sumas, restas, multiplicaciones y divisiones sobre operandos de distintos tamaños, 8 ó 16 bits, empleando las distintas instrucciones del 8085.

4. **Trabajo a nivel de bits**: Al programar en ensamblador un sistema como el uP-2000 en muchas ocasiones será necesario controlar dispositivos a un nivel muy bajo, manipulando o comprobando bits individuales. Éste es el objeto de los ejercicios de este apartado.

5. **Búsqueda de datos**: Los ejercicios de este quinto bloque aprovechan lo aprendido en los previos para abordar propuestas algo más elaboradas, en las que se combinan bucles, saltos y otros elementos a fin de localizar, contar y sustituir datos.

6. **Estructuración del código**: A medida que los programas que se pretende escribir vayan ganando en complejidad, surgirá espontáneamente la necesidad de dividirlos en trozos según su funcionalidad, creando subrutinas. Con los ejercicios de este bloque se conocerán las técnicas necesarias para escribir subrutinas, invocarlas desde un programa principal y facilitar la comunicación entre ambas partes.

7. **Ordenar datos**: En este séptimo bloque se mostrará cómo el uso conjunto de subrutinas y otras técnicas aprendidas en ejercicios precedentes, como los bucles o la búsqueda de datos, permite implementar tareas aparentemente complejas, como puede ser la ordenación de una lista de datos.

8. **Lectura del teclado y visualización en el uP-2000**: El sistema uP-2000 dispone de una pantalla, que facilita la visualización de datos, y un teclado que permite la instrucción. Los ejercicios de esta sección muestran cómo acceder a esos elementos desde programas escritos en ensamblador, de forma que el resultado de los programas pueda ser expuesto directamente, y no sea necesario inspeccionar posiciones de memoria para obtenerlo, y de igual forma pueda obtenerse información desde el teclado.

9. **Introducción de retardos**: Por regla general los programas se ejecutan en un microprocesador a una velocidad que resulta inadecuada cuando es necesario interactuar con un usuario, por ejemplo mostrando o solicitando datos, siendo por ello precisa la introducción de retardos. Éste es el tema del noveno bloque de ejercicios.

10. **Uso del PPI**: De los distintos dispositivos de E/S que componen el sistema uP-2000, y que permiten al 8085 comunicarse con el exterior, uno de los más interesantes es el 8255, conocido como PPI. Los ejercicios de este bloque demuestran cómo se puede usar dicho dispositivo para realizar tareas como la iluminación de leds o la lectura de microinterruptores.

11. **Interrupciones**: En el último bloque de ejercicios se aborda uno de los temas más complejos, y al tiempo útiles, del 8085: las interrupciones. El objetivo es aprender a crear programas preparados para atender a una solicitud de interrupción.

Dentro de cada grupo se comienza siempre con ejercicios sencillos, de los cuales se aporta la solución, que servirán como base para construir otros más elaborados cuya realización queda propuesta.

8.1 Acceso a memoria

Los microprocesadores en general, y el 8085 en particular al ser un microprocesador de 8 bits, tienen una limitada capacidad de almacenamiento, por lo que los datos sobre los que va a trabajarse tienen que transferirse desde y a la memoria de manera casi continua, de ahí la importancia de conocer las instrucciones que facilitan esas transferencias. El propósito de los ejercicios que se plantean en esta sección es adquirir dicha familiaridad. Para ello se plantean los siguientes objetivos:

- Aprender a recuperar datos de 8 bits mediante la instrucción `LDA`.
- Familiarizarse con el uso de parejas de registros como punteros a memoria para la recuperación de datos.
- Aprender a escribir datos de 8 bits mediante la instrucción `STA`.
- Usar parejas de registros como punteros para modificar el contenido de posiciones de memoria.
- Practicar la utilización conjunta de operaciones de lectura y escritura para copiar datos entre zonas de memoria.
- Familiarizarse con los formatos *little endian* y *big endian* a la hora de recuperar datos de 16 bits.
- Aprender a combinar distintas instrucciones de acceso a memoria, como `LHLD`, `SHLD` y `LDAX`, con otras como `XCHG` y `MOV` para leer y escribir datos de 8 y 16 bits.
- Conocer la forma en que pueden tratarse datos de más de 16 bits con el 8085, usando varios pares de registros e instrucciones introducidas en los ejercicios previos.
- Introducir el uso de la pila con el objetivo de hacer ver que cada problema puede tener más de una solución.

8.1.1 Lectura de datos de 8 bits

Ejercicio 8.1.1.1

Modificando exclusivamente el acumulador, recupere el contenido de la posición de memoria 20F1H y compruebe cuál es.

Indicaciones

Para resolver los primeros ejercicios resultará imprescindible tener a mano la tabla de instrucciones del 8085, en este caso para localizar la instrucción que permita recuperar el contenido de una cierta posición de memoria y llevarlo hasta el acumulador.

La resolución de este ejercicio se compone de dos pasos:

1. Leer la posición de memoria indicada y llevar su contenido al acumulador.

2. Comprobar en el uP-2000 cuál es el valor almacenado en el acumulador tras la ejecución del programa.

Cuestiones

1. ¿A qué elemento del sistema uP-2000 corresponde la dirección de memoria leída?

2. ¿Sería posible escribir un programa que, sin usar la instrucción `LDA`, efectuase la misma tarea manteniendo las mismas condiciones?

3. ¿Qué tipo de direccionamiento se ha utilizado para recuperar el dato desde la memoria y llevarlo al acumulador?

4. ¿Qué valor contiene el acumulador tras ejecutar el programa?

Solución

```
CPU "8085.TBL"
HOF "INT8"

ORG 1000H

LDA 20F1H  ; Lectura de la posición 201FH

RST 1

END
```

Ejercicio 8.1.1.2

Sin utilizar la instrucción **LDA**, lleve al acumulador el contenido de la posición de memoria 20F1H modificando los registros que necesite a excepción del par **HL**.

Indicaciones

Para completar este ejercicio será necesario:

1. Identificar la instrucción o instrucciones del 8085, a excepción de `LDA`, que permiten leer de la memoria.

2. Preparar la ejecución de la instrucción elegida, asignando a la pareja de registros que corresponda la dirección a la que va a accederse.

3. Leer el contenido de esa posición de memoria.

Cuestiones

1. ¿Qué parejas de registros pueden utilizarse con la instrucción `LDAX`? ¿Y con la instrucción `LXI`?

2. Al cargar la dirección `20F1H` en una pareja de registros, con la instrucción `LXI`, ¿qué parte de la dirección queda en cada registro?

3. ¿Cómo reescribiría este programa sin utilizar la instrucción `LXI`?

Solución

```
CPU "8085.TBL"
HOF "INT8"

ORG 1000H

LXI B, 20F1H     ; Cargar la dirección en BC
LDAX B           ; Llevar al acumulador el dato
                 ; almacenado en esa posición
RST 1

END
```

8.1.2 Modificación de datos de 8 bits

Ejercicio 8.1.2.1

Modificando exclusivamente el acumulador, escriba el valor **0CDH** en la posición de memoria **1100H**.

Indicaciones

Para resolver los primeros ejercicios resultará imprescindible tener a mano la tabla de instrucciones del 8085, en este caso para localizar la instrucción que permita almacenar el contenido del acumulador en una cierta posición de memoria.

La resolución de este ejercicio se compone de dos pasos:

1. Asignar al acumulador el dato que se pretende escribir en la memoria.

2. Escribir el contenido del acumulador en la posición de memoria indicada.

Cuestiones

1. ¿A qué elemento del sistema uP-2000 corresponde la dirección de memoria en la que se ha escrito el dato?

2. ¿Qué ocurre si se intenta escribir en direcciones correspondientes al rango 0000H–0FFFH?

3. ¿Se ve modificado el acumulador tras ejecutar la instrucción que lleva su contenido a la memoria?

Solución

```
CPU "8085.TBL"
HOF "INT8"

ORG 1000H

MVI A, 0CDH      ; Cargar en el acumulador el valor
STA 1100H        ; y escribirlo en la memoria

RST 1

END
```

Ejercicio 8.1.2.2

Usando exclusivamente las instrucciones **LDA** y **STA** del 8085, efectúe una copia de los datos contenidos en la dirección **0100H** y las tres siguientes (cuatro bytes en total) a partir de la posición de memoria **1100H**.

Indicaciones

Para completar este ejercicio no hay más que leer de las posiciones de memoria de origen y escribir en las de destino, usando para ello dos instrucciones que ya se conocen.

Cuestiones

1. ¿Resulta práctico este mecanismo de copia de datos entre zonas de memoria para bloques grandes?

2. ¿Se podría generalizar el programa para que copiase un número variable de posiciones de memoria, por ejemplo leyéndolo de una cierta dirección donde se haya escrito antes de ejecutar el programa?

Solución

```
CPU "8085.TBL"
HOF "INT8"

ORG 1000H

LDA 100H   ; Se lee el contenido de 100H
STA 1100H  ; y se lleva a 1100H

LDA 101H   ; repitiendo hasta completar el
STA 1101H  ; el número de posiciones indicado

LDA 102H
STA 1102H

LDA 103H
STA 1103H

RST 1

END
```

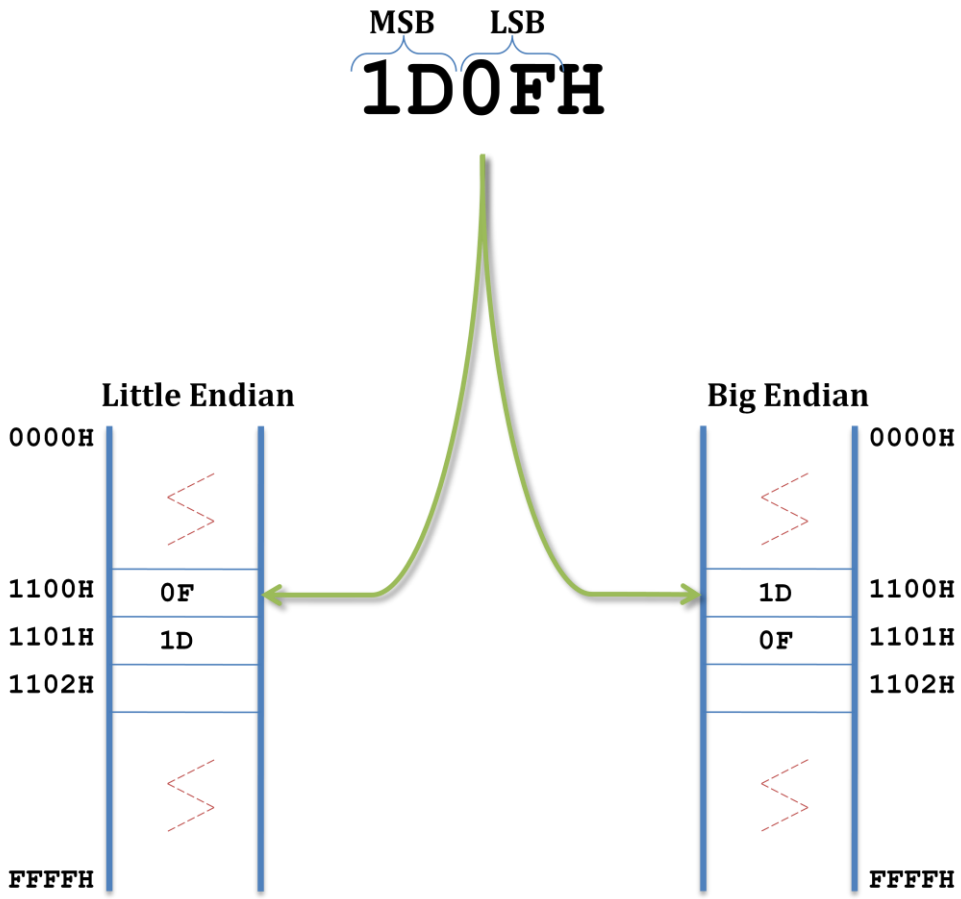
8.1.3 Lectura de datos de 16 bits

Ejercicio 8.1.3.1

En la dirección de memoria **1100H** se ha almacenado un dato de 16 bits (una dirección) en formato *little endian* que debe ser recuperado en la pareja de registros **HL**, usando a continuación este registro para llevar al acumulador el dato que haya en dicha dirección.

Indicaciones

Dado que en cada posición de memoria solamente puede almacenarse un byte, ésta es la unidad de medida básica incluso para las memorias más modernas, cuando se necesita trabajar con datos de mayor tamaño es necesario dividirlos en partes. Un dato de 16 bits, como puede ser una dirección de memoria, se dividiría en dos partes de 8 bits, almacenándose esos dos bytes en dos posiciones de memoria consecutivas. Dependiendo del orden en que se almacenen esos dos bytes se dice que el dato está almacenado en formato *little endian* o *big endian*. La figura siguiente representa las diferencias entre ambos formatos:



Almacenamiento de datos *little endian* y *big endian*.

El formato *little endian* es el que suele utilizarse en procesadores Intel y consiste en almacenar el byte de menor peso o LSB seguido del MSB. Es decir, el byte menos significativo está en la dirección de memoria más baja y el más significativo en la más alta. En el formato *big endian* el orden es el inverso, siendo el utilizado en procesadores de Motorola e IBM, por ejemplo.

Conociendo la ordenación de los bytes en la memoria, para completar este ejercicio han de reproducirse los pasos indicados a continuación:

1. Recuperar en `H` el contenido de la posición de memoria `1101H` y en el registro `L` el contenido de la posición `1100H`. De esta forma se tendrá en `HL` la dirección `1D0FH`. El 8085 dispone de una instrucción que realiza esta operación.

2. Usar `HL` como puntero y mover el dato contenido en la dirección a la que apunta al acumulador.

3. Tras ensamblar el programa y transferirlo al uP-2000, ejecutarlo varias veces modificando la dirección almacenada en `1100H`.

Cuestiones

1. ¿A la vista de cómo funciona la instrucción `LHLD`, qué formato de almacenamiento en memoria utiliza el microprocesador 8085?

2. ¿Qué ocurre si ejecuta este programa sin haber almacenado previamente una dirección en las posiciones `1000H` y `1001H`?

Solución

```
CPU "8085.TBL"
HOF "INT8"

ORG 1000H

LHLD 1100H ; Se recupera en HL la dirección
MOV A, M   ; y se lee el dato que contiene

RST 1

END
```

Ejercicio 8.1.3.2

Escriba un programa que realice la misma función que el del ejercicio 8, pero evitando el uso de la instrucción **MOV**. Puede modificar todos los registros que necesite.

Indicaciones

Recorriendo el conjunto de instrucciones del 8085, para analizar las distintas posibilidades a la hora de transferir datos desde la memoria al microprocesador, podemos encontrarnos con los casos siguientes:

1. La instrucción `LDA` permite leer el contenido de cualquier posición de memoria y llevarlo al acumulador, pero es necesario indicar la dirección a leer de forma inmediata. No sirve, puesto que en el ejercicio se indica que la dirección no es conocida en el momento de escribir en el programa, sino que éste ha de recuperarla de las posiciones `1100H` y `1101H`.

2. La instrucción `LHLD` permite recuperar la dirección, pero el uso de `HL` como puntero implica el uso de la instrucción `MOV` que se pretende evitar.

3. Para recuperar datos de la memoria usando una pareja de registros como puntero puede usarse la instrucción `LDAX`, pero solamente es posible emplearla con `BC` y `DE`, no con `HL`.

Teniendo en cuenta todas estas premisas, el ejercicio se completaría de acuerdo al siguiente guión:

1. Recuperar la dirección de la que va a leerse el dato, utilizando para ello la instrucción `LHLD`.

2. Puesto que no es posible usar `HL` como puntero, llevar el contenido de `HL` a `BC` o `DE`.

3. Leer el dato mediante la instrucción `LDAX`.

Cuestiones

1. ¿Habría alguna forma de intercambiar las parejas de registros `HL` y `DE` sin usar instrucciones `MOV` ni la instrucción `XCHG`?

2. De existir esa alternativa, ¿podría ser utilizada sin más en el programa resultado del ejercicio? ¿Por qué?

Solución

```
CPU "8085.TBL"
HOF "INT8"

ORG 1000H

LHLD 1100H ; Se lee la dirección en HL
XCHG       ; Intercambiar HL con DE
LDAX D     ; y se lee el dato apuntado por DE

RST 1

END
```

8.1.4 Modificación de datos de 16 bits

Ejercicio 8.1.4.1

Leer el dato de 16 bits almacenado en las posiciones de memoria **1100H** y **1101H** y escribirlo en las posiciones **1102H** y **1103H**.

Indicaciones

La instrucción `LHLD` tiene una complementaria: la instrucción `SHLD`. Usando ambas el ejercicio puede resolverse de manera inmediata.

Cuestiones

1. ¿Cuántas operaciones de transferencia de datos desde y hacia memoria implica la ejecución de las instrucciones `SHLD` y `LHLD` en este programa? ¿Representan en este sentido alguna ventaja respecto a instrucciones como `LDAX`/`STAX`?

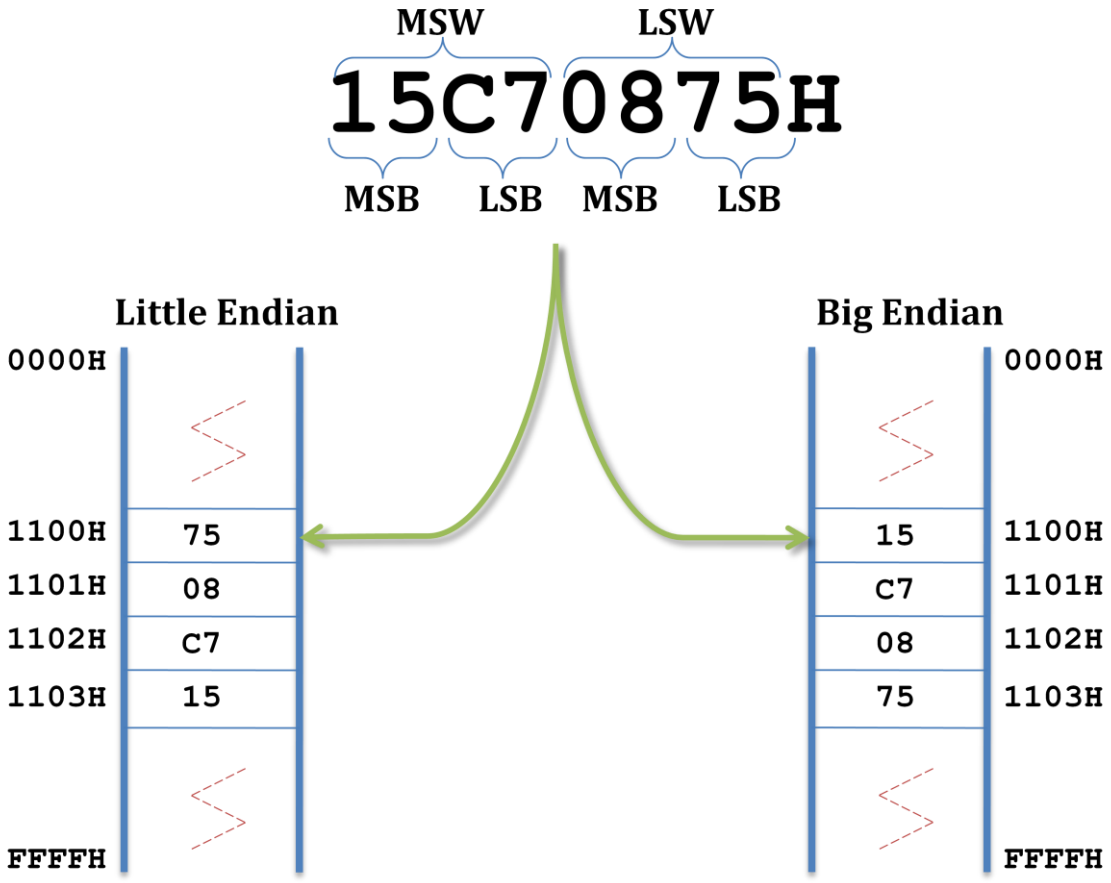
Solución

```
CPU "8085.TBL"
HOF "INT8"

ORG 1000H

LHLD 1100H ; Se lee el dato
SHLD 1102H ; y se escribe en el nuevo destino

RST 1

END
```

8.1.5 Trabajar con datos de más de 16 bits

El bus de datos del 8085 tiene un ancho de 8 bits, siendo ésta también la capacidad de la mayoría de los registros internos de este microprocesador. Varios de esos registros, sin embargo, pueden ser utilizados en parejas para trabajar con datos de 16 bits, como ha podido verse en varios ejercicios de esta sección. De manera análoga, aunque el 8085 no ofrece instrucciones concretas para ello, pueden utilizarse "parejas de parejas de registros" para trabajar con datos de 32 bits.

El aspecto más importante a la hora de trabajar con datos que ocupan varios bytes de memoria, por ejemplo un número de 32 bits, es conocer la estructura de ese dato, es decir, el formato que ha de seguirse a la hora de interpretarlo. Los formatos más usuales son los ya mencionados *big endian* y *little endian* que, al aplicarse sobre números de 32 bits, dan lugar a una primera división en dos trozos de 16 bits (MSW y LSW), cada uno de los cuales se partiría en dos bytes (MSB y LSB). El esquema siguiente muestra esta división y la forma en que se almacenaría el dato en la memoria en *little endian* y *big endian*.



Almacenamiento de datos de 32 bits.

Como puede apreciarse, en formato *little endian* los bytes que forman el dato se almacenan en orden inverso, mientras que en formato *big endian* el orden es el mismo y, por tanto, resulta más natural. El 8085, como ya sabe, está preparado para usar el primer formato y no el segundo.

Ejercicio 8.1.5.1

Examine el código del programa siguiente y responda a las cuestiones propuestas. A continuación ejecútelo en el sistema uP-2000 y verifique las respuestas dadas.

```
CPU "8085.TBL"
HOF "INT8"

ORG 1000H

LXI H, 15C7H
LXI D, 0875H
LXI SP, 1104H

PUSH H
PUSH D

RST 1

END
```

Cuestiones

1. ¿Cuál será el valor de las parejas de registros `HL` y `DE` tras la ejecución del programa?

2. ¿Qué contenido tendrán las posiciones de memoria `1100H`, `1101H`, `1102H` y `1103H` tras ejecutar el programa?

3. ¿Cuál será el valor del puntero de pila una vez finalizada la ejecución?

4. ¿Qué supone que hace este programa a la vista de las cuestiones previas?

8.1.6 Ejercicios propuestos

Ejercicio 8.1.6.1

Sin utilizar las instrucciones **LDA** y **LDAX**, lleve al acumulador el contenido de la posición de memoria 20F1H modificando exclusivamente el acumulador y el par **HL**.

Indicaciones

Para completar este ejercicio será necesario:

1. Determinar qué instrucción permite recuperar datos de memoria usando la pareja de registros HL como puntero.

2. Almacenar en HL la dirección indicada.

3. Leer el contenido de esa posición de memoria.

Cuestiones

1. ¿Qué ventajas tiene el uso de HL como puntero, para recuperar datos de la memoria, respecto a la lectura directa con la instrucción LDA?

2. ¿Qué modos de direccionamiento ha utilizado en este programa?

Ejercicio 8.1.6.2

Sin utilizar la instrucción **STA**, escriba en la posición de memoria 1100H el valor **0CDH** modificando los registros que necesite a excepción del par **HL.**

Indicaciones

Para completar este ejercicio será necesario:

1. Identificar la instrucción o instrucciones del 8085, a excepción de `STA`, que permiten escribir en la memoria. Dichas instrucciones son `STAX` y `MOV M,R`.

2. Preparar la ejecución de la instrucción elegida, asignando a la pareja de registros que corresponda la dirección a la que va a accederse y cargando el acumulador con el valor a escribir.

3. Escribir el contenido del acumulador en esa posición de memoria.

Cuestiones

1. ¿Qué modos de direccionamiento se han utilizado en el programa?

Ejercicio 8.1.6.3

Sin utilizar las instrucciones **STA** y **STAX**, escriba el valor **0CDH** en la posición de memoria **1100H**, modificando exclusivamente el acumulador y el par **HL**.

Indicaciones

Para completar este ejercicio será necesario:

1. Determinar qué instrucción permite escribir datos en memoria usando la pareja de registros `HL` como puntero.

2. Almacenar en `HL` la dirección indicada y en el acumulador el dato a escribir.

3. Escribir el contenido del acumulador en esa posición de memoria.

Cuestiones

1. ¿Aparte del hecho de que se usen diferentes parejas de registros como puntero, cuál es la diferencia fundamental entre las instrucciones `STAX` y `MOV M,R`?

2. ¿Podría completarse este ejercicio sin modificar en ningún momento el contenido del acumulador?

Ejercicio 8.1.6.4

Escriba un programa que realice la misma función que el del ejercicio 8, modificando exactamente los mismos registros: **HL** y el acumulador, pero sin utilizar la instrucción **LHLD**.

Indicaciones

Los pasos para completar este ejercicio son los mismos que para el anterior, la única diferencia es que no puede utilizarse la instrucción `LHLD` y, además, se permite la modificación exclusivamente de `HL` y el acumulador. Hay que recordar que un dato de 16 bits se compone, en realidad, de dos partes de 8 bits que pueden ser tratadas independientemente.

Cuestiones

1. ¿Qué modificaciones habría que introducir en el programa si la dirección almacenada en `1100H` y `1101H` estuviese en formato *big endian*?

2. ¿Cómo se reescribiría el programa para seguir utilizando la instrucción `LHLD` pero con la dirección almacenada en formato *big endian*?

Ejercicio 8.1.6.5

En las posiciones **1100H** y **1101H** se ha almacenado una dirección en formato *big endian*. Escribir un programa que la recupere y vuelva a escribirla en la misma posición pero en formato *little endian*.

Indicaciones

Para completar este ejercicio puede tomarse como base el programa del ejercicio 11, no hay más que intercalar entre las instrucciones `LHLD` y `SHLD` las operaciones necesarias para cambiar de formato *big endian* a *little endian*.

Cuestiones

1. ¿Cómo cambiaría el programa para que convirtiese de formato *little endian* a *big endian*?

2. Escriba un enunciado (distinto al original) que describa la finalidad del programa obtenido.

Ejercicio 8.1.6.6

Se tiene en los registros **HL** (MSW) y **DE** (LSW) un dato de 32 bits, el número **15C70875H**, resultado de una operación previa, y se quiere almacenar en memoria en formato *little endian* a partir de la posición de memoria **1100H**.

Indicaciones

La realización de este ejercicio puede dividirse en los siguientes pasos:

1. Asignar a las parejas de registros HL y DE la parte que les corresponde del dato a tratar, según se indica en el propio enunciado del ejercicio.

2. Llevar a las posiciones de memoria 1100H y 1101H la LSW, contenida en DE.

3. Llevar a las posiciones de memoria 1102H y 1103H la MSW, contenida en HL.

Cuestiones

1. ¿Cuántos datos de 32 bits podrían mantenerse simultáneamente en los registros de uso general del 8085?

2. Efectúe manualmente las operaciones adecuadas, tomando como referencia el contenido de las posiciones de memoria 1100H a 1103H, para obtener el dato como número decimal (base 10).

8.2 Implementación de bucles

Salvo en los casos más simples, como los de los ejercicios propuestos en el apartado 8.1, las instrucciones de un programa raramente se ejecutan desde la primera a la última, secuencialmente, de forma que la ejecución genera siempre exactamente el mismo resultado. Es habitual que ciertas porciones del programa se ejecuten únicamente si se cumple una cierta condición, así como que ciertas partes del programa se ejecuten más de una vez.

Tanto la ejecución condicionada (condicionales) como la ejecución repetitiva (bucles) precisan de un mismo elemento: la alteración del flujo del programa, desviándolo en caso de que se cumpla o no se cumpla una condición. Los ejercicios que se proponen en esta sección permitirán conocer las instrucciones necesarias y su puesta en práctica. Los objetivos a alcanzar son los siguientes:

- Aprender a implementar un condicional simple, tipo `IF-THEN`, mediante saltos condicionales.
- Implementar un condicional compuesto, tipo `IF-THEN-ELSE`, mediante saltos condicionales.
- Aprender a escribir condicionales más complejos, con varias condiciones.
- Saber cómo usar saltos condicionales para implementar bucles por contador, repitiendo la ejecución de bloques de instrucciones.
- Practicar la implementación de bucles recurriendo a diferentes técnicas.
- Aprender a transferir bloques de datos entre zonas de la memoria del sistema usando bucles.
- Practicar la copia de bloques de datos usando métodos alternativos.
- Introducir el uso de una pareja de registros (16 bits) como contador de un bucle para poder ejecutar más de 256 ciclos.

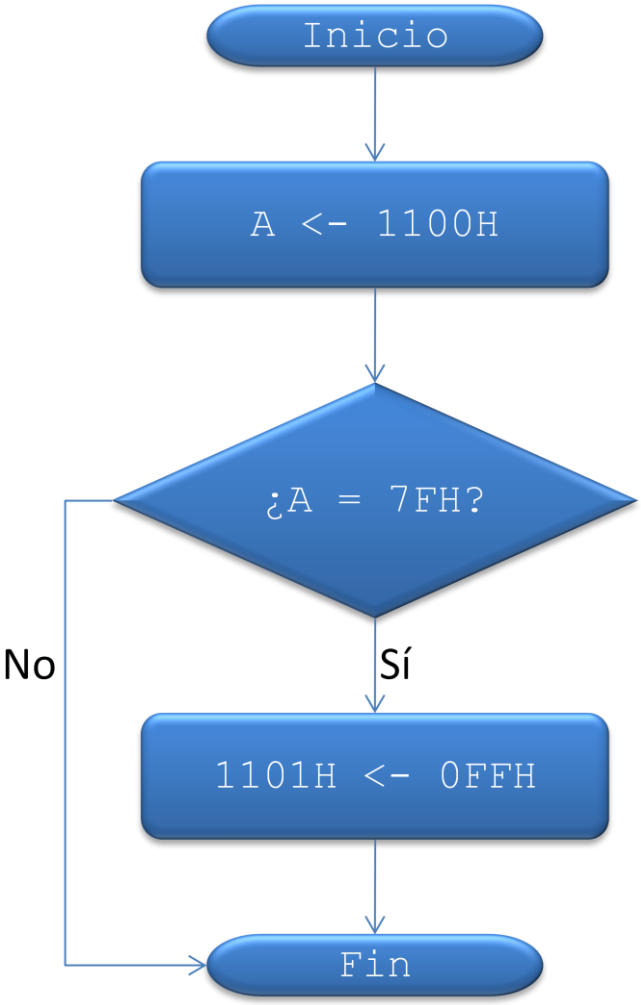
8.2.1 Saltos condicionales

Ejercicio 8.2.1.1

Leer el contenido de la posición de memoria **1100H** y, en caso de que el valor sea igual a **7AH**, escribir en la posición **1101H** el valor **0FFH**.

Indicaciones

El objeto de este ejercicio es simular la instrucción `IF-THEN` con la que cuentan todos los lenguajes de alto nivel, esquematizada en el siguiente diagrama de flujo, de forma que una cierta parte del programa se ejecute únicamente en caso de que se cumpla una condición.



Ordinograma del ejercicio 8.2.1.1.

Los pasos a seguir para completar este ejercicio serán:

1. Recuperar el contenido de la posición de memoria `1100H`, empleando para ello cualquiera de las instrucciones que se han conocido en la sección 8.1.

2. Encontrar una instrucción del 8085 que permita comparar el contenido del acumulador con un dato inmediato: `7FH`. Dicha instrucción afectará al registro de indicadores activando unos u otros dependiendo del resultado.

3. Según el diagrama previo el programa continuará su flujo normal si el dato del acumulador es `7FH`, desviándose la ejecución en caso contrario. Habrá que localizar, por tanto, una instrucción de salto condicional que altere el flujo en caso de que el bit `Z` del registro de indicadores sea igual a 0, lo que ocurrirá cuando el contenido del acumulador no sea igual a `7FH`.

Cuestiones

1. ¿Qué operación lleva a cabo la instrucción `CPI/CMP` para determinar la relación entre los datos comparados?

2. ¿Cuál es la razón de que se active el bit `Z` del registro de indicadores cuando el contenido del acumulador coincide con el dato inmediato?

3. ¿Se verá alterado el contenido del acumulador al compararlo con el dato inmediato `7FH`?

Solución

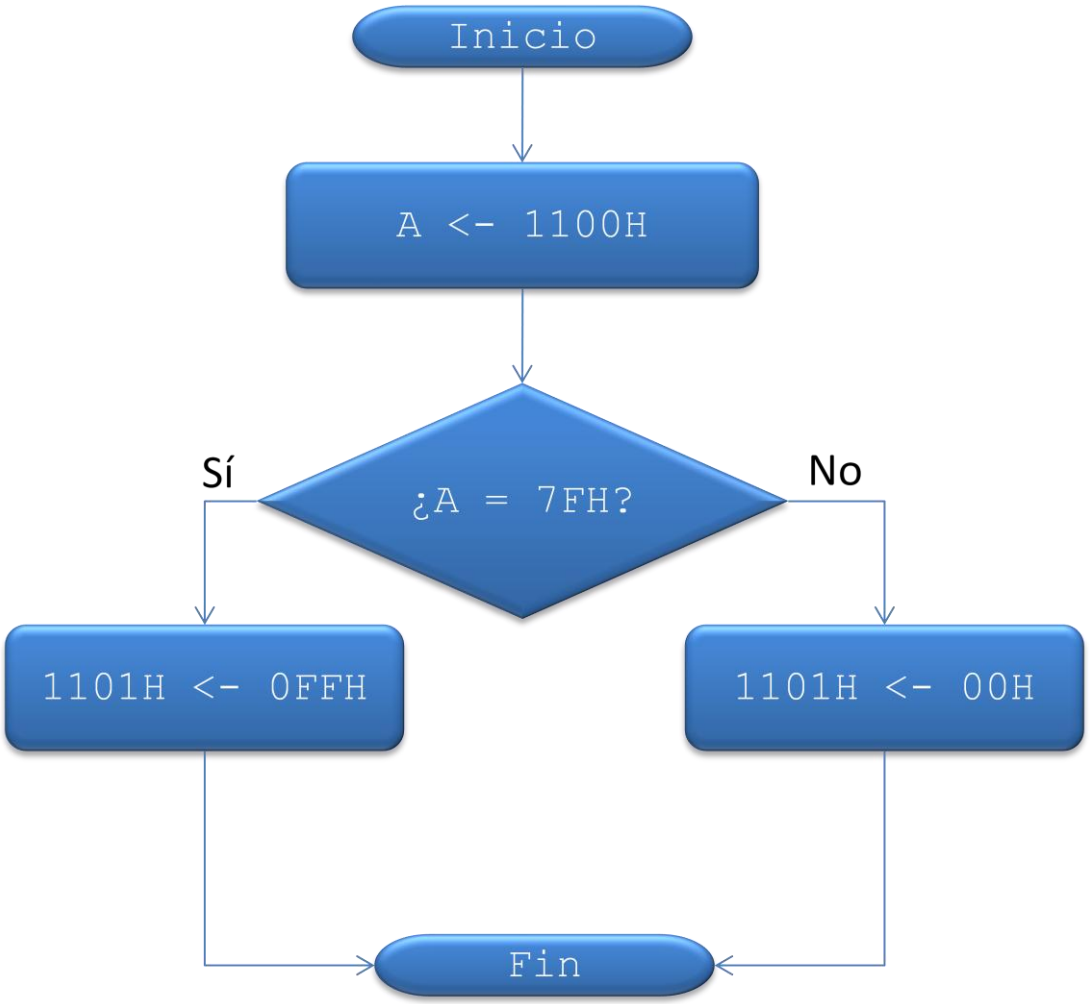
```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H
     CPI 7FH  ; Se compara A con 7FH
     JNZ FIN  ; Si no coincide, salta

     MVI A, 0FFH
     STA 1101H

FIN: RST 1

END
```

Ejercicio 8.2.1.2

> Leer el contenido de la posición de memoria **1100H** y, en caso de que el valor sea igual a **7AH**, escribir en la posición **1101H** el valor **0FFH**, en caso contrario escribir en esa misma posición el valor **00H**.

Indicaciones

Con este ejercicio se persigue reproducir la instrucción `IF-THEN-ELSE` habitual en la mayoría de los lenguajes de programación, de forma que una porción de código se ejecute si se cumple una cierta condición y otra porción exclusivamente si no se cumple. Posteriormente las dos ramas, `THEN` y `ELSE`, convergen a un mismo punto que, en este ejercicio, sería el final del programa como se aprecia en el siguiente diagrama.



Ordinograma del ejercicio 8.2.1.2.

Cuestiones

1. Al ejecutar el programa correspondiente a los dos ejercicios previos, ¿qué ocurre con la posición de memoria `1101H` si el acumulador no contiene el valor `7FH`?

2. ¿Cómo podría interpretarse el contenido de la posición de memoria `1101H` en la versión del programa que corresponde a este ejercicio?

3. ¿De qué otra forma podría escribirse este programa manteniendo el mismo resultado?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H
     CPI 7FH     ; Se compara A con 7FH
     JZ COINC    ; Si coincide, salta

     MVI A, 00H  ; Corresponde a la parte ELSE
     JMP FIN

COINC:
     MVI A, 0FFH ; Corresponde a la parte THEN

FIN:
     STA 1101H
     RST 1

END
```

Ejercicio 8.2.1.3

Comparar el contenido de la posición de memoria **1100H** con el de la posición **1101H**, de forma que se escriba en la dirección **1102H** un 0 si son iguales, un 1 si el primero es mayor que el segundo o un 2 si el primero es menor que el segundo.

Indicaciones

En este ejercicio los dos datos a comparar se encuentran alojados en posiciones de memoria, no son datos inmediatos, por lo que es preciso recuperarlos antes de poder efectuar la comparación. Además se trata de comprobar no exclusivamente la igualdad o desigualdad, sino que si se da esta última interesa saber cuál de los dos datos es mayor (o menor).

Para completar el ejercicio sería preciso seguir los pasos indicados a continuación:

1. Recuperar el contenido de las celdillas `1100H` y `1101H` y alojarlo en dos registros, uno de ellos el acumulador.

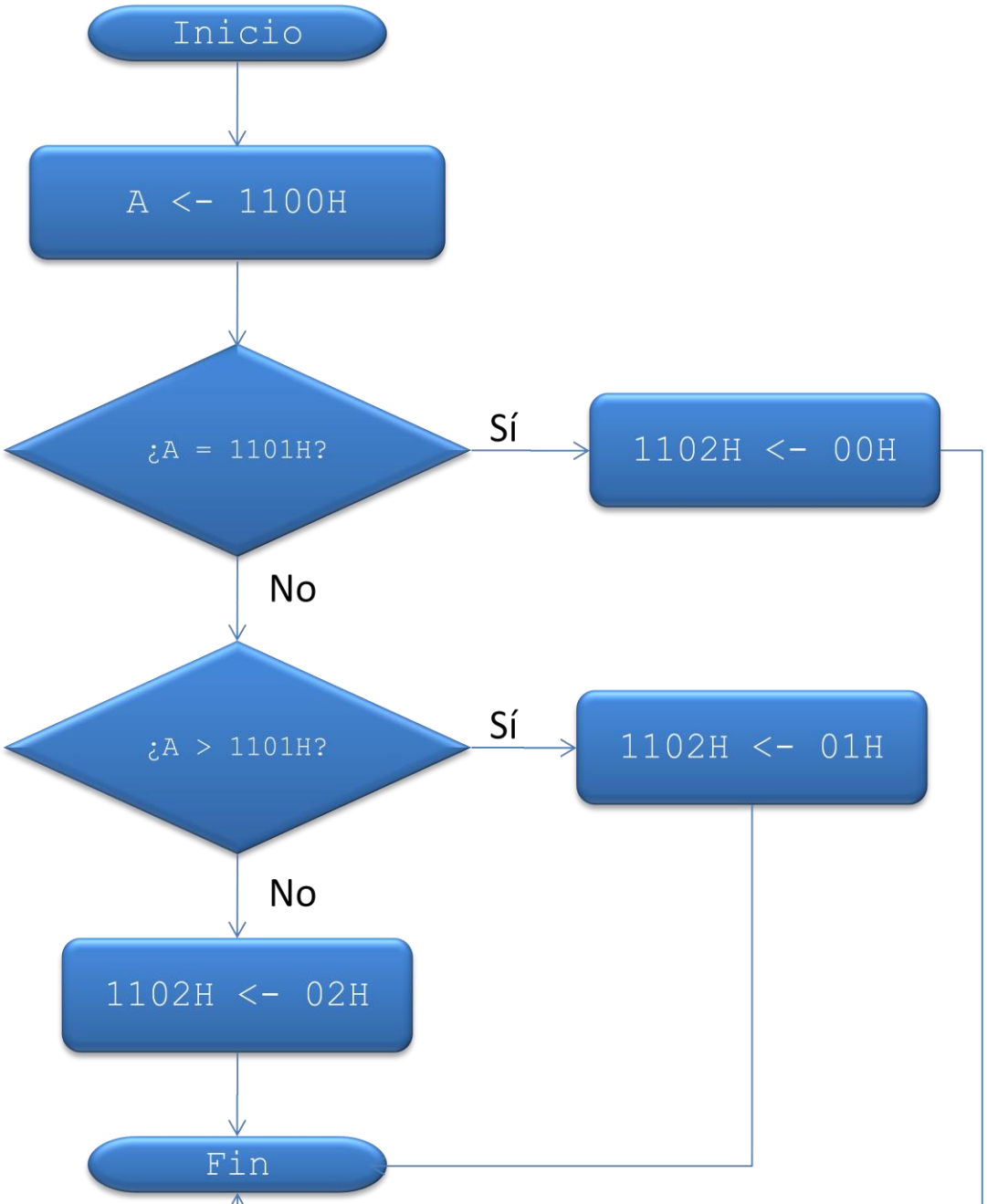
2. Comparar el contenido del acumulador con el registro que contiene el otro valor, obteniendo en el registro de indicadores información sobre el resultado.

3. Si los datos son iguales se activará el bit `Z` del registro de indicadores, por lo que se podría efectuar un salto condicional teniendo en cuenta dicho indicador.

4. Si el dato que contiene el acumulador es menor que el del otro registro, el bit que se activará en el registro de indicadores será `CY`, pudiendo utilizarse los saltos condicionales `JC`/`JNC` asociados a dicho bit.

5. Por último, si los datos no son iguales ni el del acumulador es mayor que el otro, la única posibilidad es que el contenido en el acumulador sea menor.

Estos pasos son los representados esquemáticamente en el diagrama de flujo de la página siguiente, que puede utilizarse como base para escribir el programa solicitado por el ejercicio. Ejecute el programa varias veces modificando el contenido de las posiciones de memoria `1100H` y `1101H` para verificar que, en efecto, funciona como se espera.



Ordinograma del ejercicio 8.2.1.3.

Cuestiones

1. ¿A qué estructura de control de alto nivel equivaldría el código de este programa?

2. ¿Cuál es la razón de que se active el bit `CY` al comparar el contenido del acumulador con otro que es mayor?

3. ¿Qué aplicación podría tener una subrutina que efectuase el trabajo que hace este programa?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1101H  ; Se recupera el primer dato
     MOV B, A   ; en A y el segundo en B
     LDA 1100H

     CMP B      ; Comparar A con B
     JZ IGUAL
     JC MENOR   ; Si CY activo, es que A < B

     MVI A, 01H ; Si no es igual ni menor, es mayor
     JMP FIN

IGUAL:
     MVI A, 00H
     JMP FIN

MENOR:
     MVI A, 02H

FIN:
     STA 1102H
     RST 1

END
```

8.2.2 Bucles

Ejercicio 8.2.2.1

Preparar un programa que escriba el valor **7AH** en las posiciones de memoria **1100H** a **110FH** (16 bytes en total), usando para ello un bucle.

Indicaciones

Al implementar un bucle que debe repetirse un número concreto de veces, en este caso 16, es necesario utilizar algún recurso como contador de repeticiones. Lo habitual es usar con este fin algún registro que, dependiendo del número de repeticiones, puede ser de 8 bits o bien una pareja de registros si dicho número excede de 256. El objetivo es simular una construcción equivalente a `FOR inicio TO fin/NEXT` o similar de la que disponen la mayoría de los lenguajes de alto nivel.

Si además en el bucle va a ser necesario ir recorriendo posiciones consecutivas de memoria, como ocurre en el actual ejercicio, también será preciso emplear una pareja de registros como puntero, de lo contrario habría que especificar manualmente cada una de las direcciones y el bucle ya no sería tal.

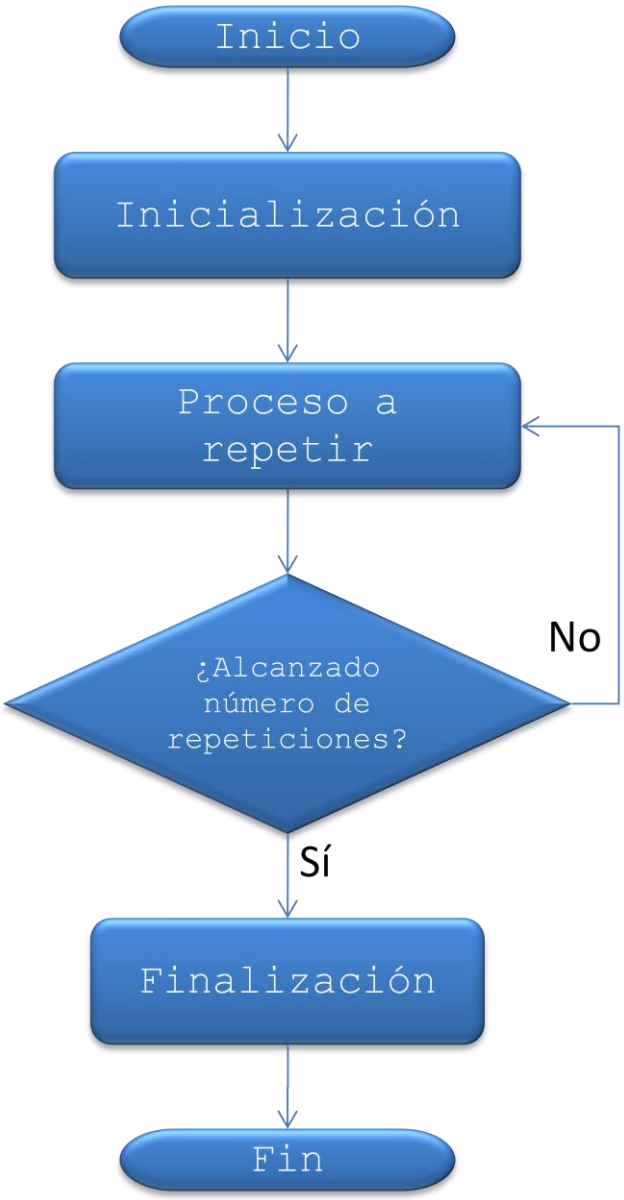
Partiendo de las dos premisas anteriores, la resolución del ejercicio podría dividirse en tres bloques:

1. Inicialización de los registros que van a actuar como contador y puntero a memoria. Este bloque se ejecutaría únicamente una vez, al ponerse en marcha el programa.

2. Escritura del valor indicado en la dirección a la que apunta la pareja de registros elegida y actualización del contador, comprobando si se ha alcanzado el número de repeticiones establecido. Mientras no sea así, este paso seguirá repitiéndose.

3. Finalización del programa una vez que se haya alcanzado el número de repeticiones deseado. Este último bloque puede reducirse a la finalización del programa en sí, con una instrucción `RST 1`, o implicar algunos pasos adicionales, como puede ser el almacenamiento de un resultado final.

Todo programa que tenga que implementar un bucle con un número concreto de repeticiones, por tanto, tendrá una estructura similar a la que esquematiza el siguiente diagrama de flujo:



Ordinograma del ejercicio 8.2.2.1.

Cuestiones

1. Atendiendo al código de la solución que se ofrece a continuación, ¿sería posible utilizar el registro `E`, en lugar del registro `C`, como contador del bucle? ¿Por qué?

2. ¿Qué modificaciones habría que introducir para poder usar el registro `E` como contador?

3. ¿Cómo reescribiría el programa para evitar el uso de la instrucción `STAX`?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI D, 1100H  ; Posición de destino
     MVI C, 0H   ; Se usará C como contador

BUCLE:
     MVI A, 7AH  ; Se escribe el valor
     STAX D      ; indicado en esa posición

     INR C       ; Ya se ha escrito un valor más
     INX D       ; Avanzar a la siguiente posición

     MOV A, C    ; Comparar el contador con 10H (16d)
     CPI 10H

     JNZ BUCLE   ; Si aún no se han escrito 16 valores

     RST 1
END
```

Ejercicio 8.2.2.2

Escribir un programa equivalente al del ejercicio anterior pero en el que no se utilice el acumulador ni la instrucción **`CPI`** para controlar el número de ciclos del bucle.

Indicaciones

La resolución de este problema exige plantearse las siguientes cuestiones:

1. **¿Qué instrucción es la que controla el bucle, la que hace que éste siga repitiéndose o termine?** La respuesta es la instrucción `JNZ`, cuyo salto condicional es el que determina si se vuelve a la etiqueta `BUCLE` o, por el contrario, el programa sigue avanzado o termina.

2. **¿De qué depende que esa instrucción siga repitiendo el bucle o lo finalice?** La instrucción `JNZ` salta a la dirección indicada (de manera simbólica por la etiqueta) dependiendo de cuál sea el estado de un cierto bit del registro de estado, en este caso el bit `Z` o *de cero*. Si dicho bit está a 0 se producirá el salto, en caso contrario no.

3. **¿Qué instrucciones afectan a ese bit del registro de estado aparte de `CPI`?** La mayoría de instrucciones aritméticas y lógicas, incluyendo las de incremento y decremento de 8 bits (`INR`/`DCR`), dependiendo de que el valor final en el registro donde se guarde el resultado sea o no cero.

La instrucción `CPI`/`CMP` lleva a cabo una operación de resta e implica el uso del acumulador. Si el contenido de éste es igual al del operando que acompañe a la instrucción el resultado de esa resta será cero y, en consecuencia, se activará el bit `Z`. Para que ese mismo bit se active al incrementar o decrementar el valor de un registro, sea o no el acumulador, es necesario que el resultado sea cero. El número de repeticiones de un bucle será exactamente el mismo contando desde 0 a n que, a la inversa, contando hacia atrás desde n hasta 0.

Para completar este ejercicio, por tanto, habría que optar por introducir en el registro que vaya a actuar como contador no el valor 0, sino el número de repeticiones que debe ejecutar el bucle. Dentro de éste se tendrá utilizar una instrucción `DCR` para ir reduciendo dicho número, de forma que en cada momento el contador indique cuántas repeticiones faltan. Cuando ese número sea 0 habrá llevado el momento de terminar.

Como se aprecia en la solución, el resultado es un programa apreciablemente más corto, y más eficiente, que las versiones previas.

Cuestiones

1. ¿Qué ocurriría si el valor inicial del registro que actúa como contador fuese cero? ¿Cuántas veces se repetiría el bucle?

2. ¿Cuáles son las diferencias fundamentales entre las parejas de instrucciones `INR/DCR` e `INX/DCX`?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI H, 1100H  ; Posición de destino
     MVI C, 10H    ; Se usará C como contador

BUCLE:
     MVI M, 7AH ; indicado en esa posición

     DCR C      ; Queda un valor menos por escribir
     INX H      ; Avanzar a la siguiente posición

     JNZ BUCLE  ; Si aún no se han escrito 16 valores

     RST 1
END
```

8.2.3 Copia de bloques de datos

Ejercicio 8.2.3.1

Se quieren transferir 32 bytes de memoria desde la dirección **1100H** a la dirección **1200H**. Escribir un programa que lleve a cabo la copia de esos datos.

Indicaciones

Un programa que tiene que copiar un bloque de datos de una dirección a otra tendrá como nudo principal un bucle y, en consecuencia, los tres bloques ya citados en ejercicios previos: inicialización, cuerpo del bucle y finalización. Puesto que en el cuerpo del bucle hay que ir leyendo de unas posiciones de memoria y escribiendo en otras, será preciso recurrir a dos punteros, dos pares de registros que actúen como dirección de origen (de la que leer) y de destino (en la que escribir).

Para completar el ejercicio será necesario dar los pasos siguientes:

1. En el bloque de inicialización habrá que cargar en sendos pares de registros, que van a ser utilizados como punteros, las direcciones de origen y destino de la copia. También habrá que inicializar el contador del bucle.

2. El cuerpo del bucle se encargará de leer una posición de memoria del origen, escribirla en el destino y actualizar los punteros.

3. Como en ejercicios previos, a cada ciclo se decrementará el contenido del registro que actúa como contador y, si no es cero, se pasará al ciclo siguiente.

Cuestiones

1. ¿Cuál es el tamaño máximo de los bloques de memoria que podrían copiarse con la solución dada?

2. ¿Cómo podrían copiarse bloques de mayor tamaño?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     ; Inicialización
     LXI B, 1100H
     LXI D, 1200H
     MVI L, 20H ; 32d = 20H

     ; Bucle
BUCLE:
     LDAX B   ; Leer un byte de origen
     STAX D   ; y escribirlo en destino

     DCR L    ; Queda un valor menos por copiar
     INX B    ; Avanzar a la siguiente posición
     INX D    ; de origen y destino

     JNZ BUCLE   ; Si aún no se ha llegado a cero

     ; Finalización
     RST 1
END
```

Ejercicio 8.2.3.2

Examine el código del programa siguiente y responda a las cuestiones propuestas. A continuación ejecútelo en el sistema uP-2000 y verifique las respuestas dadas.

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H
     LHLD 1101H
     MOV B, H
     MOV C, L
     LHLD 1103H
     XCHG
     LHLD 1105H

BUCLE:
     XCHG
     LDAX D
     XCHG
     STAX D

     INX H
     INX D
     DCX B

     MOV A, B
     CPI 00H
     JNZ BUCLE

     CMP C
     JNZ BUCLE

     SHLD 1105H
     XCHG
     SHLD 1103H

     RST 1
END
```

Cuestiones

1. Explique cuál es la finalidad de este programa y el trabajo que lleva a cabo cuando se ejecuta.

2. ¿Qué datos deben introducirse en las posiciones de memoria `1100H`, `1101H-1102H`, `1103H-1104H` y `1105H-1106H` antes de ejecutarlo?

3. ¿Cuál será el contenido de las posiciones de memoria `1103H-1104H` y `1105H-1106H` tras la ejecución del programa?

4. Introduzca a partir de la posición de memoria `1100H` los datos `AAH`, `00H`, `02H`, `00H`, `14H`, `00H` y `00H` y ejecute el programa. ¿Cuántos bytes se han copiado? ¿Qué representa la información que se ha copiado?

5. ¿Existen instrucciones superfluas en el programa, sentencias que si se eliminasen no afectarían a la funcionalidad? ¿Cuáles?

6. Explique detalladamente cómo se lleva a cabo el control del bucle que existe en el programa.

7. Añada al programa los comentarios necesarios para facilitar su comprensión.

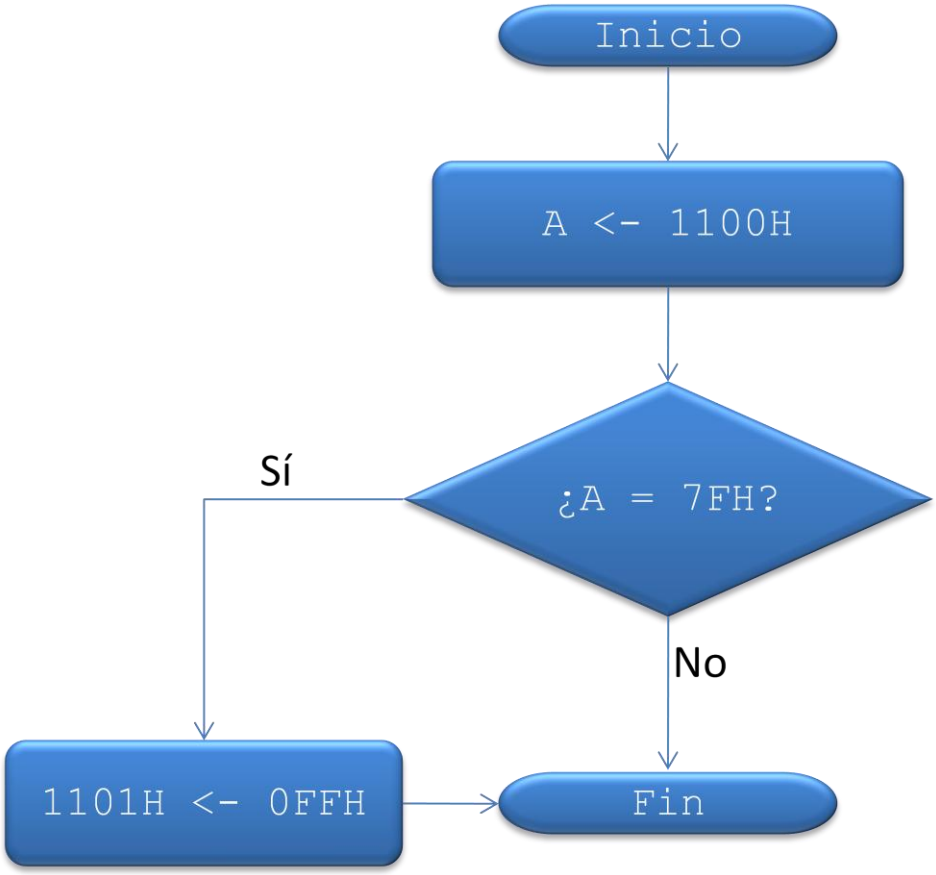
8.2.4 Ejercicios propuestos

Ejercicio 8.2.4.1

Modifique el programa del ejercicio previo para que funcione igual que lo hace pero usando la instrucción **`JZ`** en lugar de **`JNZ`**.

Indicaciones

Si en el ejercicio 8.1 el flujo por defecto del programa es el que asume que el acumulador contendrá el valor `7FH`, saltándose en caso de que no sea así, en este caso la suposición es la contraria, es decir, se asume que el acumulador tendrá un valor distinto a `7FH` y por ello el flujo por defecto es el de finalización, desviándose la ejecución en caso contrario. Es la situación representada en el siguiente diagrama de flujo.



Ordinograma del ejercicio 8.2.4.1.

La solución a este ejercicio no es única, pudiéndose alcanzar el fin del programa de distintas formas: un salto adicional, la duplicación de la instrucción `RST 1`, etc.

Cuestiones

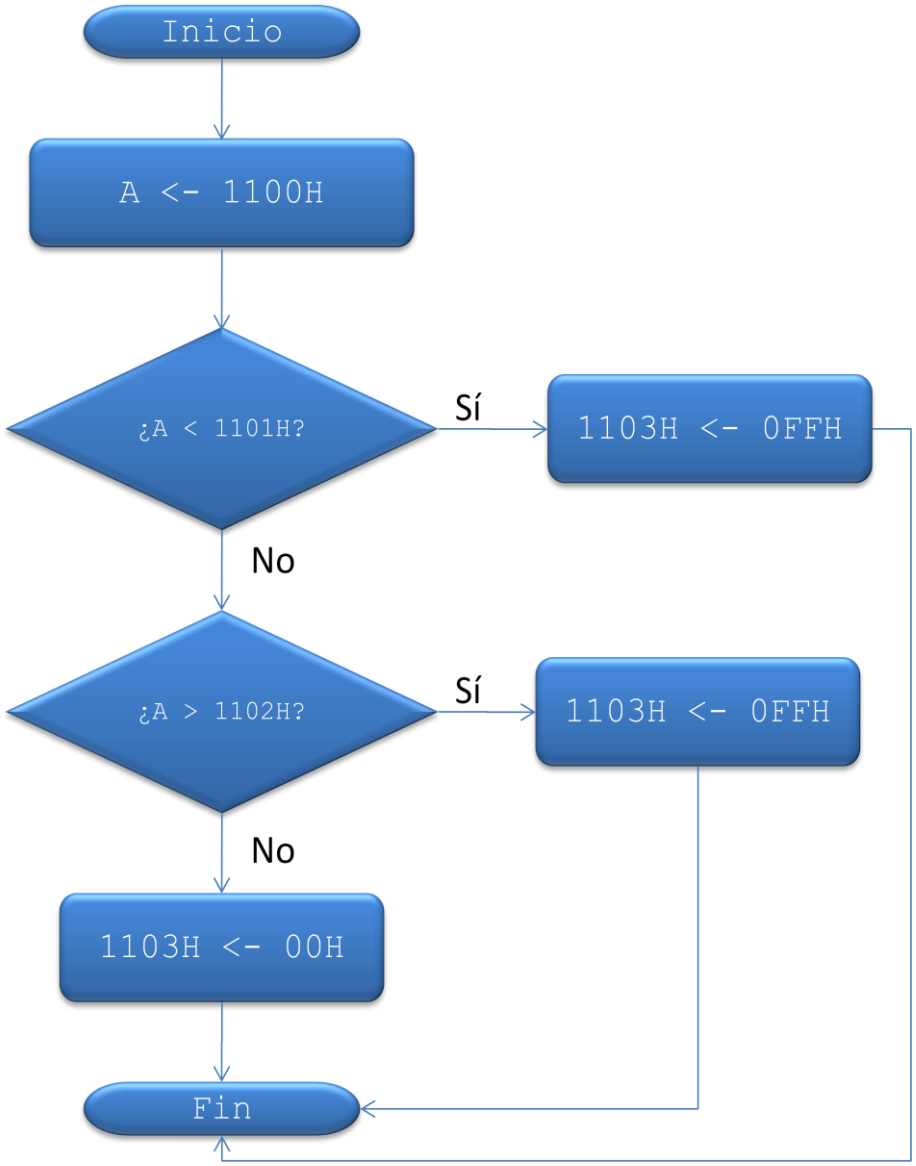
1. ¿Para qué puede resultar útil comparar el contenido del acumulador con un valor concreto?

Ejercicio 8.2.4.2

Se quiere saber si el contenido de la posición de memoria **1100H** se encuentra en el rango indicado por las dos posiciones siguientes, fijando el valor de **1101H** el mínimo y el de **1102H** el máximo. Escribir en **1103H** el valor **00H** en caso afirmativo o el valor **0FFH** en caso contrario.

Indicaciones

El objetivo de este ejercicio es la implementación de un condicional compuesto que, en un lenguaje de alto nivel, sería equivalente a la construcción `IF A>=B AND A<=C THEN R=0 ELSE R=0FFH`. El siguiente diagrama de flujo puede servir como guía para completar el ejercicio.



Ordinograma del ejercicio 8.2.4.2.

En el diagrama puede verse que, tras recuperar en el acumulador el contenido de la posición de memoria `1100H`, se opta por comprobar si está por debajo del límite inferior, en cuyo caso no es necesario hacer más comprobaciones para saber que el valor está fuera del rango, y a continuación si está por encima del límite inferior. Únicamente si estas dos comprobaciones son falsas se sabe que el valor está dentro del rango. Es importante tener en cuenta que en las comparaciones se emplean las relaciones $<$ (menor estricto) y $>$ (mayor estricto), ya que si el contenido del acumulador fuese igual a uno de los límites se consideraría dentro del rango.

Cuestiones

1. ¿Sería posible comparar el acumulador con el contenido de las posiciones de memoria que tienen los límites sin necesidad de llevar éstos a sendos registros? ¿Cómo?

2. ¿De qué forma podría escribirse este programa utilizando instrucciones de salto distintas a las de la solución propuesta?

Ejercicio 8.2.4.3

Utilizando como base los dos ejercicios previos, obtener un programa que haga lo mismo pero que tenga en cuenta lo siguiente:

- El valor a escribir en memoria se encuentra almacenado en la dirección **1100H**.

- El número de posiciones a escribir es el indicado en la dirección **1101H**.

- La dirección de memoria a partir de la que deben escribirse en los datos está en las direcciones **1102H** y **1103H** en formato *little endian* (como es habitual).

Indicaciones

La estructura del programa que se pide en este ejercicio será la misma que en los casos anteriores, constando de una inicialización, un cuerpo del bucle formado por las instrucciones a repetir y una finalización. La única diferencia se encuentra en el proceso de inicialización, ya que la dirección, dato y número de repeticiones han dejado de ser datos inmediatos. En ese bloque de inicialización el programa deberá dar los pasos siguientes:

1. Obtener el contenido de la dirección `1100H` y guardarlo en el registro que se haya elegido como dato a escribir en la memoria.

2. Recuperar el contenido de la dirección `1101H` y llevarlo al registro que se use como contador del bucle.

3. Leer la dirección de memoria alojada en `1102H-1103H` colocándola en la pareja de registros a usar como puntero.

Efectuada la inicialización el resto del programa será prácticamente idéntico al de ejercicios previos.

Ejercicio 8.2.4.4

Escribir un programa que haga el mismo trabajo que del ejercicio anterior pero sin usar las instrucciones **LDAX** y **STAX**.

Indicaciones

Además de estas dos instrucciones, que son las que no deben utilizarse y que emplean como punteros las parejas de registros BC y DE, el 8085 puede usar la pareja HL como puntero a través del registro simbólico M. Mediante dicho registro, y la instrucción MOV, puede leerse y escribirse en memoria. El problema que se plantea es que la pareja HL debería, en este caso, apuntar en unas ocasiones al bloque de origen y en otras al de destino. Para ello, tras el proceso de inicialización, los pasos a seguir en el cuerpo del bucle serían:

1. Usando M como puntero al bloque de origen, se lee un byte.
2. Se guarda el contenido actual de HL y se recupera el puntero de destino.
3. Usando M como puntero al bloque de destino, se escribe el byte leído antes.
4. Se guarda el contenido actual de HL y se recupera el puntero de origen.
5. Repetir los pasos previos hasta completar el número de ciclos establecido.

Es necesario, por tanto, emplear algún almacenamiento temporal para el puntero a memoria que no se tiene en ese momento en la pareja HL. Para ello puede utilizarse el recurso que se crea más adecuado, ya que el ejercicio no impone limitaciones en este sentido.

Cuestiones

1. ¿Qué otra alternativa podría utilizarse para almacenar temporalmente los punteros a memoria?

8.3 Operaciones aritméticas

Una de las partes fundamentales de un microprocesador es su UAL (*Unidad Aritmético Lógica*), el componente que se encarga de realizar todas las operaciones aritméticas y lógicas, generando un resultado y alterando los bits del registro de estado. Dependiendo del microprocesador se contemplarán operaciones más o menos complejas y con operandos de diferentes tamaños. En el caso del 8085 las dos únicas operaciones aritméticas para las que existen instrucciones son la suma y la resta, con operandos de 8 bits y, en el caso concreto de la suma, también con operandos de 16 bits.

Mediante estas dos operaciones básicas, no obstante, es posible realizar otras más complejas como el producto y la división. Los objetivos que persiguen los ejercicios de esta sección son los siguientes:

- Aprender a sumar datos de 8 y 16 bits, empleando para ello las distintas instrucciones del 8085.
- Conocer el significado del bit de acarreo y aprender a usarlo en las operaciones.
- Aprender a restar datos de 8 y 16 bits, tanto con acarreo como sin acarreo.
- Combinar operaciones de suma y bucles para conseguir efectuar operaciones de multiplicación.
- Aprender a efectuar divisiones combinando operaciones de resta y bucles.
- Introducir las bases para poder realizar operaciones aritméticas con operandos de mayor tamaño.

8.3.1 Suma con y sin acarreo (8 y 16 bits)

Ejercicio 8.3.1.1

Escribir un programa que, utilizando la instrucción **ADI**, sume los números **7AH** y **1FH** y almacene el resultado en la dirección de memoria **1100H**.

Indicaciones

Todas las operaciones aritméticas de 8 bits toman como operando implícito al acumulador y, tras ejecutarse, dejan el resultado en ese mismo registro. En este ejercicio, puesto que nos dan los datos que se quieren sumar, los pasos a seguir serían:

1. Colocar el primer operando en el acumulador.

2. Sumar al acumulador el segundo operando, mediante la instrucción `ADI` tal y como se indica en el enunciado.

3. Mover el resultado que está en el acumulador a la posición de memoria señalada en el ejercicio.

Cuestiones

1. ¿Cuál es el contenido de la posición de memoria `1100H` tras ejecutar el programa? ¿Cómo queda el registro de estado 8085?

2. ¿Qué ocurre si se cambia el segundo operando, actualmente `1FH`, por el valor `AAH`? ¿Qué resultado queda en la posición `1100H`? ¿Cómo queda el registro de estado del 8085?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     MVI A, 7AH ; Primer operando en el acumulador
     ADI 1FH    ; Se suma el segundo operando

     STA 1100H  ; Y se guarda el resultado

     RST 1

END
```

Ejercicio 8.3.1.2

Tomar dos datos alojados en las direcciones de memoria **1100H** y **1101H**, ambos de 8 bits, efectuar la suma y almacenar el resultado en la dirección **1102H**, poniendo en **1103H** un 1 si se ha producido acarreo o un 0 en caso contrario.

Indicaciones

A diferencia de lo que ocurre en el ejercicio 8.3.1.1, en este caso los datos a sumar no son conocidos de antemano sino que es necesario recuperarlos previamente de la memoria. Además se pide como resultado de la ejecución no solamente la suma de los operandos, sino también una indicación de si se ha producido o no acarreo en la misma.

Para completar este ejercicio habrá que reproducir los pasos siguientes:

1. Recuperar de la memoria los dos operandos de 8 bits que van a sumarse, almacenando uno de ellos en el acumulador y el segundo en otro registro.

2. Sumar al acumulador el contenido del otro registro.

3. Almacenar el resultado de la suma en la posición de memoria indicada.

4. Comprobar si está activo el bit de acarreo del registro de estado y almacenar en la siguiente posición de memoria el valor 1 si lo está o el valor 0 en caso contrario. Es un condicional simple que puede implementarse con un salto, tal y como se aprendió en la sección previa de ejercicios.

Cuestiones

1. ¿Qué ocurriría si en el programa se llevase el segundo operando al registro `B` y el primero al acumulador, es decir, a la inversa de como se ha hecho? ¿Sería distinto el resultado?

2. Al interpretar el contenido de las posiciones de memoria `1102H` y `1103H` como un dato de 16 bits en formato *little endian* ¿qué es lo que se obtiene?

Solución

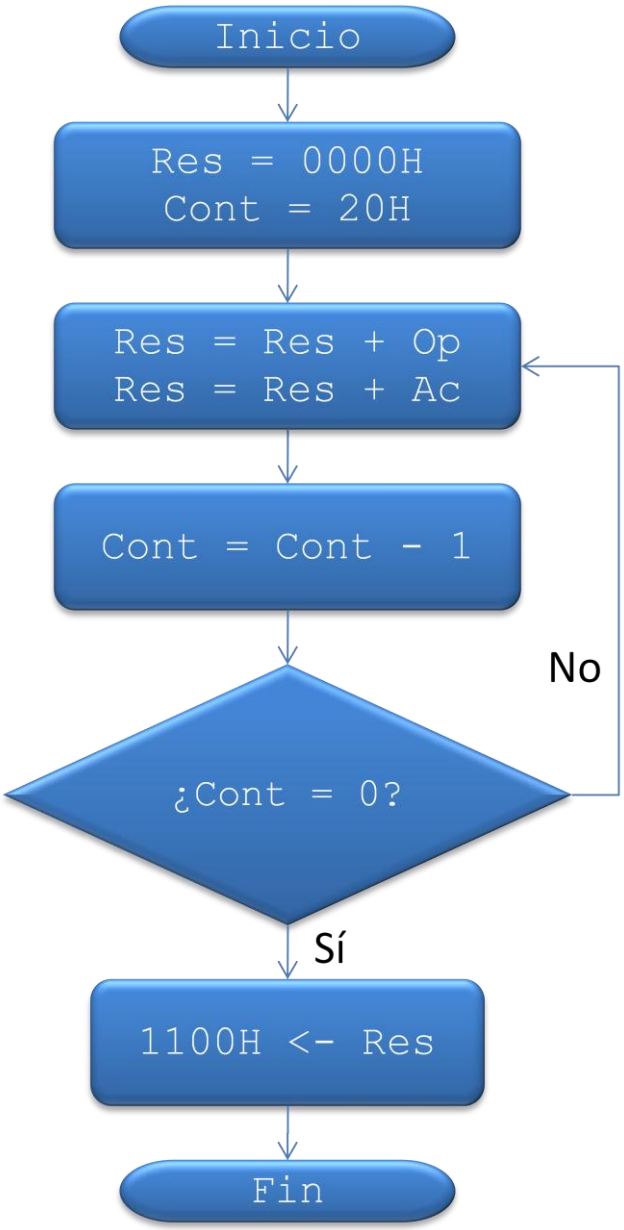
```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Recuperar un operando
     MOV B, A   ; Llevarlo a B
     LDA 1101H  ; Recuperar el otro operando

     ADD B ; Se suma el operando de B al de A

     STA 1102H  ; Y se guarda el resultado

     MVI A, 00H ; Poner un 0 en A
     JNC FIN    ; y saltar si no ha habido acarreo

     MVI A, 01H ; Si ha habido acarreo poner un 1

FIN:
     STA 1103H  ; Guardar el indicador de acarreo

     RST 1

END
```

Ejercicio 8.3.1.3

Se quiere obtener como un dato de 16 bits la suma del contenido de las celdillas de memoria comprendidas entre las direcciones **0100H** y **011FH**, un total de 32 bytes. Almacenar el resultado en las direcciones **1100H** y **1101H**, en formato *little endian*.

Indicaciones

En este caso no se trata de sumar solamente dos datos, sino una secuencia de 32 bytes consecutivos, por lo que será preciso utilizar un bucle para recorrerlos. Además el resultado ha de ser un dato de 16 bits, por lo que habrá que tener en cuenta el acarreo en cada ciclo del bucle. El siguiente diagrama de flujo esquematiza la solución:



Ordinograma del ejercicio 8.3.1.3.

`Res` sería un registro de 16 bits en el que va a ir guardándose el resultado parcial, un registro que es preciso inicializar con el valor cero. `Cont` será el registro que actúe como contador del bucle. En cada ciclo se suma a `Res` el operando que corresponda y, a continuación, se suma también el acarreo. El proceso se repite hasta que el contador llegue a cero, momento en que se almacena el resultado en la dirección indicada.

Cuestiones

1. ¿Cuál es el resultado que genera la ejecución del programa?

2. ¿Podría desbordarse el resultado de 16 bits al sumar 32 datos de 8 bits?

3. ¿Cuántos datos de 8 bits podrían llegar a sumarse obteniendo un resultado de 16 bits?

4. Intente rescribir el programa reduciendo su extensión. Puede recurrir a las instrucciones que desee del 8085.

Solución

```
HOF "INT8"

     ORG 1000H

     LXI D, 00H       ; El resultado estará en DE
     LXI H, 100H      ; Puntero a datos
     MVI C, 20H       ; 20H=32d

REP:
     MOV A, E    ; Se lleva a A el LSB
     ADD M       ; y se suma el siguiente byte
     MOV E, A    ; Guardar LSB

     MOV A, D    ; Se lleva a A el MSB
     ACI 00H     ; y se acumula el acarreo
     MOV D, A    ; Guardar MSB

     INX H       ; Avanzar al siguiente operando
     DCR C       ; Queda un byte menos

     JNZ REP     ; Repetir hasta terminar

     XCHG        ; Tomar resultado en HL
     SHLD 1100H  ; y guardarlo

     RST 1

END
```

Ejercicio 8.3.1.4

Se tienen almacenados dos números de 16 bits, en formato *little endian*, en las direcciones **1100H-1101H** y **1102-1103H**. Obtener la suma y guardar el resultado de 16 bits en la dirección **1104-1105H**.

Indicaciones

La resolución de este ejercicio es análoga a la del ejercicio 8.3.1.2, con la diferencia de que habrá que usar el acumulador de 16 bits (la pareja de registros `HL`) y las instrucciones de transferencia y suma de 16 bits. Puesto que el ejercicio indica que el resultado es de 16 bits, no se precisa tener en cuenta el posible acarreo.

Cuestiones

1. Almacene como operando los valores `9AA0H` y `BACAH`, ejecute el programa y compruebe el resultado. ¿Se ha activado el bit de acarreo en el registro de estado?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LHLD 1100H  ; Se toma el primer operando
     XCHG        ; y se lleva a DE
     LHLD 1102H  ; El segundo en HL

     DAD D       ; HL = HL + DE

     SHLD 1104H  ; Se almacena el resultado

     RST 1
END
```

8.3.2 Resta con y sin acarreo (8 y 16 bits)

Ejercicio 8.3.2.1

Tomar dos datos alojados en las direcciones de memoria **1100H** y **1101H**, ambos de 8 bits, efectuar la diferencia entre el primero y el segundo y almacenar el resultado en la dirección **1102H**.

Indicaciones

Para completar este ejercicio habrá que reproducir los pasos siguientes:

1. Recuperar de la memoria los dos operandos de 8 bits que van a restarse, almacenando uno de ellos en el acumulador y el segundo en otro registro.

2. Restar del acumulador el contenido del otro registro.

3. Almacenar el resultado de la suma en la posición de memoria indicada.

Cuestiones

1. ¿Qué ocurriría si en el programa se llevase el primer operando al registro `B` y el segundo al acumulador, es decir, a la inversa de como se ha hecho? ¿Sería distinto el resultado?

2. ¿Qué sucede cuando el dato almacenado en `1101H` es mayor que el contenido en `1100H`? ¿Cómo se ve afectado el registro de estado?

3. Introduzca en la posición `1100H` el valor `AAH`, en la posición `1101H` el valor `1DH`, ejecute el programa e interprete el resultado obtenido en la posición `1103H`. Haga lo mismo pero cambiando el contenido de la posición `1100H` por `1DH` y el de la posición `1101H` por `AAH`.

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1101H   ; El sustraendo
     MOV B, A    ; en B
     LDA 1100H  ; El minuendo en A

     SUB B       ; A = A - B

     STA 1102H   ; y se guarda

     RST 1
END
```

Ejercicio 8.3.2.2

Tomar dos datos alojados en las direcciones de memoria **1100H** y **1101H**, ambos de 8 bits, efectuar la diferencia entre ambos y almacenar el valor absoluto del resultado en la dirección **1102H**.

Indicaciones

Respecto al ejercicio anterior éste solamente plantea una diferencia: se quiere como resultado el valor absoluto de la diferencia entre los dos operandos, es decir, el resultado de la resta pero sin signo. Además no se indica si hay que restar el segundo del primero o viceversa. En realidad es algo que no importa, ya que al tomar el valor absoluto daría lo mismo calcular `15-5` que `5-15`, en ambos casos el resultado sería `10`.

La mayoría de los lenguajes de programación cuentan con una función, llamada habitualmente `abs()`, que toma como parámetro un número y devuelve su valor absoluto. En ensamblador no existe nada parecido, pero dicha función puede simularse de distintas formas. Puesto que el orden en que se tomen daría igual, podría resolverse el ejercicio de la siguiente forma:

1. Recuperar los dos operandos en dos sendos registros, asumiendo cuál hará las veces de sustraendo y cuál las de minuendo.

2. Comparar el sustraendo con el minuendo, de forma que si el primero es mayor que el segundo se intercambien.

3. Efectuar la operación de resta y almacenar el resultado, que siempre será positivo.

Cuestiones

1. Repita las acciones indicadas en la cuestión 3 del ejercicio previo. ¿Cuál es el resultado?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1101H   ; El sustraendo
     MOV B, A    ; en B
     LDA 1100H  ; El minuendo en A

     CMP B       ; Se comparan
     JNC NOCAM   ; Si B <= A saltar

     MOV C, B    ; Intercambio de A con B
     MOV B, A
     MOV A, C

NOCAM:

     SUB B ; A = A - B

     STA 1102H   ; y se guarda

     RST 1
END
```

Ejercicio 8.3.2.3

En las posiciones **1100H-1001H** y **1102H-1103H** se tienen dos datos de 16 bits. Hallar la diferencia entre el primero y el segundo y guardar el resultado, como número de 16 bits, en las direcciones **1104H-1105H.**

Indicaciones

A diferencia de lo que ocurre con la suma, el 8085 no dispone de instrucciones para realizar la sustracción entre registros de 16 bits. Al igual que con la suma, sin embargo, una resta de 16 bits puede realizarse en dos pasos independientes:

1. Se resta el LSB del sustraendo del LSB del minuendo, obteniendo el LSB del resultado.

2. A continuación se resta del MSB del sustraendo el MSB del minuendo y también el bit de acarreo.

Aunque el minuendo sea mayor que el sustraendo es necesario tener en cuenta el acarreo al restar los LSB, ya que el LSB del minuendo puede ser inferior al del sustraendo. Un ejemplo serían los valores `A010H` y `7125H`. El primero es mayor que el segundo, pero el LSB del primero es `10H` y el del segundo `25H`, por lo que al restar el segundo del primero se producirá un acarreo.

Cuestiones

1. ¿Qué ocurriría si el sustraendo fuese mayor que el minuendo? ¿Generaría la solución un resultado un correcto?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LHLD 1100H  ; El minuendo se
     XCHG        ; almacena en DE
     LHLD 1102H ; y el sustraendo en HL

     MOV A, E    ; Se restan los LSB
     SUB L
     STA 1104H   ; y se guarda

     MOV A, D    ; Se restan los MSB
     SBB H ; y el bit de acarreo
     STA 1105H

     RST 1
END
```

8.3.3 Técnicas para multiplicar números

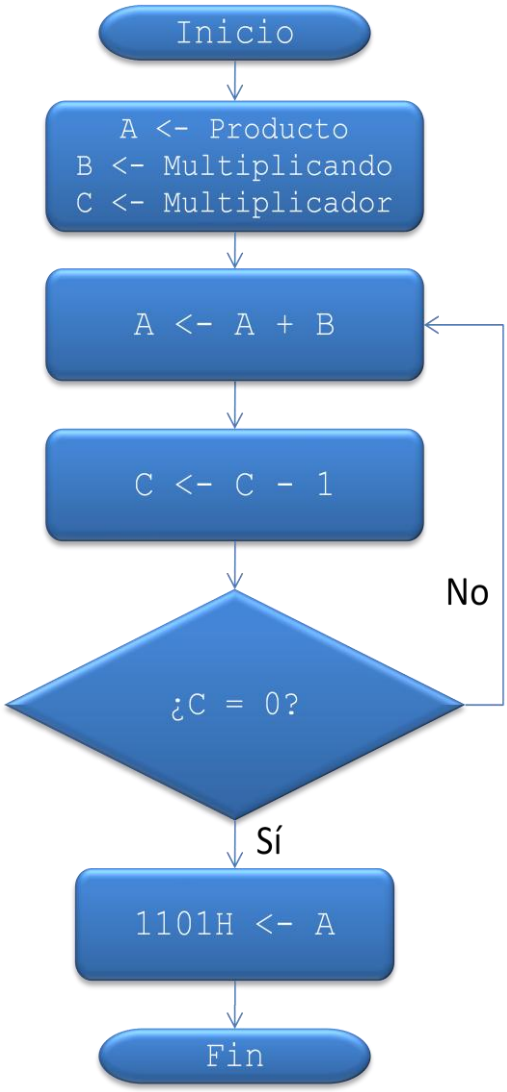
Ejercicio 8.3.3.1

> Tomar el dato de 8 bits contenido en la posición de memoria **1100H**, multiplicarlo por 5 y almacenar el resultado de 8 bits en la posición **1101H**.

Indicaciones

El 8085 no cuenta con instrucciones que permitan calcular el producto de dos números, algo que sí pueden hacer microprocesadores más modernos. Multiplicar un número cualquiera por 5, no obstante, es un proceso muy simple. Planteándolo de forma manual podría decirse que: `N x 5 = N + N + N + N +N`.

Un producto puede descomponerse como una sucesión de sumas y ésta es la forma de plantear la solución al ejercicio que se propone, según se aprecia en el siguiente diagrama de flujo:



Ordinograma del ejercicio 8.3.3.1.

Los pasos a seguir, por tanto, serían los siguientes:

1. Recuperar el contenido de la posición `1100H`, que es el número a multiplicar, y llevarlo al registro `B`.

2. Inicializar `A` con el valor 0, ya que el acumulador irá conteniendo el producto parcial a cada ciclo y final cuando termine el programa.

3. Llevar al registro `5` el multiplicador, es decir, el número de ciclos que tendrá el bucle.

4. En cada ciclo del bucle sumar al acumulador el contenido del registro `B`, el multiplicando.

5. Al final escribir en la posición `1101H` el resultado contenido en el acumulador.

Cuestiones

1. ¿Cómo se podría escribir una versión más eficiente de este ejercicio concreto? ¿Sería aplicable de manera general?

2. Almacene en la posición `1100H` un número mayor que `33H`, ejecute el programa y examine el resultado. ¿Qué ha ocurrido?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H   ; Se lleva el multiplicando
     MOV B, A    ; a B

     MVI A, 00H  ; Producto en A

     MVI C, 05H  ; Multiplicador en C

REP:
     ADD B       ; A = A + B
     DCR C
     JNZ REP     ; C > 0, seguir

     STA 1101H

     RST 1
END
```

Ejercicio 8.3.3.2

Tomar los datos de 8 bits almacenados en las posiciones de memoria **1100H** y **1101H** y, usándolos como multiplicando y multiplicador, respectivamente, calcular el producto usando 16 bits. Almacenar el resultado en las posiciones **1102H-1103H**.

Indicaciones

Este ejercicio propone una generalización del programa del ejercicio previo, de forma que el multiplicador, el número de ciclos que tendrá el bucle, no está determinado de antemano, sino que se recupera de memoria al igual que el multiplicando.

Siguiendo el ordinograma del ejercicio 8.3.3.1, lo único a cambiar sería el hecho de que el multiplicador se recupera de la posición `1101H` y de que el resultado ha de ser un número de 16 bits. Los pasos, por lo demás, serán idénticos.

Cuestiones

1. Al ser multiplicando y multiplicador dos valores cualesquiera de 8 bits, ¿existe la posibilidad de que el resultado precise más de 16 bits para su almacenamiento?

2. ¿Qué ocurre si el dato almacenado en la posición de memoria `1101H` en el momento de ejecutar el programa es 0? ¿Es el resultado correcto?

3. ¿Qué modificaciones habría que introducir en el programa para que el resultado fuese siempre correcto?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H   ; Se lleva el multiplicando
     MVI D, 00H
     MOV E, A    ; a DE

     LXI H, 00H  ; Producto en HL

     LDA 1101H   ; Multiplicador en A

REP:
     DAD D       ; HL = HL + DE
     DCR A
     JNZ REP     ; A > 0, seguir

     SHLD 1102H

     RST 1
END
```

8.3.4 Técnicas para dividir números

Ejercicio 8.3.4.1

> Tomar el dato de 8 bits contenido en la posición de memoria **1100H**, dividirlo entre 4 y almacenar el resultado de 8 bits en la posición **1101H**.

Indicaciones

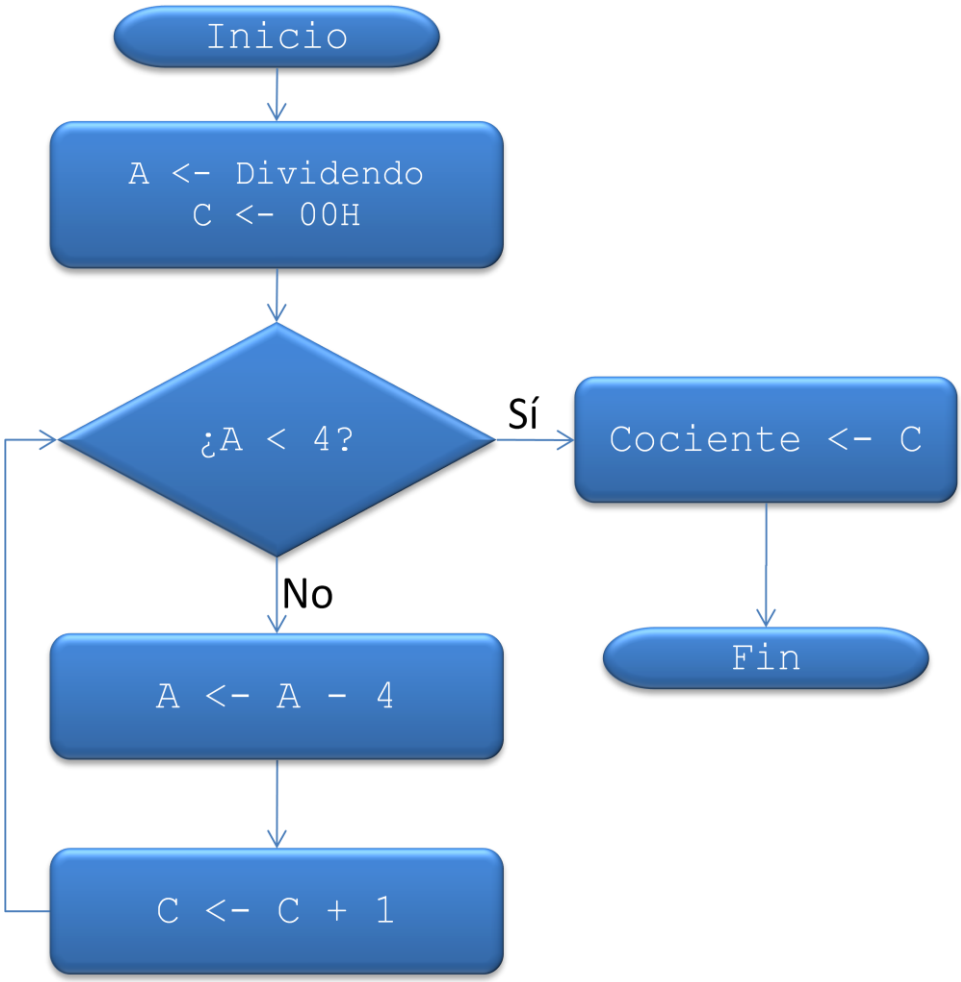
Como ocurre con el producto, el 8085 tampoco cuenta con instrucciones específicas para llevar a cabo la operación de división. La resolución de este ejercicio, por tanto, parte del planteamiento de algún algoritmo que permita realizar dicha operación repitiendo otras más básicas.

Una división puede verse como una sucesión de restas. Concretamente se resta al dividendo el divisor tantas veces como sea posible, obteniéndose el cociente como el número de restas que han podido efectuarse. Para dividir el número 20 entre 4, por ejemplo, se procedería de la siguiente forma:

```
20 - 4 = 16      -> 1
16 - 4 = 12      -> 2
12 - 4 =  8      -> 3
 8 - 4 =  4      -> 4
 4 - 4 =  0      -> 5
```

El algoritmo se detiene en el momento en que el resto parcial obtenido tras la sustracción es inferior al divisor. El cociente en este caso sería 5. El resto final es 0, por lo que la división es entera. De haberse obtenido cualquier otro valor en la última resta, ése sería el resto de la división.

Expresado de una manera más formal, este algoritmo, para el caso concreto que plantea el ejercicio, daría lugar al siguiente diagrama de flujo:



Ordinograma del ejercicio 8.3.4.1.

Cuestiones

1. ¿Qué ocurre si el valor almacenado en la posición `1100H` como dividendo es el `00H`? ¿Es correcto el resultado que genera el programa?

2. ¿Es posible que al dividir algún dato de 8 bits, usando un divisor también de 8 bits, se obtenga un resultado que necesite más de 8 bits para ser almacenado?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Dividendo en A

     MVI C, 0   ; Cociente en C

REP:
     CPI 04H    ; ¿Es A < 4?
     JC FIN     ; Si es así se ha terminado

     SUI 04H    ; A = A - 4
     INR C      ; Incrementar cociente

     JMP REP

FIN:
     MOV A, C   ; Almacenar el cociente
     STA 1101H

     RST 1
END
```

Ejercicio 8.3.4.2

Generalizar el programa del ejercicio previo, de forma que tome como dividendo el dato almacenado en la dirección **1100H** y como divisor el contenido de la posición **1101H**. El cociente debe quedar almacenado en la dirección **1102H** y el resto de la división en **1103H**.

Indicaciones

En el programa anterior el divisor era un valor inmediato, el número 4, y el resto se descartaba al final de la operación. El esquema seguido habría que adaptador de la siguiente forma:

1. Recuperar el dato de la posición de memoria `1101H` y almacenarlo en un registro que hará las veces de divisor.

2. Recuperar el dividendo de la posición de memoria `1100H` y dejarlo en el acumulador.

3. Inicializar el registro que actuará como cociente.

4. En el interior del bucle se restará del dividendo el divisor, como se hace en el ejercicio 8.3.4.1.

5. Al finalizar el bucle el acumulador contendrá un valor, inferior al divisor, resultado de la última resta. Dicho dato es el resto de la división.

Cuestiones

1. ¿Funciona correctamente en todos los casos el programa dado como solución? ¿Qué ocurre si el dato almacenado en la posición `1101H` es cero?

2. ¿Cómo podría modificarse el programa para solventar ese caso particular? ¿De qué forma se lograría transmitir el error que permitiese interpretar adecuadamente esta situación?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1101H  ; Divisor en B
     MOV B, A

     LDA 1100H  ; Dividendo en A

     MVI C, 0   ; Cociente en C

REP:
     CMP B      ; ¿Es A < B?
     JC FIN     ; Si es así se ha terminado

     SUB B      ; A = A - B
     INR C      ; Incrementar cociente

     JMP REP

FIN:
     STA 1103H  ; Almacenar el resto

     MOV A, C   ; Almacenar el cociente
     STA 1102H

     RST 1
END
```

8.3.5 Operandos y resultados de más de 16 bits

Escribir programas que traten con operandos y resultados de más de 16 bits, teniendo en cuenta que el 8085 es un microprocesador con un conjunto limitado de registros de 8 bits, representa principalmente dos problemas:

1. No es posible almacenar varios datos de 32 bits o más en los registros del microprocesador, a fin de poder trabajar con ellos, ya que no hay registros suficientes ni del tamaño necesario. Uniendo `HL-DE`, por ejemplo, se podría tener un dato de 32 bits, pero los únicos registros de propósito general que quedarían disponibles serían `BC` y `A`. Por ello para operar sobre datos de estos tamaños es necesario usar como almacenamiento intermedio la memoria. En ciertos casos también es posible servirse del puntero de pila.

2. Salvo para el caso de la suma de 16 bits, el 8085 no cuenta con instrucciones que faciliten la realización de cálculos con datos de más de 8 bits. Esto implica que cualquier operación con datos de mayor tamaño habrá de efectuarse necesariamente partiendo esos datos en trozos.

Los ejercicios de este punto son solamente una introducción a la confección de programas que trabajan con datos o resultados de más de 16 bits.

Ejercicio 8.3.5.1

Se tiene en las posiciones de memoria **1100H-1003H** un número de 32 bits y en las posiciones **1104H-1007H** un segundo operando del mismo tamaño. Obtener la suma y almacenar el resultado de 32 bits a partir de la dirección **1108H**.

Indicaciones

Al igual que en ejercicios previos se efectuaban sumas de 16 bits usando instrucciones como ADD/ADC, que operan sobre 8 bits, en este caso podría recurrirse a la instrucción DAD, que suma datos de 16 bits, para en dos pasos realizar la suma de los operandos de 32 bits. Lo que se obtendría, sin embargo, sería otra solución específica, adecuada para este caso concreto. Sería mucho más útil conseguir una solución más genérica que, con pocas modificaciones, pudiera adaptarse a otros casos. Para ello no tenemos más que analizar el procedimiento general que empleamos para sumar manualmente.

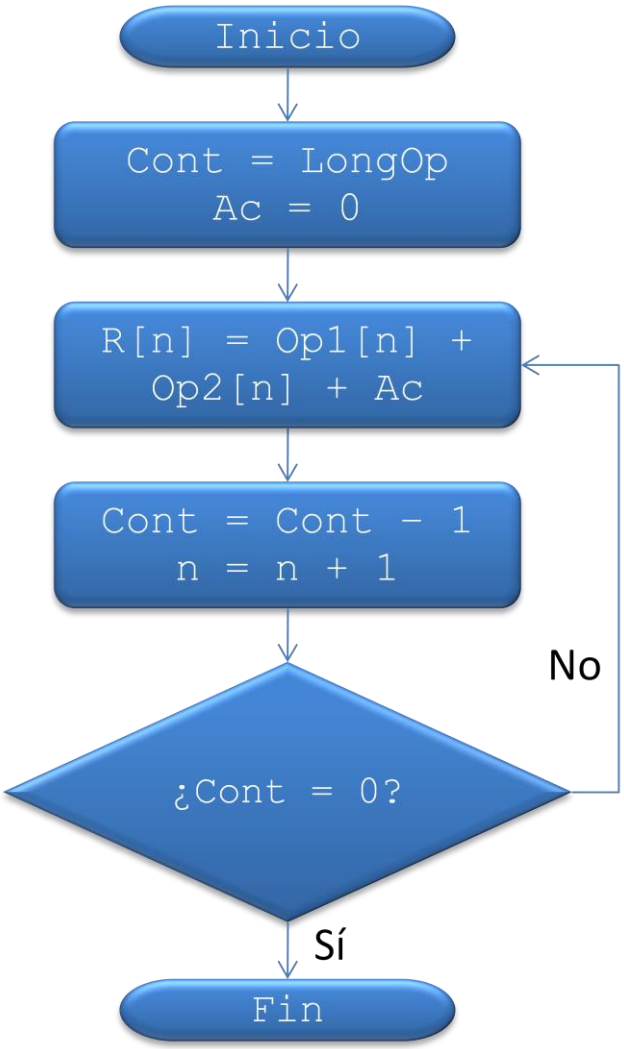
Cuando se quieren sumar a mano dos números, sin que importe el número de dígitos que tengan, siempre se sigue el mismo algoritmo:

1. Se toma el dígito menos significativo (las unidades) de ambos números, se suman y, si es necesario, se recuerda el acarreo ("me llevo una").

2. A continuación se mueve el puntero (lápiz/bolígrafo) al dígito que está a la izquierda, que es el siguiente en cuanto a unidad de magnitud. Se efectúa la suma de los dos dígitos y, si en el paso 1 hubo acarreo, también se tiene en cuenta.

3. Se repite el paso 2 hasta que ya no quedan dígitos por sumar. En ese momento se tendrá el resultado: la suma de dos números de cualquier número de cifras.

Este algoritmo puede adaptarse de forma que se tomen como dígitos bytes completos. En un dato de 32 bits las unidades equivaldrían al LSB de la palabra menos significativa, a continuación se tendría el MSB de esa misma palabra, después el LSB de la palabra más significativa y, por último el MSB de esa palabra. La misma técnica podría aplicarse para datos de más de 32 bits. El siguiente organigrama representa ese método:



Ordinograma del ejercicio 8.3.5.1.

El índice `n` representa la dirección de cada byte del resultado y de los dos operandos que intervienen en la suma, una dirección que va incrementándose a cada paso. `Ac` representa el acarreo. Es importante inicializarlo a cero, ya que de lo contrario alteraría el resultado de la suma. De esta forma en la suma de los dos primeros bytes el acarreo es 0, en la de los siguientes tendrá el acarreo que se haya generado en la suma previa.

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI SP, 2000H    ; Pila al final de memoria

     LXI D, 1100H     ; Primer operando
     LXI H, 1104H     ; Segundo operando
     LXI B, 1108H     ; Resultado

     STC         ; Asegurar que el acarreo está 0
     CMC         ; para que no influya en primera suma

     PUSH PSW    ; Guardar registro de estado

BUCLE:
     POP PSW     ; Recuperar estado anterior
     LDAX D      ; A <- [1100H]
     ADC M       ; A = A + [1104H]
     STAX B      ; [1108H] <- A

     PUSH PSW    ; Guardar estado

     INX D       ; Siguiente byte
     INX H       ; de cada operando
     INX B       ; y del resultado

     MOV A, E    ; Comprobar si se han
     CPI 04H     ; sumado los 4 bytes
     JNZ BUCLE

     POP PSW     ; Al terminar extraer la pila
     RST 1
END
```

Un aspecto destacable en este código es el uso de la pila para preservar el registro de estado después de cada suma, ya que de lo contrario el estado del bit de acarreo se perdería en la comparación que lleva a cabo la instrucción `CPI 04H`. Esto provocaría que la suma no fuese correcta.

Ejercicio 8.3.5.2

Se quiere obtener el producto de dos datos de 16 bits, obteniendo un resultado de 32 bits. El multiplicando está almacenado en las posiciones de memoria **1100H-1001H**, el multiplicador en **1102H-1103H** y el resultado debe quedar en las posiciones **1104H-1107H**.

Indicaciones

Si el anterior ejercicio planteaba el problema de actuar sobre operandos cuyo tamaño era mayor que el de los registros del microprocesador, teniendo por tanto que llevarla a cabo en trozos, el problema que conlleva este ejercicio es el insuficiente número de registros como para mantener en la memoria del microprocesador todos los datos: multiplicando, multiplicador y resultado. Los dos primeros precisan sendas parejas de registros, por ejemplo `BC` y `DE`, mientras que el resultado necesita otras dos parejas, pero la única disponible es `HL`.

En situaciones así habrá que guardar el contenido de una pareja de registros en memoria para, de esta forma, poder ir alternándolo con los datos apropiados en cada caso. Otra posibilidad sería utilizar la pila. Usando este recurso, y sabiendo que los operandos son de 16 bits y el resultado de 32, el ejercicio podría plantear de la siguiente forma:

1. El registro `BC` contendrá el multiplicador, actuando como contador de 16 bits.

2. El registro `DE` almacenará el multiplicando, siendo el dato de 16 bits que habría que sumar sucesivamente.

3. El resultado de 32 bits se almacenará en dos partes: la LSW en la pareja de registros `HL` y la MSW en la pila.

4. La parte principal del programa será un bucle en el que se sumará el contenido de `DE` a `HL`, llevando el posible acarreo a la MSW. Para ello habrá que intercambiar el dato almacenado en la pila con el contenido de `HL`, añadir el acarreo y deshacer el intercambio.

5. Al terminar, habrá que llevar `HL` y el dato almacenado en la pila a las direcciones indicadas para el resultado.

Cuestiones

1. ¿Cómo es posible intercambiar el contenido de la pareja `HL` con un dato almacenado en la pila sin que resulte afectado el estado del microprocesador?

2. ¿Habría alguna otra forma de alternar en la pareja de registros `HL` la parte alta y baja del resultado de 32 bits?

3. ¿Cómo se podría comprobar si el contenido de la pareja de registros `BC` es cero sin efectuar ninguna comparación con dicho valor?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI SP, 2000H    ; Pila al final de memoria

     LHLD 1100H       ; Multiplicando en DE
     XCHG

     LHLD 1102H  ; Multiplicador en BC
     MOV B, H
     MOV C, L

     LXI H, 0000H     ; LSB acumulador 32 bits
     PUSH H           ; MSB acumulador 32 bits

BUCLE:
     DAD D            ; HL = HL + DE
     JNC NOAC         ; Si no hay acarreo, saltar

     XTHL  ; [SP] <-> HL
     INX H            ; Incremento MSB acumulador
     XTHL

NOAC:
     DCX B            ; Queda un ciclo menos

     MOV A, B         ; ¿B = 0 y C = 0?
     ORA C

     JNZ BUCLE        ; No, continuar

     SHLD 1104H       ; Guardar LSB resultado
     POP H
     SHLD 1106H       ; y MSB del resultado

     RST 1
END
```

Ejercicio 8.3.5.3

Estudiar el código del programa mostrado a continuación y responder a las cuestiones planteadas.

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H
     LDA 1104H

     LHLD 1105H
     XCHG

     LHLD 1100H
     XCHG

     PUSH H
     XTHL

     LHLD 1102H
     MOV C, A

     STC
     CMC
     PUSH PSW

BUCLE:
     POP PSW

     LDAX D
     ADC M
     XTHL

     MOV M, A
     INX H
     XTHL

     PUSH PSW

     INX D
     INX H
     DCR C
     JNZ BUCLE

     POP PSW
     POP H

     RST 1
END
```

Cuestiones

1. ¿Qué significado tiene el contenido de las siguientes posiciones de memoria: 1100H–1101H, 1102H–1103H, 1104H y 1105H–1106H?

2. Escriba a partir de la dirección 1100H la secuencia de bytes 00H 12H 02H 12H 02H 00H 13H y a partir de la dirección 1200H la secuencia 12H 34H 34H 12H, a continuación ejecute el programa. ¿Cuál es el contenido de las posiciones de memoria 1300H–1301H? ¿Qué ha hecho el programa?

3. Describa brevemente cuál es la finalidad del programa.

4. ¿Contiene el programa algunas operaciones superfluas? ¿Cuáles?

8.3.6 Ejercicios propuestos

Ejercicio 8.3.6.1

Escribir un programa equivalente al del ejercicio 8.3.1.2 pero sin usar registros intermedios para almacenar los operandos, evitando el uso de la instrucción **LDA**.

Indicaciones

Dado que no es posible recuperar los operandos desde la memoria con anterioridad, será necesario recurrir a la operación de suma con direccionamiento indirecto por pareja de registros: `ADD M`. Los pasos elementales serían:

1. Preparar el acumulador con un contenido que no interfiera en la suma de los dos operandos.

2. Inicializar la pareja de registros que se va a utilizar como puntero a los operandos.

3. Efectuar la suma y finalizar de manera similar al ejercicio 8.3.1.2.

Cuestiones

1. ¿Con qué valor ha inicializado el acumulador para poder realizar correctamente la suma? ¿Por qué?

2. ¿Sería posible usar una pareja de registros distinta para realizar la suma de los operandos directamente desde la memoria?

3. Al efectuar la suma del primer operando con el valor inicial del acumulador, ¿existiría la posibilidad de que se activa el bit de acarreo del registro de estado? ¿Por qué?

Ejercicio 8.3.6.2

Escribir un programa equivalente al del ejercicio 8.3.1.2 pero sin usar saltos condicionales para determinar qué valor debe almacenarse en la dirección **1103H**.

Indicaciones

La resolución de este ejercicio requiere plantearse qué instrucciones existen en el repertorio del 8085 que tengan en cuenta, a la hora de ejecutarse, el estado del bit de acarreo. Dos de esas instrucciones son `JC`/`JNC`, encargadas de desviar la ejecución si dicho bit está a 1 o a 0, respectivamente. Son precisamente las instrucciones que no deben utilizarse.

Revisando la tabla 2 (conjunto de instrucciones del 8085), concretamente la sección de instrucciones aritméticas, se encuentran dos instrucciones de suma que tienen en cuenta el acarreo: `ADC` y `ACI`. Éstas suman al acumulador el operando indicado y, además, el bit de acarreo del registro de estado. El ejercicio podría plantearse de la siguiente forma:

1. Efectuar la suma de los dos operandos, como en el ejercicio 8.3.1.2. El bit de acarreo del registro de estado se verá afectado.

2. Poner a 0 el acumulador tras haber guardado el resultado.

3. Usando una de las dos instrucciones mencionadas antes, sumar el bit de acarreo al acumulador.

Ejercicio 8.3.6.3

Se quiere calcular la suma de los números 0 a 255, obteniendo el resultado como un dato de 16 bits que debe quedar almacenado en las direcciones **1100H** y **1101H**.

Indicaciones

Tomando como base el ejercicio 8.3.1.3, en el que se suman una serie de operandos de 8 bits, solamente habría que tener en cuenta los siguientes aspectos:

1. Los operandos no se recuperan de la memoria sino que han de ir calculándose a medida que se ejecuta el programa.

2. El resultado no excederá de 16 bits, puesto que se suma un total de 256 datos de 8 bits cada uno.

3. El propio contador de ciclos del bucle puede actuar como operando a sumar, ya que los datos que quieren sumarse son los mismos que irá tomando ese contador.

Cuestiones

1. ¿Cómo podría sumar valores no consecutivos, por ejemplo los números pares existentes entre 0 y 255?

2. ¿Cómo podría sumar más de 256 valores?

Ejercicio 8.3.6.4

Se tienen almacenados dos números de 16 bits, en formato *little endian*, en las direcciones **1100H-1101H** y **1102-1103H**. Obtener la suma y guardar el resultado de 16 bits en la dirección **1104-1105H**. No utilizar la instrucción **DAD**.

Indicaciones

Al no poder usar la instrucción `DAD`, las únicas instrucciones de suma disponibles serán las de 8 bits: `ADD/ADC/ADI/ACI`. Un número de 16 bits se puede interpretar como dos partes de 8 bits, dos bytes a los que se conoce como LSB y MSB. Es posible sumar esas partes por separado siempre que el acarreo del LSB se acumule en el MSB. Si la suma de los MSB también produce acarreo se descartará, puesto que se pide un resultado de 16 bits.

Cuestiones

1. ¿Produciría un resultado correcto el programa que se ha escrito si los números se hubiesen almacenado en formato *big endian* en lugar de *little endian*? ¿Por qué razón?

Ejercicio 8.3.6.5

Tomar dos datos alojados en las direcciones de memoria **1100H** y **1101H**, ambos de 8 bits, efectuar la diferencia entre ambos y almacenar el valor absoluto del resultado en la dirección **1102H**. No utilizar comparaciones ni invertir el orden de los operandos.

Indicaciones

La finalidad de este ejercicio es conseguir un programa análogo al del ejercicio 8.3.2.2, que facilite el valor absoluto de una diferencia, pero evitando el uso de la instrucción `CMP` y `el` intercambio de los registros que se daba a dicho ejercicio. ¿Cómo obtener entonces el valor absoluto? Teniendo en cuenta las siguientes reglas:

1. Si el resultado es >= 0, su valor absoluto es el propio resultado.

2. Si el resultado es < 0, su valor absoluto será el complemento a 2 del resultado.

Lo que debe hacer el programa, por tanto, es efectuar la resta y, en caso de que el resultado sea negativo, llevar a cabo el complemento a 2 del resultado.

Cuestiones

1. Según las reglas anteriores, el complemento a 2 debería efectuarse cuando el resultado obtenido de la resta sea menor estricto que 0. ¿Qué ocurriría si se llevase a cabo también en caso de que la diferencia entre los operandos fuese 0? ¿Sería correcto el resultado?

Ejercicio 8.3.6.6

Tomar dos datos alojados en las direcciones de memoria **1100H** y **1101H**, se sabe que el primero es mayor que el segundo, y calcular la diferencia entre el primero y el segundo, dejando el resultado en la dirección **1102H**. No utilizar las instrucciones de resta del 8085.

Indicaciones

Desde una perspectiva matemática la operación de resta en sí misma no existe, sino que se trata de un caso específico de suma en el que el segundo operando, el sustraendo, ha de tomarse como el simétrico del número indicado. Es decir:

```
A - B --> A + (-B)
```

En esta expresión `-B` es el simétrico de `B`, asumiendo una notación en la que el símbolo − denota el simétrico, como ocurre habitualmente en la base decimal. Al trabajar en binario el simétrico de un número se obtiene mediante complemento a 2, por lo que la expresión anterior se rescribiría como:

```
A - B --> A + (~B + 1)
```

El operador `~` indica la negación del operando que le sigue, en este caso `B`, invirtiéndose cada uno de sus bits.

Tomando como base estos conceptos puede resolverse el ejercicio de manera simple.

Ejercicio 8.3.6.7

En las posiciones **1100H-1001H** y **1102H-1103H** se tienen dos datos de 16 bits. Hallar la diferencia entre el primero y el segundo y guardar el resultado, como número de 16 bits, en las direcciones **1104H-1105H**. No usar las operaciones de resta del 8085.

Indicaciones

Al igual que en el ejercicio 8.3.6.6, lo que pretende el ejercicio es que se calcule el simétrico del sustraendo y, a continuación, se realice una suma. La diferencia es que en este caso se opera con datos de 16 bits en lugar de hacerlo con datos de 8 bits. Básicamente pueden seguirse dos caminos posibles:

1. Obtener el LSB y el MSB de sustraendo por separado, calcular el complemento a dos de cada uno y utilizar dos sumas de 8 bits teniendo en cuenta el bit de acarreo.

2. Obtener el sustraendo completo, los 16 bits, calcular el complemento a dos y recurrir a la instrucción de suma de 16 bits del 8085. Es el camino más corto.

Ejercicio 8.3.6.8

Escribir un programa análogo al del ejercicio 8.3.3.1, que multiplique el dato almacenado en **1100H** por 5, pero generando un resultado de 16 bits que debe ser almacenado en **1101H** y **1102H**.

Indicaciones

El planteamiento para resolver este ejercicio será el mismo que en el 8.3.3.1 en el que se basa, la única diferencia será que al ir sumando el multiplicando al registro donde se acumule el producto será preciso tener en cuenta el acarreo. Para ello puede seguir básicamente dos vías:

1. Seguir utilizando el acumulador de 8 bits para almacenar el producto, usando un registro adicional para sumar el acarreo a cada paso.

2. Recurrir al acumulador de 16 bits, la pareja `HL`, no teniendo que preocuparse por el acarreo a cada ciclo. La inicialización, sin embargo, habrá que adaptarla.

Cuestiones

1. ¿Existe la posibilidad de que se produzca un desbordamiento, un resultado superior a 16 bits, si el multiplicando contenido en `1100H` fuese muy grande? ¿Por qué?

Ejercicio 8.3.6.9

Se tiene en las posiciones de memoria **1100H-1101H** un dato de 16 bits que actuará como multiplicando, y en **1102H** un dato de 8 bits que hará las veces de multiplicador. Calcular el producto de 16 bits y almacenarlo en **1103H-1104H.**

Indicaciones

Partiendo de la solución dada al ejercicio previo, el 8.3.6.8, habrá que hacer unos cambios mínimos para conseguir que el multiplicando sea de 16 bits en lugar de tener un tamaño de 8 bits, especialmente si se utilizó el acumulador de 16 bits y la instrucción de suma `DAD`. Dichos cambios serán:

1. Cambiar la lectura del multiplicando de 8 bits de la posición `1100H` para que ahora se lean las posiciones `1100H-1101H` interpretándolo como multiplicando de 16 bits.

2. Ajustar la lectura del multiplicador, que antes estaba en la posición `1101H` y ahora se encuentra en la posición `1102H`.

3. Ajustar la escritura del resultado para llevarlo a las posiciones `1103H-1104H` y que sigue siendo de 16 bits.

Cuestiones

1. ¿Cuál debería ser el número de bits del resultado para que no se perdiese parte del producto dependiendo de los datos que actúan como operandos?

Ejercicio 8.3.6.10

Se tiene en las posiciones de memoria **1100H-1101H** un dato de 16 bits que actuará como dividendo, y en **1102H** un dato de 8 bits que hará las veces de divisor. Efectuar la división obteniendo un cociente de 8 bits, que quedará almacenado en **1103H**, y un resto de 8 bits, a guardar en la posición **1104H**. El bit de acarreo debe activarse si se produce una división por cero.

Indicaciones

Efectuar la división de un operando de 16 bits supone una dificultad añadida, ya que a la carencia de instrucciones para dividir o multiplicar en el 8085, hay que agregar el hecho de que tampoco es posible restar datos de más de 8 bits, al menos de manera directa. Como consecuencia el bucle central de la solución, en el que van efectuándose las restas sucesivas, será más complejo ya que es preciso restar un valor de 8 bits de otro de 16 bits y, además, comprobar cuándo dicha operación ya no será posible.

Para la consecución del ejercicio resultarán útiles las siguientes indicaciones:

1. El dividendo es de 16 bits y, por tanto, deberá ser almacenado en una pareja de registros. El divisor, por el contrario, puede seguir guardándose en un registro de 8 bits.

2. Para saber si el dividendo es menor que el divisor, y por tanto no es posible seguir dividiendo, debe tenerse en cuenta que el primero es de 16 bits y el segundo de 8. Únicamente cuando el MSB del dividendo sea igual a cero y el LSB sea inferior al divisor deberá terminarse. Si el LSB del dividendo es menor que el divisor pero el MSB no es cero, el dividendo seguirá siendo mayor que el divisor.

3. A la hora de restar un dato de 8 bits (el divisor) de uno de 16 (el dividendo) habrá que usar el LSB del dividendo como minuendo y el divisor como sustraendo, teniendo en cuenta el indicador de acarreo para ir decrementando el MSB del dividendo.

Ejercicio 8.3.6.11

Escribir un programa que calcule la suma de 10 datos de 32 bits, almacenados en formato *little endian* a partir de la dirección **0000H**. Almacenar el resultado, también de 32 bits, a partir de la dirección **1100H**.

Indicaciones

Dado que no es posible mantener dos datos de 32 bits simultáneamente en los registros del 8085, para realizar la suma habrá que recurrir a la técnica explicada en el ejercicio 8.3.5.1. La diferencia es que en este caso no son solamente dos números, sino 10, por lo que al bucle que recorre los bytes de cada operando habrá que agregar otro, externo al anterior, que recorra los operandos.

Cuestiones

1. ¿Cuál es el resultado que se obtiene a partir de la suma?

2. ¿Puede ser distinto dicho resultado al ejecutar el programa varias veces? ¿Por qué?

8.4 Trabajo a nivel de bits

Cuando se programa un sistema basado en microprocesador utilizando un lenguaje de bajo nivel, como es el ensamblador, ciertas operaciones requieren el trabajo no solamente con registros internos del microprocesador y posiciones individuales de memoria, sino que llegan hasta el punto de controlar bits individuales, ya sea para comprobar su estado o para alterarlo. Es lo que ocurre, por ejemplo, al trabajar con ciertos dispositivos de E/S como el PPI (8255), el controlador de teclado (8279), y también con datos de estado del microprocesador, como puede ser la máscara de interrupciones.

El 8085 dispone de todas las instrucciones necesarias para operar al nivel de bits y, para alcanzar su conocimiento, los ejercicios de esta sección tienen los siguientes objetivos:

- Saber cómo puede activarse, desactivarse o invertirse un cierto bit de un byte.
- Aprender a comprobar el estado de un bit determinado, realizando una tarea u otra dependiendo de que se encuentre a 0 o a 1.
- Conocer la utilidad que pueden tener las rotaciones y desplazamientos de bits, aplicándolas a diversos problemas.
- Aprender a combinar distintas operaciones sobre bits para componer o extraer patrones de bits.

8.4.1 Activar y desactivar bits concretos

Ejercicio 8.4.1.1

> Se tiene en la posición de memoria **1100H** un dato de 8 bits. Escribir un programa que active (ponga a 1) el quinto bit de dicho dato y lo deposite en la posición **1101H**.

Indicaciones

Para completar este ejercicio deben tenerse en cuenta los aspectos siguientes:

1. Los bits de un byte se cuentan de derecha a izquierda, es decir, desde el bit menos significativo al más significativo. El primer bit suele denominarse bit 0 y el octavo, bit 7.

2. Todas las operaciones de manipulación de bits se llevan a cabo mediante el conjunto de instrucciones lógicas del 8085, que pueden revisarse en la sección *Operaciones lógicas* de la tabla 2.

3. Habrá que determinar qué operación es la apropiada para activar un bit y, además, con qué dato hay que utilizarla para conseguir activar el bit indicado en el ejercicio.

El programa en sí es muy simple y se limitará a leer el contenido de la posición `1100H`, aplicar la operación lógica con el valor adecuado y devolver el resultado a la posición `1101H`.

Cuestiones

1. ¿Qué operación lógica se ha utilizado para activar el bit?

2. ¿Cuál ha sido el dato utilizado para activar el quinto bit? ¿Por qué razón?

3. ¿Qué ocurrirá si el bit que se quiere activar ya está activo?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Se recupera el dato

     ORI 10H    ; Activar el quinto bit

     STA 1101H  ; Y se guarda el resultado

     RST 1

END
```

Ejercicio 8.4.1.2

Se tiene en la posición de memoria **1100H** un dato de 8 bits. Escribir un programa que desactive (ponga a 0) el quinto bit de dicho dato y lo depo-site en la posición **1101H**.

Indicaciones

Son aplicables las indicaciones dadas en el ejercicio 8.4.1.1, pero lógicamente determinando cuál sería la operación lógica y el dato que permitirán desactivar un bit en lugar de activarlo.

Cuestiones

1. ¿Qué operación lógica se ha utilizado para desactivar el bit?

2. ¿Cuál ha sido el dato utilizado para desactivar el quinto bit? ¿Por qué razón?

3. ¿Qué ocurrirá si el bit que se quiere desactivar ya está a cero?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Se recupera el dato

     ANI 0EFH   ; Desactivar el quinto bit

     STA 1101H  ; Y se guarda el resultado

     RST 1

END
```

Ejercicio 8.4.1.3

Se tiene en la posición de memoria **1100H** un dato de 8 bits. Escribir un programa que invierta el quinto bit de dicho dato y lo devuelva a la misma posición de memoria.

Indicaciones

Son aplicables las indicaciones dadas en el ejercicio 8.4.1.1, pero lógicamente determinando cuál sería la operación lógica y el dato que permitirán invertir un bit en lugar de activarlo o desactivarlo.

Cuestiones

1. ¿Qué operación lógica se ha utilizado para invertir el bit?
2. ¿Cuál ha sido el dato utilizado para invertir el quinto bit? ¿Por qué razón?
3. ¿Qué ocurre al ejecutar el programa varias veces consecutivas?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Se recupera el dato

     XRI 10H    ; Invertir el quinto bit

     STA 1100H  ; Y se guarda el resultado

     RST 1

END
```

8.4.2 Comprobar el estado de un bit

Ejercicio 8.4.2.1

En las posiciones de memoria **1100H** y **1101H** se han almacenado dos datos de 8 bits de los cuales quiere calcularse la diferencia. Se tiene en la posición de memoria **1102H** un indicador. Escribir un programa que compruebe el estado del quinto bit de dicho indicador, de forma que si está a 1 se halle la diferencia entre el primer operando y el segundo y si está a 0 se calcule la diferencia entre el segundo y el primero. Almacenar el resultado en la posición **1103H**.

Indicaciones

El programa que se solicita en este ejercicio tiene dos partes comunes: la recuperación de los operandos y cálculo de la diferencia entre ellos, y una parte que se ejecutará dependiendo del indicador: el intercambio de los operandos. Se podría, por tanto, estructurar en tres bloques:

1. Transferencia de los operandos desde la memoria a registros del microprocesador. El acumulador será usado a continuación para comprobar el indicador, por lo que los operandos deben quedar en otros registros.

2. Lectura del indicador, comprobación del quinto bit y, en caso de que esté activo/inactivo (según interese), intercambio de los operandos que se ya se tenían en los registros.

3. Cálculo de la diferencia y almacenamiento del resultado.

Cuestiones

1. ¿Qué operación lógica se ha usado para comprobar el estado del bit y qué valor se ha aplicado como máscara?

2. ¿Habría alguna otra forma de comprobar el estado de un cierto bit en un dato?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Se recupera el primer operando
     MOV E, A   ; y se guarda en E
     LDA 1101H  ; Se recupera el segundo operando
     MOV L, A   ; y se guarda en L

     LDA 1102H  ; Se obtiene el indicador y se ponen
     ANI 10H    ; a 0 todos los bits menos el quinto

     ; Si el resultado es cero, no estaba activo
     JZ RESTA
     XCHG  ; En caso contrario invertir operandos

RESTA:
     MOV A, L   ; Recuperar un operando
     SUB E ; y restar el otro

     STA 1103H  ; Y se guarda el resultado

     RST 1

END
```

8.4.3 Rotaciones

Ejercicio 8.4.3.1

Escribir un programa que tome el dato de 8 bits almacenado en la posición **1100H**, intercambie el *nibble* alto con el bajo y deposite el resultado en la posición **1101H**.

Indicaciones

La única dificultad que plantea este ejercicio estriba en cómo llevar los cuatro bits de menor peso del dato, el *nibble* bajo, a la posición de los cuatro bits de mayor peso y viceversa. Aunque podrían seguirse diferentes métodos para realizar esa operación, la más simple y rápida consiste en recurrir a las operaciones de rotación con que cuenta el 8085. De ellas hay dos que pueden utilizarse para desplazar bits dentro de un byte sin que se pierda ninguna información.

Cuestiones

1. ¿Habría alguna diferencia entre utilizar la instrucción `RRC` en lugar de `RLC` en este caso concreto?

2. ¿Habría alguna diferencia entre utilizar la instrucción `RAL` en lugar de `RLC` en este caso concreto?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Se recupera el primer operando

     RLC        ; Se rotan cuatro bits
     RLC
     RLC
     RLC

     STA 1101H  ; Y se guarda el resultado

     RST 1

END
```

Ejercicio 8.4.3.2

Se quiere obtener la división entera del dato de 8 bits almacenado en la posición **1100H** entre 4. Colocar el resultado en la posición **1101H**. Usar exclusivamente operaciones lógicas, no aritméticas.

Indicaciones

Ciertas operaciones aritméticas, entre ellas el producto y la división, pueden efectuarse mediante desplazamientos de dígitos y agregando ceros a la derecha o la izquierda, respectivamente. Trabajando en base diez, por ejemplo, si se toma el número 12, se desplazan sus dígitos una posición a la izquierda y se agrega un 0 al final, lo que se ha conseguido es multiplicar el dato por la base, por diez. De manera análoga podría efectuarse la división, desplazando los dígitos hacia la derecha y colocando un 0 a la izquierda.

Cuando se opera con bits (dígitos binarios) la base no es 10 sino 2, por lo que un desplazamiento a izquierda o derecha equivaldrá a una multiplicación o división por 2. Para multiplicar o dividir por 4, por tanto, bastará con desplazar los bits dos posiciones en el sentido correcto.

Puesto que se solicita una división entera, el sentido de la rotación deberá ser hacia la derecha. Los bits que salgan del dato pueden ser descartados, puesto que no se pide el resto, y los nuevos bits que entren por la izquierda deberán estar a 0 para no alterar el resultado.

Cuestiones

1. ¿Se podría haber utilizado la instrucción `RRC` en lugar de `RAR` para efectuar la rotación en este programa? ¿Había que introducir algún cambio adicional si se pudiera utilizar?

2. ¿Qué ocurre si el operando a dividir, facilitado en la dirección `1100H`, es igual a cero?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Se recupera el primer operando

     ORI 00H    ; Acarreo = 0
     RAR
     ORI 00H
     RAR

     STA 1101H  ; Y se guarda el resultado

     RST 1

END
```

8.4.4 Extracción y composición de patrones de bits

Ejercicio 8.4.4.1

Se tiene un programa que procesa tablas de datos, realizando distintas operaciones con ellas. Dichas tablas se almacenan en la memoria del sistema y su primer byte es una cabecera que tiene la siguiente estructura:

Nº bit	7	6	5	4	3	2	1	0
Significado	**Pr**	**F**	**F**	**F**	**C**	**C**	**C**	**N**

El bit de menor peso indica si en la tabla hay o no datos nuevos, mientras que el bit de mayor peso comunica si la tabla ya ha sido procesada o está por procesar. Los tres bits **F** establecen el número de filas de la tabla y los tres bits **C** el número de columnas. No pueden existir tablas nulas, con 0 filas o 0 columnas, por lo que el valor **000** en estos bits se interpretará como 8 en lugar de 0.

Escribir un programa que componga esa cabecera a partir de los datos almacenados en las siguientes posiciones de memoria:

1100H –> Número de filas
1101H –> Número de columnas
1102H –> Indicador de datos nuevos
1103H –> Indicador de tabla procesada

Dejar el byte con la cabecera codificada en la posición **1104H**.

Indicaciones

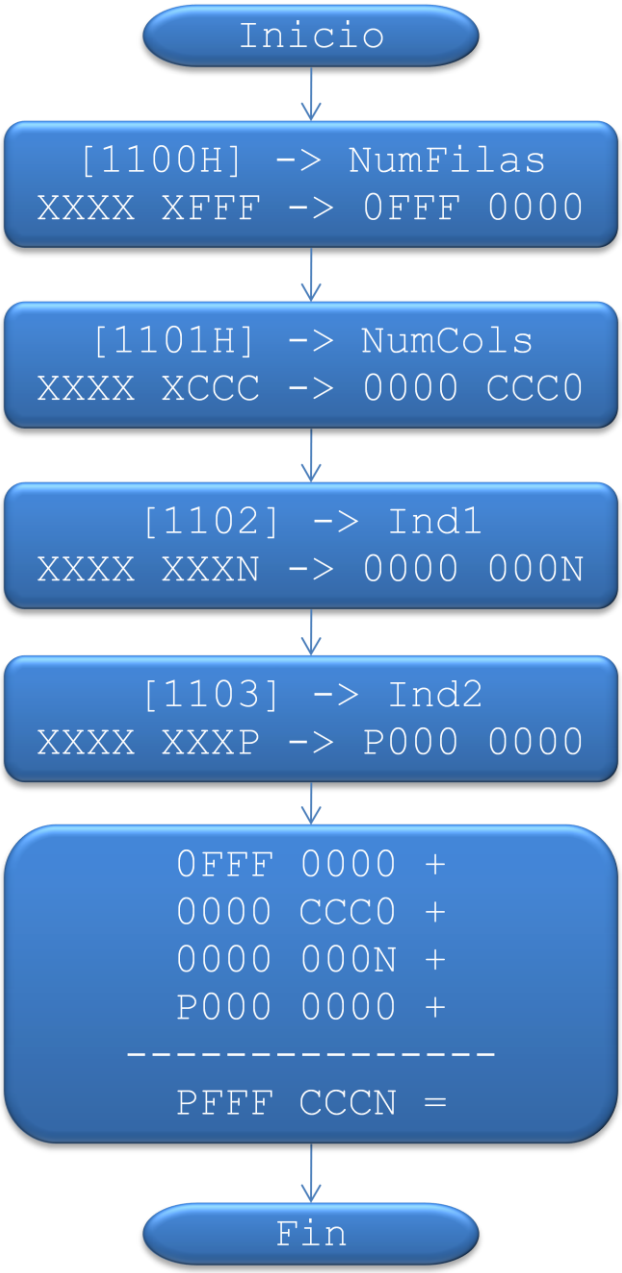
Lo que se pretende es compactar en un único byte distintos trozos de información que, a pesar de encontrarse repartidos en varias posiciones de memoria porque es la forma más fácil en que puede introducirlos un usuario, cada una de ella solamente necesita unos pocos bits para ser representada.

El programa para realizar la tarea encomendada en el ejercicio ha de dividirse en dos partes:

1. Se tomará cada trozo de información, cada dato individual de las direcciones indicadas, y usando instrucciones de rotación se colocarán sus bits en las posiciones que les corresponden, dejando el resto de los bits del byte a 0. Estos datos ya ajustados en cuanto a posición se irán guardando en distintos registros del 8085.

2. Una vez se tengan todos los datos preparados, bastará con combinarlos en un único byte mediante la operación lógica adecuada. Ésta debe ir activando en el acumulador los bits que estén activados en cada dato.

Las operaciones a llevar a cabo quedan resumidas en el siguiente diagrama de flujo:



Ordinograma del ejercicio 8.4.4.1.

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Número de filas

     RLC        ; Colocarlo en la posición
     RLC        ; que le corresponde
     RLC
     RLC
     ANI 70H    ; Dejar a 0 los demás bits
     MOV D, A   ; y guardar

     LDA 1101H  ; Número de columnas

     RLC        ; Colocar en su posición
     ANI 0EH    ; Dejar a 0 los demás bits
     MOV E, A   ; y guardar

     LDA 1103H  ; Indicador tabla procesada

     RRC        ; Colocar en su posición
     ANI 80H    ; Dejar a 0 los demás bits
     MOV B, A   ; y guardar

     LDA 1102H  ; Indicador de datos nuevos

     ORA D      ; Agregar el resto de bits
     ORA E
     ORA B

     STA 1104H  ; Guardar la cabecera

     RST 1

END
```

Ejercicio 8.4.4.2

Partiendo del mismo supuesto descrito en el ejercicio 8.4.4.1, escribir el programa que llevaría a cabo la descodificación de la cabecera de la tabla. Esa cabecera se encuentra almacenada en la posición **1104H**, debiendo extraerse el número de filas en la posición **1100H**, el de columnas en **1101H**, el indicador de datos nuevos en **1102H** y el indicador de tabla procesada en **1103H**.

Indicaciones

El programa que solicita este ejercicio deberá efectuar el trabajo inverso al del ejercicio 8.4.4.1, extrayendo de un único byte varios trozos de información. Las instrucciones a usar son básicamente las mismas: rotaciones y operaciones lógicas para poner a 0 los bits que no interesan.

El diagrama de flujo del mismo ejercicio 8.4.4.1 servirá como guía, simplemente hay que invertir cada paso y obviar el último en el que se unían las distintas partes.

Cuestiones

1. Tal y como se utilizan en estos dos últimos ejercicios las operaciones lógicas OR y AND, ¿a qué operaciones aritméticas equivaldrían?

2. ¿Se podrían haber utilizado las instrucciones de rotación `RAL/RAR` en lugar de `RLC/RRC`? ¿Habría que hacer algún cambio adicional al sustituirlas?

3. ¿Qué mejoras podrían introducirse en el programa que se ofrece como solución?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1104H  ; Número de filas

     RRC        ; Colocarlo en la posición
     RRC        ; que le corresponde
     RRC
     RRC
     ANI 07H    ; Dejar a 0 los demás bits
     STA 1100H  ; y guardar

     LDA 1104H  ; Número de columnas

     RRC        ; Colocar en su posición
     ANI 07H    ; Dejar a 0 los demás bits
     STA 1101H  ; y guardar

     LDA 1104H  ; Indicador tabla procesada

     RLC        ; Colocar en su posición
     ANI 01H    ; Dejar a 0 los demás bits
     STA 1103H  ; y guardar

     LDA 1104H  ; Indicador de datos nuevos
     ANI 01H    ; Dejar a 0 los demás bits
     STA 1102H  ; y guardar

     RST 1

END
```

8.4.5 Ejercicios propuestos

Ejercicio 8.4.5.1

Escribir un programa que tome el contenido de la posición **1100H** e invierta su quinto bit, dejando el resultado en la misma dirección. No utilizar la operación lógica XOR.

Indicaciones

Dado que el ejercicio indica que no debe utilizarse la operación lógica XOR, el programa deberá necesariamente comprobar el estado del bit y, a continuación, tomar dos caminos distintos: si está a 0 se pasa a ponerlo a 1 y viceversa. Para ello se usarán instrucciones lógicas AND y OR.

Ejercicio 8.4.5.2

Se quiere cifrar una información, concretamente una secuencia de 8 bytes almacenada a partir de la posición **1100H**, utilizando como clave de cifrado la memoria EPROM del uP-2000. Para ello el programa debe hacer una operación lógica XOR byte a byte entre mensaje y clave, dejando el criptograma (mensaje cifrado) a partir de la dirección **1200H**.

Indicaciones

Muchas técnicas de cifrado de información basan su funcionamiento en la aplicación de una operación lógica XOR entre los bits del mensaje original y una clave, mantenida normalmente en secreto, obteniendo así una versión cifrada a la que se denomina criptograma. Este ejercicio propone utilizar como clave de cifrado el contenido de la memoria EPROM del sistema uP-2000, de forma que solamente alguien que tenga este equipo, y sepa la dirección de memoria de la que se ha partido para llevar a cabo el cifrado, podría descifrar el mensaje y leerlo.

El programa tendrá que usar varias parejas de registros como punteros, de forma que pueda:

1. Leer un byte del mensaje original.
2. Leer un byte de la clave (memoria EPROM)
3. Efectuar la operación lógica XOR entre ambos datos.
4. Depositar el resultado en la posición adecuada.
5. Hacer avanzar los tres punteros y repetir la operación 8 veces.

Por tanto el programa se dividirá en dos grandes bloques: inicialización de los registros a usar como punteros y del contador del bucle, por una parte, y el bucle en el que se llevará a cabo el cifrado, por otra.

Cuestiones

1. Introduzca a partir de la dirección `1100H` la secuencia `FA BA DA 0D E0 CE BA DA` y ejecute el programa. Examine el resultado a partir de la dirección `1200H`. ¿Cuál es la secuencia de bytes obtenida? ¿Resultaría fácil deducir el mensaje original?
2. Copie los 8 bytes del criptograma, que está en `1200H`, a partir de la dirección `1100H`, sustituyendo al mensaje original, y ejecute el programa. ¿Qué

resultado obtiene a partir de la dirección 1200H? ¿A qué se debe dicho resultado?

3. ¿Qué ocurriría si para descifrar se tomase como clave una secuencia de bytes de la EPROM distinta, por ejemplo a partir de la dirección 0100H en lugar de 0000H?

Ejercicio 8.4.5.3

En la posición de memoria **1100H** se tiene almacena un dato de 8 bits del que quiere saberse el número de bits que tiene a 1. Contarlos y dejar el resultado en la posición de memoria **1101H**.

Indicaciones

Contar los bits que están a 1 o a 0 en un byte es una tarea que puede efectuarse de distintas formas pero, en cualquier caso, habrá que recorrer esos bits uno a uno y comprobar cuál es su estado. Para ello puede recurrirse a operaciones lógicas, del tipo AND y OR, o bien a las instrucciones de rotación de bits a través del bit de acarreo del registro de estado. Esta última seguramente sea la opción más rápida y simple.

Ejercicio 8.4.5.4

En la posición de memoria **1100H** se tiene almacena un dato de 8 bits. Escribir un programa que extraiga los bits individuales y los coloque cada uno en un byte, a partir de la posición de memoria **1101H**.

Indicaciones

El nudo central del programa que solicita este ejercicio será prácticamente idéntico al del ejercicio 8.4.5.3, con la salvedad de que ahora no habrá que incrementar un contador cada vez que se encuentre un bit a 1, sino que habrá que escribir ese 1, o un 0 si el bit está a 0, en la posición de memoria adecuada, avanzando a continuación a la siguiente.

Cuestiones

1. ¿Se podría realizar el mismo programa sin utilizar operaciones de rotación de bits, exclusivamente con operaciones lógicas y aritméticas?

Ejercicio 8.4.5.5

Escribir un programa que tome el dato de 8 bits almacenado en la posición **1100H**, intercambie el *nibble* alto con el bajo y deposite el resultado en la posición **1101H**. No usar instrucciones de rotación de bits.

Indicaciones

En ejercicios previos de esta sección se ha visto cómo las rotaciones pueden utilizarse para multiplicar o dividir un número en potencias de 2, de forma que desplazar un bit hacia la izquierda supone multiplicar el operando por 2, desplazarlo dos bits multiplicar por 4 y así sucesivamente. De manera análoga, al desplazar a la derecha se realizan divisiones.

Si las rotaciones pueden actuar como productos y divisiones, el proceso inverso también es válido y mediante productos y divisiones es posible desplazar los bits de un byte. Por ello este ejercicio puede plantearse de la siguiente forma:

1. Tomar el byte original, dividirlo entre 16 (número de posibles valores que se forman con 4 bits, 2^4=16) y quedarse con los cuatro bits menos significativos del cociente. Éstos serán exactamente los cuatro bits de mayor peso del byte original.

2. Tomar el byte original, quedarse con los cuatro bits menos significativos (poniendo a 0 los de mayor peso) y multiplicarlo por 16, de forma que se desplacen a la izquierda 4 bits.

3. Unir las dos partes que se han obtenido en los pasos previos, obteniendo el byte con los *nibbles* invertidos.

Ejercicio 8.4.5.6

Se tiene en la posición de memoria **1100H** un dato de 8 bits, del cual habrá que tomar el bit indicado en la dirección **1101H** (0 a 7) y activarlo o desactivarlo, según que en la posición **1102H** haya un 1 o un 0, respectivamente. Dejar el resultado en la posición **1103H**.

Indicaciones

Para completar este ejercicio plantee el programa como una división de los siguientes problemas:

1. A partir del dato alojado en la posición 1101H, un número entre 0 y 7 que indica el bit que va a manipularse, calcular la máscara necesaria para ponerlo a 1. Para ello es necesario recordar el valor de un bit según su posición o bien recurrir a las instrucciones de rotación.

2. Si el indicador de la posición 1102H dice que hay que poner el bit a 0, no a 1, hay que invertir la máscara obtenida en el paso anterior. Si para poner a 1 el bit 4 se usa la máscara 0001 0000, para poner a 0 ese mismo bit la máscara adecuada es 1110 1111.

3. Leer el dato de la posición 1100H y aplicar la máscara con la operación lógica que corresponda, según el indicador de la dirección 1102H, dejando el resultado en 1103H.

8.5 Búsqueda de datos

En las secciones previas de ejercicios se han puesto en práctica multitud de instrucciones del 8085: transferencia de datos desde y hacia la memoria, comparación de datos, realización de operaciones aritméticas y lógicas, saltos condicionales, etc. En esta sección lo que se persigue es afianzar lo aprendido poniendo en práctica la mayoría de esas instrucciones en distintos ejercicios, en los que no se introducen nuevas instrucciones.

La finalidad de los ejercicios de esta sección es la siguiente:

- Aplicar las operaciones aritméticas en el cálculo de direcciones de memoria a fin de direccionar tablas de datos.
- Aprender a implementar bucles con más de 256 ciclos.
- Saber cómo pueden combinarse distintas instrucciones ya conocidas para efectuar búsquedas de datos en la memoria.
- Combinar la búsqueda de datos con otras operaciones, como puede ser el conteo de coincidencias o la sustitución de unos datos por otros.

8.5.1 Cálculo de direcciones en tablas

Ejercicio 8.5.1.1

A partir de la dirección **200H** se tiene una tabla de datos formada por 32 filas de 8 columnas cada una y conteniendo cada elemento un byte. Calcular la dirección del elemento cuya fila se indica en la posición de memoria **1100H** y su columna en la posición **1101H**, almacenándola en las posiciones **1102H-1103H**. Leer el dato contenido en ese elemento y dejarlo en la posición **1104H**. Los índices de fila y columna vienen expresados a partir de 1, por lo que pueden tomar los valores 1 a 32 y 1 a 8, respectivamente.

Indicaciones

El cálculo de la dirección que corresponde al elemento requerirá los pasos siguientes:

1. Colocar la dirección base, en la que comienza la tabla, en una pareja de registros que va a actuar como puntero.

2. Sumar a dicha dirección base 32 bytes por cada fila completa a saltar para buscar el elemento deseado. Dado que ese elemento se encuentra en la fila indicada en la posición `1100H`, el número de filas a saltar será dicho número menos 1, de forma que la pareja de registros quede con la dirección del primer elemento de dicha fila.

3. A continuación habrá que sumar a la pareja de registros el índice de columna menos uno, momento en el que se tendrá la dirección del elemento deseado.

Para comprobar que el programa opera correctamente introduzca en las posiciones de memoria los valores `01H` y `01H`, es decir, para leer el primer elemento de la primera fila de la tabla. La dirección calculada, devuelta en las posiciones `1102H-1103H`, debe ser `0200H` y el valor leído de esa posición ha de ser `01H`.

Cuestiones

1. ¿Cuántos elementos tiene en total la tabla indicada?

2. Sabiendo cuál es la dirección de inicio de la tabla, ¿cuál es su dirección de fin, la dirección que corresponde al último elemento?

3. Si la tabla comenzase en la posición `1200H` y terminase en la posición `13FFH` ¿cuántos bytes la formarían? ¿Cuántas filas tendría si cada una se compone de 8 columnas?

Solución

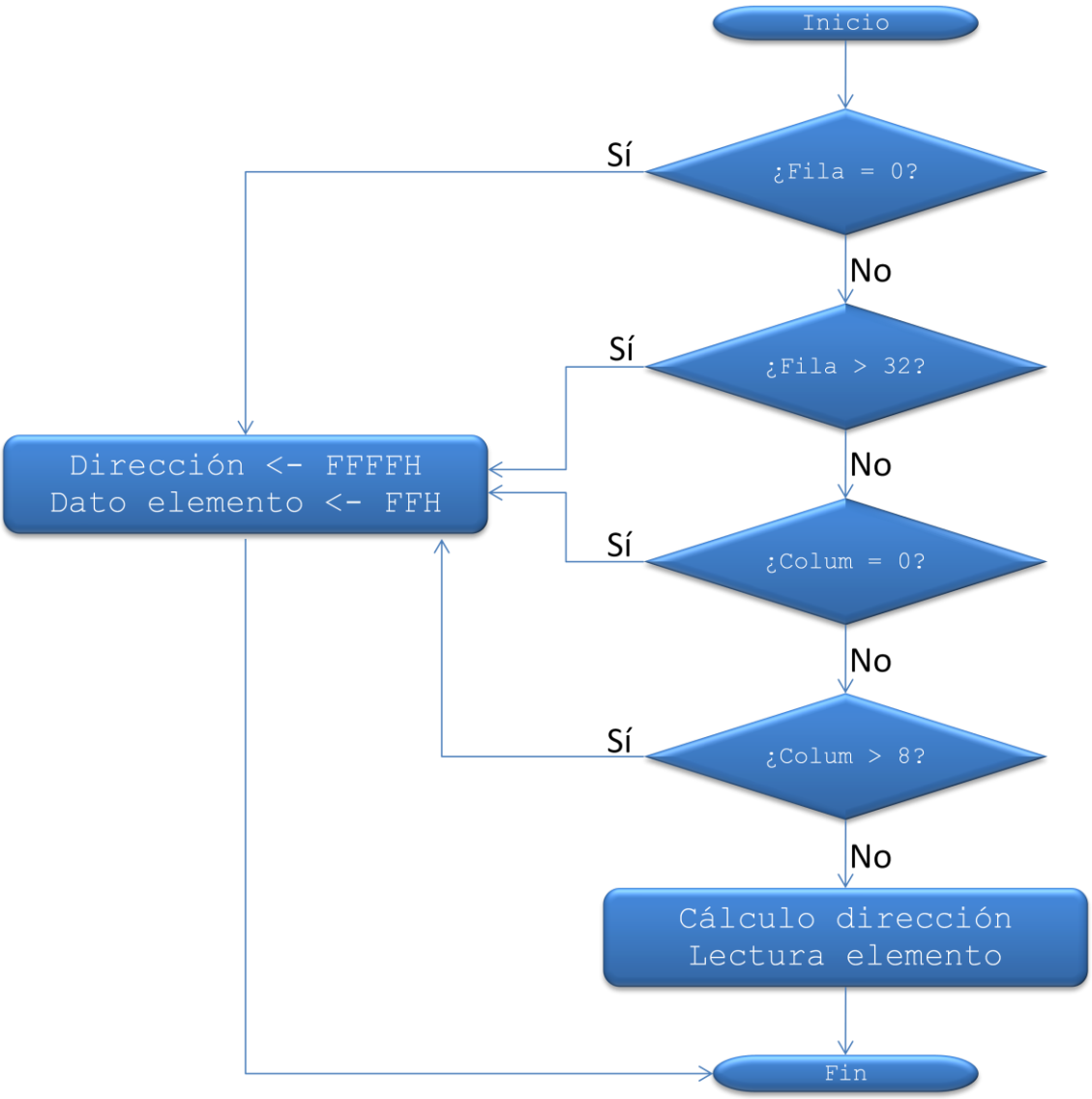
```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H   ; Se recupera la fila

     LXI H, 200H ; Inicio de la tabla
     LXI D, 20H  ; Elementos por fila

BUCLE:
     DCR A       ; Reducir número fila
     JZ FIN      ; Si es 0, terminar

     DAD D       ; Si no, sumar a dirección
     JMP BUCLE

FIN:
     LDA 1101H   ; Obtener número de columna
     DCR A
     MOV E, A
     DAD D       ; y sumarlo a la dirección

     SHLD 1102H ; Se guarda la dirección
     MOV A, M   ; Se lee el dato
     STA 1104H  ; y se guarda

     RST 1

END
```

Ejercicio 8.5.1.2

Aunque en el ejercicio anterior se indican las dimensiones de la tabla, ¿qué ocurre si se coloca el valor **FFH** en las posiciones **1100H** y **1101H**? El programa calcula la dirección y recupera el dato de una posición de memoria que no corresponde a la tabla. Adecuar el programa para que nunca acceda a datos fuera de la tabla, de forma que si los índices son incorrectos se escriba en **1102H-1103H** la dirección **FFFFH** y en **1104H** el valor **FFH**.

Indicaciones

El proceso de verificación de los límites aplicables a los índices no tiene que ver directamente con el núcleo del programa, cuya finalidad es calcular la dirección del elemento y devolver su valor, por lo que puede ser planteado como una parte separada que se ejecuta al inicio, según puede verse en el siguiente diagrama de flujo:



Ordinograma del ejercicio 8.5.1.2.

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LDA 1100H  ; Examinar la fila
     ORA A      ; Comprobar que no sea 0
     JZ INCOR
     CPI 21H    ; Ni mayor que 32
     JNC INCOR

     LDA 1101H  ; Examinar la columna
     ORA A      ; Comprobar que no sea 0
     JZ INCOR
     CPI 09H    ; Ni mayor que 8
     JNC INCOR

     ; Procedimiento de cálculo previo

     LDA 1100H  ; Se recupera la fila

     LXI H, 200H ; Inicio de la tabla
     LXI D, 20H  ; Elementos por fila

BUCLE:
     DCR A      ; Reducir número fila
     JZ FIN     ; Si es 0, terminar

     DAD D      ; Si no, sumar a dirección
     JMP BUCLE

FIN:
     LDA 1101H  ; Obtener número de columna
     DCR A
     MOV E, A
     DAD D      ; y sumarlo a la dirección

     SHLD 1102H ; Se guarda la dirección
     MOV A, M   ; Se lee el dato
     STA 1104H  ; y se guarda

     RST 1
INCOR:
     LXI H, 0FFFFH ; Señalizar error
     SHLD 1102H
     MOV A, L
     STA 1104H

     RST 1
END
```

8.5.2 Recorrer bloques de más de 256 bytes

Ejercicio 8.5.2.1

Se tiene una tabla de elementos de 8 bits de tamaño y de 32 filas por 32 columnas alojada a partir de la dirección **0000H**. Hallar la suma de los elementos que ocupan las últimas 16 filas de dicha tabla y depositarla como un dato de 16 bits en las posiciones **1100H-1101H**.

Indicaciones

Al operar con tablas de datos no puede asumirse que el número de elementos que contienen es inferior a 256, por lo que si es preciso recorrer un bloque de elementos habrá que hacerlo pensando que el número de ciclos del bucle puede ser superior a ese límite. En consecuencia el contador deberá tener un tamaño de 16 bits y no de 8.

En este ejercicio la estructura de la tabla y la dirección donde se encuentra son datos constantes, por lo que puede calcularse la directamente la dirección donde comienza la segunda mitad de la tabla, así como el número de elementos que es necesario recorrer, de forma que el programa se limitaría a un bucle que recorrería una franja de memoria sumando su contenido.

Cuestiones

1. ¿Cuál es el valor obtenido en las posiciones `1100H-1101H` al ejecutar el programa?

2. ¿Sería suficiente un contador de 16 bits para recorrer cualquier tabla de datos que pudiera alojarse en un sistema basado en el microprocesador 8085?

3. ¿Por qué tras decrementar el contenido de la pareja de registros que actúa como contador se usa el acumulador y la operación lógica OR a fin de comprobar si es cero? ¿Podría efectuarse la comprobación de otra forma?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     ; Dir inicial = 0000H + 10H filas x 20H columnas
     LXI D, 0200H
     ; Núm de elementos = 10H filas x 20H columnas
     LXI B, 0200H
     LXI H, 0000H     ; Suma

BUCLE:
     LDAX D      ; Leer un elemento de la tabla
     ADD L       ; y sumarlo con LSB suma
     MOV L, A    ; Guardar LSB suma
     MOV A, H    ; Tomar MSB suma
     ACI 00H     ; y sumar acarreo
     MOV H, A    ; Guardar MSB

     INX D       ; Siguiente elemento de la tabla
     DCX B       ; Queda un elemento menos

     MOV A, B    ; ¿BC = 0?
     ORA C

     JNZ BUCLE

     SHLD 1100H ; Guardar suma

     RST 1
END
```

8.5.3 Búsqueda del máximo y mínimo

Ejercicio 8.5.3.1

Escribir un programa que examine toda la EPROM del sistema uP-2000 buscando el valor mínimo y el máximo, que devolverá almacenándolos en las posiciones **1100H** y **1101H** respectivamente.

Indicaciones

Lo que pretende este ejercicio es consolidar el aprendizaje en cuanto al uso de bucles de 16 bits se refiere, combinándolo con la utilización de comparaciones y saltos condicionales. La solución requiere dar los pasos siguientes:

1. Elegir un registro para guardar el máximo y otro para el mínimo mientras está ejecutándose el programa. Esos dos registros deberán tener un valor inicial que pueda compararse con los valores que vayan leyéndose de la EPROM.

2. Inicializar un puntero y un contador para recorrer la memoria EPROM del sistema uP-2000 que, como se sabe, comienza en la dirección `0000H` y termina en la dirección `0FFFH`.

3. En el interior de un bucle recuperar cada byte de la EPROM, comparándolo con los registros donde se tiene el menor y el mayor valor hasta el momento. Si el dato leído es menor que el menor, habrá que actualizar el registro que guarda el menor valor encontrado. Lo mismo es aplicable para el mayor.

4. Al finalizar el bucle habrá que escribir el contenido de los registros que contienen el mayor y menor valor en las direcciones de memoria indicadas.

Cuestiones

1. ¿Cuáles son los valores obtenidos tras la ejecución del programa?

2. ¿Cómo se ha calculado la longitud de la memoria EPROM en el uP-2000?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI D, 0000H     ; Inicio de la ROM
     LDAX D           ; Tomar el primer byte

     MOV H, A         ; Y establecer como mayor
     MOV L, A         ; y menor hasta el momento

     LXI B, 1000H     ; Longitud de la ROM

BUCLE:
     LDAX D           ; Se lee un byte

     CMP L            ; ¿Menor que L?
     JNC NOME         ; si no es así saltar

     MOV L, A         ; Dato menor

NOME:
     CMP H ;          ¿Mayor que H?
     JC NOMA          ; si no es así saltar

     MOV H, A         ; Dato mayor

NOMA:
     INX D            ; Siguiente posición

     DCR C            ; Continuar hasta el final
     JNZ BUCLE        ; de la EPROM

     DCR B
     JNZ BUCLE

     SHLD 1100H       ; Guardar máximo y mínimo

     RST 1
END
```

Ejercicio 8.5.3.2

De la ejecución del programa del ejercicio anterior se deduce que en la EPROM es posible encontrar un valor que sea menor o igual a cualquier otro dado, así como un dato mayor o igual a cualquier otro. Escriba un programa que tome el dato existente en la posición de memoria **1100H** y recorra la EPROM hasta encontrar uno mayor, devolviendo la dirección donde se encuentra en las posiciones **1101H-1102H** y el dato encontrado en la posición **1103H**.

Indicaciones

Partiendo del ejercicio 8.5.3.1, la solución a éste se basa en el mismo programa pero introduciendo los cambios siguientes:

1. El dato a usar como referencia se encuentra en la posición `1100H`, por lo que no hay que inicializar registros con el primer byte leído de la EPROM. Además se trata de un único dato, no dos.

2. En el bucle interesa encontrar un dato que sea mayor, no menor o igual, al dado como referencia. En el momento en que se encuentre ese valor superior al entregado en la posición `1100H` el programa llegará a su fin, no es necesario seguir hasta el final de la EPROM.

3. Además del dato en sí, será necesario guardar también la dirección de la EPROM en la que se ha localizado.

Cuestiones

1. ¿Qué ocurre si introduce en la posición `1100H` el valor `00H` y ejecuta el programa? ¿Y si escribe el valor `FFH`?

2. ¿Cuál será el contenido de la pareja de registros `DE` en caso de que se finalice el bucle y el programa termine pasando por la segunda instrucción `RST 1`? ¿En qué casos terminará el programa por ese camino?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Recuperar el dato de referencia
     MOV L, A

     LXI D, 0000H     ; Inicio de la ROM
     LDAX D           ; Tomar el primer byte

     LXI B, 1000H     ; Longitud de la ROM

BUCLE:
     LDAX D     ; Se lee un byte

     CMP L      ; Comparar con dato referencia
     JC NOES    ; Si L > A el dato no interesa
     JZ NOES    ; Si L = A el dato no interesa

     XCHG
     SHLD 1101H ; Guardar dirección

     MOV A, M   ; Recuperar dato y guardarlo
     STA 1103H

     RST 1

NOES:
     INX D      ; Siguiente posición

     DCR C      ; Continuar hasta el final
     JNZ BUCLE  ; de la EPROM

     DCR B
     JNZ BUCLE

     RST 1
END
```

8.5.4 Búsqueda y conteo

Ejercicio 8.5.4.1

Escribir un programa que recorra toda la EPROM del sistema y cuente el número de apariciones del dato de 8 bits indicado en la posición **1100H**. Devolver el contador como un valor de 8 bits alojado en la posición **1101H**.

Indicaciones

Al igual que los ejercicios del apartado previo, éste tiene que recorrer la memoria EPROM del uP-2000 e ir efectuando comparaciones, pero en este caso no interesa encontrar datos mayores o inferiores a uno dado, sino contabilizar las coincidencias con un cierto valor. El esquema general del programa sería el siguiente:

1. Inicialización del puntero al inicio de la EPROM y de dos contadores: el de coincidencias, por un lado, y el de bytes restantes por procesador. El primero será de 8 bits y el segundo de 16.

2. A cada ciclo del bucle se recuperará un byte de la EPROM y se comparará con el dato buscado, de forma que si son iguales se incremente el contador de coincidencias.

3. Al finalizar la búsqueda el contador de coincidencias debe ser almacenado en la posición `1101H`.

Cuestiones

1. Las últimas instrucciones de la solución dada extraen un dato de la pila pero no depositan ningún resultado en la posición de memoria `1100H`. Explicar cómo llega el contador de coincidencias a dicha dirección y qué papel juegan la pila y la pareja de registros `HL`.

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI SP, 2000H

     LXI D, 0000H     ; Inicio de la ROM
     LXI B, 1000H     ; Longitud de la ROM

     LXI H, 1101H     ; Contador en 1101H
     MVI M, 00H       ; Inicializado a 0
     PUSH H           ; Puntero en la pila
     LXI H, 1100H     ; Dato de referencia en 1100H


BUCLE:
     LDAX D           ; Se lee un byte

     CMP M            ; ¿Es el byte buscado?
     JNZ NOES         ; si no es así saltar

     ; Se ha encontrado una coincidencia

     XTHL        ; HL apuntando al contador
     INR M       ; Para incrementarlo
     XTHL  ; HL de nuevo apuntando al dato

NOES:
     INX D       ; Siguiente posición

     DCR C       ; Continuar hasta el final
     JNZ BUCLE   ; de la EPROM

     DCR B
     JNZ BUCLE

     POP H       ; Sacar dato de la pila

     RST 1
END
```

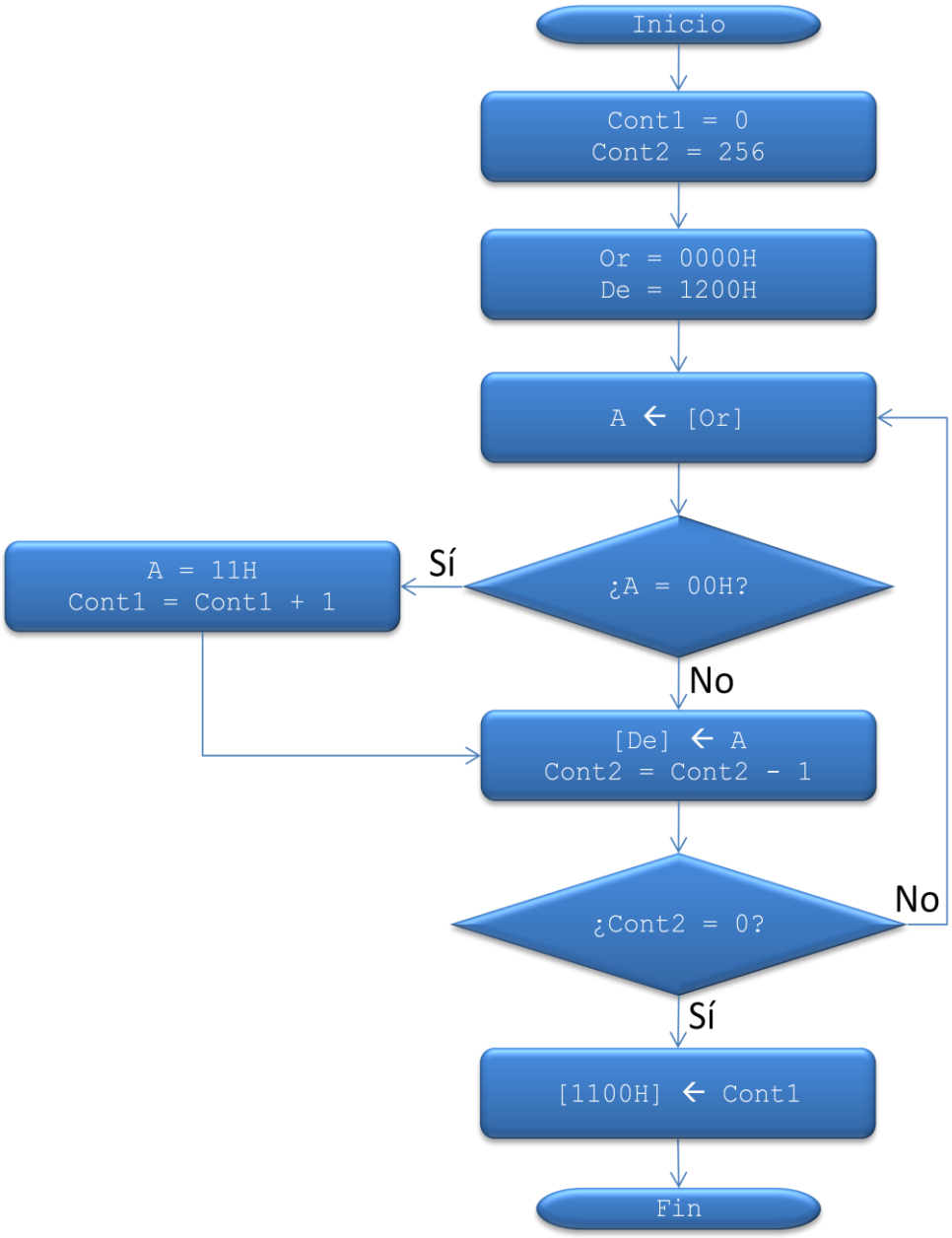
8.5.5 Búsqueda y sustitución

Ejercicio 8.5.5.1

Se quieren copiar los primeros 256 bytes de memoria EPROM del sistema a partir de la posición **1200H**, sustituyendo los bytes **00H** por **11H** durante el proceso de copia. Indicar en la posición **1100H** cuántas sustituciones se han llevado a cabo.

Indicaciones

Para completar este ejercicio utilizar como guía el siguiente diagrama de flujo:



Ordinograma del ejercicio 8.5.5.1.

Cuestiones

1. Tras ejecutar el programa, ¿cuántas sustituciones indica que se han efectuado?

2. Si se copiase toda la EPROM a memoria RAM, con un proceso similar al de este programa, ¿sería posible ejecutar el programa monitor del uP-2000 desde RAM? ¿Por qué?

3. ¿Qué cambios habría que realizar durante la copia de la EPROM para que fuese posible ejecutarla desde RAM?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI B, 0000H     ; Inicio EPROM
     LXI D, 1200H     ; Destino de copia
     LXI H, 0000H     ; Contadores

BUCLE:
     LDAX B           ; Leer un byte
     ORA A            ; y comprobar si es 0
     JNZ NOSUST       ; si no es así, no sustituir

     MVI A, 11H       ; Sustituir
     INR H            ; y contabilizar sustitución

NOSUST:

     STAX D           ; Guardar en destino

     INX B            ; Actualizar punteros
     INX D
     DCR L            ; y contador del bucle

     JNZ BUCLE

     MOV A, H         ; Guardar el contador de
     STA 1100H        ; sustituciones

     RST 1
END
```

Ejercicio 8.5.5.2

Examinar el código del siguiente programa y responder a las cuestiones planteadas.

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI SP, 2000H

     LHLD 1100H
     XCHG

     LHLD 1105H
     MOV B, H
     MOV C, L

BUC1:
     PUSH B

     LHLD 1102H

     LDA 1104H
     MOV C, A

     LDAX D

BUC2:
     CMP M
     JNZ NOCOI

     INX H
     MOV A, M
     STAX D

     JMP FIN

NOCOI:
     INX H
     INX H

     DCR C
     JNZ BUC2

FIN:
     POP B

     INX D

     DCR C
     JNZ BUC1
```

```
      DCR B
      JNZ BUC1

      RST 1
END
```

Cuestiones

1. ¿Cuántos bucles existen en el programa? ¿Están anidados?

2. ¿Qué registros actúan como controladores de esos bucles?

3. ¿Cuántos ciclos ejecuta cada bucle?

4. ¿Cuál es la función que tienen las parejas de registros DE y HL en el programa?

5. ¿Qué datos se sustituyen, los apuntados por DE o los apuntados por HL? ¿Por qué otros valores son sustituidos?

6. Introduzca a partir de la dirección 1100H la secuencia de bytes 00H 12H 00H 13H 04H FFH y a partir de la dirección 1300H la secuencia 00H AAH 1FH BDH 20H 15H 11H EDH. A continuación examine el contenido de algunos bytes a partir de la posición 1200H antes y después de ejecutar el programa. Escriba a partir de la dirección 1200H los bytes 11H 20H 00H 1FH, ejecute el programa y compruebe qué ha ocurrido con esos valores.

7. Agregue comentarios aclaratorios al código del programa y describa globalmente cuál es su finalidad.

8.5.6 Ejercicios propuestos

Ejercicio 8.5.6.1

Escribir un programa análogo al del ejercicio 8.5.1.1, de forma que calcule la posición de un elemento en una tabla y devuelva su dirección y contenido, pero teniendo en cuenta las siguientes premisas:

- La dirección de inicio de la tabla está en las posiciones **1100H-1101H**.

- El número de filas de la tabla se indica en la posición **1102H**.

- El número de columnas de la tabla se indica en **1103H**.

- La fila a buscar es la indicada en la posición **1104H**.

- La columna a buscar está indicada en la posición **1105H**.

- La dirección resultante debe quedar en **1106H-1107H** y el dato contenido en el elemento en la posición **1108H**.

Indicaciones

Conociendo la solución dada a los ejercicios 8.5.1.1 y 8.5.1.2, para obtener el programa que propone este ejercicio habría que introducir los cambios siguientes:

1. La estructura de la tabla: número de filas y columnas, ya no son datos inmediatos que puedan introducirse en el programa, sino que será necesario leerlos de las posiciones indicadas y guardarlos en registros para realizar el resto de operaciones.

2. Habrá que ajustar las direcciones de memoria de las que se leían el número de fila y número de columna, así como las posiciones donde se depositaban los resultados.

Ejercicio 8.5.6.2

Modificar el programa dado como solución al ejercicio 8.5.3.1 para que, además del mínimo y máximo valor de la EPROM, escriba en las posiciones **1102H-1103H** y **1104H-1105H** las direcciones de memoria donde han sido encontrados.

Indicaciones

Tomando como base el programa dado como solución al ejercicio 8.5.3.1, habría que agregar las instrucciones apropiadas para que cada vez que se encuentre un valor mayor o menor que los que se tienen hasta el momento se conserve no únicamente el nuevo dato, sino también la dirección en que ha sido localizado. El objetivo puede alcanzarse agregando, en los puntos adecuados, cuatro instrucciones `XCHG` y dos instrucciones `SHLD`.

Ejercicio 8.5.6.3

Modificar el programa del ejercicio 8.5.4.1 de forma que no se utilice la pila y no se efectúen accesos continuamente a memoria, a través del registro **M**.

Indicaciones

A pesar de que el objetivo del programa que solicita el ejercicio es exactamente el mismo que en el ejercicio 8.5.4.1, la forma de implementarlo habrá de ser necesariamente distinta para evitar el uso de la pila y de la pareja de registros `HL` como puntero para leer el dato de referencia o actualizar el contador.

El aspecto fundamental es cómo distribuir los registros con que cuenta el 8085 de forma que puedan realizarse todas las operaciones internamente, sin necesidad de acceder a la memoria nada más que para ir leyendo bytes de la EPROM. Una indicación útil: para buscar todas las coincidencias es preciso recorrer toda la EPROM, pero no necesariamente desde el principio al final.

Cuestiones

1. Describa las ventajas que tiene la solución dada a este ejercicio respecto al programa original del ejercicio 8.5.4.1.

8.6 Estructuración del código

A medida que se van abordando problemas más difíciles, el nivel de complejidad de los programas escritos en ensamblador también se incrementa de manera considerable. Es lógico puesto que se trata de un lenguaje de bajo nivel, de forma que estructuras habituales como los condicionales o los bucles precisan de múltiples instrucciones para su implementación. La mayoría de los programas pueden dividir su código en varias tareas más o menos simples, cada una de las cuales podría ser implementada como una subrutina independiente. De esta forma es más fácil estructurar el código de manera que resulte más fácil de escribir y mantener con posterioridad.

La finalidad de los ejercicios de esta sección es la siguiente:

- Aprender a escribir subrutinas y a invocarlas desde el cuerpo principal de un programa.
- Hacer que las subrutinas devuelvan información básica sobre el resultado utilizando para ello el registro de estado 8085.
- Aprender a facilitar parámetros de entrada a una subrutina y recoger resultados generados por ésta.
- Poner en práctica la estructuración del código de un programa desarrollando diversos ejercicios.

8.6.1 Escritura de subrutinas

Ejercicio 8.6.1.1

Modificar el programa del ejercicio 8.5.3.1 de forma que quede estructurado en tres grandes bloques: un núcleo del programa, una subrutina que se encargue del proceso de inicialización y otra que realice la búsqueda del dato superior al entregado como referencia.

Indicaciones

Tomar el código de un programa ya existente y dividirlo en varias partes es una tarea relativamente sencilla, siempre que se tengan en cuenta los siguientes aspectos:

1. Hay que comenzar delimitando claramente cuál es el inicio (primera instrucción) y final (última instrucción) de cada una de las partes en que se dividirá el código original.

2. Disponer delante de la primera instrucción una etiqueta que identifique la finalidad de esa parte del programa. Esta etiqueta será la se utilice como operando de la instrucción `CALL` y permitirá invocar a la subrutina.

3. Agregar tras la última instrucción lógica (aquella que marca el fin de ejecución, puede estar físicamente al final o no) de cada bloque una instrucción `RET`. Ésta se encargará de salir de la subrutina y devolver el control al punto desde el que se efectuó la llamada, concretamente a la instrucción siguiente a `CALL`.

4. Rescribir el inicio del programa para que actúe como nudo central, invocando a las subrutinas definidas en los pasos previos.

Cuestiones

1. Indique un paso que resulta imprescindible a la hora de escribir programas en los que se usan subrutinas.

2. ¿Puede llamar una subrutina a otra subrutina?

3. ¿Se puede llamar una subrutina a sí misma? ¿Qué aplicación tendría dicho comportamiento?

4. ¿Existe límite en el número de llamadas a subrutinas en un programa?

5. Represente esquemáticamente los pasos en que se divide la llamada a la primera subrutina de programa, su ejecución y retorno.

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     CALL INIC
     CALL BUSCA

     RST 1

; ----- Subrutina de inicialización
INIC:
     LDA 1100H  ; Recuperar el dato de referencia
     MOV L, A

     LXI D, 0000H     ; Inicio de la ROM
     LDAX D           ; Tomar el primer byte

     LXI B, 1000H     ; Longitud de la ROM

     RET
; ----- Fin de la inicialización

; ----- Subrutina de búsqueda
BUSCA:
     LDAX D           ; Se lee un byte

     CMP L       ; Comparar con dato referencia
     JC NOES     ; Si L > A el dato no interesa
     JZ NOES     ; Si L = A el dato no interesa

     XCHG
     SHLD 1101H       ; Guardar dirección
     MOV A, M         ; Recuperar dato y guardarlo
     STA 1103H

     RET

NOES:
     INX D            ; Siguiente posición

     DCR C            ; Continuar hasta el final
     JNZ BUSCA        ; de la EPROM
     DCR B
     JNZ BUSCA

     RET
; ----- Fin de la búsqueda
END
```

8.6.2 Uso de indicadores del registro de estado para comunicar resultados

Ejercicio 8.6.2.1

Escribir una subrutina que compruebe si el contenido de la pareja de registros **BC** es **0000H** y, en caso afirmativo, active el bit **Z** del registro de estado. Usar esta subrutina para controlar el bucle de la subrutina **BUSCA** del programa del ejercicio previo.

Indicaciones

Las subrutinas no siempre se limitan a hacer una tarea invariable asociada a un programa concreto, como es el caso de `INIC` y `BUSCA` en el ejercicio anterior, sino que pueden tener una aplicación más genérica que las haga útiles en otros programas. Es el caso de la subrutina que se propone en este ejercicio, cuya finalidad será comprobar si la pareja de registros `BC`, usada habitualmente como contador de bucles, contiene el valor `0000H`. Es una subrutina que podría ser utilizada en cualquier programa que requiera un bucle controlador por esa pareja de registros.

Al escribir una subrutina más o menos genérica es recomendable que:

1. La etiqueta de entrada, que será la que se emplee para llamar a la subrutina, sea lo más significativa posible dentro de las limitaciones que impone la longitud máxima de estas etiquetas.

2. Indicar claramente delante del código de la subrutina, a modo de cabecera, los registros que se verán afectados al invocarla. De esta forma quien vaya a utilizarla sabrá qué datos debe preservar si los tiene almacenados en dichos registros.

3. Si la subrutina afecta al registro de indicadores, describir qué bits se verán modificados y de qué forma.

Una vez se tenga preparada la subrutina, deberán introducirse en el programa los cambios necesarios para utilizarla.

Cuestiones

1. ¿Cuál es la ventaja fundamental de utilizar el registro de estado del 8085 para devolver un resultado, en este caso la confirmación de que `BC=0` o no, en lugar de usar para ello un registro cualquiera?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H  ; Pila al final de la RAM

     CALL INIC
     CALL BUSCA

     RST 1

; ----- Subrutina de inicialización
INIC:
     LDA 1100H  ; Recuperar el dato de referencia
     MOV L, A

     LXI D, 0000H     ; Inicio de la ROM
     LDAX D           ; Tomar el primer byte

     LXI B, 1000H     ; Longitud de la ROM

     RET
; ----- Fin de la inicialización

; ----- Subrutina de búsqueda
BUSCA:
     LDAX D           ; Se lee un byte

     CMP L       ; Comparar con dato referencia
     JC NOES     ; Si L > A el dato no interesa
     JZ NOES     ; Si L = A el dato no interesa

     XCHG
     SHLD 1101H ; Guardar dirección

     MOV A, M   ; Recuperar dato y guardarlo
     STA 1103H

     RET

NOES:
     INX D ; Siguiente posición

     CALL LIM_BC; Continuar hasta el final
     JNZ BUSCA  ; de la EPROM

     RET
; ----- Fin de la búsqueda

; ----- Subrutina de comprobación de contador
;
; Modifica los registros: A, Estado
```

```
; Devuelve: Z=1 si BC = 0, Z=0 caso contrario
;
LIM_BC:
     MOV A, B
     ORA C

     RET
; ----- Fin de la comprobación de contador
END
```

8.6.3 Recepción de argumentos y devolución de resultados

Ejercicio 8.6.3.1

Se quiere comprobar si el dato almacenado en la posición de memoria **1100H** se encuentra en el intervalo marcado por los valores alojados en las posiciones **1101H** (menor) y **1102H** (mayor). Si se considera válido escribir el dato **01H** en la posición **1103H**, en caso contrario el dato a escribir será **00H**. Emplear una subrutina genérica para validar el dato.

Indicaciones

Al escribir una subrutina hay que tratar de que sea lo más genérica posible, de esa forma resultará de utilidad en más programas. Una subrutina que verifique si un dato alojado en la posición de memoria `1100H` está en el rango indicado por las posiciones `1101H` y `1102H`, por ejemplo, servirá únicamente para programas que almacenen los datos en dichas direcciones de memoria. Si la misma subrutina se escribe de manera que reciba la información en registros concretos, existen más posibilidades de que pueda ser reutilizada en otros programas.

El número de parámetros que una subrutina puede recibir a través de registros está limitado, lógicamente, por los registros disponibles (aquellos que no están ya en uso) en el momento de efectuar la llamada. Siempre se tiene la alternativa, no obstante, de utilizar la pila como almacenamiento temporal, guardando en ella los datos actuales de los registros para restaurarlos cuando la subrutina devuelva el control.

La resolución de este ejercicio puede iniciarse por la escritura de la subrutina, decidiendo qué registros contendrán los datos necesarios y cómo se devolverá el resultado. Dado que tiene que evaluar si un cierto dato está o no en un rango de valores, parece lógico solicitar que el dato se encuentre en el acumulador y que los límites, inferior y superior, se entreguen en los registros `L` (*Low*) y `H` (*High*), respectivamente. El resultado de esta subrutina es simplemente una confirmación, una indicación de si el valor está o no el rango, por lo que puede utilizarse un bit del registro de estado para devolverlo. El bit de acarreo resulta adecuado.

Una vez escrita la subrutina la atención se centrará en codificar el programa principal, cuya tarea será recuperar los datos, invocar con ellos a la subrutina y escribir el resultado, tal y como se indica en el enunciado.

Cuestiones

1. Si no hubiese registros suficientes para facilitar todos los parámetros necesarios a una subrutina ¿qué solución podría aplicarse?

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI SP, 2000H

     LDA 1100H        ; Leer el dato
     LHLD 1101H       ; y los límites

     CALL VRF_RNG     ; Verificar
     MVI A, 01H       ; Asumir que será válido
     JNC VALIDO       ; Si lo es, saltar

     MVI A, 00H       ; En caso contrario A=00H

VALIDO:
     STA 1103H

     RST 1

; ----- Subrutina que comprueba si un cierto dato
;       está comprendido en un rango de valores
;
; Modifica los registros: Ninguno salvo el de estado
; Devuelve: CY=1 si el dato no está en el rango
;           CY=0 el dato está en el rango
;
VRF_RNG:
     CMP L       ; ¿ A < L ?
     JC NO_RNG   ; Si es así, está fuera del rango

     CMP H       ; Comprobar límite superior
     JZ EN_RNG   ; Si A = H es válido
     JNC NO_RNG  ; Si A > H, está fuera del rango

EN_RNG: ; Dato está en el rango
     ORA A       ; Borrar CY
     RET

NO_RNG: ; Dato no está en el rango
     STC         ; CY = 1
     RET

; ----- Fin de la subrutina que comprueba el rango
END
```

Ejercicio 8.6.3.2

Escribir una rutina genérica para obtener el producto de dos datos de 8 bits, devolviendo el resultado como un valor de 16 bits. Toda la transferencia de datos, tanto parámetros de entrada como resultados de salida, ha de efectuarse a través de registros. La subrutina no deberá modificar ningún registro, ni siquiera el de estado, salvo la pareja en la que vaya a devolverse el resultado. Comprobar el funcionamiento de la subrutina desde un programa simple.

Indicaciones

Una subrutina como la que solicita este ejercicio no puede, obviamente, devolver el resultado mediante el registro de estado, sino que tiene que recurrir a almacenarlo en uno o más registros de uso general. En este caso, puesto que el resultado será de 16 bits, habrá que recurrir a una pareja de registros.

Para completar el ejercicio habrá que seguir los pasos indicados a continuación:

1. Establecer cuáles serán los registros que se emplearán para facilitar a la subrutina el multiplicando y el multiplicador, así como la pareja de registros en la que se devolverá el resultado. Esta información debe estar documentada al inicio de la subrutina.

2. Escribir la subrutina de multiplicación usando los registros que se necesiten. Una vez que se tenga el código, comprobar qué registros se han modificado para efectuar la multiplicación y agregar las sentencias `PUSH` necesarias para guardarlos al inicio y restaurarlos con `POP` al final, de esta forma sus contenidos quedarán preservados.

3. Teniendo terminada la subrutina, preparar un programa que efectúe una multiplicación y debe el resultado alojado en la memoria. También deberá guardar datos en algunos registros, antes de llamar a la subrutina, a fin de verificar que, en efecto, ésta no los modifica.

Cuestiones

1. ¿Qué ocurriría si en la subrutina se olvidase extraer algún dato, previamente introducido en la pila, y se ejecutase la instrucción `RET`?

2. ¿Cuál es la finalidad de las instrucciones `PUSH PSW/POP PSW`?

3. ¿Podría utilizarse la pareja `PUSH PSW/POP PSW` en una subrutina que devolviese un resultado a través del registro de estado? ¿De qué manera?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H

     MVI B, 20H       ; Datos que quieren multiplicarse
     MVI C, 32H
     MVI A, 0AAH      ; Datos para comprobar que los
     LXI D, 7BCDH     ; demás registros no se ven afectados

     CALL MULT8X8

     SHLD 1100H       ; Escribir el resultado del producto
     XCHG
     SHLD 1102H       ; y los valores de DE y A
     STA 1104H        ; que siempre serán 7BCDH y AAH

     RST 1

; ----- Subrutina para obtener el producto de dos datos
;       de 8 bits como un resultado de 16 bits
;
; Parámetros de entrada:
;     B = Multiplicando, C = Multiplicador
; Parámetros de salida: HL = Producto de BxC
; Modifica los registros: Únicamente HL
;
MULT8X8:
     PUSH PSW    ; Guardar los registros que van a
     PUSH D      ; a ser usados en la subrutina

     MVI D, 00H ; Multiplicando
     MOV E, B    ; a DE

     LXI H, 00H ; Producto en HL
     MOV A, C    ; Multiplicador en A

REP_8X8:
     DAD D       ; HL = HL + DE
     DCR A
     JNZ REP_8X8; A > 0, seguir

     POP D
     POP PSW
     RET
; ----- Fin de la subrutina de multiplicación 8X8

END
```

Ejercicio 8.6.3.3

Examine el código del siguiente programa y responda a las cuestiones planteadas a continuación.

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H

     LDA 1100H
     CALL F1
     SHLD 1101H

     RST 1

F1:
    LXI H, 0001H

F2:
     CPI 02H
     JC FF
     PUSH PSW
     DCR A
     CALL F2
     POP PSW
     CALL MA

FF:
     RET

MA:
     PUSH PSW
     PUSH D

     MOV D, H
     MOV E, L
     LXI H, 0000H
     DAD D
     DCR A
     JNZ RH

     POP D
     POP PSW
     RET
END
```

Cuestiones

1. ¿Cuántas instrucciones `CALL` aparecen en el programa? En consecuencia, ¿cuántas subrutinas existen?

2. Identifique el inicio y fin de cada una de las subrutinas. Cambie las etiquetas de entrada por otras más descriptivas.

3. ¿Cuál es la finalidad de la subrutina marcada con la etiqueta `MA`? Identifique los parámetros de entrada y de salida que le corresponden. ¿Qué registros modifica?

4. Para que la última subrutina pueda realizar su trabajo falta una etiqueta, a la que hace referencia la instrucción `JNZ RH`. Sitúela apropiadamente.

5. ¿Cuál es la finalidad de la subrutina a la que se invoca desde la parte inicial del programa? Identifique los parámetros de entrada y de salida correspondientes. ¿Qué registros modifica?

6. Describa de forma general qué es lo que hace el programa. Agregue los comentarios adecuados al código para facilitar su comprensión.

7. ¿Habría alguna forma de conseguir el mismo resultado sin necesidad de recurrir a la técnica que emplea la primera subrutina?

8.6.4 Ejercicios propuestos

Ejercicio 8.6.4.1

Tomar el código del ejercicio 8.5.1.2 e implementar como subrutina el trabajo de comprobación que se lleva al inicio del programa, de forma que el código del programa pueda sencillamente invocar a esa subrutina y decidir si debe terminar en función del resultado que devuelva.

Indicaciones

El objetivo es hacer más simple el código de este ejercicio, de forma que la verificación de los límites aplicables a los parámetros aparezca como una parte independiente, una subrutina a la que se invoca desde el bloque principal sin tener necesariamente que saber cómo funciona. Bastará con atender al indicador que emplee para devolver el resultado.

Una manera habitual de que una rutina indique si ha finalizado correctamente, y el programa puede continuar, o por el contrario se ha producido un fallo, consiste en activar o desactivar el bit de acarreo del registro de estado. Puede utilizar esa técnica en este ejercicio, de forma que la subrutina de comprobación de límites active dicho bit se hay algún parámetro incorrecto, desactivándolo en caso contrario.

Ejercicio 8.6.4.2

Tomar el código del ejercicio 8.5.5.2, en el que se efectuaba la sustitución de una serie de valores en un bloque de memoria mediante una tabla XLAT, y estructúrelo de forma que el bloque principal del programa se limite a recuperar los datos de las posiciones de memoria adecuadas y entregarlos como parámetros a una subrutina. Ésta se puede, en caso de ser necesario, dividir en varias subrutinas. En cualquier caso ningún registro debe quedar afectado por la operación.

Indicaciones

Para resolver este ejercicio comience por aislar lo que será el programa principal, formado por el primer bloque de instrucciones y todas aquellas que recuperen datos de posiciones específicas de memoria. Esos datos habrán de entregarse ahora en registros a la subrutina que se encargará de procesar el bloque de bytes y llevar a cabo las sustituciones.

A fin de facilitar la implementación, la subrutina principal puede encargarse únicamente del bucle externo, el que recorre el bloque de memoria indicado, dejando en manos de otra subrutina el tratamiento de cada byte de ese bloque, recorriendo la tabla XLAT.

8.7 Ordenar datos

Ciertas tareas pueden parecer a primera vista demasiado complejas como para poder ser implementadas en un lenguaje de bajo nivel, como es el ensamblador, pero en realidad no hay ningún trabajo que no pueda realizar un programa escrito en dicho lenguaje, siempre que se estructure adecuadamente.

El único ejercicio de esta sección tiene el objetivo de mostrar cómo, a través de la división de un problema en partes (subrutinas), es posible implementar una funcionalidad aparentemente compleja: ordenar una lista de números. El desarrollo del ejercicio servirá, además, para poner en práctica gran parte de lo aprendido en secciones previas: comparaciones y saltos condicionales, bucles (en este caso anidados), intercambio de datos en memoria, subrutinas, etc.

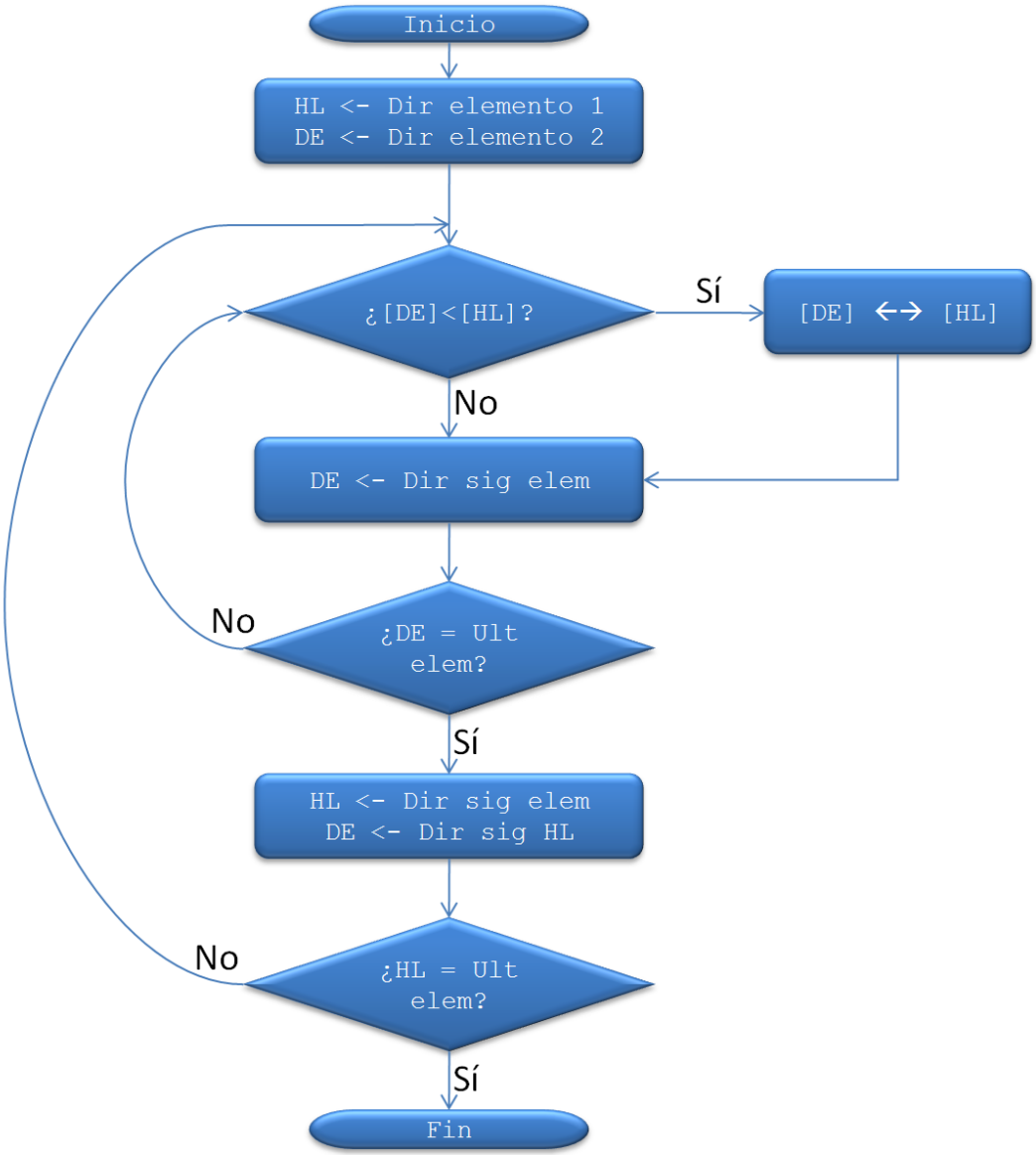
8.7.1 Implementación del algoritmo de la burbuja

Ejercicio 8.7.1.1

Se tiene en memoria una lista de datos de 8 bits, a partir de la dirección indicada en las posiciones **1100H-1101H**, con tantos elementos como se indica en las posiciones **1102-1103H**. Escribir un programa que ordene esos datos de menor a mayor manteniendo la lista en el mismo bloque de memoria. Utilizar el algoritmo de la burbuja.

Indicaciones

El siguiente diagrama de flujo representa esquemáticamente el algoritmo de la burbuja para ordenar una lista de elementos.



Ordinograma correspondiente al algoritmo de la burbuja.

Como puede apreciarse en el ordinograma, el algoritmo de la burbuja se basa en el uso de dos bucles anidados y dos punteros a elementos. La técnica consiste, básicamente, en comparar el primer elemento con todos los demás de la lista, de forma que cada vez que se encuentre uno menor que el primero se produce un intercambio. De esta forma, al terminar el primer ciclo es seguro que el primer elemento es el menor. En ese momento el puntero del bucle exterior avanza para apuntar al segundo elemento de la lista, mientras que el del bucle interior siempre tendrá la dirección del siguiente. De esta forma se compara el segundo elemento con el tercero y siguientes, hasta el final de la lista, colocando el menor al inicio de la sublista. Repitiendo el proceso hasta el final se conseguirá tener la lista ordenada. No es precisamente el algoritmo más eficiente pero sí uno de los más simples de implementar.

A la hora de abordar la resolución de este ejercicio puede dividirse la tarea en los pasos siguientes:

1. En todo momento se necesita tener un puntero al dato que está comparándose con el resto de la lista, otro puntero que facilite el recorrido de ese resto de la lista y un contador de 16 bits. Serán precisas, por tanto, tres parejas de registros. Comenzar decidiendo qué pareja asumirá cada una de las funciones.

2. Escribir el bloque correspondiente al programa principal, que se limitará a recuperar los datos de las posiciones de memoria indicadas, dejándolos en las parejas de registros elegidas previamente, y a invocar a una subrutina que llevará a cabo la ordenación de la lista de datos.

3. A continuación escribir como subrutina de ordenación el bucle exterior (observar el diagrama de flujo anterior), encargado de ir actualizando los punteros adecuadamente e invocar a una segunda subrutina que corresponderá al bucle interior.

4. La segunda subrutina (bucle interior o anidado) se encargará de efectuar la comparación de los datos y, si fuese necesario, su intercambio. Para ello delegará estas acciones en dos subrutinas, de forma que su tarea principal será controlar el bucle interior e ir actualizando el puntero que recorre la lista.

5. Finalmente implementar las subrutinas encargadas de comparar los datos y de realizar el intercambio, invocadas desde la subrutina anterior.

Esta forma de tratar el problema usa la técnica conocida como *divide y vencerás*, consistentes en ir dividiéndolo en partes cada vez más pequeñas y fáciles de implementar.

Una vez terminado el programa introduzca una lista de valores desordenados en la memoria, indique su posición y número de valores en las posiciones `1100H-1101H` y `1102-1103H`, ejecute el programa y, finalmente, verifique que, en efecto, están ordenados.

Cuestiones

1. ¿Cuántos elementos podría ordenar como máximo, teóricamente, este programa? Y en la práctica, en el sistema uP-2000, ¿cuántos puede ordenar?

2. Asumiendo que la lista de datos a ordenar contiene 100 elementos, ¿cuántos ciclos ejecuta cada uno de los bucles?, ¿cuántas comparaciones se llevan a cabo?

3. ¿Cuál es la finalidad de la instrucción `CC` que aparece tras la etiqueta `B_BUR`, en la subrutina auxiliar de ordenación? Si no existiese, ¿cómo habría que reescribir esa parte del código?

4. ¿Cómo quedan los registros del microprocesador una vez que se ha finalizado la ordenación de la lista de datos?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H

     LHLD 1102H  ; Número de datos a ordenar
     MOV C, L    ; en BC
     MOV B, H

     LHLD 1100H  ; Dir del bloque de datos

     CALL ORDENA ; Ordenar los datos

     RST 1 ; y terminar

; ----- Rutina de ordenación de datos 8 bits
;
; Entradas:
;    HL = Dirección de la lista de datos a ordenar
;    BC = Número de elementos que contiene la lista
;
; Salidas:
;    La lista de datos ordenada en la misma posición
;    de memoria donde se encontraba originalmente
;
; Modifica:
;    Nada
;
ORDENA:
     PUSH PSW    ; Guardar todos los registros en pila
     PUSH B
     PUSH H
     PUSH D

     PUSH H      ; Copiar HL en DE
     POP D

     ; Se necesitará un ciclo menos que
     ; elementos existan en la lista
     DCX B

B_ORD: ; Bucle principal del algoritmo

     INX D       ; DE siempre = HL + 1

     CALL BURB   ; Invocar al bucle interior

     INX H       ; Avanzar al dato siguiente
     DCX B       ; Queda un dato menos

     MOV A, C    ; Hasta que BC = 0
     ORA B
```

```
     JNZ B_ORD

     POP D
     POP H       ; Restaurar registros
     POP B
     POP PSW

     RET         ; y terminar
; ----- Fin del cuerpo principal de la rutina

; ----- Subrutina auxiliar de ordenación
;
; Entradas:
;    HL = Dato que va a compararse con el resto
;    DE = Dato siguiente a HL
;    BC = Número de elementos que quedan por tratar
;
; Salidas:
;    El dato apuntado por HL será el menor de todos
;    los que quedan en la lista
;
; Modifica:
;    Nada
;
BURB:
     PUSH B      ; Guardar registros que se
     PUSH D      ; modificarán

B_BUR:
     CALL COMPA  ; Comparar [HL] con [DE]
     CC INTERC   ; Si es necesario, intercambiarlos

     INX D       ; Siguiente dato de la lista

     DCX B       ; Queda un dato menos

     MOV A, C    ; Repetir hasta comparar el
     ORA B       ; apuntado por HL con el resto
     JNZ B_BUR


     POP D       ; Restaurar registros
     POP B

     RET         ; y terminar
; ----- Fin de la subrutina auxiliar de ordenación

; ----- Subrutina que se encarga de comparar dos datos
;
; Entradas:
;    DE = Apunta a un elemento de la lista
;    HL = Apunta al elemento donde quedará el menor
;
; Salidas:
```

```
;    CY = 1 si hay que intercambiarlos
;
; Modifica:
;     A = Queda con el valor apuntado por DE
;     Registro de estado
;
COMPA:
     LDAX D      ; Recuperar el dato apuntado por DE
     CMP M       ; y compararlo con el apuntado por HL

     RET         ; Volver

; ----- Subrutina encargada de intercambiar dos elementos
;
; Entradas:
;    A = Contiene uno de los elementos
;   HL = Apunta al otro elemento
;
; Salidas:
;    A = Contiene el dato al que apuntaba HL
;   HL = El dato apuntado por HL ahora es el que contenía A
;   DE = El dato contenido en A se almacena en el
;        elemento apuntado por DE
;
INTERC:
     PUSH B      ; Preservar BC

     MOV B, A    ; Porque B se usará para
     MOV A, M    ; realizar el intercambio
     MOV M, B

     STAX D      ; Guardar [DE] <- A

     POP B       ; Restaurar BC

     RET         ; y terminar
END
```

8.8 Lectura del teclado y visualización en el uP-2000

Los ejercicios de las secciones previas necesitan datos de entrada para su funcionamiento y, asimismo, generan resultados. Esa información hasta el momento se está entregando y recogiendo a través de posiciones concretas de la memoria, pero sería mucho más efectivo que los programas solicitasen y mostrasen los datos de manera interactiva. Con este fin se pueden usar distintas subrutinas alojadas en la memoria EPROM del sistema uP-2000.

La finalidad de los ejercicios de esta sección es la siguiente:

- Aprender a mostrar datos de 8 bits en el campo de datos de la pantalla, así como datos de 16 bits en el campo de direcciones.
- Conocer cuáles son los códigos asociados a las teclas del uP-2000 y cómo leer el código de una tecla pulsada.
- Saber cómo puede componerse un dato de 8 bits a partir de dos pulsaciones de tecla sucesivas.
- Utilizar el teclado del uP-2000 para dirigir el comportamiento de un programa de manera interactiva.
- Aprender a solicitar datos de 16 bits desde el teclado mediante las subrutinas del uP-2000.

8.8.1 Visualización en el campo de direcciones

Ejercicio 8.8.1.1

Escribir un programa que tome la dirección almacenada en las posiciones **1100H-1101H** (en formato *little-endian*) y la muestre en el campo de direcciones del *display* del uP-2000.

Indicaciones

El *display* o pantalla del uP-2000 está dividida en dos partes bien diferenciadas: el campo de direcciones, que ocupa 4 dígitos, y el de datos, que ocupa 2. Para mostrar información en esa pantalla hay dos alternativas:

1. Programar directamente el integrado 8279 (descrito en la primera parte), un componente relativamente complejo.

2. Recurrir a las subrutinas que el sistema uP-2000 tiene alojadas en EPROM.

Obviamente la segunda opción es la más sencilla. No hay más que saber en qué dirección se encuentra cada subrutina (véase *Tabla 4. Rutinas de servicio del uP-2000*) y cuáles son sus parámetros de entrada y salida, así como su finalidad. La técnica, por tanto, es idéntica a la que se emplearía para llamar a una subrutina propia.

Para la realización de este ejercicio deben darse los pasos siguientes:

1. Obtener la dirección de la subrutina adecuada para mostrar valores en el campo de direcciones de la pantalla del uP-2000, comprobando cómo hay que entregar los parámetros de entrada.

2. Recuperar de las posiciones `1100H-1101H` el dato a mostrar y colocarlo en los registros adecuados para entregarlo a la subrutina anterior.

3. Invocar a la subrutina y comprobar el resultado en la pantalla del uP-2000.

Cuestiones

1. ¿Puede ver el resultado que genera la ejecución del programa? ¿Por qué?

2. ¿Qué ocurre si al llamar a la subrutina para mostrar la dirección se tienen datos almacenados en otros registros? ¿Cómo podrían conservarse de manera segura?

3. ¿Podría llamarse varias veces seguidas a esta subrutina para mostrar varios datos? ¿Cuál sería el efecto?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H  ; Pila al final de la RAM

     LHLD 1100H ; Recuperar la dirección
     CALL 04C9H ; y mostrarla

     RST 1

END
```

8.8.2 Visualización en el campo de datos

Ejercicio 8.8.2.1

Escribir un programa que tome la dirección almacenada en las posiciones **1100H-1101H**, recupere el dato existente en esa posición de memoria y muestre la dirección en el campo de direcciones y el dato en el campo de datos de la pantalla del uP-2000.

Indicaciones

Tomando como punto de partida el programa del ejercicio previo, la realización de éste solamente supone añadir las siguientes instrucciones:

1. Preservar el contenido de `HL`, la dirección obtenida de `1100H-1101H`, antes de llamar a la subrutina para mostrarla, ya que de lo contrario se perdería.

2. Recuperar el contenido de `HL` y usarlo como puntero para leer en `A` el dato contenido en esa dirección.

3. Llamar a la subrutina encargada de visualizar un dato de 8 bits en el campo de datos de la pantalla del uP-2000.

Cuestiones

1. Almacene en `1100H-1101H` el dato de 16 bits `1000H` y ejecute el programa. ¿Qué obtiene en la pantalla del uP-2000?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H  ; Pila al final de la RAM

     LHLD 1100H ; Recuperar la dirección
     PUSH H     ; guardarla
     CALL 04C9H ; y mostrarla
     POP H

     MOV A, M   ; Recuperar el dato de esa dirección
     CALL 04D5H ; y mostrarlo

     RST 1

END
```

Ejercicio 8.8.2.2

Se tienen en las posiciones **1100H** y **1101H** dos datos de 8 bits. Se quiere obtener la suma de ambos, con el resultado también de 8 bits, mostrando en el campo de direcciones del uP-2000 los dos operandos y en el campo de datos su suma.

Indicaciones

En este ejercicio el contenido de las posiciones 1100H y 1101H no deben interpretarse como una dirección, sino como dos datos independientes de 8 bits. La forma de mostrar ambos operandos en el campo de direcciones del uP-2000, no obstante, será la misma utilizada en los dos ejercicios anteriores.

Efectuada la suma de los dos datos, cuyo resultado quedará en el acumulador, su visualización en el campo de datos se llevará a cabo como en el ejercicio previo, mediante la subrutina situada en la dirección 04D5H.

Cuestiones

1. Si se utilizase la instrucción LHLD para recuperar los dos operandos de la memoria, ¿cómo afectaría al resultado de la operación de suma? ¿Cómo afectaría a la visualización de la información?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H  ; Pila al final de la RAM

     LDA 1100H  ; Primer operando
     MOV H, A   ; en H
     LDA 1101H  ; Segundo operando
     MOV L, A   ; en L

     PUSH H     ; guardar HL
     CALL 04C9H ; y mostrar los operandos
     POP H

     MOV A, L   ; Efectuar la suma
     ADD H
     CALL 04D5H ; y mostrarlo

     RST 1

END
```

8.8.3 Mostrar el código de la tecla pulsada

Ejercicio 8.8.3.1

Escribir un programa que espere la pulsación de una tecla del uP-2000 y a continuación muestre en el campo de datos el código que le corresponda. Esta operación debe repetirla continuamente hasta que el código de la tecla leída sea **0FH**, correspondiente a la tecla `F` del uP-2000.

Indicaciones

Mostrar datos en la pantalla del uP-2000 resulta útil, ya que nos ahorra tener que inspeccionar manualmente el contenido de la memoria para recuperar los resultados que genera un programa. De manera análoga, el uso del teclado puede aportar interactividad a la ejecución de ese programa, de forma que los datos sobre los que opere sean introducidos en el momento en que se necesitan. Saber qué código corresponde a cada tecla permitirá crear programas que respondan adecuadamente y que usen diferentes teclas para distintas funciones.

Lo primero que hay que hacer para completar este ejercicio es recordar lo siguiente:

1. El teclado del uP-2000 es un dispositivo externo al microprocesador, comunicándose con éste a través de interrupciones. La mayor parte de las teclas generan una interrupción `RST 5.5`.

2. Como es explicó en el apartado *1.9 Interrupciones*, el 8085 dispone de un registro de interrupciones que establece cuáles están enmascaradas (deshabilitadas) y cuáles no (habilitadas). Para que el teclado responda es necesario que la `RST 5.5` esté desenmascarada, de lo contrario el 8085 la ignorará.

3. Existen dos instrucciones cuya finalidad es facilitar la lectura y escritura del registro de interrupciones del 8085: `RIM` y `SIM`. Para escribir en el registro es preciso poner a 1 el bit `WR` (véase *Figura 12. Máscara de establecimiento de interrupciones*), indicando con los tres bits de menor peso qué interrupciones estarán enmascaradas/desenmascaradas.

El programa que solicita el ejercicio, por tanto, se dividirá en dos partes bien diferenciadas:

1. Una inicialización que configurará el registro de interrupciones del 8085.

2. Un bucle que esperará pulsaciones de tecla y mostrará sus códigos en el campo de datos de la pantalla del uP-2000, hasta que el código leído sea `0FH`.

Cuestiones

1. ¿Se podría efectuar el proceso de inicialización de otra forma?

2. ¿Qué códigos corresponden a las teclas que representan códigos hexadecimales?

3. ¿Qué códigos tienen las teclas de función del teclado (color azul)?

4. ¿Generan códigos las teclas **`INTR VECT`** e **`INIC`**? ¿Por qué?

5. Aunque en el campo de datos de la pantalla del uP-2000 van apareciendo los códigos de las teclas pulsadas, en el campo de direcciones permanece visible otra información que no tiene nada que ver. ¿Cómo podría eliminarse?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H  ; Pila al final de la RAM

     RIM         ; Leer registro estado interrupciones
     ORI 08H     ; Activar bit WR
     ANI 0FEH    ; Desactivar bit M5.5
     SIM         ; Escribirla de nuevo

BUCLE:
     CALL 044EH  ; Recoger pulsación

     CPI 0FH     ; Si es F
     JZ FIN      ; terminar

     CALL 04D5H  ; En caso contrario mostrar el valor
     JMP BUCLE   ; y repetir

FIN:
     RST 1

END
```

Ejercicio 8.8.3.2

Escribir un programa que tome directamente del teclado del uP-2000 una secuencia de datos de 8 bits, mostrándolos en el campo de datos a medida que se introducen y almacenándolos a partir de la posición de memoria **1100H** a medida que vayan introduciéndose. Terminar cuando el valor introducido sea **FFH** sin llegar a almacenarlo.

Indicaciones

La subrutina `044EH` usada en el ejemplo anterior solamente recupera una pulsación de tecla que, ajustándonos a lo que son dígitos hexadecimales, representaría un dato de 4 bits, comprendidos entre `0000` (`0H`) y `1111` (`FH`). Para leer un valor de 8 bits, por tanto, será necesario recuperar dos pulsaciones de tecla, dos *nibbles* que habría que combinar para formar un byte.

Para completar este ejercicio habrá que reproducir los siguientes pasos:

1. Inicializar el registro de interrupciones del 8085 para habilitar la `RST 5.5` y borrar la pantalla.

2. Inicializar un puntero que permita ir almacenando los datos leídos a partir de la posición de memoria indicada.

3. Recuperar una pulsación de tecla, mostrarla en el campo de datos y preservarla.

4. Recuperar una segunda pulsación de tecla y formar con la anterior un byte. Si es `FFH` terminar.

5. Mostrar el byte completo en el campo de datos del uP-2000.

6. Guardar el byte, incrementar el puntero y volver al paso 3.

Algunas de las tareas pueden ser aisladas como subrutinas, facilitando la implementación. El proceso para formar un byte a partir de los dos *nibbles*, por ejemplo, es una tarea candidata a convertirse en subrutina.

Cuestiones

1. ¿Son necesarias todas las instrucciones `PUSH` y `POP` introducidas en el bucle principal del programa?

2. ¿Cuántos datos de 8 bits podrían procesarse con este programa?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     CALL 031FH  ; Borrar la pantalla

     MVI A, 0EH  ; Desenmascarar RST 5.5
     SIM         ; Escribir registro interrupciones

     LXI H, 1100H ; Puntero de destino
     PUSH H      ; Guardar puntero

BUCLE:
     CALL 044EH  ; Recoger pulsación
     PUSH PSW    ; y conservarla
     CALL 04D5H  ; antes de mostrarla en pantalla

     CALL 044EH  ; Recoger segundo dígito
     MOV B, A    ; Ponerlo en B
     POP PSW     ; para recuperar el acumulador

     CALL FRM_BYTE ; Formar el byte con A y B

     POP H       ; Recuperar puntero
     MOV M, A    ; Guardar dato leído
     INX H       ; Incrementar puntero
     PUSH H      ; y volver a guardarlo

     CPI 0FFH    ; Si es FFH
     JZ FIN      ; terminar

     CALL 04D5H  ; En caso contrario mostrar el valor
     JMP BUCLE   ; y repetir

FIN:
     POP H       ; Descartar el puntero

     RST 1

; ----- Subrutina para formar un byte con dos nibbles
; Entradas:
;    A = Nibble de mayor peso
;    B = Nibble de menor peso
; Salidas:
;    A = Byte formado con los dos nibbles
; Modifica:
;    Acumulador y registro de estado
;
FRM_BYTE:
     RLC         ; Tomar el dígito leído antes
```

```
     RLC         ; y desplazarlo cuatro bits
     RLC         ; hacia la izquierda para
     RLC         ; colocarlo como nibble alto

     ANI 0F0H    ; Eliminar cuatro bits de menor peso
     ORA B       ; para agregar segundo dígito leído

     RET
; ----- Fin de la subrutina
END
```

8.8.4 Sumar números introducidos por teclado

Ejercicio 8.8.4.1

Tomando como base el programa del ejercicio 8.8.3.2, introducir las modificaciones necesarias para que los datos introducidos no se escriban en la memoria sino que vayan sumándose. El resultado de la suma debe ser de 16 bits y las sumas parciales, con cada dato introducido, han de mostrarse en el campo de direcciones del uP-2000.

Indicaciones

Para escribir el programa que solicita este ejercicio puede aprovecharse una parte del 8.8.3.2. Las tareas en que se dividiría, cada una de las cuales podría codificarse como una subrutina independiente, serían las siguientes:

1. Proceso de inicialización, consistente en desenmascarar la `RST 5.5`, borrado de la pantalla del uP-2000 e inicialización del acumulador para la suma.

2. Lectura de un byte completo desde el teclado del uP-2000.

3. Suma del byte anterior al acumulador que tiene la suma y visualización.

De esta forma el programa principal se limitaría básicamente a un bucle desde el que se iría llamando a las subrutinas en el orden adecuado, decidiendo cuándo se ha de finalizar.

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     CALL INICIA      ; Inicialización del programa

BUCLE:
     CALL LEE_BYTE    ; Leer un byte del teclado

     CPI 0FFH         ; Si es FFH
     JZ FIN           ; terminar

     CALL VIS_SUMA    ; Si no, sumar y mostrar

     JMP BUCLE        ; y repetir

FIN:
     RST 1


; ----- Subrutina que lleva a cabo la inicialización
;       del programa. No necesita parámetros de entrada
;       ni produce resultados.
;
INICIA:
     CALL 031FH ; Borrar la pantalla

     MVI A, 0EH ; Desenmascarar RST 5.5
     SIM        ; Escribir registro interrupciones

     LXI H, 0000H  ; Acumulador de la suma

     RET
; ----- Fin de la subrutina de inicialización

; ----- Subrutina para sumar HL + A y mostrar la suma
; Entradas:
;    HL = Suma acumulada hasta el momento
;    A = Dato de 8 bits a sumar
; Salidas:
;    Ninguna
; Modifica:
;    Todos menos HL
;
VIS_SUMA:

     MOV E, A
     MVI D, 00H
     DAD D
```

```
      PUSH H
      CALL 04C9H
      POP H

      RET
; ----- Fin de la subrutina de suma

; ----- Subrutina para leer un byte del teclado
; Entradas: Ninguna
; Salidas:
;    A = Byte formado con los dos nibbles
; Modifica:
;    Todos menos HL
;
LEE_BYTE:
      PUSH H

      CALL 044EH  ; Recoger pulsación
      PUSH PSW    ; y conservarla
      CALL 04D5H  ; antes de mostrarla en pantalla

      CALL 044EH  ; Recoger segundo dígito
      MOV B, A    ; Ponerlo en B
      POP PSW     ; para recuperar el acumulador

      CALL FRM_BYTE ; Formar el byte con A y B

      PUSH PSW
      CALL 04D5H  ; En caso contrario mostrar el valor

      POP PSW
      POP H

      RET
; ----- Fin de la subrutina que lee un byte del teclado

; ----- Subrutina para formar un byte con dos nibbles
; Entradas:
;    A = Nibble de mayor peso
;    B = Nibble de menor peso
; Salidas:
;    A = Byte formado con los dos nibbles
; Modifica:
;    Acumulador y registro de estado
;
FRM_BYTE:
      RLC   ; Tomar el dígito leído antes
      RLC   ; y desplazarlo cuatro bits
      RLC   ; hacia la izquierda para
      RLC   ; colocarlo como nibble alto

      ANI 0F0H    ; Eliminar cuatro bits de menor peso
      ORA B       ; para agregar segundo dígito leído
```

```
     RET
; ----- Fin de la subrutina
END
```

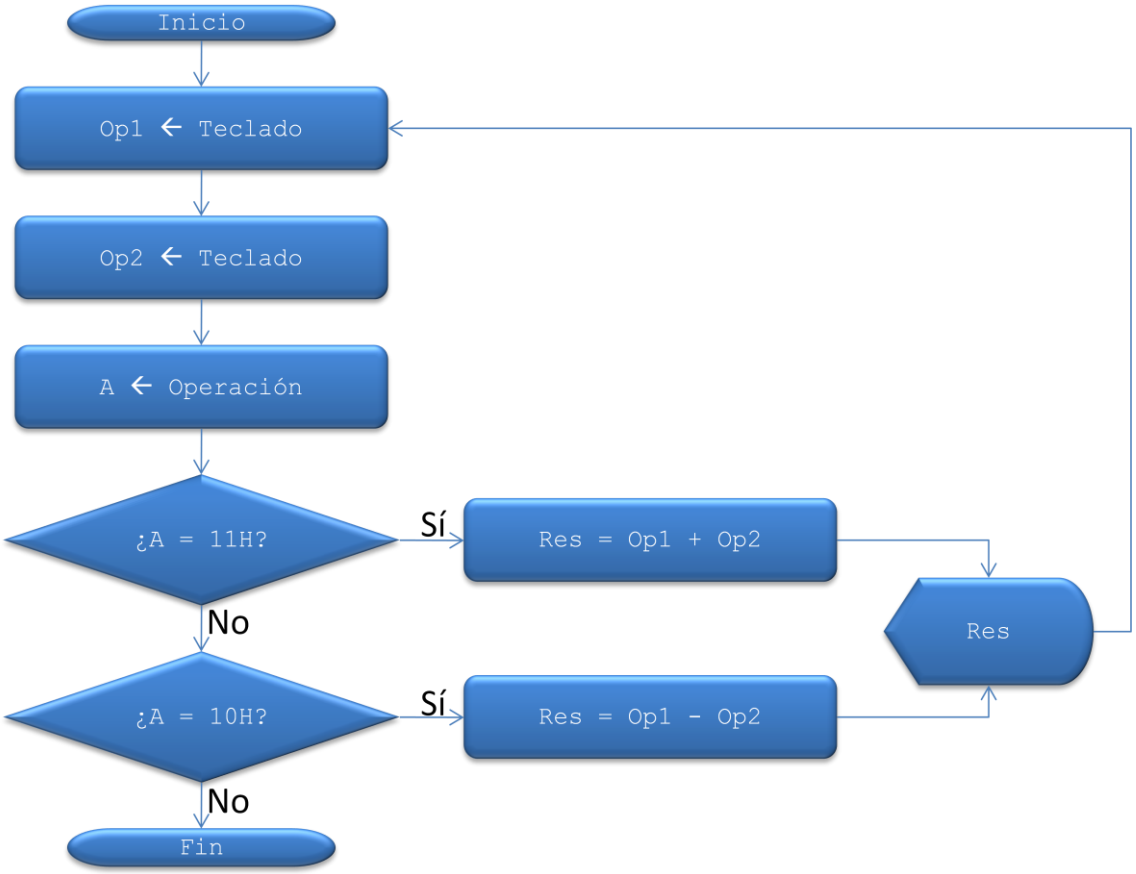
8.8.5 Uso del teclado para elegir opciones de ejecución

Ejercicio 8.8.5.1

Escribir un programa que acepte dos datos de 8 bits introducidos por teclado, deben visualizarse en el campo de datos a medida que se escriban, efectuando la suma si se pulsa a continuación la tecla **POST** o bien restando el segundo del primero si la tecla pulsada es **EJEC**. Mostrar el resultado en el campo de direcciones y repetir la operación hasta que se pulse una tecla distinta a **POST** y **EJEC**.

Indicaciones

En algunos ejercicios previos la introducción de un cierto valor, como `FFH`, ha determinado el fin de la ejecución del programa. En este caso la pulsación de una tecla será la que decida qué operación se llevará a cabo: la suma de los operandos, la resta o bien la finalización del programa. El siguiente diagrama de flujo resume cuál sería el funcionamiento del programa:



Ordinograma del ejercicio 8.8.5.1.

Para escribir el programa pueden reutilizarse las subrutinas que leen un byte del teclado, forman un byte a partir de dos pulsaciones de tecla y llevan a cabo la inicialización. El resto de las tareas pueden completarse en el bucle principal del programa.

Cuestiones

1. En la solución propuesta se hace un uso intensivo de la pila para conservar datos intermedios hasta alcanzar el final de cada ciclo del bucle, en el que se efectúa la suma o resta. ¿Qué otro medio podría emplearse para conservar esos datos inmediatos?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     CALL INICIA      ; Inicialización del programa

BUCLE:
     CALL LEE_BYTE    ; Leer los operandos
     PUSH PSW         ; desde el teclado
     CALL LEE_BYTE    ; y guardarlos
     PUSH PSW

     CALL 044EH       ; Esperar operación

     CPI 11H          ; Si se ha pulsado POST
     JZ SUMA          ; efectuar la suma

     CPI 10H          ; Si se ha pulsado EJEC
     JZ RESTA         ; efectuar la resta

     RST 1            ; Cualquier otra tecla, termina

SUMA:
     POP PSW          ; Rescatar datos
     MOV B, A
     POP PSW

     CALL VIS_SUMA    ; Efectuar la suma
     JMP BUCLE        ; y repetir

RESTA:
     POP PSW          ; Rescatar datos
     MOV B, A
     POP PSW

     CALL VIS_RESTA   ; Efectuar la resta
     JMP BUCLE


; ----- Subrutina que lleva a cabo la inicialización
;       del programa. No necesita parámetros de entrada
;       ni produce resultados.
;
INICIA:
     CALL 031FH  ; Borrar la pantalla

     MVI A, 0EH  ; Desenmascarar RST 5.5
     SIM         ; Escribir registro interrupciones

     RET
```

```
; ----- Fin de la subrutina de inicialización

; ----- Subrutina para sumar o restar y mostrar resultado
; Entradas:
;    A, B = Datos a sumar
; Salidas:
;    Ninguna
; Modifica:
;    Todos
;
VIS_SUMA:
     ADD B
     JMP VIS

VIS_RESTA:
     SUB B

VIS:
     MOV L, A
     MVI H, 00H
     CALL 04C9H

     RET
; ----- Fin de la subrutina de suma

; ----- Subrutina para leer un byte del teclado
; Entradas: Ninguna
; Salidas:
;    A = Byte formado con los dos nibbles
; Modifica:
;    Todos menos HL
;
LEE_BYTE:
     PUSH H

     CALL 044EH ; Recoger pulsación
     PUSH PSW   ; y conservarla
     CALL 04D5H ; antes de mostrarla en pantalla

     CALL 044EH ; Recoger segundo dígito
     MOV B, A   ; Ponerlo en B
     POP PSW    ; para recuperar el acumulador

     CALL FRM_BYTE ; Formar el byte con A y B

     PUSH PSW
     CALL 04D5H ; En caso contrario mostrar el valor

     POP PSW
     POP H

     RET
; ----- Fin de la subrutina que lee un byte del teclado
```

```
; ----- Subrutina para formar un byte con dos nibbles
; Entradas:
;    A = Nibble de mayor peso
;    B = Nibble de menor peso
; Salidas:
;    A = Byte formado con los dos nibbles
; Modifica:
;    Acumulador y registro de estado
;
FRM_BYTE:
     RLC   ; Tomar el dígito leído antes
     RLC   ; y desplazarlo cuatro bits
     RLC   ; hacia la izquierda para
     RLC   ; colocarlo como nibble alto

     ANI 0F0H    ; Eliminar cuatro bits de menor peso
     ORA B       ; para agregar segundo dígito leído

     RET
; ----- Fin de la subrutina
END
```

8.8.6 Ejercicios propuestos

Ejercicio 8.8.6.1

Escribir un programa que solicite una dirección de memoria a través del teclado del uP-2000, lea su contenido y lo muestre en el campo de datos. A partir de ahí debe permitir avanzar o retroceder una posición mediante las teclas **POST** y **EJEC**, respectivamente. Cualquier otra tecla provocará la finalización del programa.

Indicaciones

Para completar este ejercicio solamente se necesita saber cómo puede leerse una dirección de memoria desde el teclado del uP-2000. Existen al menos dos vías posibles:

1. Modificar la subrutina `LEE_BYTE` de ejercicios previos de forma que lea no la pulsación de dos teclas, sino de cuatro, y vaya mostrando el resultado parcial en el campo de direcciones en lugar de en el campo de datos de la pantalla del uP-2000.

2. Usar la subrutina del uP-2000 situada en la dirección `02BFH`, que leerá una dirección desde el teclado, permitiendo hacer correcciones, devolviéndola en la pareja de registros `DE` en cuanto se pulse la tecla **POST**.

Obviamente la segunda alternativa es más sencilla y efectiva, ya que basta con una instrucción `CALL` para obtener en la pareja `DE` la dirección introducida, sin tener que preocuparse de nada más.

Una vez que se tenga la dirección el programa deberá dar los pasos siguientes:

1. Mostrar la dirección en el campo de direcciones del uP-2000.

2. Recuperar el dato existente en esa dirección y mostrarlo en el campo de datos.

3. Esperar la pulsación de una tecla.

4. Si la tecla es **POST** incrementar la dirección y volver al paso 1.

5. Si la tecla es **EJEC** decrementar la dirección y volver al paso 1.

6. Cualquier otra tecla finalizará la ejecución del programa.

Cuestiones

1. Ejecute el programa, introduzca la dirección `F000H` y utilice la tecla **`POST`** para ir avanzando. Observe los datos recuperados. ¿Puede deducir qué es lo que está ocurriendo?

Ejercicio 8.8.6.2

Escribir un programa que muestre en el campo de direcciones del uP-2000 el dato **0001H**, permitiendo utilizar las teclas **POST** y **EJEC** para desplazar el 1 hacia la izquierda o la derecha, respectivamente. La pulsación de cualquier otra tecla pondrá fin al programa.

Indicaciones

El objetivo de este ejercicio es combinar en un mismo programa el uso de algunas subrutinas del uP-2000, como la que muestra datos en el campo de direcciones o recupera una pulsación de teclado, con el uso de instrucciones y técnicas ya conocidas. La parte más importante del programa será la encargada de separar el dato de 16 bits, mostrado en pantalla y que inicialmente será `0001H`, en cuatro *nibbles* o porciones de 4 bits, recombinándolas según la tecla pulsada para simular un desplazamiento del dígito `1` hacia la izquierda o hacia la derecha.

Una vez que se haya separado el dato de 16 bits en cuatro *nibbles* independientes, la forma de combinarlos, con independencia de que se vaya a rotar hacia la izquierda o hacia la derecha, guarda muchas similitudes, como se aprecia en el siguiente diagrama:

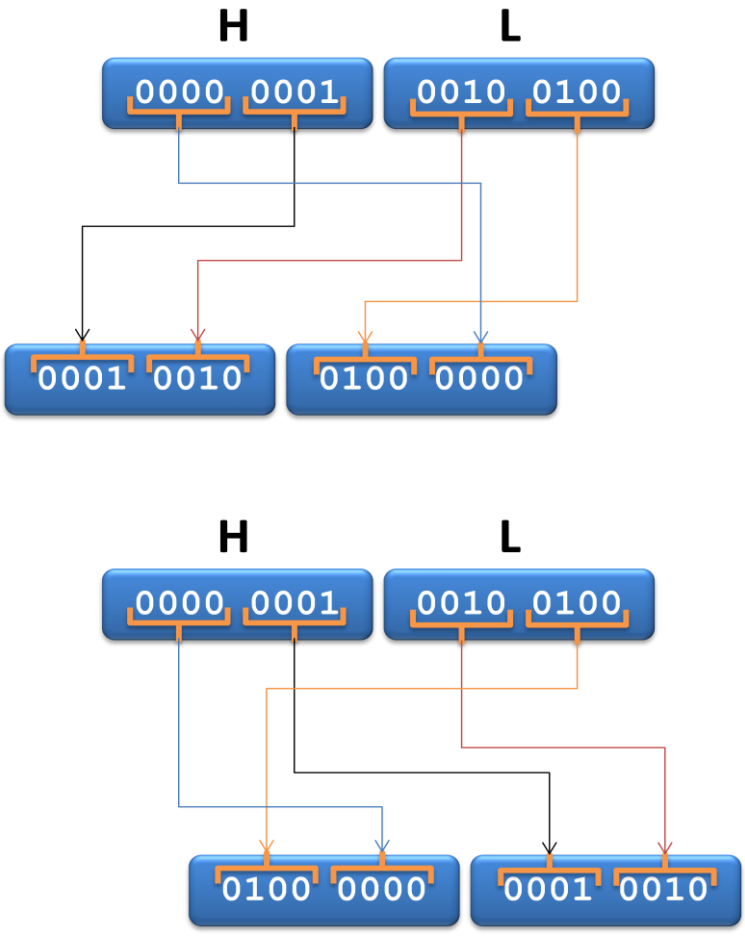


Diagrama esquemático del intercambio de *nibbles*.

En la parte superior del diagrama se ha representado la rotación a la izquierda de un *nibble*, mientras que en la inferior está la rotación a la derecha. El valor de cada *nibble* no es significativo, se han asignado valores distintos para diferenciarlos claramente.

Observando las dos partes del diagrama es fácil ver que los *nibbles* extremos del dato de 16 bits, el situado más a la izquierda y el extremo derecho, se combinan siempre en un mismo byte. Obviamente los *nibbles* centrales se combinan en el segundo byte. La única diferencia es que en un caso ese byte es el más significativo y en otro el menos significativo. Se deduce, por tanto, que el código para combinar los *nibbles* será común, difiriendo la rotación a la izquierda de la rotación a la derecha únicamente en qué byte se toma como LSB o MSB a la hora de invocar a la subrutina de visualización en el campo de direcciones.

Para facilitar el desarrollo del programa puede efectuarse la siguiente división del código:

1. Una subrutina que dado un byte, por ejemplo en el acumulador, lo separe en dos partes y desplace sus bits a izquierda o derecha para facilitar su posterior combinación, haciendo que los cuatro bits altos pasen a ser lo bajos y a la inversa.

2. Una segunda subrutina, sirviéndose de la anterior, se encargará de tomar el dato de 16 bits, obtener sus cuatro *nibbles* y recombinarlos según el esquema esbozado en el diagrama previo, devolviéndolos en un par de registros.

3. El programa principal llamará a la subrutina del punto 2 y, según la tecla pulsada, colocará un byte como LSB y otro como MSB o a viceversa, dependiendo del sentido de la rotación.

8.9 Introducción de retardos

A pesar de que el 8085 es un microprocesador lento, si se le compara con los actuales, su velocidad sigue siendo excesiva para ciertas tareas que se miden en la escala de tiempos humana, en lugar de en la de las máquinas. Por ello en ocasiones es necesario introducir en los programas retardos, código que tarda un cierto tiempo en ejecutarse para ajustar la velocidad a lo que espera el usuario.

La finalidad de los ejercicios de esta sección es la siguiente:

- Observar cuál es la problemática que plantea la ejecución de algunas tareas a la velocidad habitual del microprocesador, efectuando una primera aproximación a cuál podría ser la solución.
- Aprender a calcular cuánto tiempo tarda en ejecutarse un grupo de instrucciones.
- Saber cómo crear subrutinas que provoquen retardos de tiempo fijo o variable, según los parámetros recibidos.
- Utilizar la rutina de retardo que ofrece el propio uP-2000 en su memoria EPROM.
- Usar las subrutinas de demora en diversos ejemplos, facilitando la lectura de datos en la pantalla del uP-2000.

8.9.1 Calcular el tiempo de ejecución de las instrucciones

Ejercicio 8.9.1.1

Se quiere mostrar en el campo de direcciones del uP-2000 la secuencia de valores **0000H** a **FFFFH**, uno tras otro sucesivamente. Escribir un programa con esa finalidad, ejecutarlo y comprobar el resultado.

Indicaciones

El objetivo de este programa no es otro que plantear el problema que supone la visualización de datos por parte de un programa sin ninguna intervención del usuario, es decir, sin esperar a que se pulse alguna tecla para ir avanzando.

El programa en sí es muy simple, ya que no tiene más que implementar un bucle que recorra los valores indicados, llamando a la subrutina de visualización en el campo de direcciones a cada ciclo.

Cuestiones

1. Al ejecutar el programa, ¿puede distinguir los cambios en los dos últimos dígitos del campo de direcciones?

2. ¿Cómo podría ralentizarse el proceso de visualización para que los dígitos pudieran leerse satisfactoriamente?

3. ¿Es posible establecer un tiempo de demora concreto a cada ciclo del bucle?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     CALL 031FH       ; Borrar pantalla

     LXI H, 0000H     ; Configuración inicial

BUCLE:
     PUSH H           ; Guardar HL
     CALL 04C9H       ; Mostrarlo en campo direcciones


     POP H            ; Recuperar HL
     INX H            ; e incrementar su contenido

     MOV A, H         ; Si aún no se ha vuelto a 0000H
     ORA L            ; continuar
     JNZ BUCLE

     RST 1            ; Fin del programa

END
```

8.9.2 Creación de una rutina de retardo

Ejercicio 8.9.2.1

> Escribir una subrutina de retardo genérica que, sin modificar ningún registro, tarde aproximadamente una décima de segundo en retornar. Utilizarla en el programa del ejercicio 8.9.1.1 para ajustar el retraso en el ciclo de visualización de los datos.

Indicaciones

Las rutinas de retardo suelen implementarse como bucles en los que no se hace nada *útil*, simplemente se va incrementando o reduciendo el contenido de un registro y comprobando si se ha llegado al final. Para calcular el tiempo que tardaría en ejecutarse un bucle así es preciso conocer los siguientes elementos:

1. La frecuencia de la señal de reloj externa que se emplea para estabilizar el 8085.

2. El número de ciclos que consume cada instrucción de la rutina de retardo.

3. El número de veces que se repetirá el bucle de esa rutina.

En el uP-2000 la frecuencia del reloj externo es de 4 Mhz, por lo que internamente el 8085 operará a 2 Mhz, es decir, dos millones de ciclos máquina por segundo. Esto significa que cada ciclo empleará en su ejecución 0.5×10^{-6} segundos o, lo que es lo mismo, 0.5 microsegundos. Con estos datos, es fácil deducir una fórmula como la siguiente que nos permita saber cuánto tardaría en ejecutarse una cierta rutina de retardo:

```
[(N-1)*ciclos_bucle + resto_ciclos] * 0.5
```

Aquí `N` representa el número de veces que se ejecutaría el bucle de la rutina, `ciclos_bucle` el número de ciclos máquina correspondientes a las instrucciones que se ejecutan en ese bucle y `resto_ciclos` el número de ciclos máquina del resto de instrucciones de la rutina. El resultado será el número de microsegundos que se tarda en ejecutar esa rutina. Dividiendo entre un millón se obtienen los segundos.

Puesto que en este caso el tiempo que tardará la rutina de retardo lo sabemos de antemano: una décima de segundo (100.000 microsegundos), obteniendo el número de ciclos del bucle y despejando en la anterior fórmula `N` se sabrá cuantas veces ha de repetirse el bucle.

Cuestiones

1. Explicar cómo se ha calculado el número de veces que ha de repetirse el bucle con datos concretos.

2. ¿Cuánto tarda exactamente la rutina de retardo en ejecutarse?

3. ¿Cómo se podría introducir en el programa principal un retardo de dos décimas de segunda entre la visualización de dato y dato?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     CALL 031FH       ; Borrar pantalla

     LXI H, 0000H     ; Configuración inicial

BUCLE:
     PUSH H           ; Guardar HL
     CALL 04C9H       ; Mostrarlo en campo direcciones

     CALL RET1DEC     ; Rutina de retardo

     POP H            ; Recuperar HL
     INX H            ; e incrementar su contenido

     MOV A, H         ; Si aún no se ha vuelto a 0000H
     ORA L            ; continuar
     JNZ BUCLE

     RST 1            ; Fin del programa

; ----- Subrutina de retardo de una décima de segundo
;
RET1DEC:
    PUSH B ; Guardar registros que se modificarán
    PUSH PSW

    LXI B, 208DH      ; Número de repeticiones

BRET:
    DCX B             ; 6 ciclos
    MOV A, B          ; 4 ciclos
    ORA C             ; 4 ciclos
    JNZ BRET          ; 10 ciclos

    POP PSW
    POP B

    RET
; ----- Fin de la subrutina de retardo
END
```

8.9.3 Uso de la rutina de retardo del uP-2000

Ejercicio 8.9.3.1

Adaptar el programa del ejercicio 8.9.2.1 para prescindir de la subrutina de retardo propia, usando en su lugar la que ofrece el propio uP-2000 en su EPROM. El tiempo de demora en la visualización de los datos debe ser el mismo: una décima de segundo.

Indicaciones

El sistema uP-2000 cuenta con una subrutina, alojada a partir de la dirección `0B74H`, que produce una demora de un milisegundo. Puesto que se pretende que el programa siga mostrando los datos a la misma velocidad, para conseguir un retardo de una décima de segundo será preciso invocar a la subrutina mencionada aproximadamente 100 veces. Si se quiere ser estricto, habría que calcular también el tiempo que se empleará en llamar a esa subrutina N veces, descontándolo para realizar un ajuste perfecto.

Lo único que hay que hacer es eliminar del programa la subrutina de retardo original, sustituyéndola por otra que llame a la del uP-2000 el número de veces adecuado.

Cuestiones

1. La subrutina de retardo del uP-2000 se encuentra a partir de la dirección de memoria `0B74H` y tiene una longitud de 11 bytes. Desensamble ese breve código para comprobar cómo genera la rutina el retardo de un milisegundo.

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     CALL 031FH       ; Borrar pantalla

     LXI H, 0000H     ; Configuración inicial

BUCLE:
     PUSH H           ; Guardar HL
     CALL 04C9H       ; Mostrarlo en campo direcciones


     CALL RETARDO     ; Rutina de retardo

     POP H            ; Recuperar HL
     INX H            ; e incrementar su contenido

     MOV A, H         ; Si aún no se ha vuelto a 0000H
     ORA L            ; continuar
     JNZ BUCLE

     RST 1            ; Fin del programa

; ----- Subrutina de retardo de una décima de segundo
;       usando la subrutina que ofrece el uP-2000
;
RETARDO:
    PUSH B ; Guardar registros que se modificarán

    MVI C, 60H   ; Número de repeticiones

BRET:
    CALL 0B74H   ; 1 milisegundo

    DCR C  ; Hasta completar la décima de segundo
    JNZ BRET

    POP B
    RET
; ----- Fin de la subrutina de retardo
END
```

8.9.4 Visualización temporizada de datos

Ejercicio 8.9.4.1

Realizar un programa que solicite por teclado dos datos: primero una dirección de memoria y a continuación un dato de 8 bits que actuará como contador. El programa debe mostrar en la pantalla del uP-2000 las direcciones y sus contenidos, recorriendo el número de posiciones indicado, con una pausa de 1 segundo entre dato y dato.

Indicaciones

La finalidad de este ejercicio es combinar varias de las técnicas que ha aprendido en ésta y secciones previas, utilizando la lectura de datos por teclado, visualización de datos en pantalla y uso de rutinas de retardo.

Para completar el programa, éste puede dividirse en los pasos siguientes:

1. Desenmascarar la `RST 5.5` y borrar la pantalla.

2. Solicitar la dirección de memoria desde la que se partirá, usando la subrutina del uP-2000 situada en la dirección `02BFH`.

3. Leer del teclado el dato de 8 bits con el que se indicará el número de posiciones de memoria a recorrer. Puede utilizarse para este fin la subrutina `LEE_BYTE` escrita en ejercicios de la sección previa.

4. En el interior de un bucle, que se repetirá tantas veces como indique el dato obtenido en el paso 3, repetir las acciones indicadas a continuación:

    1. Mostrar la dirección de la que se va a leer en el campo de direcciones de la pantalla.

    2. Recuperar el dato apuntado por esa dirección y mostrarlo en el campo de datos de la pantalla.

    3. Esperar durante 1 segundo. Puede reutilizarse cualquiera de las subrutinas de retardo de ejercicios previos.

    4. Incrementar la dirección y reducir el contador, volviendo al paso 1.

Un aspecto fundamental en este programa es la preservación de los datos que emplea, la dirección de memoria leída y el contador de posiciones a tratar, ya que las subrutinas de visualización de datos alteran todos los registros.

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     MVI A, 0EH       ; Desenmascarar RST 5.5
     SIM

     CALL 031FH       ; Borrar pantalla

     CALL 02BFH       ; Solicitar dirección
     PUSH D

     CALL LEE_BYTE    ; Solicitar contador

     MOV B, A         ; Contador en B
     POP H            ; Dirección en HL

BUCLE:
     PUSH B      ; Guardar dirección y contador
     PUSH H      ; para evitar perderlos

     CALL 04C9H  ; Mostrar dirección

     POP H       ; Recuperar dirección para poder
     PUSH H
     MOV A, M    ; leer el dato que contiene
     CALL 04D5H  ; y mostrarlo también

     CALL RETARDO ; Rutina de retardo

     POP H       ; Recuperar dirección y contador
     POP B

     INX H       ; Avanzar a la siguiente dirección
     DCR B       ; Decrementar el contador
     JNZ BUCLE

     RST 1       ; Fin del programa

; ----- Subrutina de retardo de un segundo
;
RETARDO:
    PUSH B ; Guardar registros que se modificarán

    MVI C, 0AH        ; Número de repeticiones

BRET:
    CALL RET1DEC      ; Una décima

    DCR C             ; Hasta completar el segundo
```

```
    JNZ BRET

    POP B
    RET
; ----- Fin de la subrutina de retardo

; ----- Subrutina de retardo de una décima de segundo
;
RET1DEC:
    PUSH B ; Guardar registros que se modificarán
    PUSH PSW

    LXI B, 208DH ; Número de repeticiones

BRET1:
    DCX B       ; 6 ciclos
    MOV A, B    ; 4 ciclos
    ORA C       ; 4 ciclos
    JNZ BRET1   ; 10 ciclos

    POP PSW
    POP B

    RET
; ----- Fin de la subrutina de retardo

; ----- Subrutina para leer un byte del teclado
; Entradas: Ninguna
; Salidas:
;    A = Byte formado con los dos nibbles
; Modifica:
;    Todos
;
LEE_BYTE:
     CALL 044EH ; Recoger pulsación
     PUSH PSW   ; y conservarla
     CALL 04D5H ; antes de mostrarla en pantalla

     CALL 044EH ; Recoger segundo dígito
     MOV B, A   ; Ponerlo en B
     POP PSW    ; para recuperar el acumulador

     CALL FRM_BYTE ; Formar el byte con A y B

     PUSH PSW
     CALL 04D5H ; Mostrar el valor

     POP PSW

     RET
; ----- Fin de la subrutina que lee un byte del te-
clado

; ----- Subrutina para formar un byte con dos nibbles
```

```
; Entradas:
;    A = Nibble de mayor peso
;    B = Nibble de menor peso
; Salidas:
;    A = Byte formado con los dos nibbles
; Modifica:
;    Acumulador y registro de estado
;
FRM_BYTE:
     RLC   ; Tomar el dígito leído antes
     RLC   ; y desplazarlo cuatro bits
     RLC   ; hacia la izquierda para
     RLC   ; colocarlo como nibble alto

     ANI 0F0H    ; Eliminar cuatro bits de menor peso
     ORA B       ; para agregar segundo dígito leído

     RET
; ----- Fin de la subrutina
END
```

8.9.5 Ejercicios propuestos

Ejercicio 8.9.5.1

Preparar una subrutina de retardo parametrizable, de forma que al invocarla se le indique con un argumento el número de décimas de segundo que debe emplear antes de devolver el control.

Escribir un programa que la utilice, mostrando un contador de **0100H** a **0000H** en el campo de direcciones del uP-2000 a una velocidad variable, que podrá ajustarse previamente con las teclas **POST** y **EJEC**, respectivamente. La tecla **GO** lanzará el contador con la velocidad elegida.

Indicaciones

Partiendo de la subrutina desarrollada para el ejercicio 8.9.2.1, adaptarla para que acepte un parámetro con el que pueda indicarse el tiempo de espera, en décimas de segundo, no es algo difícil. El parámetro de entrada puede ser cualquier registro de 8 bits, de forma que el tiempo de espera estaría comprendido entre 1 décima de segundo (valor `01H`) y unos 25.6 segundos (valor `00H`).

El programa debe dar los pasos siguientes:

1. Inicializar el registro de interrupciones, borrar la pantalla del uP-2000 y asignar un retardo inicial que podría ser el valor `0AH` (un segundo).

2. Mostrar en el campo de datos del uP-2000 el valor actual del retardo y esperar la pulsación de una tecla.

3. Si se pulsa la tecla **EJEC** reducir el retardo y volver al paso 2.

4. Si se pulsa la tecla **POST** incrementar el retardo y volver al paso 2.

5. Si se pulsa la tecla **GO** ejecutar el bucle que mostrará los valores `0100H` a `0000H` en el campo de direcciones, usando el retardo establecido. Terminado el bucle volver al paso 2.

6. Cualquier otra tecla provocará la finalización del programa.

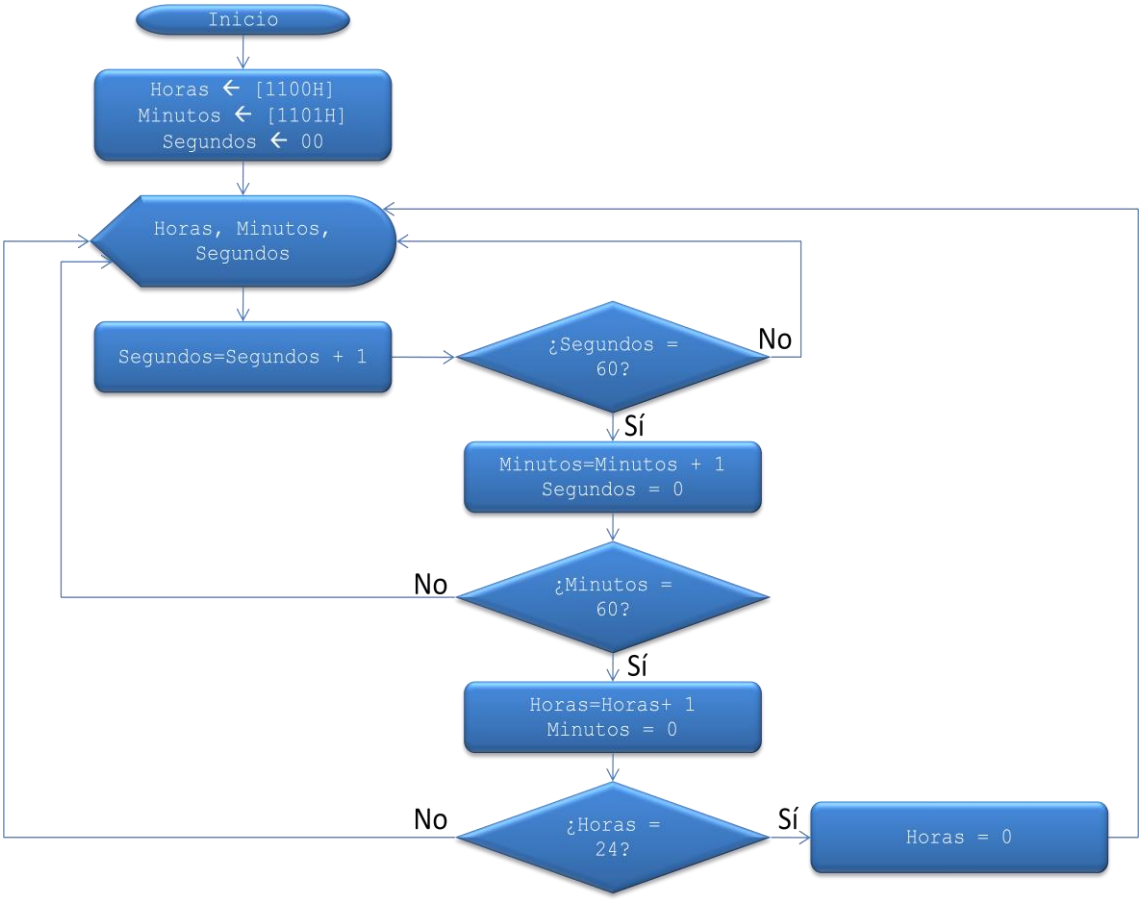
De esta forma el programa permitirá establecer un retardo, ejecutar el bucle, modificar ese retardo, volver a ejecutarlo y así sucesivamente hasta que se desee salir.

Ejercicio 8.9.5.2

> Escribir un programa que muestre en la pantalla del uP-2000 un reloj, colocando horas y minutos en el campo de direcciones y los segundos en el campo de datos. Las horas y minutos de partida se tomarán de las direcciones **1100H** y **1101H**, respectivamente, mientras que los segundos partirán siempre de cero.

Indicaciones

Un reloj como el que solicita el ejercicio no es más que un bucle que emplea un segundo exacto en cada ciclo, incrementándose al final del mismo la cuenta de segundos. Si ésta llega a 60 se pone a 0 y se incrementa la cuenta de minutos, y si ésta llega a 60 se pone a 0 y se incrementa la cuenta de horas. Éstas, al llegar a 24, se ponen de nuevo a 0. El siguiente diagrama de flujo muestra cuál sería la lógica de control del reloj.



Ordinograma del ejercicio 8.9.5.2 para controlar el reloj.

La única dificultad que plantea este ejercicio es cómo visualizar la cuenta de segundos, minutos y horas como lo haría un reloj digital, es decir, usando exclusivamente dígitos decimales, no hexadecimales. No resulta natural que el segundo y el minutero pasen de `3B` (60 en hexadecimal) a `00`. Sería necesario que cuando uno de los contadores tenga el valor `09` y se incremente no pase a ser `0A`, sino `10`. Esto equivale a utilizar de manera independientes los

dos *nibbles* que forman un byte, usando 4 bits para representar un dígito decimal, comprendido entre 0 y 9, en lugar de un valor binario. En un byte, por tanto, se podrían representar los valores `00` a `99` como máximo. Este tipo de representación es conocida como BCD (*Binary Coded Decimal*) y resulta tan habitual que el 8085 cuenta con una instrucción de ajuste, la instrucción `DAA`, capaz de realizar las correcciones indicadas.

El programa, por tanto, no tendría más que ir actualizando los contadores y, tras cada incremento, utilizar la instrucción `DAA` para realizar el ajuste a BCD.

8.10 Uso del PPI

El sistema uP-2000 está preparado para comunicarse con dispositivos externos a través de diferentes vías, entre ellas el puerto serie (USART) que facilita la transmisión de los programas desde el PC al equipo para su ejecución. Otra de esas vías es el PPI, un integrado que cuenta con tres puertos de 8 bits capaces de transmitir datos en paralelo, ya sea desde el exterior hacia el microprocesador o a la inversa.

En la sección *3. El módulo de E/S PPI 8255* se describió la arquitectura del PPI, sus diferentes modos de operación y los puertos que tiene asignados. La finalidad de los ejercicios de esta sección es usar esa información para escribir programas que se comuniquen con el exterior, alcanzando los siguientes objetivos:

- Aprender a configurar el PPI en modo 0 estableciendo el sentido en el que fluirá la información en cada puerto.
- Enviar información a través de los puertos del PPI para iluminar leds del módulo externo del uP-2000.
- Recibir información desde los puertos del PPI para conocer el estado de los microinterruptores del módulo externo.
- Combinar el uso de leds, microinterruptores, teclado y pantalla del uP-2000.

8.10.1 Configuración de los puertos del PPI

Ejercicio 8.10.1.1

Preparar una subrutina que se encargue de configurar adecuadamente los puertos **A** y **B** del PPI (siempre en modo 0), aceptando como parámetros dos indicadores que le permitirán saber si cada uno de ellos se usará como entrada o como salida de datos. Comprobar su funcionamiento.

Indicaciones

La configuración del PPI, estableciendo su modo de funcionamiento general y la tarea asignada a cada uno de sus puertos, requiere que se conozca el formato del registro de control de este circuito integrado, detallado en su momento en la *Figura 18. Registro de control del 8255*. Dado que solamente se va a utilizar el modo 0, el más simple, los tres puertos: `A`, `B` y `C`, pueden ser configurados independientemente.

Típicamente se usarán siempre la misma configuración a la hora de trabajar con el PPI, por lo que el valor a enviar al registro de configuración sería siempre idéntico. Disponer de una rutina que se encargue de establecer esa configuración, sin embargo, resulta más flexible. Esta subrutina tendrá que hacer lo siguiente:

1. Recibirá en los registros `A` y `B` sendos indicadores, que serán 0 si el puerto del mismo nombre ha de establecerse como salida o 1 para configurarlo como entrada.

2. El indicador recibido en el registro `B` ha de colocarse en el segundo bit del byte de configuración.

3. El indicador recibido en el registro `A` tiene que colocarse en el quinto bit.

4. Para habilitar la configuración de modo de funcionamiento del PPI, el octavo bit (el de mayor peso) debe ponerse a 1.

5. Formado el byte de configuración habrá que enviarlo al registro del PPI, localizado en el puerto de E/S `3BH`.

La subrutina devolverá en el acumulador el byte de configuración que se haya utilizado. De esta forma el programa podrá invocar a la rutina con unos ciertos parámetros y, a continuación, mostrar dicho byte en la pantalla del uP-2000. Hay que efectuar dos configuraciones distintas, visualizándolas en la pantalla esperando a que se pulse una tecla entre una y otra.

Cuestiones

1. ¿Qué valor de configuración se obtiene al poner el puerto `A` como salida y el `B` como entrada? ¿Y cuando la configuración es la inversa?

2. Además del puerto `3BH`, mediante el que se establece la configuración del PPI, ¿qué otros puertos se utilizarán habitualmente con este dispositivo de E/S?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     MVI A, 0EH       ; Desenmascarar RST 5.5
     SIM

     MVI A, 00H       ; Puerto A como salida
     MVI B, 01H       ; y B como entrada

     CALL CONF_PPI    ; Configurar PPI
     CALL 04D5H       ; Mostrar byte de configuración
     CALL 044EH       ; Esperar una tecla

     MVI A, 01H       ; Puerto A como entrada
     MVI B, 00H       ; y B como salida

     CALL CONF_PPI    ; Configurar PPI
     CALL 04D5H       ; Mostrar byte de configuración
     CALL 044EH       ; Esperar una tecla

     RST 1            ; Fin del programa

; ----- Subrutina de configuración del PPI,
;       siempre en modo 0
;
; Entradas:
;     A -> Puerto A: 0-Salida, 1-Entrada
;     B -> Puerto B: 0-Saluda, 1-Entrada
;
CONF_PPI:

     PUSH PSW   ; Guardar A temporalmente

     MOV A, B   ; Para llevar la configuración
     RLC        ; de B un bit a la izquierda
     ANI 02H    ; Resto de bits a 0
     MOV B, A   ; De vuelta a B

     POP PSW

     RLC        ; La configuración
     RLC        ; del puerto A
     RLC        ; en el nibble alto
     RLC
     ANI 010H   ; Resto de bits a 0

     ORA B      ; Agregar puerto B, en el nibble bajo

     ORI 80H    ; Activación del bit de configuración
```

```
      OUT 3BH     ; Escritura en registro del PPI
      RET
END
```

8.10.2 Envío de datos a través del PPI (iluminación de leds)

Ejercicio 8.10.2.1

Escribir un programa que tome el dato contenido en la posición de memoria **1100H** y lo escriba en el puerto **A** del PPI, configurándolo previamente de forma adecuada. Ejecutar el programa varias veces teniendo conectado al puerto **A** del PPI (**J3** en el uP-2000) el módulo de leds y microinterruptores, modificando el contenido de la posición **1100H** para ver el resultado.

Indicaciones

Utilizando la subrutina creada en el ejercicio previo, que facilita la configuración del PPI, y conociendo el número del puerto de E/S que corresponde al puerto A del PPI, la realización de este programa no representa ninguna dificultad especial.

Cuestiones

1. ¿Cómo se ha configurado el puerto A del PPI para poder enviar datos a través del mismo? ¿Se ha establecido alguna configuración para el puerto B?

2. ¿Qué es lo que puede verse en el panel de leds cuando se ejecuta el programa?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H  ; Pila al final de la RAM

     MVI A, 00H       ; Puerto A como salida
     CALL CONF_PPI    ; Configurar PPI

     LDA 1100H        ; Recuperar dato
     OUT 38H          ; y enviarlo por el puerto A

     RST 1            ; Fin del programa

; ----- Subrutina de configuración del PPI,
;       siempre en modo 0
;
; Entradas:
;     A -> Puerto A: 0-Salida, 1-Entrada
;     B -> Puerto B: 0-Saluda, 1-Entrada
;
CONF_PPI:

     PUSH PSW    ; Guardar A temporalmente

     MOV A, B    ; Para llevar la configuración
     RLC         ; de B un bit a la izquierda
     ANI 02H     ; Resto de bits a 0
     MOV B, A    ; De vuelta a B

     POP PSW

     RLC         ; La configuración
     RLC         ; del puerto A
     RLC         ; en el nibble alto
     RLC
     ANI 010H    ; Resto de bits a 0

     ORA B       ; Agregar puerto B, en el nibble bajo

     ORI 80H     ; Activación del bit de configuración

     OUT 3BH     ; Escritura en registro del PPI

     RET

END
```

Ejercicio 8.10.2.2

Manteniendo la configuración del ejercicio 8.10.2.1, con el puerto **A** en modo de salida y teniendo conectado al mismo la tarjeta de leds, escribir un programa que vaya iluminando los leds de manera individual, de derecha a izquierda, haciendo una pausa de medio segundo entre cambios. A cada ciclo mostrar en el campo de datos del uP-2000 el valor enviado al puerto. Al terminar apagar todos los leds.

Indicaciones

Para completar este ejercicio reproducir los pasos siguientes:

1. Configurar el PPI de forma que el puerto A sea de salida.
2. Inicializar el acumulador con el valor adecuado para iluminar el primer led.
3. Enviar el valor contenido en el acumulador por el puerto A del PPI y mostrarlo también en la pantalla del uP-2000.
4. Introducir un retardo de medio segundo. Pueden emplearse las subrutinas de retardo escritas en secciones previas.
5. Rotar el contenido del acumulador para activar el siguiente bit, situado a la izquierda del anterior.
6. Volver al paso 3. Terminar cuando el valor contenido en el acumulador sea el mismo que se estableció en el paso 2.
7. Terminar desactivando todos los leds y escribiendo el mismo valor en el campo de datos del uP-2000.

Cuestiones

1. ¿Cuál es la secuencia de valores que corresponde a la iluminación de cada led individual?
2. ¿Cómo podría invertirse el sentido en que van iluminándose los leds?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     MVI A, 00H       ; Puerto A como salida
     CALL CONF_PPI    ; Configurar PPI

     MVI A, 01H       ; Valor para activar primer led

REPET:
     OUT 38H          ; Enviar por el puerto A

     PUSH PSW         ; y mostrarlo en la pantalla
     CALL 04D5H
     POP PSW

     MVI C, 05H       ; Y esperar 5 décimas de segundo
     CALL RETARDO

     RLC              ; Rotar el bit hacia la izquierda
     CPI 01H          ; Si no se ha vuelto al principio
     JNZ REPET        ; Repetir la operación

     MVI A, 00H       ; Desactivar todos los leds
     OUT 38H
     CALL 04D5H  ; y enviar el mismo dato a la pantalla

     RST 1       ; Fin del programa

; ----- Subrutina de configuración del PPI,
;       siempre en modo 0
;
; Entradas:
;     A -> Puerto A: 0-Salida, 1-Entrada
;     B -> Puerto B: 0-Saluda, 1-Entrada
;
CONF_PPI:

     PUSH PSW    ; Guardar A temporalmente

     MOV A, B    ; Para llevar la configuración
     RLC         ; de B un bit a la izquierda
     ANI 02H     ; Resto de bits a 0
     MOV B, A    ; De vuelta a B

     POP PSW

     RLC         ; La configuración
     RLC         ; del puerto A
     RLC         ; en el nibble alto
```

```
     RLC
     ANI 010H    ; Resto de bits a 0

     ORA B       ; Agregar puerto B, en el nibble bajo

     ORI 80H     ; Activación del bit de configuración

     OUT 3BH     ; Escritura en registro del PPI

     RET

; ----- Subrutina de retardo variable
;
; Entradas: C = Número de décimas de segundo a esperar
RETARDO:
    PUSH B       ; Guardar registros que se modificarán

BRET:
    CALL RET1DEC; Una décima

    DCR C        ; Hasta completar el número indicado
    JNZ BRET

    POP B
    RET
; ----- Fin de la subrutina de retardo

; ----- Subrutina de retardo de una décima de segundo
;
RET1DEC:
    PUSH B ; Guardar registros que se modificarán
    PUSH PSW

    LXI B, 208DH ; Número de repeticiones

BRET1:
    DCX B        ; 6 ciclos
    MOV A, B     ; 4 ciclos
    ORA C        ; 4 ciclos
    JNZ BRET1    ; 10 ciclos

    POP PSW
    POP B

    RET
; ----- Fin de la subrutina de retardo

END
```

8.10.3 Lectura de datos del PPI (microinterruptores)

Ejercicio 8.10.3.1

Escribir un programa que lea del puerto **B** del PPI un dato y lo muestre en el campo de datos del uP-2000, habiendo establecido previamente la configuración adecuada. Conectar a dicho puerto la batería de microinterruptores y ejecutar varias veces el programa modificando la disposición de éstos.

Indicaciones

En este ejercicio va a utilizar el PPI no para enviar un dato al módulo de leds de la tarjeta conectada al uP-2000, sino para leer la posición de cada uno de los microinterruptores de esa misma tarjeta. Lógicamente es necesario efectuar las conexiones adecuadas para que esa comunicación sea posible.

Asumiendo que esas conexiones están hechas, el programa debe dar los pasos siguientes:

1. Configurar el PPI para que permita la escritura a través del puerto `B`.
2. Leer un byte de dicho puerto.
3. Enviar el dato leído al campo de datos del uP-2000.

Cuestiones

1. ¿Cómo interpreta el dato que aparece en la pantalla del uP-2000 cuando ejecuta el programa? ¿Cuál es la influencia de cada microinterruptor en dicho dato?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     MVI B, 01H       ; Puerto B como entrada
     CALL CONF_PPI    ; Configurar PPI

     IN 39H           ; Leer el puerto B

     CALL 04D5H       ; Mostrar dato en pantalla

     RST 1            ; Fin del programa

; ----- Subrutina de configuración del PPI,
;       siempre en modo 0
;
; Entradas:
;     A -> Puerto A: 0-Salida, 1-Entrada
;     B -> Puerto B: 0-Saluda, 1-Entrada
;
CONF_PPI:

     PUSH PSW    ; Guardar A temporalmente

     MOV A, B    ; Para llevar la configuración
     RLC         ; de B un bit a la izquierda
     ANI 02H     ; Resto de bits a 0
     MOV B, A    ; De vuelta a B

     POP PSW

     RLC         ; La configuración
     RLC         ; del puerto A
     RLC         ; en el nibble alto
     RLC
     ANI 010H    ; Resto de bits a 0

     ORA B       ; Agregar puerto B, en el nibble bajo

     ORI 80H     ; Activación del bit de configuración

     OUT 3BH     ; Escritura en registro del PPI

     RET

END
```

Ejercicio 8.10.3.2

Introducir en el programa del ejercicio 8.10.3.1 los cambios necesarios para que el dato leído del puerto **B** del PPI, además de mostrarse en pantalla, sea enviado por el puerto **A** para iluminar los leds, repitiéndose la operación hasta que todos los microinterruptores estén puestos en la posición 0.

Indicaciones

Para comenzar, es preciso asegurar la correcta conexión entre el uP-2000 y el módulo externo de microinterruptores y leds, comprobando que el puerto `A` está conectado a los leds y el puerto `B` a los microinterruptores. Partiendo de este supuesto, el programa debe hacer lo siguiente:

1. Inicializar apropiadamente el PPI para poder leer por el puerto `B` y escribir por el puerto `A`.

2. Leer el estado de los microinterruptores del puerto `B`.

3. Enviar el dato que se ha obtenido tanto al puerto `A` como al campo de datos del uP-2000.

4. Si el dato obtenido indica que todos los microinterruptores están a 0 terminar, en caso contrario volver al paso 2.

Cuestiones

1. A la vista de cómo funciona este programa, ¿puede deducirse que la ejecución de la instrucción que lee del puerto `B` provoca una espera hasta que se cambia el estado de un microinterruptor?

2. ¿Qué modificación habría que introducir en el programa para que los leds apareciesen en estado invertido al de los microinterruptores, es decir, apagados los que corresponden a microinterruptores a 1 y viceversa?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     MVI B, 01H       ; Puerto B como entrada
     MVI A, 00H       ; Puerto A como salida
     CALL CONF_PPI    ; Configurar PPI

BUCLE:
     IN 39H           ; Leer el puerto B
     OUT 38H          ; Escribir dato en el puerto A

     PUSH PSW
     CALL 04D5H       ; Y mostrarlo en pantalla
     POP PSW

     CPI 00H          ; Hasta que se desactiven todos
     JNZ BUCLE        ; los microinterruptores

     RST 1            ; Fin del programa

; ----- Subrutina de configuración del PPI,
;       siempre en modo 0
;
; Entradas:
;     A -> Puerto A: 0-Salida, 1-Entrada
;     B -> Puerto B: 0-Saluda, 1-Entrada
;
CONF_PPI:

     PUSH PSW    ; Guardar A temporalmente

     MOV A, B    ; Para llevar la configuración
     RLC         ; de B un bit a la izquierda
     ANI 02H     ; Resto de bits a 0
     MOV B, A    ; De vuelta a B

     POP PSW

     RLC         ; La configuración
     RLC         ; del puerto A
     RLC         ; en el nibble alto
     RLC
     ANI 010H    ; Resto de bits a 0

     ORA B       ; Agregar puerto B, en el nibble bajo

     ORI 80H     ; Activación del bit de configuración

     OUT 3BH     ; Escritura en registro del PPI
```

```
        RET

END
```

8.10.4 Ejercicios propuestos

Ejercicio 8.10.4.1

Escribir un programa que vaya leyendo pulsaciones del teclado del uP-2000 y envíe los códigos por el puerto **A** del PPI, al que se habrá conectado el módulo de leds. Finalizar cuando se pulse cualquiera de las teclas de función (no dígitos hexadecimales).

Indicaciones

El objetivo de este ejercicio es combinar en un programa la lectura del teclado del uP-2000, a través de la correspondiente subrutina de la EPROM, con el envío de datos hacia el PPI. Son tareas ambas que se han llevado a cabo, por separado, en programas de ejercicios previos, por lo que su desarrollo no debe presentar ninguna dificultad.

Cuestiones

1. En el ejercicio 8.10.3.2 el bucle principal del programa se repetía continuamente. ¿Ocurre lo mismo en el programa correspondiente al ejercicio actual?

Ejercicio 8.10.4.2

Se quiere mostrar en la pantalla del uP-2000, concretamente en el campo de direcciones, un contador que pueda ser controlado mediante microinterruptores externos. Para ello se conectará el módulo externo al puerto **B** del PPI, utilizándose el microinterruptor situado más a la izquierda para pausar/reanudar el contador, y el que está justo a su derecha para invertir el sentido del contador.

Indicaciones

Los pasos que han de reproducirse para completar este ejercicio son:

1. Borrar la pantalla del uP-2000, inicializar apropiadamente el PPI y asignar un valor inicial al contador que va a mostrarse.

2. Mostrar el contador en la pantalla del uP-2000.

3. Comprobar el estado del microinterruptor del extremo izquierdo. Si está a 0 no continuar hasta que se ponga a 1.

4. Comprobar el estado del microinterruptor situado a la derecha del anterior, de forma que si está a 1 el contador se incremente y en caso contrario se decremente.

5. Volver al paso 2.

Ejercicio 8.10.4.3

Ampliar el programa propuesto en el ejercicio 8.10.4.2 de forma que se utilicen los tres microinterruptores del extremo derecho, que representan los bits de menor peso, para introducir a cada ciclo del contador un retardo de entre 1 y 8 décimas de segundo.

Indicaciones

En la secuencia de pasos indicada para el ejercicio previo habría que intercalar uno adicional, entre los pasos 2 y 3, que consistiría en los siguientes subpasos:

1. Leer el puerto `B` del PPI para obtener el estado en que se encuentran los microinterruptores.

2. Poner a 0 los cinco bits de mayor peso para quedarse únicamente con el valor de los tres últimos.

3. Puesto que el estado `000` va a asumirse como una décima de segundo y el valor `111` como 8, será preciso incrementar el valor en una unidad.

4. Finalmente se invoca a la rutina de retardo indicando el número de milisegundos.

El resto del programa seguirá funcionando igual. Comprobar que el retardo puede ajustarse sin necesidad de detener el contador, basta con modificar la posición de los microinterruptores para ir ajustando la velocidad.

8.11 Interrupciones

El 8085 cuenta con un mecanismo de interrupciones que permite a dispositivos externos, como puede ser el teclado, interrumpir temporalmente la ejecución de un programa para permitir que el sistema atienda solicitudes externas. Las interrupciones son atendidas por porciones de códigos, por regla general muy breves, denominadas *rutinas de servicio de interrupción* o ISR.

El objetivo del único ejercicio de esta sección es demostrar cómo puede escribir una ISR que intervenga en la ejecución de un programa, descubriendo los problemas que puede plantear la interrupción de un programa en curso y cómo superarlos.

8.11.1 Rutinas de atención a interrupciones

Ejercicio 8.11.1.1

Se quiere mostrar en el campo de direcciones del uP−2000 un contador que se incremente constantemente, teniendo la posibilidad de pausarlo temporalmente mediante el teclado.

Indicaciones

La subrutina que facilita la lectura del teclado del uP-2000, empleada en diversos ejercicios de secciones previas, tiene el inconveniente de que detiene la ejecución del programa que la invoca. No es una función válida, por tanto, cuando lo que se quiere es tener un proceso en funcionamiento continuo (en este caso el contador) que pueda ser interrumpido puntualmente.

En el sistema uP-2000 la mayor parte de las teclas generan una interrupción tipo `RST 5.5`, atendida por el programa monitor alojado en EPROM. La tecla **`INTR VECT`**, sin embargo, está conectada a la señal correspondiente a la `RST 7.5`, con mayor prioridad que la anterior. Lo interesante es que en el uP-2000 esa interrupción, la `RST 7.5`, está pensada para que la utilice el usuario del sistema, siempre que sepa cómo escribir su ISR (*Interrupt Service Routine*) o Rutina de servicio de interrupción.

El objetivo de este ejercicio es que se combinen en un mismo código un programa con una finalidad concreta: la de mostrar un contador que cambia continuamente, y una ISR capaz de interrumpir ese programa de manera puntual. Para ello puede dividirse el trabajo en los siguientes pasos:

1. Obtener la dirección a la que transfiere el control la pulsación de la tecla **`INTR VECT`**, leyendo para ello el correspondiente vector de interrupción.

2. Introducir en la dirección obtenida en el paso 1 una instrucción de salto que transfiera el control a un punto del programa que va a controlar el contador.

3. El código del programa se dividirá en dos partes bien diferenciadas: la encargada de ir mostrando y actualizando el contador, en un bucle infinito, y la rutina de atención a la interrupción. Ésta puede detener la ejecución del programa con la instrucción `HLT` si previamente activa las interrupciones, con la instrucción `EI`, de forma que la pulsación de cualquier tecla reanudará la ejecución.

Un aspecto a tener en cuenta es que aparte de desenmascarar las interrupciones `RST 7.5` y `RST 5.5`, la primera para permitir que la pulsación de **`INTR VECT`** interrumpa el

programa y la segunda para que cualquier tecla facilite la salida de la instrucción `HLT`, este programa debe habilitar explícitamente las interrupciones allí donde sea necesario, usando para ello la instrucción `EI`. En los programas de ejercicios previos que han leído datos del teclado esto no ha sido necesario, puesto que la subrutina a la que se llama para obtener una pulsación de tecla se ocupa de activar las interrupciones.

Cuestiones

1. ¿Cuál es la dirección que corresponde al vector de la interrupción `RST 7.5`? ¿Cuál es su contenido?

2. ¿Cómo puede introducirse una dirección de salto a una ISR propia en la dirección concreta a la que se transfiere el control desde el vector de interrupción?

3. Ejecutar el programa, pulsar la tecla **`INTR VECT`** y a continuación cualquier otra tecla. El contador se detiene temporalmente y la pulsación de otra tecla lo reanuda. Si se repite la operación, sin embargo, ya no es posible detener el contador. ¿Cuál es la razón? ¿Cómo podría solucionarse este problema?

4. Ejecutar el programa, pulsar cualquier tecla y a continuación **`INTR VECT`**. ¿Qué ocurre? ¿A qué se debe este comportamiento?

5. Tras introducir en el programa los cambios que permitan pausar y reanudar el contador tantas veces como se quiera, se observará que al detener y reactivar el contador en ocasiones su valor cambia arbitrariamente. ¿A qué se debe que el contenido de la pareja `HL` se pierda y tome valores arbitrarios? ¿Qué cambios habría que introducir para solucionar este problema?

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     MVI A, 08H ; Desenmascarar RST 5.5 y RST 7.5
     SIM
     EI               ; Activar interrupciones

     LXI H, 0000H     ; Valor inicial del contador

BUCLE:
     PUSH H           ; Se guarda el contador
     CALL 04C9H       ; para mostrarlo
     POP H

     INX H            ; Incrementarlo
     JMP BUCLE        ; y continuar

ISR: ; Rutina de atención a la RST 7.5

     EI               ; Activar interrupciones
     HLT              ; Y tener el programa

     JMP BUCLE        ; Volver al bucle

; Las instrucciones siguientes se ejecutarán
; al generarse una RST 7.5
ORG 20CEH

     JMP ISR          ; Saltar a la ISR

END
```

9 Soluciones a las cuestiones y ejercicios de la sección *6. Familiarizarse con el entorno*

9.1 Acceso a la memoria del uP-2000

Ejercicio 6.1.1

Respuestas a las cuestiones

1. La tecla **S.ME ANT**. Mientras se examina la memoria, las teclas **S.ME ANT** y **POST** permiten reducir e incrementar, respectivamente, la dirección de memoria actual en una unidad.

2. Puede reiniciarse el uP-2000, mediante la tecla **INIC**, o bien cancelarse el comando que está utilizándose en ese momento, usando la tecla **EJEC**. En este caso no aparecerá la indicación **- 8085** pero el sistema quedará a la espera de un nuevo comando.

Ejercicio 6.1.2

Respuestas a las cuestiones

1. Utilizando el sistema de numeración hexadecimal, que es el empleado por defecto en el uP-2000, dos dígitos permiten usar el rango de valores `00H` a `FFH` que, en decimal, sería de 0 a 255. Éstos son los 256 valores posibles que pueden almacenarse en una celdilla de memoria, ya que cada una de ellas tiene una capacidad de 8 bits.

2. La tecla **INIC** está cableada con la patilla **RESET** del 8085, por lo que al pulsarse reinicia el microprocesador. La memoria, sin embargo, permanece inalterada ya que no se ha perdido la alimentación eléctrica. Si tras completar el ejercicio se desconecta el uP-2000 un par de segundos y se vuelve a conectar, lo más probable es que el contenido de las celdillas de memoria indicadas se haya perdido.

Ejercicio 6.1.3

Respuestas a las cuestiones

1. El rango de direcciones de memoria **0000H** a **0FFFH** se encuentra en el uP-2000 ocupada por una memoria EPROM, no por memoria RAM, por ello no es posible escribir nada en las posiciones indicadas.

Ejercicio 6.1.1.1

Solución

Siguiendo el diagrama de flujo que se facilita, se obtendrían los pasos siguientes como resultado de la resolución del ejercicio:

1. Se pulsa la tecla **INIC** para iniciar el uP-2000.
2. Se pulsa la tecla **S.ME ANT** y se introduce la dirección **0000H**.
3. Se pulsa la tecla **POST**.
4. Lectura en campo de datos del código de operación: **20H**.
5. Búsqueda en la referencia de instrucciones del código **20H** para determinar que corresponde a la instrucción **RIM**, anotándolo aparte. La instrucción no tiene operandos.
6. Se pulsa la **POST** para avanzar a la dirección siguiente.
7. Lectura del campo de datos del código de operación: **07H**.
8. La instrucción correspondiente es **RLC** y no tiene operandos.
9. Se avanza a la dirección siguiente para leer el código de operación **DA**.
10. La instrucción es **JC** y según la referencia de instrucciones del 8085 tiene un operando de 16 bits, por lo que hay que pulsar **POST** dos veces para leer y anotar el contenido de las dos celdillas siguientes de memoria: **05F5H**.
11. Se repiten los pasos 9 y 10, obteniendo la instrucción **JMP** y su operando de 16 bits: **0339H**.
12. Ya se tienen cuatro instrucciones, por tanto se da por terminado el proceso.

    Al finalizar el ejercicio se debe haber anotado lo siguiente:

```
RIM
RLC
JC 05F5H
JMP 0339H
```

Respuestas a las cuestiones

1. La instrucción **RIM** lee el byte de estado que contiene la máscara de interrupciones, cuyo bit de mayor peso contiene el valor de la patilla de entrada **SID** del 8085, y lo deja en el acumulador. La instrucción **RLC** rota a la izquierda el contenido del acumulador, copiando además el bit de mayor peso en el indicador de acarreo. La instrucción **JC 05F5H** salta a la dirección indicada si el acarreo está activo, es decir, si el bit leído de la patilla **SID** del 8085 era 1. Por último, la instrucción **JMP 0339H** solamente será ejecutada si no se ha ejecutado la anterior, cuando el bit indicado sea 0, desviando la ejecución a la dirección **0339H**.

2. La patilla **SID** del 8085 se encuentra en el uP-2000 conectada de tal forma que se recibe la posición del microinterruptor que establece el modo de uso: teclado o conexión a PC. Lo que se hace durante la inicialización, por tanto, es seguir un camino u otro dependiendo de cuál sea la posición de ese microinterrruptor, activando el teclado o bien preparando la comunicación serie con el PC.

9.2 Acceso a los registros del 8085

Ejercicio 6.2.1

Respuestas a las cuestiones

1. Si se examinan los registros justo después de inicializar el sistema con la tecla **INIC**, los valores obtenidos son, efectivamente, siempre los mismos. La razón es que al pulsar dicha tecla se ejecuta siempre el mismo código de inicialización, que finaliza mostrando el mensaje **- 8085** en la pantalla y quedando a la espera de una acción en el teclado, por lo que los registros siempre quedan en el mismo estado.

2. Cuando se conecta el uP-2000 no se ejecuta el código de inicialización de la tecla **INIC**, el proceso de puesta en marcha sigue otra ruta, de ahí que los valores de los registros se diferente.

Ejercicio 6.2.2

Respuestas a las cuestiones

1. El valor obtenido es la dirección del tope de pila, situado en **20E3H**.

2. La dirección **20E3H** se encuentra en el rango **2000H-27FFH**, asignado al integrado 8155 situado en la parte superior derecha del uP-2000. Dicho integrado incorpora varias funciones y cuenta con 256 bytes de memoria RAM que ocupan las posiciones **2000H-20FFH**, por lo que la dirección **20E3H** apunta a esa zona de RAM.

Ejercicio 6.2.3

Respuestas a las cuestiones

1. El valor del acumulador no se pierde ante operaciones básicas, como la lectura de registros o de posiciones de memoria, ya que el uP-2000 preserva el estado de ese registro mientras se trabaja con él.

2. Al pulsar la tecla **INIC** se ejecuta un código de inicialización que altera el contenido del acumulador, entre otros registros, por lo que se perderá el valor asignado.

Ejercicio 6.2.1.1

Solución

Para completar este ejercicio reproduzca los pasos siguientes:

1. Pulsar la tecla **INIC** para inicializar el sistema.

2. Pulsar la tecla **E.REG** para explorar los registros del procesador.

3. Pulsar la tecla **F** para seleccionar el registro de indicadores.

4. Leer del campo de datos el contenido del registro.

5. Convertir el valor obtenido a binario para identificar el estado de cada bit.

Respuestas a las cuestiones

1. El valor del registro de indicadores tras la inicialización es **A0H**. Al convertir este valor a binario se obtiene la secuencia **1010 0000**, por lo que el único indicador activo es el de signo (**S**), correspondiendo el otro bit a 1 a una posición sin significado establecido.

2. Usando el teclado del uP-2000 solamente pueden introducirse bytes completos, ya sea en los registros o en posiciones de memoria, por lo que si se quiere modificar un bit hay que descomponer la operación en las siguientes partes:

    a. Leer el valor actual que contiene el registro.

    b. Calcular cuál sería el nuevo valor tras modificar el bit deseado.

    c. Introducir el nuevo valor sustituyendo el contenido previo.

    Si partiendo del valor que tiene el registro de indicadores tras la inicialización, que es **A0H**, se quiere activar el indicador **Z**, al que corresponde el sexto bit, el valor resultante sería **1110 0000** que, convertido a hexadecimal, equivale a **E0H**. Éste sería el valor a escribir en el registro de indicadores.

Ejercicio 6.2.1.2

Solución

Para completar este ejercicio reproduzca los pasos siguientes:

1. Comprobar que el uP-2000 está preparado para la introducción de un nuevo comando.

2. Pulsar la tecla **E.REG** y a continuación la tecla **A**. Leer del campo de datos el valor del acumulador.

3. Pulsar la tecla **EJEC** para salir del comando.

4. Pulsar la tecla **S.ME ANT** e introducir la dirección **20EEH**.

5. Pulsar la tecla **POST** y leer en el campo de datos el valor contenido en esa posición de memoria. Es el mismo que contiene el acumulador.

6. Introducir un nuevo valor en esa posición de memoria mediante el teclado numérico, pulsando la tecla **POST** para confirmarlo.

7. Pulsar la tecla **EJEC** para salir del comando.

8. Pulsar la tecla **E.REG** y a continuación la tecla **A**. Leer del campo de datos el valor del acumulador. Es el mismo que se había introducido en la dirección 20EEH.

9. Introducir un nuevo valor en el acumulador mediante el teclado numérico, pulsando la tecla **POST** para confirmarlo.

10. Pulsar la tecla **EJEC** para salir del comando.

11. Pulsar la tecla **S.ME ANT** e introducir la dirección **20EEH**.

12. Pulsar la tecla **POST** y leer en el campo de datos el valor contenido en esa posición de memoria. Es el mismo que acaba de introducirse en el acumulador.

Respuestas a las cuestiones

1. El contenido del acumulador y de la celdilla de memoria **20EEH** siempre coinciden. En esa posición de memoria es donde el programa monitor del uP-2000, que permite operar con el sistema a través del teclado, guarda el último valor del acumulador.

2. De igual forma el programa monitor del uP-2000 conserva el contenido del registro de indicadores en la dirección de memoria **20EDH**, por lo que si modifica el valor de ese registro aparecerá también en la celdilla indicada y viceversa.

9.3 Ejecución de programas

Ejercicio 6.3.1

Respuestas a las cuestiones

1. En la pantalla aparece el mensaje **– 8085** y el valor de ciertos registros, como el procesador o el puntero de pila, se modifican, lo mismo que ocurre al pulsar la tecla **INIC**.

Ejercicio 6.3.2

Respuestas a las cuestiones

1. La instrucción **RST** provoca el salto a un vector de interrupción cuya dirección se obtiene multiplicando por 8 el número dispuesto tras la instrucción. La ejecución de **RST 1**, por tanto, provoca que el puntero de programa apunte a la dirección **0008H** y se ejecute el código situado a partir de esa posición, cuya finalidad es devolver el control al programa monitor del uP-2000.

   Si se eliminase la instrucción **RST 1** hay que hacerse la siguiente pregunta: ¿cómo sabrá el 8085 que ha llegado al final del programa, tras mover el valor indicado al acumulador? Obviamente no tiene forma de saberlo, porque un programa puede tener en principio cualquier longitud (dentro de los límites de direccionamiento). En consecuencia el procesador seguiría interpretando el contenido de las celdillas de memoria sucesivas como instrucciones e intentaría ejecutarlas, deteniéndose normalmente con un error o entrando en un bucle infinito.

2. Las instrucciones del 8085 se traducen siempre por el mismo código de operación, sin que importe cuál sea la posición de memoria en que se almacenen. Los datos que acompañan a algunas instrucciones, sin embargo, pueden hacer referencia a direcciones que son relativas a la posición en que comienza el programa, en cuyo caso la relocalización de éste forzaría también el recálculo de esas direcciones. En este ejercicio el programa se compone de dos instrucciones simples, sin referencias directas a direcciones, por lo que no habría ningún problema en trasladarlo a cualquier otro punto de la memoria del sistema.

Ejercicio 6.3.1.1

Solución

Los pasos para completar este ejercicio son los indicados a continuación:

1. Comprobar que el uP-2000 está preparado para introducir un nuevo comando.

2. Pulsar la tecla **GO**. En el campo de direcciones aparece el actual valor del puntero de programa.

3. Introducir la dirección **05F5H**.

4. Pulsar la tecla **EJEC** para ejecutar a partir de esa dirección.

Respuestas a las cuestiones

1. La dirección a usar es la que sigue al salto condicional **JC**, que provoca la ejecución del código que se encuentra en **05F5H** si el microinterruptor **TEC** da el control al PC.

2. En la pantalla del uP-2000 aparece el mensaje **8085**, sin el guión en el extremo izquierdo que indica que puede utilizarse el teclado. Además el teclado no parece operativo, el sistema no acepta ningún comando ni la introducción de datos o direcciones. Únicamente pulsando la tecla **INIC**, reiniciando de nuevo el sistema, se retomará el control.

Ejercicio 6.3.1.2

Solución

Para completar este ejercicio los pasos a seguir son los indicados a continuación:

1. Escribir el programa que, en este caso, se compondría de las tres instrucciones siguientes:

```
MOV C,A
INR C
RST 1
```

2. Buscar en la hoja de instrucciones del 8085 cada una de las instrucciones para hallar sus códigos de operación: **4FH 0CH CFH**. Ésta es la secuencia de bytes que componen el programa.

3. Pulsar la tecla **INIC** para inicializar el uP-2000.

4. Pulsar la tecla **S.ME ANT** e introducir la dirección **1000H**, terminando con la tecla **POST**.

5. Ir introduciendo los códigos de operación confirmando cada uno con la tecla **POST**.

6. Pulsar la tecla **EJEC** para salir de la función. El programa está preparado ya en memoria.

7. Pulsar la tecla **E.REG** y a continuación la tecla **A** para modificar el contenido del acumulador, introduciendo el valor que se desee.

8. Pulsar la tecla **GO** e introducir la dirección **1000H**, lanzando la ejecución con la tecla **EJEC**.

9. Pulsar la tecla **E.REG** y comprobar el contenido del acumulador, el registro **C** y el registro de estado.

10. Repetir los pasos 6 a 9 para probar con distintos valores.

Respuestas a las cuestiones

1. Tras asignar el valor **1FH** al acumulador y ejecutar el programa el registro de estado contiene el valor **10H**, por lo que se sabe que está activo el indicador **AC** o acarreo auxiliar. Éste indica que se ha producido un arrastre desde los cuatro bits de menor peso (que contenían el valor **FH** y ahora son **0**) a los cuatro bits de mayor peso. Puede verificarse comprobando el valor del registro **C**, que será **20H**.

2. Los registros del 8085 son de 8 bits, por lo que no pueden contener un valor superior a **FFH** en hexadecimal. Al asignar este valor al acumulador y ejecutar el programa, llevando ese dato al registro **C** e incrementándolo, se produce un desbordamiento. El registro **C** contendrá tras la ejecución el valor 0, mientras que el registro de estado contiene el valor **54H**. Los indicadores activos son **Z** (bit de cero), AC (acarreo auxiliar) y **P** (bit de paridad).

   El programa podría comprobar si se ha producido un desbordamiento, por tanto, comprobando el bit **Z** del registro de estado, con una instrucción del tipo **JZ** o **JNZ**.

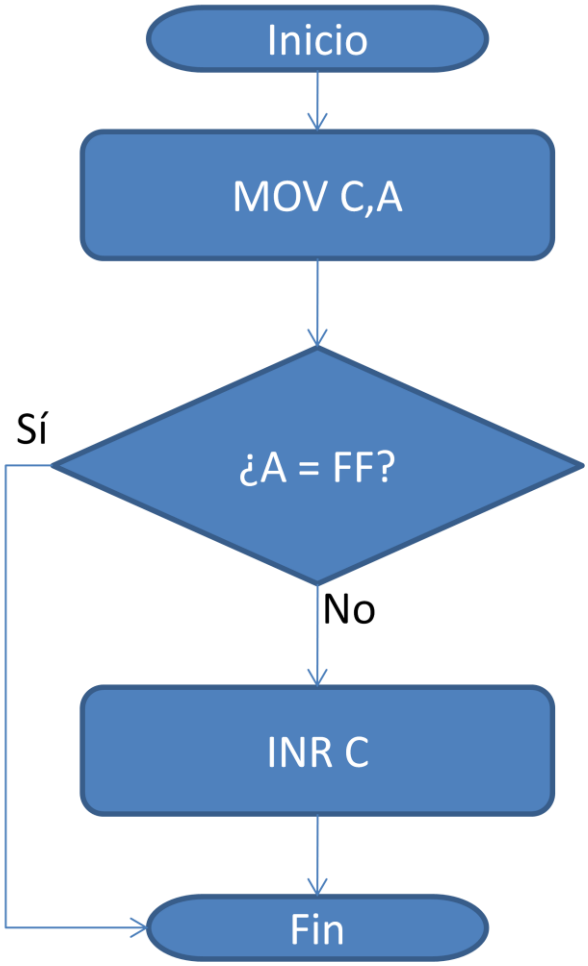
3. Siempre que las direcciones de memoria en las que se alojan no se solapen, es posible alojar en la memoria del uP-2000 tantos programas como se deseen. Se podría, por ejemplo, situar el código del ejercicio anterior a partir de la di-

rección **1000H** y la de éste a partir de la dirección **1100H**, pudiendo ejecutar uno u otro según se precise.

Ejercicio 6.3.1.3

Solución

Para completar este ejercicio, algo más complejo que los previos, se parte del siguiente diagrama de flujo que representa los pasos a dar:



Ordinograma del ejercicio 6.3.1.3.

1. Tomando como referencia el diagrama anterior se escribe el programa en lenguaje ensamblador 8085, en este caso disponiendo a la izquierda de cada instrucción la dirección de memoria que le correspondería:

```
1000H MOV C,A
1001H CPI 0FFH
1003H JZ 1007H
1006H INR C
1007H RST 1
```

2. Usando la referencia de instrucciones del 8085 se traduce el programa a código máquina. La secuencie de bytes sería la siguiente:

```
1000  MOV C,A          4FH
1001  CPI 0FFH         FEH FFH
1003  JZ 1007H         CAH 07H 10H
1006  INR C            0CH
1007  RST 1            CFH
```

3. Se introduce el programa en la memoria del uP-2000 siguiendo el mismo procedimiento de los dos ejercicios previos.

4. Con el programa ya en memoria, se modifica el contenido del acumulador asignando un valor y se ejecuta el programa.

5. Asignar al acumulador el valor **FFH**, ejecutar el programa y verificar que el valor del registro **C** no ha cambiado.

Respuestas a las cuestiones

1. A diferencia de los programas de ejercicios anteriores, el obtenido en éste no puede copiarse sin más a cualquier posición de memoria ya que contiene una instrucción de salto cuya dirección de destino es relativa al inicio del programa. Lo que interesa es que la instrucción **JZ** transfiera el control a la instrucción **RST 1**, saltándose la instrucción de incremento. La posición en la que se encontrará esa instrucción es **1007H** en la solución dada, pero si se alojase el programa a partir de la dirección de memoria **1500H** la instrucción se encontraría en la dirección **1507H**. Debería modificarse, por tanto, la línea **JZ 1007H** por **JZ 1507H**.

10 Soluciones a las cuestiones y ejercicios en la sección *8. Ejercicios de programación*

10.1 Acceso a la memoria

Ejercicio 8.1.1.1

Respuestas a las cuestiones

1. Según el mapa de memoria del uP-2000 (véase la figura 33), el rango de direcciones `2000H-27FFH`, al que pertenece la dirección `20F1H`, corresponde al integrado 8155 que, además de un reloj y unas puertas de E/S, también cuenta con una zona de memoria RAM de 256 bytes de tamaño.

2. Las demás instrucciones del 8085 que permiten recuperar un dato de la memoria y llevarlo al acumulador, como `LDAX` o `MOV A,M`, implican haber cargado primero en una pareja de registros la dirección de la que va a leerse, modificándolos. Para escribir un programa que hiciese exactamente lo mismo, por tanto, sería necesario guardar antes el contenido de esa pareja de registros, recuperándolo al final.

3. El modo de direccionamiento que utiliza la instrucción `LDA` es por registro directo (véase la sección *2.2. Modos de direccionamiento*).

4. La posición de memoria `20F1H` la utiliza el programa monitor del sistema uP-2000 para guardar una copia del registro de interrupciones del 8085, por lo que su valor puede variar según el estado del sistema. Tras haberlo reiniciado, no obstante, el valor usual es `07H`.

Ejercicio 8.1.1.2

Respuestas a las cuestiones

1. La instrucción `LDAX` lee el contenido de la posición de memoria cuya dirección se encuentra almacenada en una pareja de registros, y lo lleva hasta el acumulador, pareja que puede ser `BC` o `DE`. La instrucción `LXI` copia un dato inmediato de 16 bits en una pareja de registros que, en este caso, puede ser `BC`, `DE`, `HL` o `SP`.

2. Los registros del 8085 son de 8 bits, si bien pueden utilizarse en parejas para tratar datos de 16 bits. Todas las instrucciones del 8085 que operan sobre direcciones, entre ellas `LXI`, toman el MSB (*Most Significant Byte*) y lo llevan al primer registro del par, copiando el LSB (*Least Significant Byte*) en el segundo. La ejecución de la instrucción `LXI B, 20F1H`, por tanto, almacenaría el valor `20H` en el registro `B` y el valor `F1H` en el registro `C`.

3. Sabiendo qué hace la instrucción `LXI`, y la forma en que divide el dato de 16 bits para almacenarlo como dos porciones de 8 bits en dos registros, es fácil sustituir dicha instrucción por dos instrucciones `MVI`: `MVI B, 20H` y `MVI C, 0F1H` tendrían el mismo resultado que `LXI B, 20F1H`.

Ejercicio 8.1.2.1

Respuestas a las cuestiones

1. En el sistema uP-2000 el rango de direcciones `1000H-1FFFH` está ocupado por memoria RAM de usuario (el programa monitor no utiliza directamente esa zona de RAM), en la que pueden almacenarse tanto programas como datos.

2. Dicho rango de posiciones de memoria se encuentra ocupado por el programa monitor del uP-2000, grabado en EPROM, por lo que la escritura de datos no tendrá ningún efecto. La ejecución de una instrucción del tipo `STA 0100H` no generaría un error, de hecho el 8085 enviaría por los buses que le comunican con la memoria las señales para efectuar la operación, pero el integrado de memoria EPROM la ignoraría.

3. Cuando se ejecuta la instrucción `STA` el 8085 coloca en el bus de datos el contenido del acumulador, pero el valor de dicho registro no se ve alterado en ningún momento. Por ello en ocasiones se dice que los que se escribe es una *copia* del contenido del acumulador, aunque en realidad ambos datos, el contenido en el acumulador y el almacenado en la memoria, son secuencias de 8 bits idénticas e indistinguibles, por lo que no puede ser considerado uno copia del otro.

Ejercicio 8.1.2.2

Respuestas a las cuestiones

1. Es obvio que el procedimiento no resultará práctico en cuanto crezca un poco el número de posiciones de memoria a copiar. Para transferir 100 bytes de memoria, por ejemplo, habría que escribir un programa de más de 200 líneas y que ocuparía más de 600 bytes. Usando un bucle el proceso se reduciría a poco más de media docena de instrucciones y una docena de bytes de memoria de ocupación.

2. Dado que las direcciones de memoria a leer y escribir son datos inmediatos, que forman parte de las instrucciones del programa, la generalización del éste resultaría bastante compleja, más no imposible.

Ejercicio 8.1.3.1

Respuestas a las cuestiones

1. La instrucción `LHLD` (véase la tabla de instrucciones) va seguida de una dirección de memoria y, al ejecutarse, lee el contenido de dicha dirección y lo lleva al registro `L`, a continuación lee la posición siguiente y el valor lo asigna a `H`. El byte de mayor peso, que es el contenido en `H`, va detrás del byte de menor peso, por lo que se deduce que el formato empleado es *little endian*, como la mayoría de los microprocesadores diseñados por Intel.

2. Puesto que el programa utiliza el dato leído de `1100H` y `1101H` como una dirección, para recuperar lo que haya en ella, el resultado final sería que en el acumulador se tendría el dato contenido en la dirección `0000H` ya que, por defecto, tras iniciar el sistema uP-2000, las posiciones `1100H` y `1101H` suelen contener ambas el valor 0.

Ejercicio 8.1.3.2

Respuestas a las cuestiones

1. Cuando se opera con parejas de registros, en general con datos de 16 bits de tamaño, siempre puede recurrirse al uso de la pila como espacio de almacenamiento intermedio. Para intercambiar los valores de `DE` y `HL` bastarían las siguientes cuatro instrucciones:

```
PUSH HL     ; Guardar en la pila HL
PUSH DE     ; y DE
POP HL      ; recuperándolos en orden inverso
POP DE
```

2. Esta alternativa a la instrucción `XCHG` no puede utilizarse sin más, en un programa como el de este ejercicio en el que se no se ha inicializado el puntero de pila, ya que éste se encontrará apuntando normalmente a la zona de memoria usada por el programa monitor del sistema uP-2000. La ejecución de las instrucciones `PUSH`, en consecuencia, estarían escribiendo el contenido de los registros en posiciones de memoria sobre las que no tentemos control alguno, al contrario, podrían alterar el funcionamiento del citado programa monitor.

Ejercicio 8.1.4.1

Respuestas a las cuestiones

1. A pesar de que estas instrucciones permiten leer o escribir el contenido de una pareja de registros, en este caso HL, el bus de datos del microprocesador 8085 tiene un ancho de 8 bits, lo que implica que no es posible realizar la operación con una única transferencia. Aunque se escriba una única instrucción, su ejecución implica dos transferencias desde el microprocesador hacia la memoria o viceversa, por lo que la pareja de instrucciones LHLD/SHLD empleada en el programa desencadena cuatro transferencias. En este sentido dichas instrucciones no aportan ventaja alguna sobre cualquier otra de transferencia de datos entre procesador y memoria.

Ejercicio 8.1.5.1

Respuestas a las cuestiones

1. El contenido de HL se modifica al inicio del programa, con una instrucción LXI, al igual que el de la pareja DE, no alterándose ya hasta el final. Por tanto el valor contenido en HL será 15C7H y el de DE será 0875H.

2. En las posiciones de memoria 1100H a 1103H estarán almacenados los valores 75H, 08H, C7H y 15H, respectivamente.

3. El valor inicial del puntero de pila, asignado con la instrucción LXI, es 1104H, pero cada una de las dos instrucciones PUSH que hay a continuación restarán 2 a ese valor, por lo que finalmente el puntero de pila contendrá la dirección 1100H.

4. Si se asume que las parejas de registros HL–DE lo que contienen es un dato de 32 bits, almacenando HL la MSW y DE la LSW, lo que hace el programa es escribir ese dato en la memoria en formato *little endian*, usando para ello el puntero de pila y las instrucción PUSH. Es equivalente, por tanto, al programa propuesto en el ejercicio previo.

Localice en la Tabla 3, en la que se resumen todas las instrucciones del 8085, la instrucción PUSH y examine su descripción para comprender porqué en este programa se asigna al registro SP la dirección 1104H, y no la 1100H o la 1103H siendo éste el rango de posiciones donde quieren almacenarse los datos.

Ejercicio 8.1.6.1

Solución

```
CPU "8085.TBL"
HOF "INT8"

ORG 1000H

LXI H, 20F1H     ; Cargar la dirección en HL
MOV A,M          ; Llevar al acumulador el dato
           ; almacenado en esa posición
RST 1

END
```

Respuestas a las cuestiones

1. Una de las ventajas apreciables se da cuando es necesario leer varias posiciones de memoria consecutivas, algo que con la instrucción `LDA` requeriría escribir las direcciones una a una, por ejemplo:

```
LDA 20F1H
; Tratar el dato del acumulador
LDA 20F2H
; Tratar el dato
LDA 20F3H
...
```

Cada una de las instrucciones `LDA` ocupa tres bytes de memoria: uno correspondiente al código de operación y dos más para la dirección. Usando `HL` como puntero el proceso podría implementarse así:

```
LXI 20F1H
MOV A,M
; Tratar el dato del acumulador e incrementar HL
MOV A,M
; Tratar el dato e incrementar
MOV A,M
...
```

Cada una de las instrucciones `MOV A,M` ocupa en memoria un solo byte, correspondiente al código de operación. Además resulta más cómodo establecer la dirección de inicio e ir incrementando el contenido de un registro, en este caso `HL`, que introducir manualmente las direcciones en cada instrucción.

2. En este programa se ha utilizado el direccionamiento inmediato, en la instrucción `LXI` que introduce en `HL` el dato de 16 bits inmediato `20F1H`, y también el direccionamiento por par de registros indirecto, en la instrucción `MOV A,M`, donde `M` representa la posición de memoria apuntada por `HL`.

Ejercicio 8.1.6.2

Solución

```
CPU "8085.TBL"
HOF "INT8"

ORG 1000H

MVI A, 0CDH     ; Cargar en el acumulador el valor
LXI B, 1100H    ; Preparar dirección en BC
STAX B          ; y escribirlo en la memoria

RST 1

END
```

Respuestas a las cuestiones

1. Las instrucciones `MVI` y `LXI` emplean direccionamiento inmediato para llevar al acumulador y la pareja de registros `BC`, respectivamente, datos que se encuentran en la propia instrucción. `STAX`, como `LDAX`, usan el modo par de registros indirecto.

Ejercicio 8.1.6.3

Solución

```
CPU "8085.TBL"
HOF "INT8"

ORG 1000H

MVI A, 0CDH     ; Cargar en el acumulador el valor
LXI H, 1100H    ; Preparar dirección en BC
MOV M, A        ; y escribirlo en la memoria

RST 1

END
```

Respuestas a las cuestiones

1. La principal diferencia entre las instrucciones `STAX` y `MOV M,r` estriba en que la primera lleva implícita la transferencia del contenido del acumulador a la memoria, mientras que en la segunda se indica explícitamente qué registro es el que va a transferirse. Podría decirse que `STAX` equivale a `MOV M,A`.

2. Bastaría con asignar el dato `0CDH` a un registro distinto, por ejemplo `B`, y cambiar la instrucción `MOV M,A` por `MOV M,B`.

Ejercicio 8.1.6.4

Solución

```
CPU "8085.TBL"
HOF "INT8"

ORG 1000H

LDA 1100H  ; Se recupera en A el LSB
MOV L, A   ; y se lleva a L
LDA 1101H  ; Se recupera al MSB
MOV H, A   ; ya se tiene en HL la dirección

MOV A, M   ; y se lee el dato que contiene

RST 1

END
```

Respuestas a las cuestiones

1. En formato *big endian* el orden de los bytes que componen la dirección estaría invertido, por lo que en el programa habría que leerlos también en orden inverso (cambiar `1100H` por `1101H` en el primer `LDA` y a la inversa en el segundo) o bien seguir leyéndolos igual pero cambiar el registro al que se llevan desde el acumulador, invirtiendo la posición de las instrucciones `MOV L,A` y `MOV H,A`. Hechos estos cambios, para que el programa siga leyendo el mismo dato en el acumulador tendría que cambiarse el contenido de las direcciones `1100H` y `1101H`, guardando la dirección a leer en formato *big endian*.

2. Dado que la instrucción `LHLD` espera que el dato leído de la memoria esté en formato *little endian*, si se encuentra en *big endian* será necesario, tras la lectura, intercambiar los contenidos de `H` y `L`. El programa podría quedar como se muestra a continuación:

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

LHLD 1100H ; Se lee la dirección en HL

MOV A, L   ; Se intercambia H con L
MOV L, H   ; usando el acumulador
MOV H, A   ; como intermediario

MOV A, M   ; y se lee el dato que contiene

RST 1
END
```

Ejercicio 8.1.6.5

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

LHLD 1100H ; Se lee el dato

MOV A, L   ; Se invierten los
MOV L, H   ; registros H y L
MOV H, A

SHLD 1100H ; y se escribe en el nuevo destino

RST 1
END
```

Respuestas a las cuestiones

1. No es necesario cambiar nada. Si se ejecuta el programa habiendo almacenado en la dirección `1100H` un dato de 16 bits en formato *little endian*, el resultado obtenido será el mismo dato en formato *big endian*.

2. Un posible enunciado sería: *Escribir un programa que tome los datos almacenados en las direcciones `1100H` y `1101H` e intercambie su posición en la memoria*. Esto es realmente lo que hace el programa, invertir el contenido de dos posiciones de memoria, si bien esa operación puede interpretarse como la conversión de *little endian* a *big endian*, de *big endian* a *little endian* o cualquier otra tarea que implique esa inversión.

Ejercicio 8.1.6.6

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

LXI H, 15C7H     ; MSW en HL
LXI D, 0875H     ; LSW en DE
; se tiene el dato 15C70875H en HL-DE

SHLD 1102H ; MSW en 1102H-1103H
XCHG       ; intercambiar HL-DE
SHLD 1100H ; LSW en 1100H-1101H

RST 1
END
```

Respuestas a las cuestiones

1. Los registros de uso general del 8085 son A, B, C, D, E, H y L. Los demás registros, como PC, SP o el registro de indicadores, tienen cometidos específicos y no pueden utilizarse libremente. Puesto que tenemos 7 registros disponibles, cada uno de 8 bits, la capacidad total sería de 56 bits, lo que permitiría almacenar un único dato de 32 bits en el microprocesador.

2. Los cuatro bytes que conforman el dato de 32 bits pueden ser interpretados como cuatro dígitos en base 256, de forma que podría aplicarse el algoritmo genérico de conversión de cualquier base a base 10 que, en este caso, se resumiría en la siguiente expresión:

$$256^0 \text{x}\, B_0 + 256^1 \text{x}\, B_1 + 256^2 \text{x}\, B_2 + 256^3 \text{x}\, B_3$$

Los términos B_n son los bytes que componen el dato, correspondiendo el subíndice 0 al de menor peso y el 3 al de mayor peso. Por tanto se tiene que:

B_0=75H=117_{10}
B_1=08H=8_{10}
B_2=C7H=199_{10}
B_3=15H=21_{10}

Finalmente se obtendría:

256^0x117 + 256^1x8 + 256^2x199 + 256^3x21 = 365.365.365

10.2 Implementación de condicionales y bucles

Ejercicio 8.2.1.1

Respuestas a las cuestiones

1. Estas instrucciones llevan a cabo una operación de resta entre el contenido del acumulador y el operando indicado, un registro o un dato inmediato, de forma que los bits del registro de estado se vean afectados como en cualquier otra operación de resta.

2. El bit `Z` del registro de indicadores se activa (se pone a 1) cuando el resultado que genera una cierta operación, y que queda en el acumulador, es 0. Si el contenido del acumulador es el valor `7FH`, al restar ese mismo valor el resultado será 0, de ahí que se active ese indicador.

3. A pesar de que la instrucción de comparación efectúa una resta para determinar la relación entre el acumulador y el operando indicado, el contenido del acumulador no se verá afectado porque es preservado en un registro auxiliar del 8085, recuperándose tras la operación. De esta manera lo único que se ve alterado es el registro de indicadores.

Ejercicio 8.2.1.2

Respuestas a las cuestiones

1. En ambos casos el contenido de la posición de memoria `1101H` se modifica solo si el acumulador tiene el valor `7FH`, por lo que en caso contrario se quedaría con el contenido que tuviese en ese momento. Esto implica que la simple observación de esa posición de memoria no nos servirá para determinar si ha habido o no coincidencia, ya que podría tener el valor `0FFH` con anterioridad a la ejecución del programa.

2. Si acordamos que el valor `0FFH` equivale a `TRUE` y el valor `00H` a `FALSE`, como ocurre en muchos lenguajes de programación, el contenido de la posición de memoria `1101H` nos indicará si ha habido o no coincidencia sin ninguna ambigüedad.

3. Como suele ser habitual, es posible dar distintas soluciones al ejercicio, todas ellas válidas si bien unas serían más óptimas que otras. Una de las posibles alternativas sería la siguiente:

```
CPU "8085.TBL"
HOF "INT8"
     ORG 1000H

     LDA 1100H
     CPI 7FH       ; Se compara A con 7FH
     JZ COINC      ; Si coincide, salta a un punto
     JNZ NOCOINC   ; si no, salta a otro

COINC:
     MVI A, 0FFH ; Corresponde a la parte THEN
     STA 1101H
     RST 1

NOCOINC:
     MVI A, 00H  ; Corresponde a la parte ELSE
     STA 1101H
     RST 1

END
```

El código de este programa es más largo que la solución original, pero a cambio separa claramente la parte que se ejecutará si se cumple la condición y la parte a ejecutar cuando no se cumpla, por lo que resulta más fácil de seguir. En ocasiones puede ser más interesante escribir un código fácilmente legible que obtener el código más eficaz posible.

Ejercicio 8.2.1.3

Respuestas a las cuestiones

1. En un lenguaje de alto nivel la estructura más cercana sería la siguiente:

```
IF A = B THEN
   Resultado = 0
ELSE IF A > B THEN
   Resultado = 1
ELSE
   Resultado = 2
```

Como puede apreciarse, serían necesarias dos comparaciones entre `A` y `B`, mientras que en un programa escrito en ensamblador una única comparación permite tomar a continuación distintos caminos, según el indicador que se haya activado.

2. La instrucción `CMP` efectúa una operación de sustracción, aunque sin modificar finalmente el contenido del acumulador, restando del acumulador el operando que se indique. En caso de que ese operando sea mayor que el valor del acumulador se producirá un desbordamiento: el resultado ob-

tenido sería negativo. Por ello se activa el bit CY del registro de indicadores, para comunicar dicho desbordamiento.

3. Determinar la relación existente entre dos datos, ya sean valores numéricos o de otro tipo, resulta básico para implementar cualquier algoritmo de ordenación.

Ejercicio 8.2.2.1

Respuestas a las cuestiones

1. En el programa propuesto como solución se utiliza la pareja de registros DE como puntero a memoria, de forma que el dato contenido en el acumulador se escribe en la posición que indique dicha pareja de registros. A pesar de que en las instrucciones LXI, STAX e INX aparezca como operando solamente D, en realidad dichas instrucciones utilizan la pareja DE. No es posible, por tanto, usar E como contador del bucle, porque el mismo registro no puede tener al tiempo el número de repeticiones y también el LSB de la dirección donde se escribirá el dato.

2. Si se quiere utilizar E como contador no es posible usar la pareja DE como puntero, por lo que habría que cambiarla por otra pareja: BC.

3. La alternativa sería utilizar la pareja de registros HL como puntero, sustituyendo la instrucción STAX por una del tipo MVI M, 7AH, como se muestra en la siguiente versión del programa. Observe que ya no es necesario tener el valor 7AH en el acumulador para escribirlo en la memoria, porque puede utilizarse en su lugar el valor inmediato.

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H
     LXI H, 1100H  ; Posición de destino
     MVI C, 0H   ; Se usará C como contador

BUCLE:
     MVI M, 7AH ; indicado en esa posición

     INR C       ; Ya se ha escrito un valor más
     INX H       ; Avanzar a la siguiente posición
     MOV A, C    ; Comparar el contador con 10H (16d)
     CPI 10H
     JNZ BUCLE   ; Si aún no se han escrito 16 valores

     RST 1
END
```

Ejercicio 8.2.2.2

Respuestas a las cuestiones

1. Si el valor inicial del contador fuese 0, al llegar a la instrucción `DCR C` se restaría una unidad y su contenido pasaría a ser `0FFH`, de forma que la instrucción `JNZ` encontraría el bit `Z` a 0, es decir, indicando que el resultado que ha quedado en el registro es distinto de 0, por lo que se saltaría a la etiqueta `BUCLE`. Al final el bucle se repetiría 256 veces.

2. La primera diferencia es obvia y basta con examinar la tabla de instrucciones para percibirla: las instrucciones `INR/DCR` operan sobre cualquier registro de 8 bits de uso general: `A`, `B`, `C`, `D`, `E`, `H` o `L`, mientras que `INX/DCX` trabajan sobre parejas de registros: `BC`, `DE`, `HL` o `SP`. La segunda diferencia no resulta tan obvia a primera vista, pero es muy importante. Si se observa la columna `Estado` de la tabla de instrucción se aprecia que `INX/DCX` no afectan a ningún bit del registro de estado. Esto implica que no podría utilizarse la técnica de control de bucles de este ejercicio si el registro que actúa como contador fuese una pareja de registros y se recurriese a la instrucción `DCX` para ir decrementándola.

Ejercicio 8.2.3.1

Respuestas a las cuestiones

1. Puesto que el registro utilizado como contador del bucle (el registro `L`) tiene un tamaño de 8 bits, el número máximo de repeticiones será 2^8, es decir, 256.

2. Básicamente existen dos alternativas: copiar varios bytes por cada ciclo del bucle, incrementando así el tamaño del bloque total, o bien usar como contador del bucle una pareja de registros. En este último caso habría que establecer cómo se determinaría el fin del bucle, ya que al ir decrementando con `DCX` no se ve afectado ningún bit del registro de estado.

Ejercicio 8.2.3.2

Respuestas a las cuestiones

1. El programa lo que hace es copiar un bloque de datos desde una dirección de origen a otra de destino. Al ejecutarse recupera de la memoria el número de bytes a copiar y las direcciones, lleva a cabo el proceso de copia en un bucle y finalmente guarda en memoria las direcciones finales.

2. En las posiciones 1101H–1102H hay que indicar el número de bytes a copiar, en las posiciones 1103H–1104H la dirección de destino de la copia y las posiciones 1105H–1106H la dirección de origen de la copia. En la posición 1100H se puede introducir cualquier valor.

3. Al terminar, el programa escribe en las posiciones 1103H–1104H la dirección posterior a la última en la que se han escrito datos y en las posiciones 1105H–1106H la dirección posterior a la última de la que se ha leído.

4. Ejecutando el programa con esos datos se copiarán 512 bytes, desde el inicio de la EPROM del uP-2000 a la dirección 1400H. Al terminar la ejecución en las posiciones 1103H–1104H se habrá escrito la dirección 1600H y en las posiciones 1105H–1106H la dirección 0200H.

5. La instrucción LDA 1100H que hay al inicio del programa es superflua, ya que recupera en el acumulador un dato que en realidad no se utiliza para nada. Esa es la razón de que sea indiferente el dato que se coloque en esa posición de memoria antes de ejecutar el programa.

6. Como contador se utiliza la pareja de registros BC, que contiene inicialmente el valor recuperado de las posiciones 1101H–1102H. Al utilizarse 16 bits sería posible copiar hasta 64 kilobytes (2^{16}). Al final de cada ciclo reduce el contenido de BC en una unidad, con la instrucción DCX, y se lleva la parte alta del contador, que está en el registro B, al acumulador, comprobándose si es cero. Mientras la parte alta no sea cero aún quedarán ciclos por completar, por lo que se salta al principio del bucle. Si esa parte alta ya es cero (por tanto en el acumulador se tiene el valor 0), falta por saber si la parte baja, en el registro C, también es 0. Por ello se compara el contenido del acumulador con el registro C, si ambos coinciden es que C es 0 y, por tanto, el bucle ha terminado.

7. El código del programa incluyendo comentarios:

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LHLD 1101H  ; Se recupera el número de bytes
     MOV B, H    ; a copiar y se lleva a BC
     MOV C, L

     LHLD 1103H ; Se recupera la dirección de destino
     XCHG       ; y se lleva a DE

     LHLD 1105H  ; Se recupera la dirección de origen
```

```
     ; En este momento se tiene:
     ;
     ; HL -> Dirección de origen
     ; DE -> Dirección de destino
     ; BC -> Contador del bucle
BUCLE:
     XCHG    ; Obtener origen en DE
     LDAX D  ; para leer un byte

     XCHG    ; Obtener destino en DE
     STAX D  ; para escribir ese byte

     INX H   ; Se incrementan los dos punteros
     INX D
     DCX B   ; y se reduce el contador

     MOV A, B  ; Se comprueba si se ha
     CPI 00H   ; alcanzado el fin del bucle
     JNZ BUCLE

     CMP C
     JNZ BUCLE

     ; Se ha llegado al final
     SHLD 1105H ; Guardar última dirección
     XCHG       ; en las posiciones originales
     SHLD 1103H

     RST 1
END
```

Ejercicio 8.2.4.1

Solución

```
CPU "8085.TBL"
HOF "INT8"
     ORG 1000H

     LDA 1100H
     CPI 7FH     ; Se compara A con 7FH
     JZ COINC    ; Si coincide, salta

     RST 1

COINC:
     MVI A, 0FFH
     STA 1101H
     RST 1
END
```

Respuestas a las cuestiones

1. Las aplicaciones potenciales son múltiples, pero una típica consiste en utilizar el acumulador como contador de un bucle, sirviendo la comparación para determinar si se ha alcanzado el número de repeticiones establecido.

Ejercicio 8.2.4.2

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Se recupera en A el dato a verificar
     LHLD 1101H ; y en HL los límites

     CMP L      ; ¿Es A < L o, lo que es lo mismo, L > A?
     JC NOVALIDO     ; Si se activa CY es porque L > A

     CMP H           ; ¿Es A > H?
     JZ VALIDO       ; Si A = H es válido
     JNC NOVALIDO    ; Si A <> H y CY = 0 es que A > H

VALIDO:
     MVI A, 00H      ; Está en el rango
     JMP FIN

NOVALIDO:
     MVI A, 0FFH     ; No está en el rango

FIN:
     STA 1103H       ; Guardar el resultado
     RST 1

END
```

Cuestiones

1. Sí que resulta posible. En lugar de recuperar los límites en una pareja de registros, como se ha hecho en este caso, se podría asignar a `HL` la dirección donde está el límite y a continuación comparar, de la siguiente forma:

```
...
LXI H, 1101H
CMP M
...
```

2. Para saber si el valor del acumulador es igual a uno de los extremos se utiliza el bit `Z` del registro de indicadores, que es lo más lógico, pero para saber si dicho valor es mayor o menor que uno de los límites puede utilizarse el bit `S` en lugar del bit `CY` (en la sección *1.2. Registros* se detalla cada uno de los bits del registro de estado). Recurriendo a ese indicador el programa podría escribirse de la siguiente forma:

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H
     ; Se recupera en A el dato a verificar
     LDA 1100H

     LHLD 1101H ; y en HL los límites

     CMP L
     JZ VALIDO    ; Si A = L -> Correcto
     JP VALIDO    ; Si L < A -> Correcto

     CMP H
     JZ VALIDO    ; Si A = H -> Correcto
     JM VALIDO    ; Si H > A -> Correcto

NOVALIDO:
     MVI A, 0FFH ; No está en el rango
     JMP FIN

VALIDO:
     MVI A, 00H ; Está en el rango

FIN:
     STA 1103H   ; Guardar el resultado
     RST 1

END
```

Ejercicio 8.2.4.3

Solución

```
CPU "8085.TBL"
HOF "INT8"
     ORG 1000H

     ; Inicialización
     LDA 1101H    ; Leer el contador
     MOV C, A     ; y llevarlo a C

     LDA 1100H    ; En A el dato a escribir
     LHLD 1102H   ; y en HL la dirección
     ; Bucle
BUCLE:
     MOV M, A     ; Se escribe el dato

     DCR C        ; Queda un valor menos por escribir
     INX H        ; Avanzar a la siguiente posición

     JNZ BUCLE    ; Si aún no se ha llegado a cero
     ; Finalización
     RST 1
END
```

Ejercicio 8.2.4.4

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     ; Inicialización
     LXI H, 1100H
     LXI D, 1200H
     MVI C, 20H ; 32d = 20H

     ; Bucle
BUCLE:
     MOV A, M      ; Se lee un byte del origen
     XCHG          ; intercambiar punteros
     MOV M, A      ; se escribe en el destino
     XCHG          ; volver a intercambiar

     DCR C    ; Queda un valor menos por copiar
     INX H    ; Avanzar a la siguiente posición
     INX D    ; de origen y destino

     JNZ BUCLE     ; Si aún no se ha llegado a cero
```

```
      ; Finalización
      RST 1
END
```

Respuestas a las cuestiones

1. Si se ha utilizado, como en la solución previa, la pareja de registros DE como almacenamiento intermedio y la instrucción XCHG para realizar el intercambio con HL, una posible alternativa, similar a ésta, sería el almacenamiento en la pila de uno de los punteros (previamente habría que inicializar SP), utilizando después la instrucción XTHL para intercambiar ese valor con el contenido de HL. Otra posibilidad sería, dentro del cuerpo del bucle, guardar el actual valor de HL con SHLD en una cierta dirección de memoria y leer el otro puntero con LHLD, repitiendo la operación a cada ciclo del bucle.

10.3 Operaciones aritméticas

Ejercicio 8.3.1.1

Respuestas a las cuestiones

1. El resultado es `99H`, suma de los valores `7AH` y `1FH`. Son dos datos de 8 bits y el resultado es de 8 bits. El registro de estado queda con el valor `96H` que, en binario, sería el patrón de bits `1001 0110`. El bit que más nos interesa tras una operación de suma es el de menor peso, el último leyendo de izquierda a derecha, que es el correspondiente al *carry* o acarreo. En este caso dicho bit está a 0, lo que significa que tras ejecutar la suma no se ha producido acarreo, es decir, el resultado es de 8 bits y por tanto puede ser almacenado completo en el acumulador.

2. Al cambiar el segundo operando el resultado que se obtiene en la dirección `1100H` es `24H`, un valor muy inferior al anterior a pesar de que uno de los datos sumados es mayor. Al inspeccionar el registro de estado se aprecia que su contenido es `15H`, `0001 0101` en binario. El último bit está a 1, lo cual significa que el acarreo se ha activado para indicar que el resultado no puede ser almacenado en 8 bits, existiendo un noveno bit a 1 que sería el de mayor peso. Ese bit es el acarreo.

Ejercicio 8.3.1.2

Respuestas a las cuestiones

1. La suma es una operación aritmética con distintas propiedades, entre ellas la propiedad conmutativa, lo cual significa que no importa el orden de los operandos. Por tanto al invertir la secuencia en que se recuperan no habría ninguna diferencia, el resultado sería el mismo valor.

2. Al interpretar como un dato de 16 bits esas dos posiciones de memoria lo que se obtiene es el resultado completo de la suma. Si, por ejemplo, se colocan los valores `7AH` y `AAH` en las posiciones `1100H` y `1101H`, se efectúa la suma y a continuación se utilizase la instrucción `LHLD 1102H`, el contenido de `HL` sería `0124H`, un número de 16 bits resultado de la suma de dos valores de 8 bits.

Ejercicio 8.3.1.3

Respuestas a las cuestiones

1. La suma de los datos indicados da como resultado el valor 0AE3H.

2. Si el contenido de las 32 posiciones de memoria fuese 0FFH el resultado final sería 1FE0H, un valor que puede ser representado con 13 bits: **1 1111 1110 0000**. Por tanto no hay peligro de desbordamiento. En general el número de bits adicionales al tamaño de los operandos, que en este caso es de 8 bits, será de $\log_2 n$, siendo n el número de operandos. En este caso $\log_2 32 = 5$, por lo que hay que sumar 5 bits a los 8 que tiene cada operando, obteniendo un resultado de 13 bits que, por tanto, puede ser almacenado sin problemas en 16 bits.

3. Atendiendo a la fórmula dada en la cuestión anterior, al ser los operandos de 8 bits y el resultado disponer de 16 bits se tendrían 8 bits disponibles. Habría que evaluar la ecuación $\log_2 n = 8$, es decir, $2^8 = n$, de donde se obtiene que $n = 256$. Éste sería el máximo número de datos que podrían sumarse.

4. Un método alternativo para escribir el programa de este ejercicio sería utilizar la instrucción de suma de 16 bits DAD, en la cual la pareja de registros HL actúa como acumulador y los operandos pueden ser BC, DE o SP. Puesto que los operandos a sumar son de 8 bits, la parte alta de la pareja de registros elegida estaría siempre a 0, utilizándose únicamente la parte baja o LSB. Al realizar la suma con DAD se tiene la ventaja de que no es preciso controlar el acarreo manualmente, ya que la suma tiene en cuenta los 16 bits del resultado. El programa quedaría así:

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI D, 00H       ; Operando de 16 bits
     LXI H, 00H       ; Acumulador de 16 bits
     LXI B, 0100H     ; Dirección de partida

REP:
     LDAX B           ; Recuperar un operando
     MOV E, A         ; Llevarlo a DE
     DAD D            ; HL = HL + DE

     INX B            ; Avanzar al siguiente operando
     MOV A, C         ; Comprobar si se han
     CPI 20H          ; recorrido los 32 bytes (20H)
     JNZ REP          ; Repetir hasta terminar
```

```
      SHLD 1100H       ; Guardar el resultado

      RST 1
END
```

Ejercicio 8.3.1.4

Respuestas a las cuestiones

1. Al efectuar la suma de esos dos datos el resultado que se obtiene en las direcciones 1104H–1105H es 556AH. Obviamente se ha producido un acarreo y el resultado real de esa suma sería 1556AH. El bit de acarreo tras efectuar esta suma se ha puesto a 1, ya que la instrucción DAD actúa de manera similar a ADD/ADI, modificando el registro de estado según el resultado que se haya obtenido en el acumulador, la diferencia es que para DAD el acumulador es la pareja de registros HL y su tamaño es de 16 bits.

Ejercicio 8.3.2.1

Respuestas a las cuestiones

1. La sustracción es una operación aritmética que no cuenta con la propiedad conmutativa, por lo que el orden en que se toman los operandos afecta al resultado. Si se invirtiese el orden en el que son almacenados en los registros el resultado no sería el mismo, sino que se obtendría el dato de la posición 1101H menos el de la posición 1100H, en lugar de lo que se busca que es el dato de 1100H menos el de 1101H.

2. Si el minuendo es menor que el sustraendo el resultado obtenido en la posición 1102H no parece tener, en principio, un significado lógico. Lo que ocurre, en realidad, es que hay que interpretarlo como un número negativo (en complemento a 2) y no como un número positivo. Además en el registro de estado se habrá activado el bit de acarreo. Si en el caso de la suma dicho bit señala un exceso de un bit en el resultado, en el de la resta el significado es análogo y serían precisos 8 bits para almacenar la diferencia más 1 bit adicional para el signo.

3. En el primer caso el valor obtenido como resultado es 8DH. Dado que el bit de acarreo está a 0, ésa es la diferencia entre el primer operando y el segundo:

```
     AAH = 170d
   - 1DH =  29d
     8DH = 141d
```

En el segundo caso el resultado que se obtiene en 1102H es 73H. El bit de acarreo está a 1, lo que nos indica que ese valor ha de ser interpretado como un número negativo en complemento a 2. Para convertir el número en complemento a 2 en un número positivo hay que: a) invertir todos sus bits y b) sumar 1 al resultado:

```
      73H = 0111 0011b

Invertir - 1000 1100b

Suma 1   - 1000 1101b = 8DH = 141d
```

El resultado, por tanto, es -141 puesto que sabemos que el resultado ha sido negativo.

Ejercicio 8.3.2.2

Respuestas a las cuestiones

1. El resultado que se obtiene en la posición de memoria 1102H es 8DH, tanto si se coloca el valor AAH en 1100H y el valor 1DH en 1101H como si se hace a la inversa. El programa por tanto facilita el valor absoluto de la diferencia, que es lo que se pretendía.

Ejercicio 8.3.2.3

Respuestas a las cuestiones

1. Al igual que ocurre con la resta en 8 bits, si el minuendo es menor que el sustraendo el resultado obtenido será un número negativo en complemento a dos. Además la ejecución de la instrucción SBB, con la que se resta al MSB del minuendo el del sustraendo y el bit de acarreo, provocará que dicho bit se ponga de nuevo a 1 señalando la situación.

Ejercicio 8.3.3.1

Respuestas a las cuestiones

1. En este caso concreto, dado que el multiplicador es pequeño, resultaría mucho más eficiente escribir el programa de la siguiente forma:

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Tomar el dato a multiplicar
     MOV B, A

     ADD A ; duplicarlo
     ADD A ; cuadruplicarlo
     ADD B ; y sumarlo una quinta vez

     STA 1101H

     RST 1
END
```

Teniendo en el acumulador el número que se quiere multiplicar, al sumarlo consigo mismo se obtiene el doble. Ese resultado queda en el acumulador, de forma que al sumarlo consigo mismo una segunda vez es como si se hubiese multiplicado por 4. Finalmente se añade el multiplicando original, que se había guardado en `B`, obteniendo el mismo efecto que la multiplicación del programa original. La diferencia es que este programa ocupa 25 bytes menos y resulta más rápido. No se trata, sin embargo, de una solución que pueda ser aplicada de manera general.

2. Cualquier valor superior a `33H` como multiplicando provocará un desbordamiento del acumulador, por lo que en la posición `1101H` se tendrá únicamente el LSB del resultado.

Ejercicio 8.3.3.2

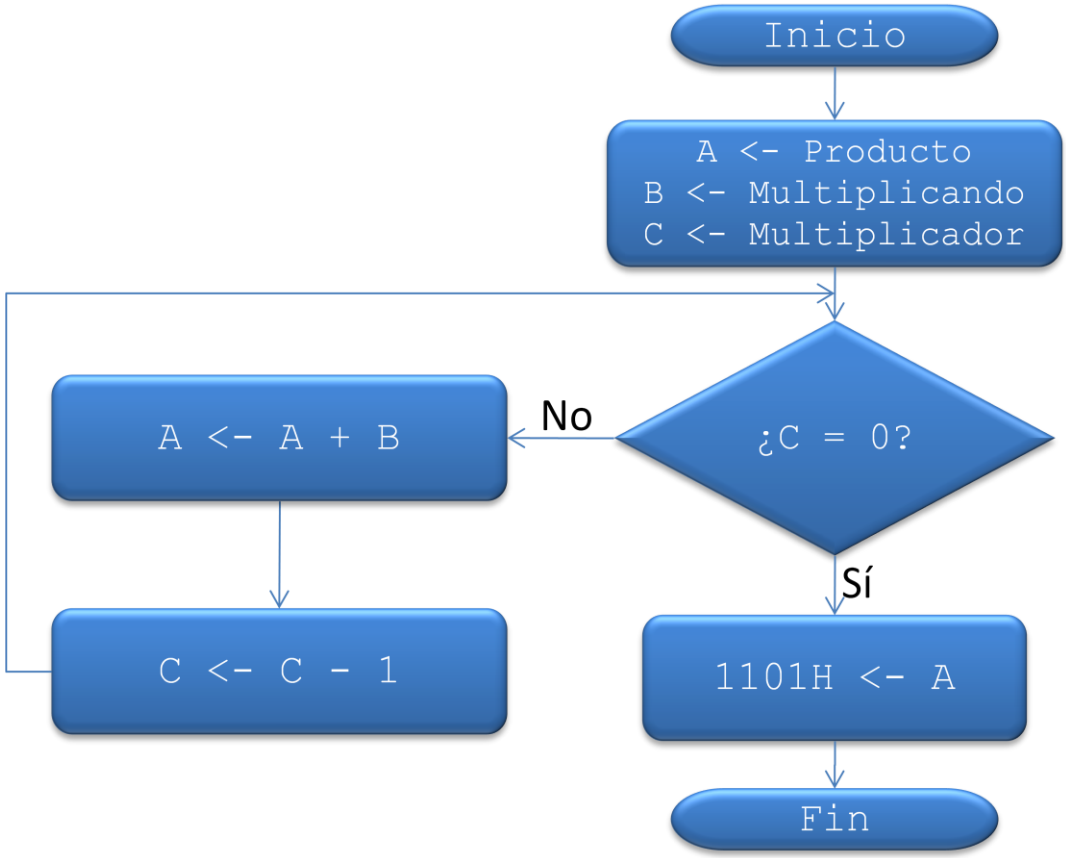
Respuestas a las cuestiones

1. Si ambos, multiplicando y multiplicador, tomasen el mayor valor posible para un dato de 8 bits: `FFH`, su producto tendría como resultado el valor `FE01H`. Es un dato de 16 bits. No existe la posibilidad de un desbordamiento.

2. Cualquier número multiplicado por 0 da como resultado 0. El programa dado como solución para el ejercicio, sin embargo, no produce dicho resul-

tado. En su lugar lo que hace es multiplicar el primer operando por 256 (100H), ya que el registro usado como contador primero se decrementa y después se comprueba, no al contrario.

3. Para evitar el caso particular en el que el multiplicador es 0, y conseguir que el programa genere siempre el resultado correcto, habría que cambiar su estructura y ajustarla al esquema representado en el siguiente diagrama de flujo:



Ordinograma de la solución alternativa al ejercicio 8.3.3.2.

Siguiendo este ordinograma el código del programa quedaría así:

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H   ; Se lleva el multiplicando
     MVI D, 00H
     MOV E, A    ; a DE

     LXI H, 00H  ; Producto en HL

     LDA 1101H   ; Multiplicador en A

REP:
```

```
      CPI 00H
      JZ FIN      ; A = 0, terminar

      DAD D       ; HL = HL + DE
      DCR A

      JMP REP

FIN:
      SHLD 1102H

      RST 1
END
```

Ejercicio 8.3.4.1

Respuestas a las cuestiones

1. En caso de que el dividendo facilitado en la posición `1100H` sea 0, la comparación que se efectúa al inicio del bucle provocará de inmediato el salto a la etiqueta `FIN`. En ella se almacena como cociente el contenido del registro `C` que, previamente, había sido inicializado con el valor 0. El resultado, por tanto, es el correcto.

2. Al contrario de lo que ocurre con el producto, cuyo resultado será normalmente mayor que los operandos que intervienen en la operación, en la división el cociente podrá ser inferior o igual (si el divisor es 1) al dividendo, pero nunca mayor. Obviamente esto es así teniendo en cuenta que hablamos de aritmética entera. Usando como divisor un número decimal, inferior a 1, sí que podría obtenerse un cociente mayor que el dividendo.

Ejercicio 8.3.4.2

Respuestas a las cuestiones

1. El programa ofrecido funciona bien en todos los casos salvo en uno concreto: cuando el divisor, almacenado en la posición `1101H`, es cero. Lo que ocurre en ese caso es que el valor del registro `B` será cero y, por tanto, el dato contenido en el acumulador no disminuirá nunca, por lo que el bucle se ejecutará de manera indefinida. Por ello el uP-2000 muestra el carácter `E` en el display y deja de responder, se encuentra en un bucle sin fin. La única solución será reiniciarlo. Ésta es la razón de que, matemáticamente hablando, se diga que cualquier valor dividido entre 0 da como resultado infinito (para ser estrictos habría que decir que tiende a infinito).

2. La solución es obvia: hay que modificar el programa para que compruebe al principio si el divisor es cero, en cuyo caso no se entraría en el bucle de división. Lo que no resulta tan obvio es cómo comunicar el hecho de que se ha producido un error de división por cero, ya que los microprocesadores no pueden representar el concepto matemático de infinito. Una forma de hacerlo sería devolver tanto el cociente como el resto con un valor imposible, por ejemplo haciendo que ambos datos sean `FFH`. Ningún número de 8 bits dividido entre otro de 8 bits podría generar ese resultado.

   Otra posibilidad es usar un bit del registro de estado para indicar si la operación se ha completado con éxito o no. Es una técnica habitual activar el bit de acarreo (ponerlo a 1) para indicar un fallo y desactivarlo para comunicar que todo ha ido bien. El programa siguiente combina las dos técnicas mencionadas, de forma que una división por cero devuelve como resultado `FFH` como cociente y resto y la activación del bit de acarreo. Con solo consultar dicho indicador se sabrá si la operación ha tenido éxito, estará a 0, o no.

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1101H   ; Divisor en B
     MOV B, A

     CPI 00H     ; Comprobar si divisor = 0
     JNZ VALE    ; Si no, se puede dividir

     ; El divisor = 0, no es posible llevar
     ; a cabo la división

     MVI A, 0FFH ; Usar como cociente y resto
     STA 1102H    ; el valor FFH
     STA 1103H

     STC         ; Poner el bit de acarreo a 1
     RST 1       ; para salir indicando error

VALE: ; Proceso de división

     LDA 1100H   ; Dividendo en A

     MVI C, 0    ; Cociente en C

REP:
     CMP B       ; ¿Es A < B?
     JC FIN      ; Si es así se ha terminado

     SUB B       ; A = A - B
```

```
      INR C       ; Incrementar cociente

      JMP REP

FIN:
      STA 1103H   ; Almacenar el resto

      MOV A, C    ; Almacenar el cociente
      STA 1102H

      STC         ; Poner el bit de acarreo a 0
      CMC         ; para salir sin indicar error

      RST 1
END
```

Ejercicio 8.3.5.2

Respuestas a las cuestiones

1. Utilizando las instrucciones `PUSH` y `POP`, que son las habituales al trabajar con la pila, al recuperar el dato almacenado en la pila habría que modificar una pareja de registros, por lo que el contenido de ésta tendría que guardarse previamente. El problema es que si se ejecuta `PUSH` y a continuación `POP` el dato que se recupera es el mismo que acaba de guardarse. La instrucción `XHTL`, por el contrario, intercambia el contenido de la pareja `HL` con el último dato almacenado en la pila sin afectar a ningún otro registro del procesador. En realidad el 8085 recurre a un registro interno al microprocesador para realizar esa operación.

2. Podría usarse una zona de memoria auxiliar para ello que, incluso, podría ser aquella en la que se almacenará el resultado. Al inicio de cada ciclo del bucle se recuperaría la parte baja con `LHLD`, a continuación se sumaría `DE`, el paso siguiente sería almacenar el resultado con `SHLD`, recuperar la parte alta, acumular el arrastre y almacenar de nuevo esa parte alta.

3. El 8085 no dispone de instrucciones que puedan comparar datos de 16 bits, por lo que la evaluación de si una pareja de registros tiene un cierto valor ha de hacerse necesariamente en dos partes: LSB y MSB. Comprobar si dicho contenido es cero, no obstante, representa un caso particular. Si se toman los bits de cada uno de los registros que forman la pareja, en este caso `B` y `C`, y se efectúa una operación `OR` bit a bit, el resultado será cero únicamente en el caso de que ambas partes sean cero.

Ejercicio 8.3.5.3

Respuestas a las cuestiones

1. El significado de las posiciones de memoria indicadas es el siguiente:

   - `1100H-1101H`: Indica la posición de memoria donde está el LSB del primer sumando.

   - `1102H-1103H`: Indica la posición de memoria donde está el LSB del segundo sumando.

   - `1104H`: Indica la dirección de un dato de 8 bits que establece la longitud de los sumandos en número de bytes.

   - `1105H-1106H`: Indica la posición de memoria a partir de la cual se almacenará el resultado.

2. El contenido de esas posiciones de memoria es `5555H`. El programa ha sumado dos datos de 16 bits, cuyas posiciones en memoria vienen indicadas por los valores escritos en las posiciones `1100H-1106H`.

3. Este programa realiza la suma de dos datos que pueden tener cualquier longitud, entre 1 y 255 bytes, y que pueden estar almacenados en cualquier posición de la memoria del sistema. El resultado se guarda, con el mismo tamaño que los operandos, en la dirección de memoria que se indique. Podría decirse que es un programa genérico para sumar dos números, independientemente de cuál sea su tamaño.

4. La instrucción `XTHL` que hay tras la instrucción `PUSH H` es superflua, ya que intercambia `HL` con el dato que acaba de introducirse en la pila, que es el propio `HL`. Su eliminación no afectará al resultado. Reordenando las instrucciones de inicialización también puede eliminarse una de las instrucciones `XCHG`.

A continuación se muestra el código del programa pero tras agregar comentarios explicativos, eliminar operaciones superfluas y reordenar algunas instrucciones:

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI SP, 2000H

     LHLD 1100H  ; HL <-- Dir operando 1
```

```
     XCHG        ; DE <-- Dir operando 1

     LHLD 1105H  ; HL <-- Destino resultado
     PUSH H      ; [SP] <-- Dir resultado

     LHLD 1102H  ; HL <-- Dir operando 2

     LDA 1104H   ; A <-- Longitud sumandos
     MOV C, A    ; C <-- Longitud sumandos

     STC         ; Se pone a 0 el acarreo
     CMC

     PUSH PSW

BUCLE:
     POP PSW     ; Restaurar registro estado

     LDAX D      ; A <-- Byte operando 1
     ADC M       ; A = A + [HL]
     XTHL        ; [SP] <--> HL

     MOV M, A    ; Guardar byte resultado
     INX H       ; siguiente byte resultado
     XTHL        ; [SP] <--> HL

     PUSH PSW    ; Guardar registro estado

     INX D       ; Siguiente byte operando 1
     INX H       ; y operando 2

     DCR C       ; Queda un byte menos
     JNZ BUCLE

     POP PSW     ; Limpiar pila al salir
     POP H

     RST 1
END
```

Ejercicio 8.3.6.1

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     MVI A, 00H ; Se inicializa el acumulador
     LXI H, 1100H ; y la pareja HL
```

```
     ADD M ; Se suma el primer operando
     INX H ; Avanzar al siguiente
     ADD M ; Se suma el segundo operando

     STA 1102H  ; Y se guarda el resultado

     MVI A, 00H ; Poner un 0 en A
     JNC FIN    ; y saltar si no ha habido acarreo

     MVI A, 01H ; Si ha habido acarreo poner un 1

FIN:
     STA 1103H  ; Guardar el indicador de acarreo

     RST 1

END
```

Respuestas a las cuestiones

1. Inicialmente se ha asignado al acumulador el valor 0, de forma que las dos sumas posteriores den como resultado la únicamente de los dos operandos. La razón es que el 0 es el elemento neutro para la suma.

2. No existe ninguna otra modalidad de la instrucción `ADD`, aparte de `ADD M`, que permita recuperar los operandos desde la memoria, por lo que no es viable usar como puntero otra pareja de registros que no sea `HL`.

3. El contenido inicial del acumulador es 0 y el primer operando será de 8 bits, por lo que el resultado, al ser el 0 el elemento neutro de la suma, será ese mismo operando y, en consecuencia, no hay ninguna posibilidad de que se produzca acarreo.

Ejercicio 8.3.6.2

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Tomar los dos operandos en A y B
     MOV B, A
     LDA 1101H

     ADD B ; Efectuar la suma

     STA 1102H  ; Y guardar el resultado

     MVI A, 00H ; Ponemos a 0 el acumulador
     ACI 00H    ; y le sumamos el valor 0 más acarreo

     STA 1103H

     RST 1

END
```

Ejercicio 8.3.6.3

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI D, 00H ; Operando de 16 bits
     LXI H, 00H ; Acumulador de 16 bits
     MVI A, 00H ; Contador

REP:
     MOV E, A   ; Llevar el contador a E
     DAD D      ; HL = HL + DE

     INR A      ; Siguiente número
     JNZ REP    ; Siempre que no se haya llegado a 0

     SHLD 1100H ; Guardar el resultado

     RST 1
END
```

Respuestas a las cuestiones

1. Para sumar valores consecutivos basta con ir incrementando, como se hace en esta solución, el contenido de un registro que va a actuar como operando a cada ciclo. Para cualquier otra sucesión de valores, ya sean pares, múltiplos de 3, potencias, etc., en cada ciclo del bucle sería necesario realizar las operaciones aritméticas apropiadas que calculasen el siguiente operando, no bastando con un simple incremento. Sumar los pares del 0 al 255 sería un caso relativamente sencillo, ya que bastaría con agregar un segundo incremento, una instrucción `INR A` adicional, de forma que se saltasen dos unidades por ciclo del bucle.

2. La suma de más de 256 datos de 8 bits requiere tener en cuenta dos aspectos distintos. El más obvio es que se precisará un registro de 16 bits que actúe como contador, en lugar de usar uno de 8 bits. Lo más importante, sin embargo, es que esa suma puede generar un resultado de más de 16 bits, por lo que debería tenerse en cuenta un posible acarreo de 16 bits, o decimoséptimo bit, que provocaría que el resultado se extendiese a 32 bits.

Ejercicio 8.3.6.4

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H   ; LSB primer operando
     MOV B, A    ; en B
     LDA 1102H   ; LSB segundo operando en A

     ADD B       ; Se suman
     STA 1104H   ; Se guarda LSB resultado

     LDA 1101H   ; MSB primer operando
     MOV B, A    ; en B
     LDA 1103H   ; MSB segundo operando en A

     ADC B       ; Se suma añadiendo acarreo
     STA 1105H   ; y se guarda

     RST 1
END
```

Respuestas a las cuestiones

1. Aunque de manera puntual el resultado de la suma de dos números en formato *big endian* podría ser correcto, siempre que no se produzca ningún acarreo al sumar los LSB ni MSB, en el caso general el programa no funcionaría bien. La razón es que al ser (en *big endian*) el primer byte el MSB, en lugar del LSB, la instrucción `ADC` estaría llevando el acarreo del MSB al LSB, es decir, al contrario de lo que debería ser. La solución obviamente sería invertir el orden en que se leen los datos de la memoria, recuperando primero las posiciones `1101H` y `1103H` y después `1100H` y `1102H`.

Ejercicio 8.3.6.5

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1101H   ; El sustraendo
     MOV B, A    ; en B
     LDA 1100H   ; El minuendo en A

     SUB B       ; A = A - B

     JNC NONEG   ; Si no es negativo salta

     CMA         ; Complementar el acumulador
     INR A       ; y sumar 1

NONEG:

     STA 1102H   ; y se guarda

     RST 1
END
```

Respuestas a las cuestiones

1. Si se toma el valor 0 en representación binaria con 8 bits se tiene `00000000b`. Para calcular el complemento a 2 se invertirían todos los bits: `11111111b`, y a continuación se sumaría 1: `00000000b`. El resultado final, por tanto, sería el mismo, el único efecto adicional sería que el acarreo se activaría al incrementar el valor en una unidad, algo que no ocurre al calcular el complemento a 2 de cualquier otro número.

Ejercicio 8.3.6.6

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1101H   ; El sustraendo,
     CMA         ; complementar el acumulador
     INR A       ; y sumar 1,
     MOV B, A    ; en B

     LDA 1100H  ; El minuendo en A

     ADD B ; A = A + (-B)

     STA 1102H   ; y se guarda

     RST 1
END
```

Ejercicio 8.3.6.7

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LHLD 1100H  ; El minuendo en HL

     LDA 1102H   ; El sustraendo se niega
     CMA
     MOV E, A   ; y se lleva a DE
     LDA 1103H
     CMA
     MOV D, A

     INX D       ; Terminar complemento a dos

     DAD D       ; HL = HL + (-DE)
     SHLD 1104H

     RST 1
END
```

Ejercicio 8.3.6.8

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Se lleva el multiplicando
     MVI D, 00H
     MOV E, A   ; a DE

     LXI H, 00H ; Producto en HL

     MVI C, 05H ; Multiplicador en C

REP:
     DAD D      ; HL = HL + DE
     DCR C
     JNZ REP    ; C > 0, seguir

     SHLD 1101H

     RST 1
END
```

Respuestas a las cuestiones

1. Ningún número de 8 bits multiplicado por 5 precisará más de 16 bits para ser almacenado. De hecho necesitará muchos menos bits, según la regla explicada en la cuestión 2 del ejercicio 8.3.1.3 se necesitarán $\log_2 n$, siendo n el multiplicador.

Ejercicio 8.3.6.9

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LHLD 1100H ; Se lleva el multiplicando
     XCHG       ; a DE

     LXI H, 00H ; Producto en HL

     LDA 1102H  ; Multiplicador en A
```

```
REP:
     CPI 00H
     JZ FIN      ; A = 0, terminar

     DAD D       ; HL = HL + DE
     DCR A

     JMP REP

FIN:
     SHLD 1103H

     RST 1
END
```

Respuestas a las cuestiones

1. Al tener uno de los operandos un tamaño de 16 bits y el otro de 8 bits, el producto puede tener una longitud de hasta 24 bits. Éste sería el tamaño que debería tener el resultado para no perder información.

Ejercicio 8.3.6.10

Solución

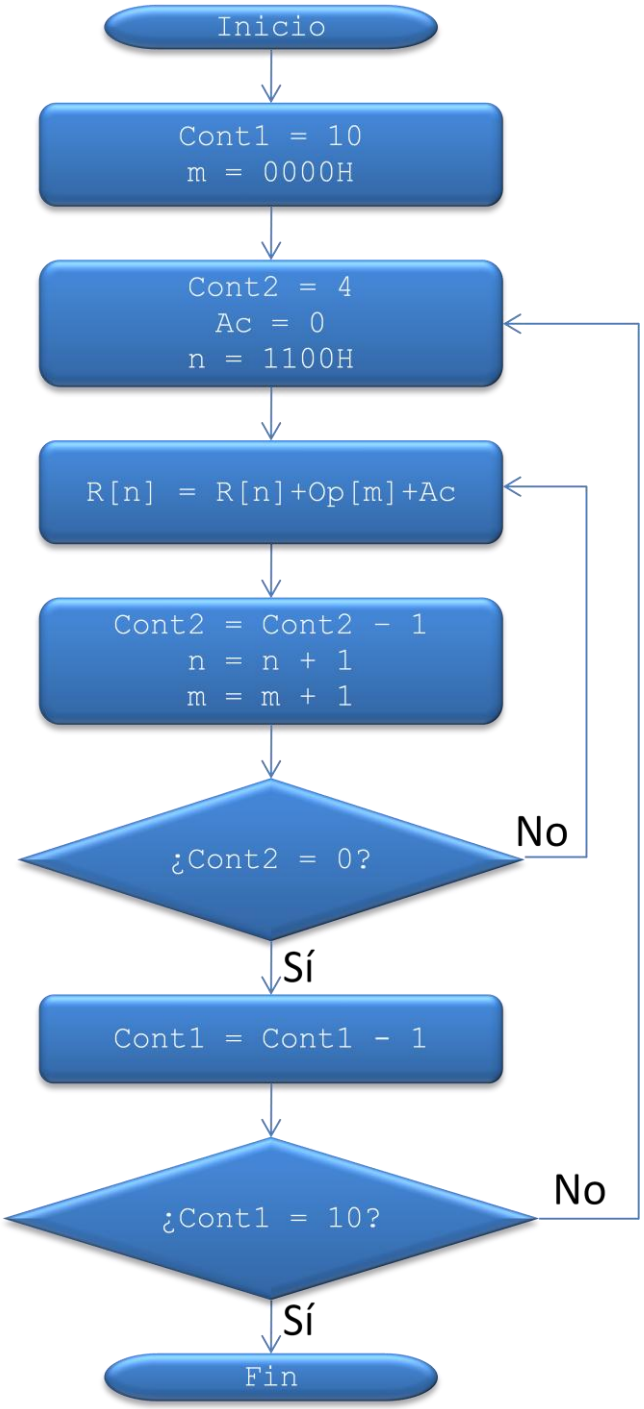
```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1102H   ; Divisor en B
     MOV B, A

     CPI 00H     ; Comprobar si divisor = 0
     JNZ VALE    ; Si no, se puede dividir

     ; El divisor = 0, no es posible llevar
     ; a cabo la división

     MVI A, 0FFH ; Usar como cociente y resto
     STA 1103H    ; el valor FFH
     STA 1104H

     STC         ; Poner el bit de acarreo a 1
     RST 1       ; para salir indicando error

VALE: ; Proceso de división

     LHLD 1100H  ; Dividendo en HL

     MVI C, 0    ; Cociente en C
```

```
REP:
     MOV A, B    ; ¿Es L > B?
     CMP L
     JC SIGUE    ; Si es así, seguir dividiendo

     ; Si L < B, se habrá llegado al final siempre
     ; que el contenido de H = 0
     MOV A, H
     CPI 00H
     JZ FIN      ; Si es así se ha terminado

SIGUE:
     MOV A, L    ; L = L - B
     SUB B
     MOV L, A
     MOV A, H    ; H = H - AC
     SBI 00H
     MOV H, A

     INR C       ; Incrementar cociente

     JMP REP

FIN:
     MOV A, C    ; Almacenar el cociente
     STA 1103H

     MOV A, L
     STA 1104H   ; Almacenar el resto

     STC         ; Poner el bit de acarreo a 0
     CMC         ; para salir sin indicar error

     RST 1
END
```

Ejercicio 8.3.6.11

Solución

Para resolver este ejercicio se usará como guía el siguiente diagrama de flujo:



Ordinograma del ejercicio 8.3.6.11.

El puntero `m`, que puede ser cualquier pareja de registros, va avanzando desde la dirección de partida: la `0000H`. En cada ciclo del bucle interior se incrementa 4 veces, y dicho bucle interior se ejecuta 10 veces. El puntero `n` señala la dirección donde quedará el resultado

y, a diferencia del anterior, siempre recorre las mismas posiciones: de la 1100H a la 1103H. De esta forma se utilizan las posiciones de memoria del resultado para ir acumulando la suma, en lugar de mantenerla en otro lugar y copiarla al final.

El código del programa sería el siguiente:

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de memoria

     LXI H, 0000H     ; Se pone a 0 el resultado
     SHLD 1100H       ; de 32 bits
     SHLD 1102H

     XCHG             ; DE apunta a los datos a sumar
     MVI C, 0AH       ; Sumar 10 datos de 32 bits

BUCLE1:
     LXI H, 1100H     ; HL apunta al resultado
     STC         ; Asegurar que el acarreo está 0
     CMC         ; para que no influya en primera suma

     PUSH PSW    ; Guardar registro de estado

BUCLE2:
     POP PSW     ; Recuperar estado anterior
     LDAX D      ; A <- [DE]
     ADC M       ; A = A + [HL]
     MOV M,A     ; [HL] <- A

     PUSH PSW    ; Guardar estado

     INX D       ; Siguiente byte
     INX H       ; de cada operando

     MOV A, L    ; Comprobar si se han
     CPI 04H     ; sumado los 4 bytes
     JNZ BUCLE2

     POP PSW     ; Al terminar extraer la pila

     DCR C
     JNZ BUCLE1  ; Sumar 10 datos

     RST 1
END
```

Respuestas a las cuestiones

1. El dato de 32 bits obtenido a partir de la suma, y almacenado en las direcciones `1100H-1103H`, es `BE23EB1CH`.

2. La suma no puede ser distinta en ningún caso siempre que el programa se ejecute sobre el sistema uP-2000, ya que los datos tomados como sumandos se encuentran almacenados en memoria EPROM y ésta no cambia entre ejecuciones.

10.4 Trabajo a nivel de bits

Ejercicio 8.4.1.1

Respuestas a las cuestiones

1. Se ha recurrido a la operación lógica OR, en este caso concreto mediante la instrucción `ORI` del 8085 puesto que el dato con el que se operará es inmediato y forma parte de la instrucción.

2. En general, para activar un cierto bit se usará como operando de la operación lógica OR un valor que tenga todos sus bits a 0 excepto el que se quiere afectar. Para activar el quinto bit se ha utilizado el valor hexadecimal `10H`, ya que en dicho dato están todos los bits a 0 excepto el quinto. Para saber qué valor hexadecimal corresponde a cada bit lo más fácil es usar una tabla como la siguiente:

Nº bit	7	6	5	4	3	2	1	0
Bit	0	0	0	1	0	0	0	0
Hex	80H	40H	20H	10H	08H	04H	02H	01H
Decimal	128	64	32	16	8	4	2	1

Tabla de correspondencia entre posiciones de los bits y valores hexadecimales.

Los valores hexadecimales son los que habría que tomar si el bit indicado estuviese a 1. En este caso el bit que interesa es el 4 (quinto bit), cuyo valor asociado es el `10H`.

3. Si el dato leído de la posición `1100H` tuviese el quinto bit ya activo, al realizar la operación lógica OR ni ese bit ni el resto del dato cambiarán, por lo que el resultado será exactamente el mismo.

Ejercicio 8.4.1.2

Respuestas a las cuestiones

1. Se ha recurrido a la operación lógica AND, en este caso concreto mediante la instrucción `ANI` del 8085 puesto que el dato con el que se operará es inmediato y forma parte de la instrucción.

2. El principio de funcionamiento de la operación lógica AND, cuando se usa para manipular bits, es justamente el inverso de la operación OR. El dato a usar, por tanto, deberá tener todos sus bits a 1 a excepción del que se quiere poner a 0. Dicho valor se obtiene sumando los valores de todos esos bits, en este caso `80H+40H+20H+08H+04H+02H+01H = FEH`.

También se puede calcular como el valor correspondiente a todos los bits a 1 menos el valor del bit a desactivar: `FFH-10H=FEH`.

3. En caso de que el bit a desactivar ya estuviese a cero el resultado será idéntico al valor original.

Ejercicio 8.4.1.3

Respuestas a las cuestiones

1. Se ha recurrido a la operación lógica XOR, en este caso concreto mediante la instrucción `XRI` del 8085 puesto que el dato con el que se operará es inmediato y forma parte de la instrucción.

2. Cuando se efectúa una operación lógica XOR entre dos bits, si el segundo es 0 el bit resultante será igual al primero, mientras que si el segundo es 1 el resultado será el inverso del primero. Es algo que puede verse fácilmente a partir de la tabla de verdad de este operador lógico:

XOR	0	1
0	0	1
1	1	0

Tabla de verdad del operador XOR.

   Basta con usar el mismo dato que se emplearía con la operación lógica OR, por tanto, pero el resultado no sería siempre la activación del bit sino la inversión de su estado.

3. Si se ejecuta el programa varias veces, puesto que el resultado se almacena en la misma posición que el dato original, el resultado será que a cada ejecución se cambiará el bit indicado, poniéndose a 0 y a 1.

Ejercicio 8.4.2.1

Respuestas a las cuestiones

1. Cuando lo que se quiere no es manipular un bit sino comprobar su estado, lo más fácil es aislarlo desactivando todos los demás. Para ello se recurre a la operación lógica AND que, con el valor `10H`, pondrá a cero todos los bits menos el quinto. Si el resultado que queda en el acumulador es 0, y por tanto se activa el correspondiente indicador del registro de estado del 8085, ello significará que el quinto bit también estaba a 0, ya que en caso contrario el resultado obtenido en el acumulador no podría ser 0.

2. Sí. Otra posibilidad sería ir rotando los bits del byte que actúa como indicador hasta obtener el quinto bit en el bit de acarreo del registro de estado.

Ejercicio 8.4.3.1

Respuestas a las cuestiones

1. En este caso concreto no habría alguna diferencia, ya que al ser 4 bits la mitad de los que tiene un byte, daría igual rotarlos hacia la izquierda que hacia la derecha para intercambiar sus posiciones.

2. Aunque la instrucción `RAL` también lleva a cabo una rotación hacia la izquierda, como `RLC`, el resultado no sería el mismo ya que en dicha rotación no intervienen solamente los 8 bytes del dato, sino que también participa el bit de acarreo. Se trata, en realidad, de una rotación de 9 bits: los 8 del acumulador más el bit de acarreo. El resultado, por tanto, no sería el mismo en términos generales, aunque de forma puntual pudiera coincidir.

Ejercicio 8.4.3.2

Respuestas a las cuestiones

1. En el programa ofrecido como solución se usado la instrucción `RAR` ya que al rotar hacia la derecha, haciendo que se pierda el bit de menor peso, el que se introduce como nuevo bit de mayor peso es el contenido en el bit de acarreo, un bit que previamente se ha puesto a 0. De utilizarse la instrucción `RRC` el bit de menor peso, que sale por la derecha, es el mismo que se introduce como bit de mayor peso, entrando por la izquierda. Esto requeriría un paso adicional, al final, que consistiría en poner a 0 los dos bits de mayor peso del acumulador. El programa quedaría entonces de la siguiente forma:

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LDA 1100H  ; Se recupera el primer operando
     RAR        ; Rotar hacia la derecha
     RAR        ; 2 veces
     ANI 3FH    ; Poner a 0 dos bits de mayor peso
     STA 1101H  ; Y se guarda el resultado
     RST 1
END
```

2. El resultado es correcto, ya que al desplazar 8 bits que son 0 el resultado seguirá siendo 0.

Ejercicio 8.4.4.2

Respuestas a las cuestiones

1. Aplicada a dos bits individuales que se pretenden combinar en uno, la opción lógica OR actúa como la suma, de forma que el resultado será 1 en cuanto uno de los dos bits que actúan como operandos sea 1. La operación lógica AND actúa como el producto y, por ello, el bit resultante es 0 si cualquiera de los dos bits que actúan como operandos es 0. La única posibilidad de obtener un 1 es que los dos operandos sean 1.

2. Las instrucciones de rotación a través del bit de acarreo son análogas a las que se han utilizado y, en consecuencia, podrían también aplicarse en la solución. Sería necesario, no obstante, agregar una rotación adicional en cada caso, ya que dichas instrucciones trabajan sobre 9 bits (según se explicó en 8.4.3.1) y no sobre 8. Se obtendría un código algo más largo sin ningún beneficio respecto a la solución original.

3. Una mejora, que reduciría tanto el tamaño del programa (una vez ensamblado) como el tiempo de ejecución, sería conservar en un registro del 8085 el byte leído de la posición `1104H` al inicio del programa, recuperándolo cuando se necesite en lugar de leerlo cada vez de la memoria.

Ejercicio 8.4.5.1

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Tomar el dato
     MOV B, A   ; y preservarlo

     ANI 10H    ; Mirar el quinto bit
     JZ PON1    ; Si está a 0, ponerlo a 1

     ; El bit está a 1, ponerlo a 0
     MOV A, B
     ANI 0EFH

     JMP FIN

PON1: ; El bit está a 0, ponerlo a 1
```

```
     MOV A, B
     ORI 10H

FIN:
     STA 1100H

     RST 1

END
```

Ejercicio 8.4.5.2

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI D, 0000H     ; DE apunta a clave
     LXI H, 1200H     ; [SP] apunta a destino
     PUSH H
     LXI H, 1100H     ; HL apunta a origen

     MVI C, 08H       ; 8 bytes

BUCLE:
     LDAX D      ; Se recupera un byte
     XRA M       ; y se hace XOR con un byte de EPROM
     XTHL        ; [SP] <--> HL
     MOV M, A    ; Guardar byte codificado

     INX H       ; Incrementar los tres punteros
     XTHL
     INX H
     INX D

     DCR C       ; Queda un byte menos

     JNZ BUCLE

     RST 1

END
```

Respuestas a las cuestiones

1. La secuencia de bytes que forma el criptograma es `DA BD 00 F8 E5 0D 83 D9`. A partir de ella resulta computacionalmente imposible recuperar el mensaje original si no se conoce la secuencia de bytes usada como clave, y dicha secuencia ni siquiera es conocida por el programa ya que está alojada en la memoria EPROM del uP-2000.

2. El resultado que se obtiene al ejecutar el programa tomando como mensaje el criptograma es el propio mensaje original. La operación lógica XOR es *invertible*, lo cual le hace especialmente adecuada para cifrar/descifrar datos.

3. Se obtendría un mensaje sin ningún sentido, a menos que la secuencia usada clave de descifrado coincidiese byte a byte con la usada para el cifrado, algo altamente improbable.

Ejercicio 8.4.5.3

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H   ; Tomar el byte en A

     LXI H, 1101H
     MVI M, 00H  ; Contador de bits a 0

     MVI C, 08H  ; Hay que recorrer 8 bits

BUCLE:
     RAL         ; Siguiente bit al carry
     JNC ES0     ; Si no se activa, el bit es 0

     INR M       ; Si es 1, incrementar cuenta

ES0:
     DCR C       ; Queda un byte menos

     JNZ BUCLE

     RST 1

END
```

Ejercicio 8.4.5.4

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Tomar el byte en A

     LXI H, 1101H

     MVI C, 08H ; Hay que recorrer 8 bits

BUCLE:
     RAL        ; Siguiente bit al carry
     JNC ES0    ; Si no se activa, el bit es 0

     MVI M, 01H ; El bit es uno
     JMP FINB

ES0:
     MVI M, 00H ; El bit es cero

FINB:
     INX H      ; Siguiente bit
     DCR C      ; Queda un bit menos

     JNZ BUCLE

     RST 1

END
```

Respuestas a las cuestiones

1. Si se prescinde de la operación de rotación a través del bit de acarreo, para comprobar el estado de cada bit será necesario usar la operación lógica AND con la máscara adecuada: una que tenga todos los bits a 0 a excepción del bit a comprobar. De esta forma se activará el bit Z del registro de estado si el bit está a 0, desactivándose si es 1. La máscara usada cambiará a cada ciclo del bucle. Si se ha comenzado por el bit de menor peso, esa máscara será inicialmente `0000 0001`, al siguiente ciclo pasará a ser `0000 0010`, al siguiente `0000 0100` y así sucesivamente. Obsérvese que cada paso representa una multiplicación por 2 del valor de la máscara, es decir, una suma consigo misma. También hay que tener en cuenta que en este caso se están recorriendo los bits de derecha a izquierda, desde el de menor peso al de mayor peso, mientras que la solución previa lo hace de izquierda a derecha. A la hora de depositar el resultado en memoria es-

to provocaría que apareciesen en orden inverso. Con todo, la solución a este problema modificado sería la siguiente:

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H   ; Tomar el byte a comprobar
     MOV E, A    ; en E

     LXI H, 1108H ; Empezar desde el final

     MVI C, 08H  ; Hay que recorrer 8 bits
     MVI A, 01H  ; Comenzar por el primer bit (bit 0)

BUCLE:
     MOV D, A    ; Guardar A
     ANA E       ; Antes de comprobar el bit
     JZ ES0      ; Si se activa, el bit es 0

     MVI M, 01H  ; El bit es uno
     JMP FINB

ES0:
     MVI M, 00H  ; El bit es cero

FINB:
     DCX H       ; Siguiente bit
     DCR C       ; Queda un bit menos

     MOV A, D    ; Recuperar bit
     ADD A       ; Desplazar una posición a izq.

     JNZ BUCLE

     RST 1

END
```

Ejercicio 8.4.5.5

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Tomar el byte en A

     MVI C, 00H ; Cociente

DIVIDE:
     CPI 10H    ; A < 10H (16d)
     JC FIND    ; Fin de la división

     SUI 10H    ; Se resta 10H
     INR C      ; e incrementa cociente
     JMP DIVIDE

FIND:
     LDA 1100H  ; Tomar de nuevo el byte en A
     ANI 0FH    ; Borrar cuatro bits más peso

     ADD A      ; Multiplicar por 2
     ADD A      ; por 4
     ADD A      ; por 8
     ADD A      ; por 16

     ORA C      ; Unir 4 bits menor peso

     STA 1101H

     RST 1

END
```

Ejercicio 8.4.5.6

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1101H   ; Tomar n° bit a tratar
     MOV C, A    ; y ponerlo en 0
     INR C       ; ajustado para entrada en bucle

     ; Colocar el bit en la posición indicada
     MVI A, 01H

DESP:
     DCR C       ; Si ya se ha llegado al bit
     JZ FINP     ; no continuar

     RLC         ; En caso contrario desplazar
     JMP DESP    ; y seguir

FINP:
     MOV C, A    ; Guardar máscara

     ; Comprobar si hay que activar/desactivar
     LDA 1102H
     ORA A
     JZ PONA0  ; Hay que poner a 1

PONA1:
     LDA 1100H   ; Se obtiene el dato
     ORA C       ; y se activa el bit con la máscara
     JMP FIN

PONA0:
     MOV A, C    ; Hay que invertir la máscara
     CMA
     MOV C, A
     LDA 1100H   ; Se toma el dato
     ANA C       ; y se desactiva el bit

FIN:
     STA 1103H
     RST 1

END
```

10.5 Búsqueda de datos

Ejercicio 8.5.1.1

Respuestas a las cuestiones

1. Si cada fila tiene 8 columnas y existen 32 filas, una simple multiplicación nos dará el número de elementos que tiene la tabla: 32x8 = 256.

2. Puesto que la tabla comienza en la dirección `0200H` y tiene 256 elementos (`100H` en hexadecimal), la dirección de fin sería la `02FFH`. En general, la dirección de fin siempre será `Inicio+N° bytes-1`.

3. A partir de la fórmula anterior se puede deducir la siguiente: `N° bytes=Fin-Inicio+1`. En este caso se tendría: `13FFH-1200H+1=400H`, es decir, la tabla tendría 512 bytes. Si las filas tienen 8 columnas, eso significaría que existen 64 filas.

Ejercicio 8.5.2.1

Respuestas a las cuestiones

1. La suma de los elementos indicados es `C119H`.

2. Cualquier sistema basado en el 8085 contará con un bus de direcciones de 16 bits, que es el número de líneas con que cuenta este microprocesador para direccionar la memoria, por lo que no podrían tratarse tablas de mayor tamaño. Un contador de 16 bits, por tanto, será suficiente en todos los casos.

3. A diferencia de la instrucción `DCR`, que actúa sobre registros de 8 bits y afecta al bit Z del registro de estado, la instrucción `DCX` empleada con registros de 16 bits no afecta en ningún momento al registro de estado. No es posible, por tanto, realizar un salto condicional tras ejecutar la instrucción `DCX`, porque el estado del bit Z dependerá de cualquier operación previa, pero no de la reducción del contador. Por eso hay que recurrir a otro tipo de comprobación, siendo una técnica habitual llevar una parte del contador al acumulador y realizar una operación lógica OR con la otra parte. El resultado solamente puede ser cero si el contador contiene el valor 0.

   Una técnica alternativa consiste en tratar el contador de 16 bits como dos partes independientes, un LSB y un MSB. A cada ciclo del bucle se decrementa el LSB y, si no es cero, se inicia un nuevo ciclo. Cuando se llega a cero se pasa a decrementar el MSB y, si no es cero, se inicia un nuevo ciclo.

En éste se reducirá de nuevo el MSB, que pasará del valor 00H a FFH. De esta forma cada ciclo del MSB producirá 256 ciclos del LSB. El código quedaría de la siguiente forma:

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     ; Dirección inicial = 0000H + 10H filas x 20H columnas
     LXI D, 0200H
     ; Número de elementos = 10H filas x 20H columnas
     LXI B, 0200H
     LXI H, 0000H     ; Suma

BUCLE:
     LDAX D      ; Leer un elemento de la tabla
     ADD L ; y sumarlo con LSB suma
     MOV L, A    ; Guardar LSB suma
     MOV A, H    ; Tomar MSB suma
     ACI 00H     ; y sumar acarreo
     MOV H, A    ; Guardar MSB

     INX D ; Siguiente elemento de la tabla

     DCR C ; Reducir LSB
     JNZ BUCLE

     DCR B ; Reducir MSB
     JNZ BUCLE

     SHLD 1100H  ; Guardar suma

     RST 1
END
```

Ejercicio 8.5.3.1

Respuestas a las cuestiones

1. El valor mínimo es 00H y el máximo FFH, algo que era de esperar puesto que se examina un área bastante extensa de la memoria y dichos valores son el menor y mayor que pueden almacenarse en un celdilla de memoria.

2. En el mapa de memoria del uP-2000 (figura 33) se indica que la dirección de inicio de la EPROM es la 0000H y la de fin la 0FFFH, usando la fórmula Número bytes = Dir final - Dir inicial + 1 se tiene que 0FFFH - 0000H + 1 = 1000H.

Ejercicio 8.5.3.2

Respuestas a las cuestiones

1. Al facilitar el dato `00H` como valor de referencia el programa finaliza en el primer ciclo del bucle, ya que el primer byte de la EPROM, que se devuelve como resultado, es `20H` y, por tanto, superior a dicho valor. La dirección escrita en `1101H-1102H` será, por tanto, la `0000H`. Si el valor de referencia es `FFH` no será posible encontrar ningún valor mayor, por lo que el programa finaliza, tras recorrer toda la EPROM, sin llegar a escribir ningún resultado en las posiciones `1101H-1102H` y `1103H`.

2. La salida por la segunda instrucción `RST 1` solamente se dará cuando el bucle llegue a su fin, es decir, cuando se haya recorrido toda la EPROM. En ese momento la pareja de registros `DE` contendrá la dirección `1000H`, es decir, apuntará al primer byte de RAM del sistema, una posición que no habrá llegado a leer. El único caso en que el programa terminará por este camino será cuando el dato de referencia dado sea `FFH`, no pudiéndose encontrar un valor mayor.

Ejercicio 8.5.4.1

Respuestas a las cuestiones

1. La solución propuesta no mantiene el contador de coincidencias en un registro que sea necesario guardar al final en la memoria, sino que emplea una pareja de registros como puntero a la posición `1101H`, cuyo contenido es incrementado directamente, sin necesidad de ser transferido previamente a un registro. Esto es posible gracias al registro `M` que representa el uso de la pareja `HL` como puntero a memoria. Dado que la pareja `HL` también se usa como puntero para recuperar el dato de referencia, y poder compararlo con los bytes leídos de la EPROM, es necesario intercambiar los punteros de forma que uno sea el activo y el otro quede almacenado temporalmente. Lo habitual es usar la pareja de registros `DE` y la instrucción `XCHG` con este fin, pero esa pareja está ocupada por la dirección de la EPROM a leer. Por eso se introduce uno de los punteros en la pila y se recurre a la instrucción `XTHL` para intercambiarlo con el contenido actual de la pareja `HL`.

Ejercicio 8.5.5.1

Respuestas a las cuestiones

1. Una vez ejecutado el programa en la posición de memoria `1100H` queda el valor `1BH`, lo que significa que se han efectuado 27 sustituciones.

2. No sería posible efectuando una simple copia de la EPROM a RAM, ya que en el programa monitor existirán instrucciones de salto cuya dirección de destino se encuentra en la ROM, de forma que al ejecutarlo desde RAM en un momento u otro se volverá de nuevo a la zona de memoria EPROM.

3. En el proceso de copia habría que localizar todas las referencias de memoria a la zona de EPROM y sustituirlas por las nuevas direcciones en RAM, solamente de esa forma sería posible la ejecución desde RAM.

Ejercicio 8.5.5.2

Respuestas a las cuestiones

1. El programa consta de dos bucles, identificados por las etiquetas `BUC1` y `BUC2`. Se encuentran anidados, de forma que `BUC2` se ejecuta por cada ciclo de `BUC1`.

2. El bucle exterior está controlado por la pareja de registros `BC`, un contador de 16 bits, mientras que el interno lo controla el registro `C`, usando 8 bits. Al inicio del bucle exterior se guarda la pareja `BC` en la pila, recuperándose al salir del bucle interior. Esto permite utilizar un mismo registro, en este caso `C`, para los dos bucles.

3. Delante de la etiqueta `BUC1` puede verse que la pareja `BC` toma el contenido de la pareja de registros `HL` que, previamente, ha recuperando el contenido de las posiciones de memoria `1105H-1106H`. Será el dato de 16 bits almacenado en dicha dirección, por tanto, el que establezca el número de ciclos del bucle exterior. En cuanto al interior, el valor de `C` se obtiene, a través del acumulador, de la posición `1104H`. Este bucle se ejecutará tantas veces como indique el valor de esa dirección de memoria multiplicado por el dato almacenado en `1105H-1106H`, ya que es un bucle anidado.

4. La pareja de registros `DE` se utiliza como puntero para ir inspeccionando la memoria, concretamente a partir de la dirección especificada en las posiciones `1100H-1101H` que se recuperan al inicio del programa. `HL` tiene una función similar, usándose como puntero de acceso a la memoria, si

bien su dirección se recupera de las posiciones 1102H–1103H al inicio de cada ciclo del bucle exterior.

5. Buscando en el código del programa instrucciones de almacenamiento en memoria, como STA, STAX o MOV M, R, puede deducirse fácilmente que los datos sobre los que se escribe son los apuntados por la pareja de registros DE, es decir, aquellos cuya dirección se indica en las posiciones de memoria 1100H–1101H. Los datos escritos en esa zona de memoria se recuperan mediante la instrucción MOV A, M, por lo que proceden de las posiciones apuntadas por la pareja de registros HL. Concretamente se sustituye un byte apuntado por DE que coincide por otro apuntado por HL por el valor siguiente a este último.

6. El efecto de la primera ejecución del programa dependerá de los valores que hubiese originalmente en la zona de memoria desde la dirección 1200H en adelante. Es posible detectar algún cambio si se han encontrado coincidencias. La segunda ejecución, sin embargo, será totalmente clara, ya que los datos introducidos en las posiciones 1200H–1203H habrán sido sustituidos por otros, procedentes de los datos alojados en 1300H.

7. El código comentado está a continuación. El objetivo del programa es inspeccionar una zona de memoria, cuya dirección de inicio se indica en 1100H–1101H y su longitud en 1105H–1106H, buscando valores que se encuentran en una tabla de doble entrada, de forma que por cada coincidencia de un byte con la primera columna de esa tabla dicho byte es sustituido por el dato de la segunda columna. La tabla de traducción (suele llamársele *tabla XLAT*) está alojada en la dirección indicada en las posiciones 1102H–1103H y tiene tantas filas como indica el dato de la posición 1104H, con dos columnas cada una. Al introducir los valores de la cuestión anterior en memoria, se ha creado una tabla de traducción de cuatro filas, colocada a partir de la dirección 1300H. Esa tabla tendría la siguiente estructura:

00H	**AAH**
1FH	**BDH**
20H	**15H**
11H	**EDH**

Estructura de la tabla XLAT.

Con esta tabla el programa examinará 100H (256d) bytes a partir de la dirección 1200H sustituyendo los bytes indicados por la primera columna de la tabla XLAT por los datos de la segunda.

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H

     LHLD 1100H  ; Dir. bloque a examinar
     XCHG        ; en DE

     LHLD 1105H  ; Longitud bloque
     MOV B, H    ; en BC
     MOV C, L
BUC1:
     PUSH B      ; Guardar contador bloque

     LHLD 1102H ; Dir de la tabla en HL

     LDA 1104H   ; Contador de tabla XLAT
     MOV C, A    ; en C

     LDAX D      ; Tomar un byte del bloque
BUC2:
     CMP M       ; Y comparar con un byte de la tabla
     JNZ NOCOI

     INX H       ; Tomar el byte a poner en su lugar
     MOV A, M    ; y sustituir
     STAX D

     JMP FIN     ; No es necesario seguir buscando
NOCOI:
     INX H       ; Siguiente elemento tabla XLAT
     INX H

     DCR C       ; Queda un elemento menos
     JNZ BUC2
FIN:
     POP B       ; Restaurar contador bloque

     INX D       ; Siguiente byte del bloque

     DCR C       ; Hasta procesarlo completo
     JNZ BUC1
     DCR B
     JNZ BUC1

     RST 1
END
```

Ejercicio 8.5.6.1

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI H, 1102H     ; Apunta al número de filas

     LDA 1104H  ; Examinar la fila
     ORA A      ; Comprobar que no sea 0
     JZ INCOR
     CMP M      ; Ni mayor que el número máximo
     JNC INCOR

     INX H      ; Avanzar al número de columnas

     LDA 1105H  ; Examinar la columna
     ORA A      ; Comprobar que no sea 0
     JZ INCOR
     CMP M      ; Ni mayor que el número máximo
     JNC INCOR

     ; El número de fila y columna es correcto

     LHLD 1100H ; Dirección inicio de la tabla
     MVI D, 00H
     LDA 1103H
     MOV E, A   ; Elementos por fila

     LDA 1104H  ; Se recupera la fila

BUCLE:
     DCR A      ; Reducir número fila
     JZ FIN     ; Si es 0, terminar

     DAD D      ; Si no, sumar a dirección
     JMP BUCLE

FIN:
     LDA 1105H  ; Obtener número de columna
     DCR A
     MOV E, A
     DAD D      ; y sumarlo a la dirección

     SHLD 1106H ; Se guarda la dirección
     MOV A, M   ; Se lee el dato
     STA 1108H  ; y se guarda

     RST 1

INCOR:
```

```
      LXI H, 0FFFFH ; Señalizar error
      SHLD 1106H
      MOV A, L
      STA 1108H

      RST 1
END
```

Ejercicio 8.5.6.2

Solución

```
CPU "8085.TBL"
HOF "INT8"

      ORG 1000H

      LXI D, 0000H ; Inicio de la ROM
      LDAX D       ; Tomar el primer byte

      MOV H, A     ; Y establecer como mayor
      MOV L, A     ; y menor hasta el momento

      LXI B, 1000H ; Longitud de la ROM

BUCLE:
      LDAX D       ; Se lee un byte

      CMP L        ; ¿Menor que L?
      JNC NOME    ; si no es así saltar

      MOV L, A     ; Dato menor

      XCHG
      SHLD 1102H
      XCHG

NOME:
      CMP H        ; ¿Mayor que H?
      JC NOMA      ; si no es así saltar

      MOV H, A     ; Dato mayor

      XCHG
      SHLD 1104H
      XCHG

NOMA:
      INX D        ; Siguiente posición

      DCR C        ; Continuar hasta el final
      JNZ BUCLE    ; de la EPROM
```

```
     DCR B
     JNZ BUCLE

     SHLD 1100H ; Guardar máximo y mínimo

     RST 1
END
```

Ejercicio 8.5.6.3

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LDA 1100H  ; Se obtiene el dato de referencia
     MOV D, A   ; en el registro D
     MVI E, 00H ; Y el contador de coincidencias en E

     LXI B, 1000H     ; Longitud de la ROM

BUCLE:
     MOV A, B   ; Comprobar si BC = 0
     ORA C
     JZ FIN     ; Si es así, terminamos

     DCX B      ; Retrocedemos a la posición previa

     LDAX B     ; Se lee un byte

     CMP D      ; y se compara con el dato
     JNZ BUCLE  ; Si no coincide, continuar

     ; Se ha encontrado una coincidencia
     INR E      ; Incrementar la cuenta
     JMP BUCLE  ; y continuar

FIN:
     MOV A, E ; Guardar el contador de coincidencias
     STA 1101H

     RST 1
END
```

Respuestas a las cuestiones

1. La primera ventaja es visible a primera vista, ya que el código de esta solución es más corto que el del ejercicio 8.5.4.1, tanto en versión código fuente como una vez ensamblado, formato en el que su tamaño se reduce en aproximadamente un 30%. Lo más importante, sin embargo, será la velocidad de ejecución, ya que en la versión original el dato de referencia se transfería de la memoria al microprocesador una vez por ciclo (para efectuar la comparación), y el contador de coincidencias era transferido tantas veces como se encontrará un valor igual al buscado. En esta nueva versión el dato de referencia es leído una sola vez, al igual que el contador de coincidencias se escribe una sola vez. Al reducirse el número de transferencias de manera tan drástica el programa resultará mucho más rápido.

10.6 Estructuración del código

Ejercicio 8.6.1.1

Respuestas a las cuestiones

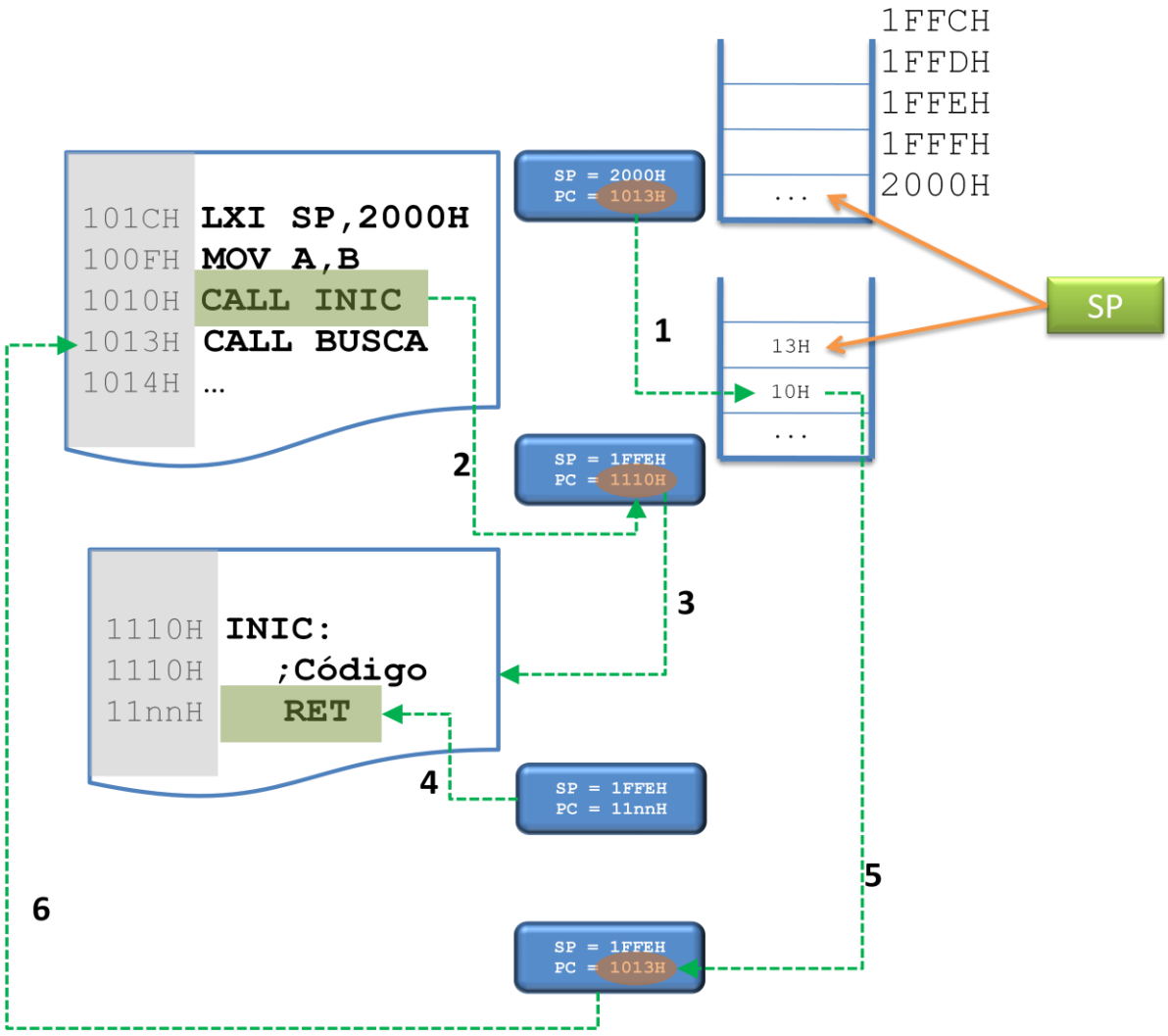
1. Cualquier programa que vaya a utilizar subrutinas debe comenzar siempre inicializando el puntero de pila (registro `SP`), ya que cada ejecución de la instrucción `CALL` implica que el contenido actual del contador de programa se almacene en la pila, recuperándose en el momento en que se ejecuta la instrucción `RET`.

2. El código de una subrutina es independiente del punto desde el que se le llame, por lo que no hay ningún problema en llamar a una subrutina desde otra.

3. Aunque no es algo habitual, una subrutina puede llamarse a sí misma siempre que exista un control sobre la realización de dicha llamada de forma que no sea incondicional, ya que en ese caso se entraría en un bucle infinito. La aplicación más lógica será la implementación de un procedimiento recursivo, técnica con la que se facilita la resolución de problemas de determinada complejidad.

4. No existe un límite de llamadas a subrutinas para un programa a menos que dichas llamadas estén anidadas, ya sea directa o indirectamente. Es decir, si en el bloque principal del programa existe un bucle que llama 1000 veces a una subrutina, por poner un ejemplo, no habrá ningún problema. Diferente sería que una rutina se llama a sí misma, o bien que llamase a otra que, a su vez, llamase a la primera. En ese caso las direcciones se irían acumulando en la pila y el valor del registro `SP` iría reduciéndose paulatinamente. Si el número de llamadas es muy grande, podría darse el caso de que los datos almacenados en la pila ocupasen toda la memoria RAM, sobrescribiendo incluso el propio programa. El límite en cuento a llamadas anidadas, por tanto, lo impone la memoria disponible para la pila.

5. El esquema correspondiente a la primera llamada podría ser el mostrado en la siguiente figura:



Esquema de funcionamiento de la primera llamada en el ejercicio 8.6.1.1.

Las direcciones de memoria que aparecen en el margen izquierdo de los bloques de código no son reales, su única utilidad es hacer ver el valor que toman el contador de programa y los datos que se almacenan en la pila.

Al llegar a la instrucción resaltada del bloque superior: `CALL INIC`, el registro `SP` apunta a la dirección `2000H` y el contador de programa tiene ya la dirección de la siguiente instrucción. En ese momento se reduce la dirección del `SP` y se almacena en ella el MSB del `PC`, repitiéndose la operación con el LSB del contador de programa. El registro `SP` queda apuntando al último dato almacenado y el `PC` se actualiza para saltar a la subrutina.

En ese momento se ejecuta la subrutina. Al llegar al final, la instrucción `RET` se encarga de tomar el dato apuntado por `SP` y almacenarlo como LSB del contador de programa, a continuación se incrementa el puntero de pila y se recupera el MSB, produciéndose un nuevo incremento de `SP` que queda de nuevo con el valor `2000H`. El contador de programa apunta a la siguiente instrucción del programa principal, que proseguirá su ejecución.

Ejercicio 8.6.2.1

Respuestas a las cuestiones

1. La principal ventaja es que el código que llama a la subrutina, cuando recupera el control, puede servirse del registro de estado para realizar un salto condicional directamente, sin necesidad de comparar si un registro, en el que se ha devuelto el resultado, está a 0 o a 1. Además también hay que tener en cuenta que no se precisa un registro adicional para devolver ese resultado.

Ejercicio 8.6.3.1

Respuestas a las cuestiones

1. Siempre cabe la posibilidad de que la subrutina espere que todos sus parámetros estén alojados en una tabla en memoria, recibiéndose en una pareja de registros la dirección de memoria donde el programa principal la haya alojado. De esta forma se ocuparía únicamente esa pareja de registros y la subrutina tendría toda la información necesaria para realizar su trabajo. Lo que no debe hacer nunca una subrutina es asumir que esa tabla de parámetros va a encontrarse en una posición de memoria concreta, ya que cada programa puede colocarla en un punto distinto según sus necesidades.

Ejercicio 8.6.3.2

Respuestas a las cuestiones

1. Cuando desde el programa principal se invoca a una subrutina, el microprocesador almacena en la pila la dirección de memoria a la que tendrá que volver cuando se ejecute la instrucción `RET`. Si en el interior de la subrutina se introduce un dato y no es extraído antes de ejecutar dicha instrucción, lo que ocurrirá es que el dato se interpretará como la dirección a la que hay que volver. El efecto, como es fácil imaginar, será imprevisible.

2. La instrucción `PUSH PSW` almacena en la pila la pareja de registros acumulador-registro de estado, de forma que en el interior de una subrutina puedan ser modificados, en este caso para comprobar si el contador ha llegado a cero, y al final restaurarse de forma que el programa principal los encuentre como los dejó. Esto es importante si, por ejemplo, el programa principal invoca a la subrutina desde el interior de un bucle, en el que también necesita usar el registro de estado, y no es deseable que éste se vea modificado.

3. Sí sería posible, siempre que se necesite preservar dicha pareja de registros. Para devolver un dato a través de un bit del registro de estado, por ejemplo el bit de acarreo, primero habría que hacer `POP PSW`, para recuperar el registro de estado, y a continuación efectuar la activación o desactivación de dicho bit. Si se hiciese a la inversa, ejecutando la instrucción `POP PSW` justo antes de salir de la subrutina, cualquier modificación del registro de estado se perderá al restaurar el contenido previo de dicho registro.

Ejercicio 8.6.3.3

1. En el programa aparecen tres instrucciones `CALL`, pero esto no implica necesariamente que existan tres subrutinas, ya que una misma subrutina se la puede llamar desde distintos puntos de un programa. Cada una de esas instrucciones, no obstante, va acompañada de una etiqueta distinta, por lo que, en principio, sí parece que ése es el número de subrutinas. Buscando el número de instrucciones `RET`, sin embargo, se comprueba que solamente hay dos, por lo que es fácil concluir que existen dos subrutinas puesto que toda subrutina ha de acabar con dicha instrucción.

2. La primera subrutina comienza en la etiqueta `F1` y finaliza con la instrucción `RET` que hay más abajo, mientras que la segunda se inicia en la etiqueta `MA` y concluye con la instrucción `RET` que hay al final. Las etiquetas `F1` y `MA` podrían cambiarse por `FACT` y `MULT_HLA`, respectivamente, algo más descriptivas según la finalidad de cada subrutina.

3. Dicha subrutina toma el contenido de la pareja de registros `HL` y lo multiplica por el contenido del acumulador, dejando el resultado en `HL`. Salvo esta pareja de registros, no se modifica ningún otro, ni siquiera el registro de estado.

4. La etiqueta `RH` debe colocarse entre la instrucciones `LXI H, 0000H` y `DAD D`, que corresponden a la inicialización de `HL` (solamente una vez) y la suma de `DE` a `HL`, primera instrucción del bucle.

5. Esa subrutina calcula el factorial de un número, que toma como entrada en el acumulador, y devuelve el resultado en la pareja de registros `HL`. Se modifican esos registros y también el registro de estado.

6. El programa toma el dato de 8 bits introducido en la posición de memoria `1100H`, obtiene su factorial y devuelve el resultado de 16 bits en las posiciones de memoria `1101H–1102H`. El código completo del programa tras agregar comentarios, y cambiar algunas etiquetas por otras más claras, es el siguiente:

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI SP, 2000H

     LDA 1100H  ; Tomar el dato de 8 bits
     CALL FACT  ; Calcular su factorial
     SHLD 1101H ; y guardar el resultado

     RST 1      ; Fin del programa

; ----- Subrutina que calcula el factorial
;
; Entradas: A = Número del que se desea
;           calcular el factorial
; Salidas: HL = Factorial del número
; Modifica: HL, A, registro de estado
;
FACT:
    LXI H, 0001H      ; Factorial de 1 = 1

FACT1:
     CPI 02H          ; ¿A < 2?
     JC FIN_FAC       ; Si es así, no continuar

     PUSH PSW         ; Guardar acumulador
     DCR A            ; Reducirlo
     CALL FACT1       ; y calcular factorial de A-1

     POP PSW          ; Recuperar acumulador
     CALL MULT_HLA    ; y multiplicar por HL

FIN_FAC:
     RET
; ----- Fin de la subrutina auxiliar FACT

; ----- Subrutina para multiplicar HL por A
;
; Parámetros de entrada:
;     HL = Multiplicando, A = Multiplicador
; Parámetros de salida: HL = Producto de HLxA
; Modifica los registros: Únicamente HL
;
MULT_HLA:
     PUSH PSW   ; Guardar los registros que van a
     PUSH D     ; a ser usados en la subrutina

     MOV D, H   ; Multiplicando
     MOV E, L   ; a DE
     LXI H, 0000H

REP_HLA:
```

```
      DAD D       ; HL = HL + DE
      DCR A
      JNZ REP_HLA ; A > 0, seguir

      POP D
      POP PSW
      RET
; ----- Fin de la subrutina de multiplicación

END
```

7. La subrutina `FACT` calcula el factorial de un número utilizando un método recursivo que se apoya en la conocida regla que dice que `N!=N(N-1)!`, es decir, el factorial de `N` se puede calcular multiplicando `N` por el factorial de `N-1`. Se sabe que el factorial de `N` es 1, por lo que basta con ir reduciendo `N` hasta llegar a ese valor y, en el proceso de vuelta de la recursividad, ir calculando el producto por cada `N`. En este caso `N` equivaldría al valor almacenado en el acumulador.

   Otra posibilidad sería usar un bucle que recorriese los valores desde 1 hasta el dato contenido en el acumulador, en lugar de realizar llamadas recursivas. De hecho sería una alternativa más eficiente. El cálculo del factorial mediante recursividad es, no obstante, un recurso didáctico habitual para explicar cómo funciona la recursividad.

Ejercicio 8.6.4.1

Solución

```
CPU "8085.TBL"
HOF "INT8"

      ORG 1000H

      CALL VRF_LIM     ; Comprobar los parámetros
      JC INCOR         ; Terminar si no son válidos

      LDA 1100H        ; Se recupera la fila

      LXI H, 200H      ; Inicio de la tabla
      LXI D, 20H       ; Elementos por fila

BUCLE:
      DCR A            ; Reducir número fila
      JZ FIN           ; Si es 0, terminar

      DAD D            ; Si no, sumar a dirección
      JMP BUCLE
```

```
FIN:
     LDA 1101H        ; Obtener número de columna
     DCR A
     MOV E, A
     DAD D            ; y sumarlo a la dirección

     SHLD 1102H       ; Se guarda la dirección
     MOV A, M         ; Se lee el dato
     STA 1104H        ; y se guarda

     RST 1

INCOR:
     LXI H, 0FFFFH    ; Señalizar error
     SHLD 1102H
     MOV A, L
     STA 1104H

     RST 1

; ----- Subrutina que comprueba los límites
;
; Modifica los registros: A, Estado
; Devuelve: CY=1 si hay error, CY=0 todo bien
;
VRF_LIM:
     LDA 1100H  ; Examinar la fila
     ORA A      ; Comprobar que no sea 0
     JZ FALLO
     CPI 21H    ; Ni mayor que 32
     JNC FALLO

     LDA 1101H  ; Examinar la columna
     ORA A      ; Comprobar que no sea 0
     JZ FALLO
     CPI 09H    ; Ni mayor que 8
     JNC FALLO

     ; Los límites son correctos, hay que
     ORA A      ; poner el acarreo a 0
     RET

FALLO:          ; Hay un fallo en los límites
     STC        ; Activar el acarreo
     RET
; ----- Fin de la subrutina que comprueba los límites
END
```

Ejercicio 8.6.4.2

Solución

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI SP, 2000H

     LHLD 1100H  ; Dir. bloque a examinar
     XCHG        ; en DE

     LHLD 1105H  ; Longitud bloque
     MOV B, H    ; en BC
     MOV C, L

     LHLD 1102H  ; Dir de la tabla en HL

     LDA 1104H   ; Elementos de la tabla XLAT

     CALL XLAT   ; Procesar

     RST 1       ; Fin del programa


; ----- Subrutina XLAT
;
; Entradas: DE = Dir bloque a procesar
;           HL = Dir de la tabla XLAT
;           BC = Longitud del bloque a procesar
;            A = Número de elementos de la tabla
;
; Salidas: Ninguna
;
; Modifica: Nada
;
XLAT:
     PUSH D      ; Preservar DE y BC, que van a ser
     PUSH B      ; modificados

BUCXL:
     CALL XLAT_A ; Procesar un byte

     INX D       ; Siguiente byte del bloque

     DCR C       ; Hasta procesarlo completo
     JNZ BUCXL
     DCR B
     JNZ BUCXL

     POP B       ; Restaurar los registros DE y BC
     POP D
```

```
     RET
; ----- Fin de la subrutina XLAT

; ----- Subrutina auxiliar XLAT
;
; Entradas: A  = Elementos tabla XLAT
;     HL = Dirección de la tabla XLAT
;     DE = Dirección donde debe leer/escribir el dato
;
; Salidas:  Ninguna
;
; Modifica: Nada
;
XLAT_A:
     PUSH H      ; Preservar HL, BC y PSW que
     PUSH B      ; van a ser modificados
     PUSH PSW

     MOV C, A    ; Número de elementos como contador

BUCXLA:

     LDAX D      ; Tomar un byte del bloque

     CMP M       ; Y comparar con un byte de la tabla
     JNZ NOCOI

     INX H       ; Tomar el byte a poner en su lugar
     MOV A, M    ; y sustituir
     STAX D

     JMP FINXA   ; No es necesario seguir buscando

NOCOI:
     INX H       ; Siguiente elemento tabla XLAT
     INX H

     DCR C       ; Queda un elemento menos
     JNZ BUCXLA

FINXA:
     POP PSW     ; Restaurar PSW, BC y HL
     POP B
     POP H

     RET
END
; ----- Fin de la subrutina auxiliar XLAT
```

10.7 Ordenar datos

Ejercicio 8.7.1.1

Respuestas a las cuestiones

1. Puesto que el número de elementos se indica mediante un valor de 16 bits, teóricamente podrían ordenarse hasta 2^{16} elementos, es decir, 65536 elementos. Hay que tener en cuenta, sin embargo, que parte de la memoria estará ocupada por la EPROM del sistema, una zona que no es modificable. En el sistema uP-2000 el número de elementos estaría limitado por la memoria RAM disponible, un total de 4096 bytes, parte de la cual está ocupada por el programa (unos 210 bytes). También hay que tener en cuenta el espacio que necesita la pila, si bien en este caso es muy reducido ya que no hay llamadas de tipo recursivo y se usa, básicamente, para preservar el contenido de los registros. Serían suficientes unos 32 bytes. Esto nos dejaría libres, finalmente, algo más de 3800 bytes. Ése sería el máximo número de elementos que, en la práctica, podría ordenar el programa.

2. Si el número de elementos indicado en `1102H-1103H` es 100, el bucle exterior se ejecutará 99 veces, que son los ciclos necesarios para comparar un elemento dado con el resto de la lista. En cuanto al bucle interior o anidado, se ejecutará 99 veces en la primera llamada, 98 en la segunda, 97 en la tercera y así sucesivamente, ya que a cada ciclo queda un elemento menos por ordenar. Se trata de una sucesión aritmética y, por tanto, puede resolverse como $(n^2+n)/2$, siendo `n` el máximo valor y asumiendo que el menor es 1. El resultado es 4950 veces. Ése será también el número de veces que se invoque a la subrutina `COMPA` para realizar comparaciones.

3. La instrucción `CC` es una llamada a subrutina condicional. Cuando el bit de acarreo está a 1, esta instrucción se comporta como lo haría `CALL`. Si dicho bit está a 0, no hace nada. Sería equivalente a tener el código siguiente:

```
    ...
    CALL COMPA

    JNC NOINT
    CALL INTERC

NOINT:
    INX D
    ...
```

Como puede verse, sería necesario introducir un salto condicional que evitase la ejecución de la instrucción `CALL`, de forma que a ésta solamente se llegase en caso de el bit de acarreo esté activo.

4. Los registros quedan inalterados, puesto que lo primero que hace la subrutina `ORDENA` es preservarlos, restaurándolos justo antes de salir. El programa principal, por tanto, los encontrará tal y como los dejó antes de efectuar la llamada a dicha subrutina.

10.8 Lectura del teclado y visualización en el uP-2000

Ejercicio 8.8.1.1

Respuestas a las cuestiones

1. Al ejecutar el programa no es posible ver el dato en el campo de direcciones del uP-2000, ya que de inmediato aparece el mensaje – `80 85` que confirma que la ejecución del programa ha finalizado. Dicho mensaje lo muestra el programa monitor del uP-2000, al que se devuelve el control mediante la instrucción `RST 1`. Si se sustituye dicha instrucción por `HLT`, al ejecutar el programa (tras volver a ensamblarlo y transferirlo al uP-2000) se podrá ver el resultado sin problemas, ya que dicha instrucción detiene el 8085. Con la tecla **INIC** podrá devolver el control al programa monitor.

2. Esta subrutina, como la mayoría de las que ofrece el sistema uP-2000 en EPROM, alteran el contenido de todos los registros, por lo que si se tiene alguna información útil en ellos ésta se perderá. Para evitarlo habría que guardar las parejas de registros adecuadas en la pila, invocar a la subrutina y, al terminar ésta, recuperar la información de la pila.

3. No hay ningún problema en invocar múltiples veces a la subrutina con distintos datos a mostrar, el problema es que únicamente podrá verse el último puesto que la ejecución es inmediata, a pesar de que la pantalla irá mostrándolos todos pero a una velocidad que no permitirá apreciarlos.

Ejercicio 8.8.2.1

Respuestas a las cuestiones

1. Al ejecutar el programa con esa información, en el campo de direcciones aparecerá el valor `1000H` y en el campo de datos el valor `31H`. Éste corresponde al código de operación de la instrucción `LXI SP`, que es el contenido de esa dirección de memoria siempre que el programa esté alojado a partir de la posición `1000H`.

Ejercicio 8.8.2.2

Respuestas a las cuestiones

1. El resultado de la operación de suma no se vería afectado, puesto que dicha operación tiene la propiedad conmutativa, pero los dos operandos e mostrarían en el campo de direcciones del uP-2000 en orden inverso, es decir, primero el alojado en la posición `1101H` y después el de la posición `1100H`.

Ejercicio 8.8.3.1

Respuestas a las cuestiones

1. En el programa ofrecido como solución se ha optado por un método de inicialización que lee el registro de interrupciones, activa el bit WR, pone a 0 el bit correspondiente a la `RST 5.5` para desenmascararla y, finalmente, escribe el valor modificado de nuevo en el registro de interrupciones. Ésta sería la forma más correcta de hacerlo, no afectando así al estado de los demás bits de ese registro. Si se asume que el programa del ejercicio tiene el control total del sistema, también podría haberse optado por escribir directamente el valor adecuado en el registro de interrupciones, sin más. Bastaría con una instrucción `MVI A, 0EH` seguida de la instrucción `SIM`. El valor `0EH` corresponde a `0000 1110` en binario, de forma que se activaría el bit `WR` y desactivaría el correspondiente a la `RST 5.5`.

2. Las teclas **0** a **F** generan los códigos `00H` a `0FH`, respectivamente. Es lógico, de esta forma se permite usar esas teclas para introducir directamente valores sobre los que se puede operar, sin necesidad de realizar traducciones.

3. Los códigos de esas teclas son:

   - **EJEC**: `10H`
   - **POST**: `11H`
   - **GO**: `12H`
   - **S.ME ANT**: `13H`
   - **E.REG**: `14H`
   - **EJEC PASO**: `15H`

4. No, esas dos teclas no generan un código. La tecla **INTR VECT** no desencadena una `RST 5.5` sino una `RST 7.5`, mientras que la tecla **INIC** tiene la función de reiniciar el sistema.

5. Bastaría con agregar al inicio del programa una llamada a la subrutina situada en la dirección `031FH`, encargada de borrar la pantalla.

Ejercicio 8.8.3.2

Respuestas a las cuestiones

1. La mayoría de las subrutinas que ofrece el programa monitor del uP-2000 modifican todos los registros del 8085, por ello es preciso guardar aquellos que contienen información útil antes de invocarlas. En este programa el acumulador contiene los códigos de las teclas leídas, de ahí que sea preciso guardarlo y recuperarlo tras las llamadas. Lo mismo ocurre con la pareja de registros `HL`, usada como puntero donde se almacenarán los datos.

2. El programa no termina hasta que se introduce el valor `FFH`, por lo que podría utilizarse para escribir tantos bytes como se quiera, hasta llenar toda la memoria disponible.

Ejercicio 8.8.5.1

Respuestas a las cuestiones

1. El uso de la pila para guardar temporalmente datos sobre los que va a operarse, como se hace en este programa, resulta algo casi natural, pero no es la única opción. Se podría haber optado para guardar los bytes devueltos por la subrutina `LEE_BYTE` en posiciones de memoria concretas, desde las que los recuperarían las rutinas `VIS_SUMA` y `VIS_RESTA`. Esto podría resultar incluso más eficiente, ya que en la pila solamente pueden almacenarse parejas de registros y cada vez que se guarda un dato contenido en el acumulador también se guarda el registro de estado, algo que en este caso no interesa para nada.

Ejercicio 8.8.6.1

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     MVI A, 0EH       ; Desenmascarar RST 5.5
     SIM

     CALL 031FH       ; Borrar pantalla

     CALL 02BFH       ; Solicitar dirección

BUCLE:
     LDAX D           ; Leer byte apuntado por DE
```

```
     PUSH D           ; Guardar DE

     CALL 04D5H       ; Mostrar dato
     POP H            ; Recuperar DE en HL
     PUSH H
     CALL 04C9H       ; Para mostrar dirección

     CALL 044EH       ; Esperar operación

     CPI 11H          ; Si se ha pulsado POST
     JZ SIGU          ; avanzar a la siguiente posi-
ción

     CPI 10H          ; Si se ha pulsado EJEC
     JZ ANTE          ; retroceder a posición previa

     POP D            ; Sacar DE de la pila
     RST 1            ; y terminar

SIGU:
     POP D            ; Recuperar puntero en DE
     INX D            ; avanzando a la posición si-
guiente

     JMP BUCLE        ; y repetir

ANTE:
     POP D            ; Recuperar puntero en DE
     DCX D            ; avanzando a la posición si-
guiente

     JMP BUCLE        ; y repetir
END
```

Respuestas a las cuestiones

1. Cuando se dan los pasos indicados, avanzando desde la dirección `F000H`, se da el caso de que aparece como dato leído un valor que siempre coincide con el LSB de la dirección. Al ejecutarse la instrucción `LDAX D` el 8085 utiliza la pareja de registros `DE` como puntero, recuperando el contenido de la dirección de memoria indicada. De esta forma es posible leer la memoria EPROM y también la RAM, por ejemplo usando las direcciones `0100H` y `1000H` respectivamente. La cuestión es: ¿qué hay a partir de la dirección `F000H`? En la configuración del sistema uP-2000 no existe memoria ni dispositivo alguno asociado a esa dirección, por lo que no es posible leer nada. En consecuencia el bus de datos queda con el estado correspondiente a la última información que ha transportado. ¿Qué información es ésa? Pues precisamente el LSB del bus de direcciones, ya que en el 8085 está multiplexado con el bus de datos. Ésa es la razón de que el dato leído coincida siempre con los 8 bits menos significativos de la dirección.

Ejercicio 8.8.6.2

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     MVI A, 0EH       ; Desenmascarar RST 5.5
     SIM

     CALL 031FH       ; Borrar pantalla

     LXI H, 0001H     ; Configuración inicial

BUCLE:
     PUSH H           ; Guardar HL
     CALL 04C9H       ; Mostrarlo en campo direcciones

     CALL 044EH       ; Esperar operación

     POP H            ; Recuperar HL

     CPI 11H          ; Si se ha pulsado POST
     JZ IZQU          ; llevar hacia la izquierda

     CPI 10H          ; Si se ha pulsado EJEC
     JZ DERE          ; llevar hacia la derecha

     RST 1       ; Cualquier otra tecla termina
```

```
IZQU:
     CALL SEP_NIB     ; Separar nibbles
     MOV L, A         ; y colocarlos en su nueva
     MOV H, B         ; disposición

     JMP BUCLE        ; y repetir

DERE:
     CALL SEP_NIB     ; Separar nibbles
     MOV H, A         ; y colocarlos en su nueva
     MOV L, B         ; disposición

     JMP BUCLE        ; y repetir

; ----- Subrutina que separa la dirección de
;       16 bits en cuatro nibbles
;
SEP_NIB:
     XCHG             ; HL <-> DE
     MOV A, E        ; Comenzar obteniendo los nibbles
     CALL NIBBLES     ; del byte bajo de la dirección
     PUSH H           ; y guardarlos

     MOV A, D         ; Obtener los nibbles
     CALL NIBBLES     ; del byte alto

     POP D            ; Recuperar los nibbles previos

     MOV A, L         ; Combinar los nibbles de
     ORA D            ; cada byte

     MOV B, A         ; Devolviendo uno en A
     MOV A, H         ; y otro en B
     ORA E

     RET

; ----- Subrutina que extrae los nibbles de un byte
;       preparándolos para ser intercambiados
;
NIBBLES:
     MOV H, A   ; Tomar el byte de A y guardarlo en H

     RLC        ; Antes de rotarlo hacia la izquierda
     RLC
     RLC
     RLC
     ANI 0F0H   ; quedándose con los 4 bits altos

     MOV L, A   ; Guardar nibble
     MOV A, H   ; para tomar en A el otro

     RRC        ; Que se rotará hacia la derecha
     RRC
```

```
      RRC
      RRC
      ANI 0FH     ; quedándose con los 4 bits bajos

      MOV H, A    ; Se devuelven en H y L

      RET
END
```

10.9 Introducción de retardos

Ejercicio 8.9.1.1

Respuestas a las cuestiones

1. El contador avanza tan deprisa que resulta imposible apreciar los cambios en los últimos dos dígitos, incluso el tercero, contando de derecha a izquierda, resulta difícil de leer ya que cambia prácticamente cada décima de segundo.

2. Podría introducirse, justo detrás de la llamara a la subrutina de visualización, un bucle de retardo que hiciese que el programa fuese más lento. Por ejemplo:

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     CALL 031FH       ; Borrar pantalla

     LXI H, 0000H     ; Configuración inicial

BUCLE:
     PUSH H           ; Guardar HL
     CALL 04C9H       ; Mostrarlo en campo direcciones


     LXI B, 1800H     ; Valor obtenido a tanteo

BRET:
     DCR C       ; Bucle que pierde tiempo
     JNZ BRET
     DCR B
     JNZ BRET

     POP H            ; Recuperar HL
     INX H            ; e incrementar su contenido

     MOV A, H         ; Si aún no se ha vuelto a 0000H
     ORA L            ; continuar
     JNZ BUCLE

     RST 1            ; Fin del programa
END
```

3. Mediante tanteo, modificando el valor asignado a la pareja de registros BC que controla el bucle de retardo y ejecutando el programa, sería posible

calcular aproximadamente un tiempo concreto entre ciclos para ajustar el funcionamiento del programa. Otra posibilidad sería calcular, dependiendo de la frecuencia de reloj del 8085 en el uP-2000 y los ciclos que consumen las instrucciones del bucle, el número de ciclos exacto según el tiempo de espera deseado.

Ejercicio 8.9.2.1

Respuestas a las cuestiones

1. Se sabe que el número de ciclos a consumir es de 200.000 para provocar un retardo de una décima de segunda. De ésos hay que descontar los que consumen las instrucciones que hay al inicio de la rutina, de inicialización, y las que hay al final, que restauran los registros. Los ciclos que consumen esas instrucciones son: `PUSH B`: 12 ciclos, `PUSH PSW`: 12 ciclos, `LXI B`: 10 ciclos, `POP PSW`: 10 ciclos, `POP B`: 10 ciclos y `RET`: 10 ciclos. En total son 62 ciclos, por lo que quedarían 200.000-62=199.938 por consumir.

   Las instrucciones que forman el bucle consumen en total 24 ciclos, por lo que dividiendo 199.938 entre 24 se obtiene el número de veces que ha de ejecutarse el bucle: unas 8331 (`208BH` en hexadecimal).

2. El número de ciclos total que consume la subrutina es de 200.006 (8331x24+64). Multiplicando por 0.5, que es el tiempo empleado por cada ciclo, obtenemos el número de microsegundos: 100.003. El tiempo empleado por la subrutina, por tanto, es de 0.100003 segundos, es decir, una décima de segundo.

3. Bastaría con llamar dos veces consecutivas a la rutina de retardo.

Ejercicio 8.9.3.1

Respuestas a las cuestiones

1. Al leer a partir de esas dirección 11 bytes consecutivos se obtiene la secuencia: `C5H 06H 8BH 05H 00H 00H C2H 77H 0BH C1H C9H`. Buscando en la tabla de instrucciones los códigos de operación, y los operandos que les corresponden, se obtiene el código siguiente:

```
0B74H       PUSH B
0B75H       MVI B, 8BH
0B77H       DCR B
0B78H       NOP
0B79H       NOP
0B7A        JNZ 0B77
0B7DH       POP D
0B7EH       RET
```

Como puede verse, es un bucle en el que el registro `B` actúa como contador, en el que se ejecutan dos instrucciones `NOP` aparte del decremento de `B` y el salto condicional, y que se repite 139 veces (`8BH`).

Las instrucciones `DCR B` y `NOP` requieren 4 ciclos cada una, mientras que el salto condicional consume 10 ciclos cada vez que vuelve a la dirección indicada. Son 22 ciclos por tanto, que repetidos 139 veces hacen un total de 3058 ciclos. Cada microsegundo se ejecutan 2 ciclos, por lo que el bucle empleará un total de 1529 microsegundos, es decir, aproximadamente 1.5 milisegundos.

Ejercicio 8.9.5.1

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     MVI A, 0EH       ; Desenmascarar RST 5.5
     SIM

     CALL 031FH       ; Borrar pantalla

     MVI C, 0AH       ; Retardo inicial

BUCLE: ; Cuerpo principal del programa

     MOV A, C    ; Se muestra en el campo de datos
     PUSH B           ; el retardo actual
     CALL 04D5H
     CALL 044EH       ; Y se espera una tecla

     POP B            ; Restaurar BC

     CPI 10H          ; Según la tecla pulsada
     JZ T_EJEC        ; se salta a la etiqueta
     CPI 11H          ; apropiada
     JZ T_POST
     CPI 12H
     JZ T_GO

     RST 1            ; Cualquier otra tecla termina

T_EJEC:
     DCR C            ; EJEC reduce el retardo
     JMP BUCLE

T_POST:
     INR C            ; POST incrementa el retardo
     JMP BUCLE

T_GO:
     CALL VIS_CONT    ; Y GO lanza la visualización
     JMP BUCLE

; ----- Subrutina que visualiza el bucle
;
VIS_CONT:
     LXI H, 0100H     ; Partir desde 0100H

BUC_VIS:
     PUSH H           ; Se guardan los registros
```

```
     PUSH B           ; con datos útiles

     CALL 04C9H       ; Mostrar dirección

     POP B            ; Recuperar el retardo en C
     CALL RETARDO     ; Rutina de retardo

     POP H            ; Recuperar dato a mostrar

     DCX H            ; Reducirlo
     MOV A, L
     ORA H

     JNZ BUC_VIS      ; Y seguir hasta 0000H

     PUSH B
     CALL 04C9H       ; Mostrar el último valor
     POP B

     RET

; ----- Subrutina de retardo variable
;
; Entradas:
;     C = Número de décimas de segundo a esperar
RETARDO:
    PUSH B ; Guardar registros que se modificarán

BRET:
    CALL RET1DEC; Una décima

    DCR C  ; Hasta completar el número indicado
    JNZ BRET

    POP B
    RET
; ----- Fin de la subrutina de retardo

; ----- Subrutina de retardo de una décima de segundo
;
RET1DEC:
    PUSH B ; Guardar registros que se modificarán
    PUSH PSW

    LXI B, 208DH ; Número de repeticiones

BRET1:
    DCX B  ; 6 ciclos
    MOV A, B    ; 4 ciclos
    ORA C  ; 4 ciclos
    JNZ BRET1   ; 10 ciclos

    POP PSW
    POP B
```

```
    RET
; ----- Fin de la subrutina de retardo

END
```

Ejercicio 8.9.5.2

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     MVI A, 00H       ; Establecer los segundos
     STA 1102H
BUCLE:
     LDA 1102H
     CALL 04D5H       ; Mostrar segundos

     LDA 1100H
     MOV H, A
     LDA 1101H
     MOV L, A
     CALL 04C9H       ; Mostrar horas y minutos

     MVI C, 0BH       ; Esperar durante un segundo
     CALL RETARDO

     LDA 1102H        ; Se recuperan los segundos
     INR A            ; y se incrementan
     DAA              ; Convertir a BCD
     STA 1102H        ; y guardar

     CPI 60H          ; Si segundos <> 60
     JNZ BUCLE        ; continuar

     MVI A, 00H       ; Si segundos = 60, ponerlos
     STA 1102H        ; a 0

     LDA 1101H        ; e incrementar el número
     INR A            ; de minutos
     DAA
     STA 1101H

     CPI 60H          ; Si minutos <> 60
     JNZ BUCLE        ; continuar

     MVI A, 00H       ; Si minutos = 60
     STA 1101H        ; ponerlos a 0
```

```
     LDA 1100H        ; e incrementar las horas
     INR A
     DAA
     STA 1100H

     CPI 24H          ; Si horas <> 24
     JNZ BUCLE        ; continuar

     MVI A, 00H       ; En caso contrario ponerlas
     STA 1100H        ; a 0 y continuar
     JMP BUCLE

     RST 1
; ----- Subrutina de retardo variable
;
; Entradas: C=Número de décimas de segundo a esperar
RETARDO:
    PUSH B ; Guardar registros que se modificarán
BRET:
    CALL RET1DEC; Una décima

    DCR C        ; Hasta completar el número indicado
    JNZ BRET

    POP B
    RET
; ----- Fin de la subrutina de retardo

; ----- Subrutina de retardo de una décima de segundo
;
RET1DEC:
    PUSH B ; Guardar registros que se modificarán
    PUSH PSW

    LXI B, 208DH ; Número de repeticiones
BRET1:
    DCX B        ; 6 ciclos
    MOV A, B     ; 4 ciclos
    ORA C        ; 4 ciclos
    JNZ BRET1    ; 10 ciclos

    POP PSW
    POP B

    RET
; ----- Fin de la subrutina de retardo
END
```

10.10 Uso del PPI

Ejercicio 8.10.1.1

Respuestas a las cuestiones

1. En el primer caso el valor es 82H, 1000 0010 en binario. El segundo bit está a 1, estableciendo el puerto B como entrada, mientras que el quinto está a 0, configurando el puerto B como salida. En el segundo caso el valor es 90H, 1001 0000 en binario. El segundo bit está a 0 y el quinto a 1, por lo que los puertos han invertido su función.

2. Los puertos que se usarán en los programas que trabajen con el PPI serán, aparte del 3BH, los puertos 38H, 39H y 3AH, utilizándolos para leer/escribir en los puertos A, B y C respectivamente, dependiendo de cuál sea su configuración.

Ejercicio 8.10.2.1

Respuestas a las cuestiones

1. Para poder enviar datos a través de un puerto del PPI es preciso configurarlo como puerto de salida, y así se ha hecho con el puerto A. Para el puerto B no se ha indicado explícitamente ninguna configuración, pero la subrutina tomará el contenido que en ese momento tenga el registro B y lo usará para determinar si debe actuar como salida o como entrada. En cualquier caso es algo que no afecta a este programa.

2. Al enviar un byte a la batería de leds éstos representan los bits del dato recibido, iluminándose aquellos que corresponden a bits a 1 y manteniéndose apagados los que el bit asociado está a 0.

Ejercicio 8.10.2.2

Respuestas a las cuestiones

1. Los valores que aparecen en el campo de datos del uP-2000 son 01H, 02H, 04H, 08H, 10H, 20H, 40H y 80H, que son los que corresponden a 2^0, 2^1 ... 2^7, es decir, a poner a 1 solamente el bit enésimo.

2. Bastaría con asignar como valor inicial al acumulador el dato 80H y cambiar la instrucción RLC que hay en el bucle principal por RRC. También habría que modificar la condición de finalización del bucle.

Ejercicio 8.10.3.1

Respuestas a las cuestiones

1. En la pantalla del uP-2000 aparece en hexadecimal el dato que se ha introducido como binario a través del PPI. Los microinterruptores representan los bits de un byte, de forma que su posición determina qué bits estarán a 0 y cuáles a 1.

Ejercicio 8.10.3.2

Respuestas a las cuestiones

1. Aunque en apariencia el programa no haga nada hasta que no se modifica un microinterruptor, momento en el que cambia tanto el estado de los leds como el dato mostrado en la pantalla del uP-2000, la lectura del puerto `B` del PPI, mediante la instrucción `IN 39H`, no detiene la ejecución del programa en ningún momento. El bucle principal está ejecutándose continuamente, de forma que el estado de los microinterruptores se lee, y el de los leds y la pantalla se modifica, varios miles de veces por segundo.

2. Bastaría con complementar el contenido del acumulador antes de enviarlo hacia el puerto `A` del PPI, intercalando una instrucción `CMA` justo delante del `OUT 38H`.

Ejercicio 8.10.4.1

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     MVI A, 00H       ; Puerto A como salida
     CALL CONF_PPI    ; Configurar PPI

     MVI A, 0EH       ; Desenmascarar RST 5.5
     SIM

BUCLE:
     CALL 044EH       ; Leer tecla

     CPI 10H          ; Si es mayor que F
     JNC FIN          ; no continuar

     OUT 38H          ; En caso contrario enviar al
PPI
```

```
     JMP BUCLE        ; y repetir

FIN:
     RST 1            ; Fin del programa

; ----- Subrutina de configuración del PPI,
;       siempre en modo 0
;
; Entradas:
;     A -> Puerto A: 0-Salida, 1-Entrada
;     B -> Puerto B: 0-Saluda, 1-Entrada
;
CONF_PPI:

     PUSH PSW    ; Guardar A temporalmente

     MOV A, B    ; Para llevar la configuración
     RLC         ; de B un bit a la izquierda
     ANI 02H     ; Resto de bits a 0
     MOV B, A    ; De vuelta a B

     POP PSW

     RLC         ; La configuración
     RLC         ; del puerto A
     RLC         ; en el nibble alto
     RLC
     ANI 010H    ; Resto de bits a 0

     ORA B       ; Agregar puerto B, en el nibble bajo

     ORI 80H     ; Activación del bit de configuración

     OUT 3BH     ; Escritura en registro del PPI

     RET

END
```

Respuestas a las cuestiones

1. El bucle principal de este programa se repite únicamente una vez por cada pulsación de tecla, ya que la llamada a la subrutina `044EH` de la EPROM del uP-2000 provoca la parada del programa hasta que se lee esa pulsación.

Ejercicio 8.10.4.2

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     MVI B, 01H       ; Puerto B como entrada
     CALL CONF_PPI    ; Configurar PPI

     CALL 031FH       ; Borrar pantalla

     LXI H, 0000H     ; Valor inicial del contador

BUCLE:
     PUSH H           ; Mostrar el contador
     CALL 04C9H
     POP H            ; sin perder HL

ESPERA:
     IN 39H           ; Leer microinterruptores

     ANI 80H          ; Comprobar el extremo izquierdo
     JZ ESPERA        ; Si está a 0, se pausa

     IN 39H           ; Leer de nuevo

     ANI 40H          ; Para comprobar el siguiente
     JZ DECRE ; y determinar el sentido del contador

     INX H            ; Incrementarlo
     JMP BUCLE        ; y repetir

DECRE:
     DCX H            ; Decrementarlo
     JMP BUCLE        ; y repetir

; ----- Subrutina de configuración del PPI,
;       siempre en modo 0
;
; Entradas:
;     A -> Puerto A: 0-Salida, 1-Entrada
;     B -> Puerto B: 0-Saluda, 1-Entrada
;
CONF_PPI:

     PUSH PSW    ; Guardar A temporalmente

     MOV A, B    ; Para llevar la configuración
     RLC         ; de B un bit a la izquierda
     ANI 02H     ; Resto de bits a 0
```

```
     MOV B, A    ; De vuelta a B

     POP PSW

     RLC         ; La configuración
     RLC         ; del puerto A
     RLC         ; en el nibble alto
     RLC
     ANI 010H    ; Resto de bits a 0

     ORA B       ; Agregar puerto B, en el nibble bajo

     ORI 80H     ; Activación del bit de configuración

     OUT 3BH     ; Escritura en registro del PPI

     RET

END
```

Ejercicio 8.10.4.3

Solución

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     MVI B, 01H       ; Puerto B como entrada
     CALL CONF_PPI    ; Configurar PPI

     CALL 031FH       ; Borrar pantalla

     LXI H, 0000H     ; Valor inicial del contador

BUCLE:
     PUSH H           ; Mostrar el contador
     CALL 04C9H
     POP H            ; sin perder HL

     IN 39H           ; Leer microinteruptores
     ; Quedarnos con los 3 menos significativos
     ANI 07H
     INR A            ; Asumir 000=1, 111=8

     MOV C, A         ; Introducir retardo
     CALL RETARDO

ESPERA:
     IN 39H           ; Leer microinterruptores
```

```
     ANI 80H          ; Comprobar el extremo izquierdo
     JZ ESPERA        ; Si está a 0, se pausa

     IN 39H           ; Leer de nuevo

     ANI 40H          ; Para comprobar el siguiente
     JZ DECRE ; y determinar el sentido del contador

     INX H       ; Incrementarlo
     JMP BUCLE   ; y repetir

DECRE:
     DCX H       ; Decrementarlo
     JMP BUCLE   ; y repetir

; ----- Subrutina de configuración del PPI,
;       siempre en modo 0
;
; Entradas:
;     A -> Puerto A: 0-Salida, 1-Entrada
;     B -> Puerto B: 0-Saluda, 1-Entrada
;
CONF_PPI:

     PUSH PSW    ; Guardar A temporalmente

     MOV A, B    ; Para llevar la configuración
     RLC         ; de B un bit a la izquierda
     ANI 02H     ; Resto de bits a 0
     MOV B, A    ; De vuelta a B

     POP PSW

     RLC         ; La configuración
     RLC         ; del puerto A
     RLC         ; en el nibble alto
     RLC
     ANI 010H    ; Resto de bits a 0

     ORA B       ; Agregar puerto B, en el nibble bajo

     ORI 80H     ; Activación del bit de configuración

     OUT 3BH     ; Escritura en registro del PPI

     RET

; ----- Subrutina de retardo variable
;
; Entradas:
;  C = Número de décimas de segundo a esperar
RETARDO:
    PUSH B ; Guardar registros que se modificarán
```

```
BRET:
    CALL RET1DEC ; Una décima

    DCR C  ; Hasta completar el número indicado
    JNZ BRET

    POP B
    RET
; ----- Fin de la subrutina de retardo

; ----- Subrutina de retardo de una décima de segundo
;
RET1DEC:
    PUSH B ; Guardar registros que se modificarán
    PUSH PSW

    LXI B, 208DH ; Número de repeticiones

BRET1:
    DCX B        ; 6 ciclos
    MOV A, B     ; 4 ciclos
    ORA C        ; 4 ciclos
    JNZ BRET1    ; 10 ciclos

    POP PSW
    POP B

    RET
; ----- Fin de la subrutina de retardo

END
```

10.11 Interrupciones

Ejercicio 8.11.1.1

Respuestas a las cuestiones

1. Para saber cuál es la dirección que corresponde al vector de una cierta interrupción no hay más que multiplicar el número de ésta por 8. En este caso sería `7.5x8=60`$_d$, que corresponde al valor `3CH`. Leyendo el contenido de esa posición de memoria se comprueba que tiene una instrucción de salto: `C3H=JMP`, por lo que los dos bytes siguientes indican la dirección de destino: `20CEH`. Esta posición corresponde a una pequeña porción de memoria RAM dentro del integrado 8155 (I/O Timer), con cuatro bytes disponibles para introducir una instrucción que transfiera el control a la ISR que interese.

2. En todos los programas escritos en ensamblador se indica, al inicio del código, la dirección a partir de la que se alojará el código, utilizando para ello la directiva `ORG`. No hay ninguna limitación en cuanto al número de veces que puede aparecer esta directiva en el código, por lo que basta con colocarla delante de la instrucción de salto a escribir en la dirección `20CEH`.

3. Cada vez que se produce una interrupción, el 8085 lo primero que es deshabilitar las interrupciones para permitir así que la ISR o rutina de servicio pueda atenderla correctamente. Al pulsar la tecla **`INTR VECT`** se llega a la dirección `20CEH`, desde la que se salta a la etiqueta `ISR`. En ésta se reactivan las interrupciones, con la instrucción `EI`, y se queda a la espera de una interrupción `RST 5.5` que saque al procesador del estado `HLT`. Dicha interrupción, al producirse, provocará que el microprocesador deshabilite las interrupciones, que no vuelven a habilitarse puesto que la última instrucción de la ISR es un salto a la etiqueta `BUCLE`, que muestra continuamente el contador sin que se vuelva a ejecutar la instrucción `EI`.

   Una posible solución sería incluir la instrucción `EI` que hay al inicio del programa al inicio del bucle, de forma que las interrupciones se rehabiliten a cada ciclo. El programa quedaría así:

```
CPU "8085.TBL"
HOF "INT8"
ORG 1000H

     LXI SP, 2000H  ; Pila al final de la RAM

     MVI A, 08H ; Desenmascarar RST 5.5 y RST 7.5
     SIM
```

```
     LXI H, 0000H ; Valor inicial del contador

BUCLE:
     EI           ; Activar interrupciones
     PUSH H       ; Se guarda el contador
     CALL 04C9H   ; para mostrarlo
     POP H

     INX H        ; Incrementarlo
     JMP BUCLE    ; y continuar

ISR: ; Rutina de atención a la RST 7.5

     EI           ; Activar interrupciones
     HLT          ; Y tener el programa

     JMP BUCLE    ; Volver al bucle

; Las instrucciones siguientes se ejecutarán
; al generarse una RST 7.5
ORG 20CEH

     JMP ISR      ; Saltar a la ISR

END
```

4. Este comportamiento se debe a la situación explicada en la cuestión anterior. Si mientras el contador está activa se pulsa cualquier tecla, produciendo una `RST 5.5`, el procesador lo primero que hace es deshabilitar las interrupciones. Esto provoca que ya no se atienda a la `RST 7.5`, por lo que no es posible pausar el programa. Con la modificación indicada, colocando la instrucción `EI` al inicio del bucle, será posible pausar y reanudar el bucle tantas veces como se quiera, independientemente de que antes se hayan pulsado o no otras teclas.

5. Al introducir la instrucción `EI` al inicio del bucle, esto significa que el programa podrá ser interrumpido en cualquier momento, dándose situaciones como las siguientes:

   i. Se guarda la pareja de registros `HL` en la pila, se produce la interrupción y la rutina de servicio devuelve el control a la etiqueta `BUCLE`, sin llegar a mostrar el contador ni recuperar `HL` de la pila. El valor del contador no varía, pero en la pila irán quedando datos no recuperados.

   ii. Tras guardar `HL` en la pila se invoca a la subrutina que visualiza su contenido en el campo de direcciones y, cuando ésta se encuentra haciendo su trabajo, se produce la interrupción. La rutina de servicio

devuelve el control a la etiqueta BUCLE habiéndose perdido el contenido de HL y, además, en la pila ha quedado tanto el contenido previo de HL como la dirección de retorno que almacenó la ejecución de la instrucción CALL.

De esto se deduce que debería haber momentos en que no fuese posible interrumpir el programa, evitando la corrupción de la pila, que queda con datos no útiles almacenados, y la pérdida de la información realmente útil, como es el contenido de la pareja de registros HL.

La solución pasa por desactivar las interrupciones justo antes de que se inicie el bloque de instrucciones que no han de ser interrumpidas, volviéndolas a habilitar al salir de dicho bloque. De esta forma el programa quedaría como se muestra a continuación:

```
CPU "8085.TBL"
HOF "INT8"

     ORG 1000H

     LXI SP, 2000H    ; Pila al final de la RAM

     MVI A, 08H ; Desenmascarar RST 5.5 y RST 7.5
     SIM

     LXI H, 0000H     ; Valor inicial del contador

BUCLE:

     DI               ; Activar interrupciones

     PUSH H           ; Se guarda el contador
     CALL 04C9H       ; para mostrarlo
     POP H

     EI          ; Volver a habilitar interrupciones

     INX H            ; Incrementarlo
     JMP BUCLE        ; y continuar

ISR: ; Rutina de atención a la RST 7.5

     EI               ; Activar interrupciones
     HLT              ; Y tener el programa

     JMP BUCLE        ; Volver al bucle

; Las instrucciones siguientes se ejecutarán
; al generarse una RST 7.5
ORG 20CEH
```

```
      JMP ISR           ; Saltar a la ISR

END
```

Ahora el ejecutar el programa podrá comprobarse que es posible pausar y reanudar el contador tantas veces como se quiera, sin que el valor de éste se vea alterado en ningún momento.