*PRÁCTICA 3*

# Programación de la familia MCS-51

*Texto basado en el documento:*

*"Procesadores de propósito general"*

*Departamento de Ingeniería Electrónica, de Sistemas Informáticos y Automática - Universidad de Huelva*

## Objetivos

- Conocer los modos de direccionamiento que soportan los miembros de la familia MCS-51, así como sus principales características y aplicaciones.
- Analizar los diferentes tipos de instrucciones que componen el repertorio de esta familia de microcontroladores.

## Contenido

3.1 Introducción ..... 1

3.2 Modos de direccionamiento ..... 1

3.2.1 Direccionamiento directo ..... 1

3.2.2 Direccionamiento indirecto ..... 2

3.2.3 Direccionamiento por registro ..... 2

3.2.4 Direccionamiento implícito ..... 3

3.2.5 Direccionamiento inmediato ..... 3

3.2.6 Direccionamiento indexado ..... 3

3.3 Tipos de instrucciones ..... 3

3.3.1 Instrucciones aritméticas ..... 4

3.3.2 Instrucciones lógicas ..... 5

3.3.3 Instrucciones de transferencia de datos ..... 6

3.3.4 Instrucciones booleanas ..... 7

3.3.5 Instrucciones de salto ..... 9

## 3.1 Introducción

Todos los microcontroladores de la familia MCS-51 ejecutan el mismo *set* o juego de instrucciones. Estas instrucciones están optimizadas para el desarrollo de aplicaciones basadas en un tamaño de palabra de 8 *bits* y soportan una buena variedad de modos de direccionamiento en el acceso a la memoria de datos interna del microcontrolador. Además de las instrucciones que operan sobre datos de tamaño *byte*, existe un conjunto de instrucciones especializadas en la gestión de estructuras de datos de tamaño *bit*.

En los apartados siguientes, en primer lugar se analizan los diferentes modos de direccionamiento soportados por los microcontroladores de la familia MCS-51 y, a continuación, se proporciona una breve descripción de las diferentes instrucciones, agrupándolas en distintas categorías según la función que realizan.

En la Tabla 3.1 se especifica el significado de la notación simbólica utilizada en adelante.

| Símbolo | Significado |
|---|---|
| R | Registro R |
| (R) | Contenido del registro R |
| [AB] | Posición de memoria ABH |
| [(R)] | Posición de memoria direccionada mediante el registro R |
| ([AB]) | Contenido de la posición de memoria ABH |
| ([(R)]) | Contenido de la posición de memoria direccionada mediante el registro R |
| ← | Transferencia de datos |
| <*byte*> | Operando de 8 *bits* especificado mediante varios modos de direccionamiento |
| <orig> | Origen de una transferencia especificado mediante varios modos de direccionamiento |
| <dest> | Destino de una transferencia especificado mediante varios modos de direccionamiento |
| dir 16 | Dirección de 16 *bits* (64 K*bytes*) |
| dir 11 | Dirección de 11 *bits* (2 K*bytes*) |
| rel | Desplazamiento de 8 *bits* en complemento a 2 utilizado en los saltos relativos |
| dato8 | Contante de 8 *bits* |
| dato16 | Contante de 16 *bits* |
| .NOT. | Complemento lógico |
| .AND. | Operación lógica AND |
| .OR. | Operación lógica OR |
| .XOR. | Operación lógica XOR |
| / | División entera |
| mod | Resto de la división entera |

**Tabla 3.1. Significado de la notación simbólica utilizada.**

## 3.2 Modos de direccionamiento

El 8051/52 soporta un total de 6 modos de direccionamiento que, de manera breve, se describen a continuación.

### 3.2.1 Direccionamiento directo

El operando se especifica en la instrucción mediante un campo de dirección de 8 *bits*. Sólo la RAM interna de datos (primeros 128 *byte*s) y la zona de SFR se pueden direccionar de esta forma.

Este modo de direccionamiento permite el acceso de fácil y rápido a las zonas de las memoria anteriormente indicadas.

*Ejemplo:*

```
ADD A,3BH   ; esta instrucción suma (ADD) al contenido del acumulador el contenido
              de la posición de memoria 3BH: A ← (A) + ([3B]).
```

### 3.2.2 Direccionamiento indirecto

En este modo de direccionamiento, la dirección del operando está contenida en un registro, cuyo nombre se especifica en la instrucción precedido por el símbolo "@". Permite acceder tanto a la memoria de datos interna (128/256 *bytes* para el 8051/52) como a la memoria de datos externa.

Para el direccionamiento de áreas de memoria (tanto internas como externas) de tamaño no superior a 256 *bytes*, se puede hacer uso de los registros R0 y R1 del banco de registros seleccionado. Si el área de memoria posee un tamaño superior, sólo puede utilizarse el registro DPTR.

Este modo de direccionamiento es especialmente útil para acceder a los 128 *bytes* altos de la memoria de datos interna (8052) y a la memoria de datos externa.

*Ejemplos:*

```
ADD A,@R0       ; Si (R0) = 3BH, la operación realizada por esta instrucción será la
                  misma del ejemplo anterior: A ← (A) + ([3B]).

MOVX @DPTR,A    ; Si (DPTR) = 4C2BH, esta instrucción carga el contenido del
                  acumulador en la posición 4C2BH de la memoria de datos externa:
                  [4C2B] ← (A).
```

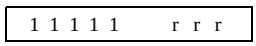
### 3.2.3 Direccionamiento por registro

Como ya se ha estudiado, los microcontroladores 8051/52 poseen cuatro bancos de registros (que pueden ser seleccionados mediante los *bits* 3 y 4 del registro PSW) cada uno de los cuales está compuesto por 8 registros (R0 - R7). Cuando se usa direccionamiento por registro, en el propio código de operación de la instrucción se especifica el registro del banco activo con que se opera mediante un campo de tres *bits*.

*Ejemplo:*

```
MOV Rn,A        ; Carga en el registro Rn el contenido del acumulador: Rn ← (A).
```

formato:




Dependiendo de qué registro sea Rn, el código de instrucción tomará distintos valores. Así, `MOV R0,A` se codificará como 11111000B = F8H, mientras que `MOV R6,A` se codificará como 11111110B = FEH.

Este direccionamiento ocupa muy poca memoria de programa y permite acceder de manera rápida a los datos más frecuentemente utilizados.

### 3.2.4 Direccionamiento implícito

Algunas instrucciones operan siempre sobre un mismo registro (acumulador, puntero de datos, etc.) el cual, en consecuencia, es seleccionado de forma implícita por el propio código de operación, no siendo necesario añadir un *byte* de dirección para referenciarlo.

Las instrucciones con este modo de direccionamiento son muy rápidas y compactas, ocupando el mínimo espacio en la memoria de programa.

*Ejemplos:*

```
INC A          ; Incrementa el contenido del acumulador: A ← (A) + 1.
INC DPTR       ; Incrementa puntero de datos: DPTR ← (DPTR) + 1.
```

### 3.2.5 Direccionamiento inmediato

En este modo de direccionamiento, el operando va incluido en la propia instrucción y se identifica precediéndolo por el símbolo "#". Dicho operando tiene una longitud de 8 *bits*, excepto en las instrucciones que afectan al DPTR, en las cuales posee un tamaño de 16 *bits*.

Este modo se usa en los casos en que el operando es una constante.

*Ejemplos:*

```
MOV A,#255          ; Carga en el acumulador el número decimal 255.
MOV DPTR,#36CAH     ; Carga en el registro DPTR el valor  36CAH.
```

### 3.2.6 Direccionamiento indexado

Este direccionamiento sólo es aplicable sobre la memoria de programas y para operaciones de lectura. Se utiliza para la lectura de tablas de valores constantes alojadas en esta zona de la memoria. La posición de memoria a acceder se especifica mediante la dirección de la base de la tabla (almacenada en un registro de 16 *bits*: DPTR o PC) y un desplazamiento u *offset* (almacenado en el acumulador). Es decir, la dirección a la que se accede se obtiene mediante la suma de los contenidos del acumulador y del puntero base (DPTR o PC). Las instrucciones `MOVC A,@A+DPTR` y `MOVC A,@A+PC` utilizan el modo de direccionamiento indexado.

Otro tipo de direccionamiento indexado es el utilizado en la instrucción `JMP @A+DPTR`, donde la dirección de salto se obtiene sumado los contenidos de los registros A y DPTR.

## 3.3 Tipos de instrucciones

El *set* de instrucciones del 8051/52 se puede dividir en las siguientes categorías:

- Instrucciones aritméticas.
- Instrucciones lógicas.
- Instrucciones de transferencia de datos.
- Instrucciones booleanas.
- Instrucciones de salto.

Las instrucciones aritméticas, lógicas, de transferencia y de salto son comunes a la mayoría de microprocesadores. Los microcontroladores poseen un área de aplicación especial, el control de procesos industriales, en la que son de gran utilidad las operaciones orientadas al *bit*. Para la realización de este tipo de operaciones, los microcontroladores 8051/52 poseen un *set* de instrucciones booleanas, que permiten la manipulación de *bits* individuales.

En las instrucciones que poseen dos operandos, el primero de ellos es siempre el que almacena el resultado.

A continuación, se describen brevemente las diferentes instrucciones pertenecientes a cada una de las categorías, indicando cuáles son sus nemónicos, las operaciones que realizan, los modos de direccionamiento que admiten, los ciclos de máquina que dura su ejecución y los *flags* del registro PSW a los que afectan. Dado que el número de *bytes* que ocupa cada instrucción depende del modo de direccionamiento empleado, este parámetro no se ha incluido en las diferentes tablas, por lo que en caso de ser necesaria esta información, se remite al lector al manual del fabricante.

### 3.3.1 Instrucciones aritméticas

Las instrucciones aritméticas soportadas por la familia MCS-51 realizan diferentes operaciones aritméticas sobre datos de 8 *bits*, salvo algunas excepciones que se comentan a continuación. En la Tabla 3.2 se muestran estas instrucciones, indicándose los modos de direccionamiento que soportan y los *flag*s de la palabra de estado a los que afectan.

| Nemónico | Operación | Modos de direccionamiento | | | | Ciclos de máquina | *Flags* afectados | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Dir | Ind | Reg | Inm | | C | OV | AC |
| ADD A, <*byte*> | A ← (A) + <*byte*> | X | X | X | X | 1 | X | X | X |
| ADDC A, <*byte*> | A ← (A) + <*byte*> + C | X | X | X | X | 1 | X | X | X |
| SUBB A, <*byte*> | A ← (A) - <*byte*> - C | X | X | X | X | 1 | X | X | X |
| INC A | A ← (A) + 1 | Sólo acumulador | | | | 1 | | | |
| INC <*byte*> | <*byte*> ← <*byte*> + 1 | X | X | X | | 1 | | | |
| INC DPTR | DPTR ← (DPTR) + 1 | Sólo puntero de datos | | | | 2 | | | |
| DEC A | A ← (A) - 1 | Sólo acumulador | | | | 1 | | | |
| DEC <*byte*> | <*byte*> ← <*byte*> - 1 | X | X | X | | 1 | | | |
| MUL AB | B:A ← (A) x (B) | Sólo acumulador y B | | | | 4 | 0 | X | |
| DIV AB | A ← (A) / (B); B ← (A) mod (B) | Sólo acumulador y B | | | | 4 | 0 | X | |
| DA A | Ajuste decimal | Sólo acumulador | | | | 1 | X | | |

**Tabla 3.2. Instrucciones aritméticas de la familia MCS-51.**

**ADD:** Añade al contenido del acumulador un dato de 8 *bits* especificado de forma explícita o el contenido de una posición de la memoria de datos interna.

**ADDC:** Actúa igual que la instrucción anterior pero añadiendo, además, el valor del *bit* de acarreo (C). Permite la realización de sumas de precisión múltiple.

**SUBB:** Resta del acumulador el valor del acarreo y, bien un dato de 8 *bits* especificado de explícitamente, o bien el contenido de una posición de la memoria de datos interna. Si se desea realizar una resta sin acarreo, éste deberá ser puesto a cero inmediatamente antes de esta instrucción.

**INC y DEC:** Estas instrucciones permiten incrementar y decrementar, respectivamente, tanto el contenido del acumulador, como el de cualquier *byte* de la memoria de datos interna sin la

intervención de este registro. Por otro lado, INC DPTR permite el incremento del puntero de datos (16 *bits*).

**MUL AB:** Multiplica el contenido de dos datos de 8 *bits*, alojados en los registros A y B, almacenando los 8 *bits* de más peso del producto (16 *bits*) en B y los 8 de menos peso en A.

**DIV AB:** Realiza la división entera del contenido del acumulador entre el contenido de B, almacenando el cociente en A y el resto en B. Por ejemplo, si A = 5FH (95) y B = 0AH (10), después de la ejecución de dicha instrucción A = 9 y B = 5.

**DA A:** Realiza un ajuste decimal del resultado expresado en binario puro producido por las instrucciones ADD y ADDC. Esta instrucción debe colocarse inmediatamente después de cualquiera de estas instrucciones (cuando actúan sobre operandos expresados en BCD) y realiza las siguientes operaciones:

Si $[(A_{3-0}) > 9]$ o $[AC = 1]$ entonces $A_{3-0} \leftarrow (A_{3-0}) + 6$

y si $[(A_{7-4}) > 9]$ o $[C = 1]$ entonces $A_{7-4} \leftarrow (A_{7-4}) + 6$

## 3.3.2 Instrucciones lógicas

La Tabla 3.3 muestra la lista de instrucciones lógicas de la familia MCS-51, los modos de direccionamiento que permiten, así como los *flags* afectados.

| Nemónico | Operación | Modos de direccionamiento | | | | Ciclos de máquina | *Flags* afectados | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Dir | Ind | Reg | Inm | | C | OV | AC |
| ANL A, <*byte*> | A ← (A) .AND. <*byte*> | X | X | X | X | 1 | | | |
| ANL <*byte*>, A | <*byte*> ← <*byte*> .AND. (A) | X | | | | 1 | | | |
| ANL <*byte*>, #dato8 | <*byte*> ← <*byte*> .AND. dato8 | X | | | | 2 | | | |
| ORL A, <*byte*> | A ← (A) .OR. <*byte*> | X | X | X | X | 1 | | | |
| ORL <*byte*>, A | <*byte*> ← <*byte*> .OR. (A) | X | | | | 1 | | | |
| ORL <*byte*>, #dato8 | <*byte*> ← <*byte*> .OR. dato8 | X | | | | 2 | | | |
| XRL A, <*byte*> | A ← (A) .XOR. <*byte*> | X | X | X | X | 1 | | | |
| XRL <*byte*>, A | <*byte*> ← <*byte*> .XOR. (A) | X | | | | 1 | | | |
| XRL <*byte*>, #dato8 | <*byte*> ← <*byte*> .XOR. dato8 | X | | | | 2 | | | |
| CLR A | A ← 00H | Sólo acumulador | | | | 1 | | | |
| CPL A | A ← .NOT. (A) | Sólo acumulador | | | | 1 | | | |
| RL A | Rota A un *bit* a la izquierda | Sólo acumulador | | | | 1 | | | |
| RLC A | Rota A un *bit* a la izquierda con C | Sólo acumulador | | | | 1 | X | | |
| RR A | Rota A un *bit* a la derecha | Sólo acumulador | | | | 1 | | | |
| RRC A | Rota A un *bit* a la derecha con C | Sólo acumulador | | | | 1 | X | | |
| SWAP A | Intercambia los *nibbles* de A | Sólo acumulador | | | | 1 | | | |

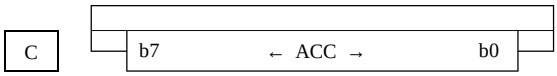
**Tabla 3.3. Instrucciones lógicas de la familia MCS-51.**

**ANL, ORL y XRL:** Realizan las operaciones AND, OR y XOR, respectivamente, sobre dos datos de 8 *bits*, que pueden residir en diferentes ubicaciones (véase la Tabla 3.3).

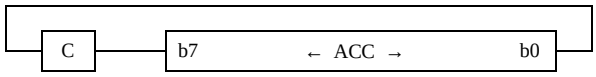
**CLR A:** Pone a cero el Acumulador.

**CPL A:** Complementa el acumulador.

**RL A y RR A:** Rotan el acumulador a la izquierda y a la derecha, respectivamente.




**RLC A y RRC A:** Rotan el acumulador a la izquierda y a la derecha, respectivamente, pero a través del acarreo.




**SWAP A:** Intercambia los *nibbles* alto y bajo del acumulador. Generalmente se utiliza en las manipulaciones de números en BCD. Por ejemplo, si previamente (A) = 93H, después de SWAP A el contenido del acumulador es 39H.

## 3.3.3 Instrucciones de transferencia de datos

Según el área de memoria que intervenga en la transferencia (memoria de datos interna, memoria de datos externa o memoria de programa), este tipo de instrucciones puede clasificarse a su vez en tres grupos.

### 3.3.3.1 Transferencia de datos sobre la memoria RAM interna

En la Tabla 3.4 se muestra el conjunto de instrucciones disponibles para mover datos dentro de los diferentes espacios de la memoria RAM interna, indicando los distintos modos de direccionamiento que soportan cada una de ellas.

| Nemónico | Operación | Modos de direccionamiento | | | | Ciclos de máquina |
|---|---|---|---|---|---|---|
| | | Dir | Ind | Reg | Inm | |
| MOV A, <orig> | A ← <orig> | X | X | X | X | 1 |
| MOV <dest>, A | <dest> ← (A) | X | X | X | | 1 |
| MOV <dest>, <orig> | <dest> ← <orig> | X | X | X | X | 2 |
| MOV DPTR, #dato16 | DPTR ← dato16 | | | | X | 2 |
| PUSH <orig> | SP ← (SP) + 1; [(SP)] ← <orig> | X | | | | 2 |
| POP <dest> | <dest> ← ([(SP)]); SP ← (SP) – 1 | X | | | | 2 |
| XCH A, <*byte*> | Intercambio de los contenidos de A y <*byte*> | X | X | X | | 1 |
| XCHD A, @Ri | Intercambio de los *nibbles* bajos de A y [(Ri)] | | X | | | 1 |

**Tabla 3.4. Instrucciones de transferencia de datos sobre la memoria RAM interna.**

**MOV:** Permite inicializar el acumulador o una posición de memoria con un valor constante, o bien transferir un dato de 8 *bits* entre éstos, en ambos sentidos, o entre dos posiciones de memoria sin la intervención del acumulador. MOV DPTR inicializa el puntero de datos mediante la carga en el mismo de un dato de 16 *bits* expresado de forma explicita.

**PUSH:** Incrementa el valor del puntero de la pila (SP) y a continuación almacena en ésta el contenido de la posición de memoria especificada.

**POP:** Extrae el dato de la cabecera de la pila almacenándolo en la posición de memoria especificada, decrementando a continuación el SP.

**XCH:** Intercambia los contenidos del acumulador y de la posición de memoria especificada.

**XCHD:** Intercambia solo los *nibbles* bajos del acumulador y de la posición de memoria especificada. Es una instrucción especializada en la manipulación de datos expresados en el código BCD.

### 3.3.3.2 Transferencia de datos sobre la RAM externa

La Tabla 3.5 muestra el conjunto de instrucciones que permiten la comunicación con la memoria de datos externa. Todas estas instrucciones poseen el mismo nemónico (MOVX) y únicamente soportan el modo de direccionamiento indirecto. Además, en ellas siempre interviene el acumulador, ya sea como fuente o como destino de la información.

| Nemónico | Operación | Puntero | Longitud de la dirección | Ciclos de máquina |
|---|---|---|---|---|
| MOVX A, @Ri | Lectura de la memoria RAM externa | R0 o R1 | 8 *bits* | 2 |
| MOVX @Ri, A | Escritura en la memoria RAM externa | R0 o R1 | 8 *bits* | 2 |
| MOVX A, @DPTR | Lectura de la memoria RAM externa | DPTR | 16 *bits* | 2 |
| MOVX @DPTR, A | Escritura en la memoria RAM externa | DPTR | 16 *bits* | 2 |

**Tabla 3.5. Instrucciones de transferencia de datos sobre la RAM externa.**

El usuario debe escoger entre realizar un direccionamiento con 8 *bits* usando las instrucciones `MOVX A,@Ri` y `MOVX @Ri,A` (donde Ri puede ser R0 o R1 del banco de registros seleccionado) o realizar un direccionamiento de 16 *bits* mediante el registro DPTR usando las instrucciones `MOVX A,@DPTR` y `MOVX @DPTR,A` (lo cual implica sacrificar el puerto P2).

Las líneas de control de lectura y escritura ( $\overline{RD}$ y $\overline{WR}$ ) sólo se activan al ejecutar las instrucciones MOVX. Si el sistema no dispone de memoria RAM externa, los pines correspondientes quedan libres para ser utilizados como líneas de E/S.

### 3.3.3.3 Transferencia de datos desde la memoria de programas externa.

La Tabla 3.6 muestra las dos instrucciones disponibles para la lectura de tablas de valores constantes ubicadas en la memoria de programa. Obsérvese que estas tablas sólo pueden ser leídas y no actualizadas. El nemónico es MOVC (MOVer Constante).

| Nemónico | Operación | Ciclos de máquina |
|---|---|---|
| MOVC A, @A+DPTR | Lectura de la memoria de programa externa mediante A y DPTR | 2 |
| MOVC A, @A+PC | Lectura de la memoria de programa externa mediante A y PC | 2 |

**Tabla 3.6. Instrucciones de transferencia de datos desde la memoria de código externa.**

Si la tabla está ubicada en la memoria de programas externa, la ejecución de estas instrucciones activa la señal de control $\overline{PSEN}$.

La instrucción `MOVC A,@A+DPTR` carga en el acumulador el contenido de la posición de memoria cuya dirección es la suma de los contenidos del DPTR (dirección base de la tabla) y del acumulador (elemento de la tabla).

La instrucción `MOVC A,@A+PC` es similar a la primera, salvo que es el PC el que actúa como indicador de la dirección base de la tabla. Esta instrucción suele colocarse dentro de una subrutina justo antes de la RET, mientras que la tabla de datos se ubica justo después de dicha instrucción de retorno. Así, el elemento de la tabla al que se accede será igual al valor del acumulador antes de saltar a dicha subrutina menos 1.

## 3.3.4 Instrucciones booleanas

Los microcontroladores de la familia MCS-51 poseen un Procesador Booleano incluido en el propio chip. Como ya se ha señalado, estos microcontroladores poseen dos áreas que

permiten el direccionamiento "*bit* a *bit*": una corresponde al segmento de RAM interna comprendido entre las direcciones 20H y 2FH (128 *bits* en total), y la otra corresponde a la columna izquierda de la zona SFR.

El conjunto de instrucciones que permiten la manipulación de *bits* individuales, tan interesantes para las aplicaciones industriales, se relacionan en la Tabla 3.7.

| Nemónico | Operación | Ciclos de máquina | *Flags* afectados | | |
|---|---|---|---|---|---|
| | | | C | OV | AC |
| ANL C, *bit* | C ← C .AND. *bit* | 2 | X | | |
| ANL C, /*bit* | C ← C .AND. .NOT. *bit* | 2 | X | | |
| ORL C, *bit* | C ← C .OR. *bit* | 2 | X | | |
| ORL C, /*bit* | C ← C .OR. .NOT. *bit* | 2 | X | | |
| MOV C, *bit* | C ← *bit* | 1 | X | | |
| MOV *bit*, C | *bit* ← C | 2 | | | |
| CLR C | C ← 0 | 1 | 0 | | |
| CLR *bit* | *bit* ← 0 | 1 | | | |
| SETB C | C ← 1 | 1 | 1 | | |
| SETB *bit* | *bit* ← 1 | 1 | | | |
| CPL C | C ← .NOT. C | 1 | X | | |
| CPL *bit* | *bit* ← .NOT. *bit* | 1 | | | |
| JC rel | Salto si C = 1 | 2 | | | |
| JNC rel | Salto si C = 0 | 2 | | | |
| JB *bit*, rel | Salto si *bit* = 1 | 2 | | | |
| JNB *bit*, rel | Salto si *bit* = 0 | 2 | | | |
| JBC *bit*, rel | Salto si *bit* = 1; *bit* ← 0 | 2 | | | |

**Tabla 3.7. Instrucciones booleanas de la familia MCS-51.**

Puede observarse como la mayoría de las instrucciones booleanas están referidas al *flag* C del registro PSW. Se podría decir que este *flag* actúa como acumulador en las instrucciones booleanas.

**ANL:** Realiza la operación AND entre los valores de C y del *bit* especificado, o su complemento.

**ORL:** Realiza la operación OR entre los valores de C y del *bit* especificado, o su complemento.

**MOV:** Transfiere información entre C y el *bit* especificado, en ambas direcciones.

**CLR:** Pone a cero el *flag* C o el *bit* especificado.

**SETB:** Pone a uno el *flag* C o el *bit* especificado.

**CPL:** Complementa el *flag* C o el *bit* especificado.

**JC y JNC:** Ejecutan un salto relativo si el valor de C es 1 y 0, respectivamente.

**JB y JNB:** Ejecutan un salto relativo si el valor del *bit* especificado es 1 y 0, respectivamente.

**JBC:** Ejecutan un salto relativo si el valor del *bit* especificado es 1, poniendo a 0 dicho *bit*.

Cuando el programa está escrito en ensamblador, la dirección de destino en los saltos relativos viene especificada por una etiqueta. Una vez ensamblado el programa, la dirección

del salto se convierte en un número (*offset* o desplazamiento) de 8 *bits* representado en complemento a dos. Por tanto, este modo de direccionamiento únicamente permite ejecutar saltos hacia posiciones comprendidas en un rango de [-128, +127] *bytes* con respecto a la dirección de la instrucción siguiente. Si el desplazamiento es positivo, el salto es hacia adelante (direcciones crecientes de memoria) y si es negativo, el salto es hacia atrás (direcciones decrecientes de memoria).

## 3.3.5 Instrucciones de salto

En una primera aproximación, un programa es un conjunto de instrucciones ubicadas en posiciones consecutivas de la memoria, que se ejecutan una a continuación de otra. Esta secuencia natural de ejecución del programa puede ser interrumpida por las denominadas instrucciones de salto o de bifurcación, las cuales pueden modificar el contenido del PC por diferentes motivos:

- **Finalización de la secuencia de instrucciones:** Se puede pasar a ejecutar una secuencia diferente o volver a ejecutar la misma secuencia (salto incondicional).
- **Naturaleza del resultado de una operación:** En función del estado de los diferentes *flags* de la PSW o de otros *bits*, o del contenido de un registro o de una posición de memoria, se puede continuar con la secuencia normal del programa o pasar a la ejecución de otra secuencia diferente (salto condicional).

Seguidamente, se estudiarán por separado los dos tipos de instrucciones de salto (incondicional y condicional) que soportan los microcontroladores de la familia MCS-51.

### 3.3.5.1 Instrucciones de salto incondicional

Este tipo de instrucciones provocan un salto hacia otra parte de la memoria cada vez que aparecen en la secuencia del programa. En la Tabla 3.8 se muestra una lista resumida del conjunto de instrucciones de salto incondicional que soporta la familia MCS-51.

| Nemónico | Operación | Ciclos de máquina |
|---|---|---|
| SJMP rel | Salto corto relativo | 2 |
| LJMP dir16 | Salto largo absoluto | 2 |
| AJMP dir11 | Salto absoluto en página de 2 K*bytes* | 2 |
| JMP @A+DPTR | Salto hasta [(A) + (DPTR)] | 2 |
| LCALL dir16 | Llamada a subrutina larga | 2 |
| ACALL dir11 | Llamada a subrutina en página de 2 K*bytes* | 2 |
| RET | Retorno de subrutina | 2 |
| RETI | Retorno de interrupción | 2 |
| NOP | No operación | 1 |

**Tabla 3.8. Instrucciones de salto incondicional de la familia MCS-51.**

**SJMP:** La dirección de salto se especifica de forma relativa al PC mediante un *offset* o desplazamiento de 8 *bits*, igual que en las instrucciones de salto booleanas. La instrucción tiene un tamaño de dos *bytes*, que corresponden al código de operación y al desplazamiento. Las direcciones de salto están limitadas, igualmente, dentro de un rango de [-128, +127] *bytes* con respecto a la dirección del primer *byte* de la instrucción siguiente a la de salto. Por tanto, la ejecución de esta instrucción realiza las operaciones:

$$PC \leftarrow (PC) + 2$$

$$PC \leftarrow (PC) + rel$$

**LJMP:** Ejecuta un salto hacia una dirección que se especifica de manera absoluta mediante una constante de 16 *bits*. La instrucción consta de 3 *bytes*, 1 para el código de operación y 2 para la dirección del salto. Por tanto, esta instrucción permite saltar a cualquier posición de la memoria de programa. La operación que realiza es:

PC ← dirección de salto (16 *bits*)

**AJMP:** La dirección de salto también es absoluta, pero en este caso viene especificada por una combinación de sólo 11 *bits*. La instrucción tiene 2 *bytes*, uno para el código de operación (que contiene los 3 *bits* de más peso de la dirección) y otro que contiene los 8 *bits* de menos peso de la misma. El formato de esta instrucción es el siguiente:

| Código de operación | | | | | | | | Operando | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $A_{10}$ | $A_9$ | $A_8$ | 0 | 0 | 0 | 0 | 1 | $A_7$ | $A_6$ | $A_5$ | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ |

Por tanto, el destino del salto está comprendido dentro del bloque de 2 K*byte*s donde se aloja la siguiente instrucción a la de salto. Las operaciones realizadas en este caso son:

PC ← (PC) + 2

$PC_{10-0}$ ← dirección de salto (11 *bits*)

En cualquier caso, cuando el programador escribe el programa en lenguaje ensamblador, especifica la dirección de destino, bien como una etiqueta o bien como una dirección constante de 16 *bits*. Es el programa ensamblador el que se encarga de asignar al destino el formato correcto, según el código de operación. Si la longitud de salto es superior a la que soporta este formato, se generará un mensaje de error similar a este: "*Destination out of range*".

**JMP:** Se trata de una instrucción de salto indirecto. Suma el *byte* contenido en el acumulador con los 16 *bits* del puntero de datos (DPTR) y carga el resultado en el contador de programa.

**LCALL:** Efectúa una llamada a una subrutina cuya dirección se especifica mediante una combinación de 16 *bits*, por lo que ésta puede ubicarse en cualquier parte de la memoria de programas. La secuencia de operaciones que realiza es la siguiente:

PC ← (PC) + 3

SP ← (SP) + 1

[(SP)] ← ($PC_{7-0}$)

SP ← (SP) + 1

[(SP)] ← ($PC_{15-8}$)

PC ← dir16

**ACALL:** Efectúa una llamada a una subrutina cuya dirección se especifica mediante una combinación de 11 *bits*, por lo que dicha subrutina debe encontrarse dentro del área de 2 K*bytes*, en la que se aloja la instrucción siguiente en la secuencia. Las operaciones realizadas por esta instrucción son:

PC ← (PC) + 2

SP ← (SP) + 1

[(SP)] ← ($PC_{7-0}$)

$SP \leftarrow (SP) + 1$

$[(SP)] \leftarrow (PC_{15-8})$

$PC_{10-0} \leftarrow dir_{10-0}$ (posición dentro del área de 2 K*bytes*)

Igual que sucede con AJMP, para esta instrucción el programador sólo debe preocuparse de colocar una etiqueta en lugar de la dirección de salto, recayendo sobre el programa ensamblador la tarea de determinar la combinación binaria a insertar en el código máquina.

**RET:** Finaliza la ejecución de una subrutina y reanuda la ejecución del programa que realizó la llamada a partir de la instrucción siguiente a CALL. Más concretamente, la secuencia de operaciones que realiza es:

$PC_{15-8} \leftarrow ([(SP)])$

$SP \leftarrow (SP) - 1$

$PC_{7-0} \leftarrow ([(SP)])$

$SP \leftarrow (SP) - 1$

**RETI:** Finaliza la ejecución de una rutina de interrupción y reanuda la ejecución del programa principal a partir de la instrucción siguiente a la última que se ejecutó. Además, restaura la lógica de interrupciones para que puedan ser aceptadas las interrupciones con el mismo nivel de prioridad que la que acaba de ser atendida. La dirección de retorno se recupera de la pila de la misma forma que en el caso anterior.

**NOP:** No realiza ninguna acción, lo cual podría interpretarse como un salto incondicional hasta la instrucción inmediatamente posterior. Se utiliza para la generación de pequeños retardos.

### 3.3.5.2 Instrucciones de salto condicional

Este tipo de instrucciones (Tabla 3.9) provocan un salto sólo si se cumple la condición especificada en las mismas. En caso contrario se continúa con la ejecución de la siguiente instrucción en la secuencia. Esto permite al microcontrolador tomar decisiones (ejecutando una parte del programa u otra) basándose en la respuesta a una cuestión que sólo admite dos soluciones perfectamente definidas (verdadero o falso).

| Nemónico | Operación | Modos de direccionamiento | | | | Ciclos de máquina | *Flags* afectados | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Dir | Ind | Reg | Inm | | C | OV | AC |
| JZ rel | Salto si (A) = 0 | Sólo acumulador | | | | 2 | | | |
| JNZ rel | Salto si (A) ≠ 0 | Sólo acumulador | | | | 2 | | | |
| DJNZ <*byte*>, rel | Decremento de <*byte*> y salto si ≠ 0 | X | | X | | 2 | | | |
| CJNE A, <*byte*>, rel | Salto si (A) ≠ <*byte*> | X | | | X | 2 | X | | |
| CJNE <*byte*>, #dato8, rel | Salto si <*byte*> ≠ dato8 | | X | X | | 2 | X | | |

**Tabla 3.9. Instrucciones de salto condicional de la familia MCS-51.**

La cuestión que determina el que se produzca o no el salto se halla implícita en código de operación de la instrucción, mientras que el operando es un desplazamiento de 8 *bits* que permite la especificación del salto relativo. Al programar en ensamblador, en lugar del operando se coloca una etiqueta o una dirección 16 *bits*.

**JZ y JNZ:** Provocan un salto si el contenido del acumulador es igual a 0 y distinto de 0, respectivamente.

**DJNZ:** Decrementa el operando <*byte*> (que actúa como contador) y provoca un salto hasta la dirección especificada si el resultado es distinto de cero. Esta instrucción está especializada en la implementación de bucles. Para que un bucle se ejecute *n* veces, basta con cargar este valor en el contador mediante una instrucción, asignar la etiqueta que marca la dirección de salto a la instrucción siguiente y, por último, colocar la instrucción DJNZ tras la última instrucción del bucle.

*Ejemplo: Ejecutar un bucle 15 veces.*

```
        MOV CONTADOR,#15
BUCLE:  (comienzo del bucle)
        ............
        ............
        (fin del bucle)
        DJNZ CONTADOR,BUCLE
```

**CJNE:** Compara el primer operando con el segundo y produce un salto a la dirección especificada si son diferentes. Además, pone C a 1 si el primer operando es menor que el segundo, y a 0 si no es así. Una de las aplicaciones de esta instrucción es establecer comparaciones de magnitud entre dos datos.