

Referencia básica del ensamblador ASM51 de Intel

1. Introducción

Este manual describe los aspectos básicos del ensamblador ASM51 de Intel (v2.2), con especial atención a la sintaxis del programa fuente, directivas y opciones disponibles.

2. Sintaxis del lenguaje ensamblador

2.1. Formato de las líneas de programa

El formato general de las líneas de instrucción es:

[etiqueta:] nemónico [operando] [,operando] [,operando] [;comentario]

en donde el número de operandos (entre 0 y 3) depende de la instrucción en particular.

Entre campos adyacentes pueden existir espacios en blanco o tabulaciones, que serán ignoradas por el ensamblador. **El ensamblador no diferencia entre mayúsculas y minúsculas.**

Los operandos pueden ser de los siguientes tipos:

1) No numéricos:

👉 **Símbolos específicos del ensamblador.** Son palabras reservadas que se emplean necesariamente en algunas instrucciones. Por ejemplo AB en la instrucción MUL AB. Son los siguientes:

A AB C DPTR PC R0 RI R2 R3 R4 R5 R6 R7

El ensamblador reconoce también los símbolos \$ (contador de posición), y AR0,...,AR7 utilizados en combinación con la directiva USING (ver apartado 3).

👉 **Direcciones indirectas.** Utilizan el nombre de un registro para indicar la dirección de un dato. Por ejemplo @R0 en la instrucción MOV A,@R0.

2) Numéricos:

👉 **Datos inmediatos.** Indican una constante. Por ejemplo #25 en la instrucción MOV A,#25.

👉 **Direcciones de byte.** Indican la posición de la memoria en la que se encuentra un dato de tipo byte. Por ejemplo, 40 en la instrucción MOV A,40.

👉 **Direcciones de bit.** Indican la posición de la memoria en la que se encuentra un dato de tipo bit. Por ejemplo 20H en la instrucción SETB 20H.

👉 **Direcciones de programa (código).** Indican la posición de la memoria en la que se encuentra una instrucción. Por ejemplo el número 346 en la instrucción AJMP 346.

Los tipos numéricos se pueden representar mediante expresiones simbólicas, con la condición de que puedan ser convertidas a un número. El proceso de conversión se denomina *evaluación de la expresión*, y se dice que una expresión *evalúa a* un número.

El ensamblador intenta evaluar las expresiones que encuentra en un programa fuente. Cuando no puede evaluar una expresión, deja esta tarea al enlazador (linker). Las expresiones evaluables por el ensamblador se

denominan *expresiones absolutas*. Las que debe evaluar el enlazador se denominan expresiones reubicables (relocatable). El estudio de las expresiones reubicables está fuera del objetivo de este manual, por lo que el lector interesado debe consultar la documentación facilitada por el fabricante.

El margen de valores permitido para un operando numérico depende de la instrucción en la que se utiliza. Las expresiones simbólicas se forman utilizando símbolos predefinidos o definidos por el usuario y números que se combinan mediante *operadores de la fase de ensamblado* (ver apartados 2.2, 2.3 y 2.4).

El área direccionable del 8051 se divide en las 5 zonas que se indican a continuación, denominadas *segmentos*.

☞ Segmento de datos (bytes) direccionables de modo directo (DATA).

☞ Segmento de bits (BIT).

☞ Segmento de datos en memoria externa (XDATA).

☞ Segmento de programa o código (CODE).

☞ Segmento de datos direccionables de modo indirecto (IDATA).

El programador puede definir símbolos de los tipos indicados entre paréntesis (DATA, etc.) para determinar su categoría. El ensamblador realiza una comprobación del tipo de operando numérico empleado en cada instrucción, y advierte de posibles errores. Si el usuario no desea que se realice esta comprobación puede emplear un tipo genérico denominado *número* que es aceptado como operando en cualquier instrucción.

En el apartado 2.4 se describe la forma en la que el proceso de evaluación de expresiones determina el tipo del resultado de la evaluación, en función de los operadores empleados.

El ensamblador permite utilizar dos nemónicos que no representan directamente ninguna instrucción del repertorio del 8051: JMP y CALL. Cuando el ensamblador encuentra el nemónico JMP lo sustituye por una instrucción de salto (SJMP, AJMP, LJMP) que sea válida para el destino dado. Si el salto es hacia atrás, elige el tipo de salto más corto posible. El nemónico CALL funciona de forma similar, pero con las instrucciones ACALL, LCALL.

2.2. Evaluación de expresiones en la fase de ensamblado (assembly-time)

Las expresiones simbólicas están formadas por símbolos, números y operadores. Su evaluación conduce siempre a un número de 16 bits. Si se utiliza un número negativo, el evaluador lo representa en complemento a 2. A todos los demás efectos, la evaluación se realiza considerando todos los números positivos en el margen 0 a 65535 (0 a FFFF).

En general, se permite la referencia a símbolos que se definen más adelante (forward reference).

Los números se pueden expresar en formato decimal, hexadecimal, octal y binario. Por defecto, el ensamblador los considera en formato decimal. Para indicar un formato diferente se debe añadir un sufijo determinado:

FORMATO	SUFIJO	OBSERVACIONES
Decimal	D o nada	
Hexadecimal	H	el valor no puede empezar por letra
Octal	O o Q	
Binario	B	

Los números no pueden empezar por una letra, ya que el ensamblador los consideraría como símbolos alfanuméricos. Por lo tanto, para indicar un valor hexadecimal que empiece por A,...,F se debe añadir un cero a la izquierda. Por ejemplo 0A5H.

No se pueden utilizar en las expresiones números mayores de 65535 (0FFFFH).

Se permite el uso de caracteres simples y cadenas en las expresiones. Se deben encerrar entre comillas simples ('a', 'abc'). Para incluir una comilla simple como parte de una cadena se deben poner 2 seguidas. Por ejemplo 'abc"d' es una cadena formada por las letras a, b, c, una comilla simple y d.

Los caracteres simples evalúan a su valor ASCII. Las cadenas de dos caracteres evalúan al valor ASCII del segundo carácter más 256 por el valor ASCII del primero. Las cadenas de más de 2 caracteres sólo se pueden utilizar con la directiva DB (ver apartado 3). Ejemplos:

'A' + 5 evalúa a 70 (el valor ascii de A es 65)

'A0' evalúa a $256 \cdot 65 + 48 = 16688$ (el valor ascii de 0 es 48)

2.3. Utilización de símbolos

Los símbolos se caracterizan por los 4 atributos cuyos posibles valores se indican a continuación (no todas las combinaciones son válidas):

👉 Tipo: register, segment, number, address.

👉 Segmento: DATA, BIT, XDATA, CODE, IDATA.

👉 Alcance (scope): local, public, extern.

👉 Valor: nombre de un registro, dirección base de un segmento, constante, dirección.

Ejemplos de diferentes tipos de símbolos:

El símbolo velocidad definido mediante la directiva EQU siguiente representa al registro R4 y es de tipo registro. Las directivas se describen en el apartado 3.

velocidad EQU R4

Los símbolos empleados en la definición de segmentos reubicables son del tipo segmento.

El símbolo recarga definido mediante la directiva siguiente es del tipo number (número genérico) y puede ser utilizado en cualquier expresión.

recarga EQU 52H

El símbolo espacio definido mediante la directiva siguiente es del tipo address (dirección), lo que significa que puede emplearse, por ejemplo, como operando en una instrucción que requiera la dirección de un dato.

espacio DATA 40H

Los símbolos del tipo address (dirección) pueden tener alcance (scope) local, public (público) o extern (externo). El atributo alcance se utiliza en la programación modular para permitir que el enlazador (linker) realice la combinación de varios módulos ensamblados por separado.

El significado del atributo valor depende del tipo de símbolo. Para el tipo registro es el nombre del registro en ASCII (p. ej. R4). Para el tipo segmento es el valor de la dirección base del segmento (determinada por el enlazador). Para el tipo número es el valor constante asignado. Para el tipo dirección, el valor puede representar la dirección absoluta (si el símbolo es absoluto) o el desplazamiento (offset) con respecto a la dirección base de su segmento (si el símbolo es reubicable).

El archivo REG51.PDF incluye la definición de un conjunto de símbolos de los tipos dirección de byte y dirección de bit. Estos símbolos son:

Direcciones de byte:

SÍMBOLO	DIRECCION HEX	SIGNIFICADO
ACC	E0	Direcc. del acumulador
B	F0	Direcc. del registro B
DPH	83	Direcc. de DPTR (MSBy)
DPL	82	Direcc. de DPTR (LSBy)
IE	A8	Direcc. de IE
IP	B8	Direcc. de IP
P0	80	Direcc. del puerto 0
P1	90	Direcc. del puerto 1
P2	A0	Direcc. del puerto 2
P3	B0	Direcc. del puerto 3
PSW	D0	Direcc. de PSW
SBUF	99	Direcc. de SBUF
SCON	98	Direcc. de SCON
SP	81	Direcc. de SP
TCON	88	Direcc. de TCON
TH0	8C	Direcc. de TH0
TH1	8D	Direcc. de TH1
TL0	8A	Direcc. de TL0
TL1	8B	Direcc. de TL1
TMOD	89	Direcc. de TMOD

Direcciones de bit:

SÍMBOLO	DIRECCION HEX	SIGNIFICADO
CY	D7	PSW.7 (flag de acarreo)
AC	D6	PSW.6 (acarreo auxiliar)
F0	D5	PSW.5 (flag 0)
RS1	D4	PSW.4 (selecc. banco MSB)
RS0	D3	PSW.3 (selecc. banco LSB)
OV	D2	PSW.2 (flag de desbordam.)
P	D0	PSW.0 (flag de paridad)
TF1	8F	TCON.7 (flag de desbord. temp. 1)
TR1	8E	TCON.6 (control avance del temp. 1)
TF0	8D	TCON.5 (flag de desbord. temp. 0)
TR0	8C	TCON.4 (control avance del temp. 0)
IE1	8B	TCON.3 (flag de interr. temp. 1)
IT1	8A	TCON.2 (nivel/flanco int. ext. 1)
IE0	89	TCON.1 (flag de interr. temp. 0)
IT0	88	TCON.1 (nivel/flanco int. ext. 0)
SM0	9F	SCON.7 (bit 2 de control p. serie)
SM1	9E	SCON.6 (bit 1 de control p. serie)
SM2	9D	SCON.5 (bit 0 de control p. serie)
REN	9C	SCON.4 (autorización recepción)
TB8	9B	SCON.3 (bit 8 a transmitir)
RB8	9A	SCON.2 (bit 8 recibido)
TI	99	SCON. 1 (flag de fin de transmisión)
RI	98	SCON.0 (flag de byte recibido)
EA	AF	IE.7 (autoriz. general de interrupc.)
ES	AC	IE.4 (autoriz. interr. p. serie)

ET1	AB	IE.3 (autoriz. interr. temp. 1)
EX1	AA	IE.2 (autoriz. interr. ext. 1)
ET0	A9	IE.1 (autoriz. interr. temp. 0)
EX0	A8	IE.0 (autoriz. interr. ext. 0)
RD	B7	P3.7 (señal de lectura memoria ext.)
WR	B6	P3.6 (señal de escrit. memoria ext.)
T1	B5	P3.5 (entrada impulsos ext. temp. 1)
T0	B4	P3.4 (entrada impulsos ext. temp. 0)
INT1	B3	P3.3 (entrada interr. ext. 1)
INT0	B2	P3.2 (entrada interr. ext. 0)
TXD	B1	P3.1 (salida p. serie)
RXD	B0	P3.0 (entrada p. serie)
PS	BC	IPA (prioridad interr. p. serie)
PT1	BB	IP.3 (prioridad interr. temp. 1)
PX1	BA	IP.2 (prioridad interr. ext. 1)
PT0	B9	IP. 1 (prioridad interr. temp. 0)
PX0	B8	IP.O (prioridad interr. ext. 0)

Los símbolos definidos en esta tabla evalúan a los valores numéricos indicados, que representan las direcciones de los registros correspondientes. Por lo tanto, la instrucción:

ADD ACC,#2

se convierte en:

ADD 0E0H,#2

y no es válida, ya que el modo de direccionamiento **direccion,valor** no existe para ADD.

Sin embargo la instrucción:

ADD A,#2

si es válida, porque A es una palabra reservada (no un símbolo a evaluar).

El símbolo \$ está predefinido internamente y representa la dirección del primer byte de la instrucción en la que se utiliza.

2.4. Operadores de la fase de ensamblado

En las expresiones simbólicas se pueden utilizar los operadores que se describen a continuación.

ARITMETICOS: +, -, *, /, MOD.

LOGICOS: OR, AND, XOR, NOT (actúan bit a bit sobre los 16 bits).

RELACIONALES: =, <>, <, <=, >, >= (verdadero=FFFF, falso=0).

ESPECIALES: SHR Y SHL (desplazamientos), HIGH y LOW (evalúan al byte alto o bajo del operando respectivamente)

Todos los operadores designados mediante una cadena de caracteres alfabéticos (MOD, OR, etc.) deben llevar espacios a ambos lados, para que el ensamblador los pueda distinguir de los símbolos.

El orden de precedencia de mayor a menor es el que se indica a continuación. Los operadores que se muestran en la misma línea poseen igual precedencia.

HIGH, LOW
*, /, MOD, SHL, SHR
+, -
=, <>, <, <=, >, >=
NOT
AND
OR, XOR

A igual precedencia, la evaluación se realiza de izquierda a derecha. Se pueden utilizar paréntesis para obtener el orden de precedencia deseado.

Para determinar de qué tipo es el resultado de una operación, se aplican las reglas siguientes:

- 1) El resultado de una operación unaria (+ y - unarios, NOT, LOW, HIGH) es del mismo tipo que su operando. Ejemplo: -(tipo DATA)= tipo DATA
- 2) El resultado de todas las operaciones binarias (entre 2 operandos) excepto + y - es del tipo genérico NUMERO (number). Ejemplo: (tipo DATA)*(tipoDATA)= tipo NUMBER.
- 3) El resultado de las operaciones binarias (entre 2 operandos) + y - es del tipo número genérico salvo que uno (y sólo uno) de los operandos no sea número genérico. En este caso se conserva el tipo del que no es número genérico. Ejemplos:

(tipo DATA)+(tipo DATA)= tipo NUMBER
(tipo DATA)+(tipo NUMBER)= tipo DATA

3. Directivas del ensamblador

Además de las líneas de instrucciones, el programa fuente puede incluir una serie de líneas que facilitan al ensamblador una información adicional o bien le piden que realice determinadas tareas. Estas líneas están formadas por directivas que pueden incluir operandos.

Las directivas se clasifican en 5 grupos y se describen en los siguientes subapartados.

3.1 Directivas de definición de símbolos

Las utiliza el programador para definir sus propios símbolos. Los símbolos deben empezar por una letra o '?' o '_'. El resto de los caracteres pueden ser además números. Aunque los símbolos pueden tener hasta 255 caracteres, sólo los 31 primeros son tenidos en cuenta por el ensamblador. Las letras minúsculas son convertidas a mayúsculas, por lo que los símbolos **bucle** y **BUCLE** son iguales para el ensamblador.

Existen una serie de palabras reservadas que no pueden ser empleadas como nombres de símbolos, porque poseen un significado especial para el ensamblador. Son todos los nemónicos de instrucciones, los

operadores de la fase de ensamblado, los símbolos predefinidos, los atributos de los segmentos y las directivas del ensamblador.

Las etiquetas asociadas a líneas del programa son un caso particular de símbolos. Deben ir seguidas de ':' y solamente se pueden asociar a líneas vacías, instrucciones, directivas DB, DW, DS y DBIT.

En el caso de las directivas EQU, SET, DATA, IDATA, XDATA, BIT, CODE y SEGMENT, el símbolo que precede a la directiva no puede llevar los dos puntos ':' finales.

A continuación se describen brevemente las directivas de definición de símbolos.

La directiva SEGMENT se utiliza para definir segmentos reubicables. Su descripción está fuera del objetivo de este manual.

La directiva EQU se utiliza con el formato:

nombre EQU expresión

se asigna al símbolo **nombre** el valor que resulta de evaluar la expresión. La expresión no debe contener referencias hacia adelante (forward references). También puede utilizarse con el formato:

nombre EQU símbolo_especial

en cuyo caso se asigna al símbolo nombre el símbolo especial indicado (por ejemplo R3).

Los símbolos definidos mediante EQU no se pueden redefinir posteriormente.

La directiva SET es similar a la directiva EQU, pero permite redefinir los símbolos. Un mismo símbolo no puede ser definido mediante EQU y SET en el mismo programa. Los formatos de SET son:

nombre SET expresión

nombre SET símbolo_especial

La directiva BIT asigna una dirección de bit a un símbolo. La expresión debe evaluar a número o dirección de bit. El formato es:

nombre BIT expresión

donde la expresión puede utilizar el formato B.b en donde B evalúa a la dirección de un byte y b a un número entre 0 y 7. El byte especificado debe ser accesible a nivel de bit. El símbolo adquiere el tipo dirección de bit.

Ejemplos:

```
motor BIT P1.0
marcha BIT 0
mayor BIT 2.7
```

La directiva DATA asocia a un símbolo una dirección de RAM interna para acceso DIRECTO. Si la dirección es mayor de 127, se refiere a un SFR. El símbolo obtiene el tipo dirección de dato (byte). Ejemplo del formato:

estado DATA 40H

La directiva IDATA asigna a un símbolo, una dirección de RAM interna para acceso INDIRECTO. En el caso del 8051 la dirección asignada no debe superar 127. Ejemplo:

estado IDATA 40H

La directiva XDATA asigna a un símbolo una dirección de dato en memoria externa. Ejemplo:

variable XDATA 2000H

La directiva CODE asigna a un símbolo una dirección de la memoria de programa (código). Ejemplo:

Intext0 CODE 03H

3.2 Directivas de inicialización y reserva de espacio

Las utiliza el programador para reservar espacio para datos o inicializarlos.

La directiva DS se utiliza para reservar espacio en cualquier segmento excepto el de bits.

DS 20

reserva 20 bytes en el segmento actual (incrementa el contador de posiciones de ese segmento en 20).

La directiva DBIT se utiliza para reservar espacio en el segmento de bits.

DBIT 4

reserva 4 bits en el segmento actual que debe ser un segmento de bits (incrementa el contador de posiciones de ese segmento en 4).

Las directivas DB y DW se utilizan para definir constantes en la memoria de programa (segmento de código). Si llevan una etiqueta, ésta apunta a la posición ocupada por el primer valor de los que se definen en la lista.

La directiva DB posee varios formatos, que se describen mediante ejemplos.

DB 50H

ubica el valor 50H en la posición actual del segmento de programa (código). El contador de posiciones se incrementa en 1 byte.

DB 20H, 46, 'A'

ubica los valores 20 hexadecimal, 46 decimal y 65 decimal (el ascii de 'A') en posiciones sucesivas del segmento de código a partir de la actual. El contador de posiciones se incrementa en 3 bytes.

DB 'alarma'

ubica los valores ascii de los caracteres que componen la cadena en posiciones sucesivas del segmento de código a partir de la actual. El contador de posiciones se incrementa en 6 bytes.

La directiva DW es similar a DB pero ubica palabras en vez de bytes (valores de 16 bits). El byte más significativo se almacena primero. No se permiten cadenas de texto con más de 2 caracteres. Si se usa el símbolo \$, evalúa a la posición de la palabra que se está definiendo.

DW 1400, 0FFFFH, 'AB', 'A'

ubica los enteros 1400 decimal, FFFF hexadecimal, $256 * \text{ascii}(A) + \text{ascii}(B)$ y $\text{ascii}(A)$. Este último valor queda con el byte más significativo a 0.

3.3 Directivas de enlace

Permiten indicar el alcance de un conjunto de símbolos. Se describen brevemente, dado que su aplicación se encuentra en el desarrollo de proyectos de software formados por varios módulos, tema que está fuera del objetivo de este manual.

La directiva PUBLIC sirve para indicar que un símbolo o conjunto de símbolos debe ser reconocido por otros módulos que se enlacen con el actual. Si no se indica de forma expresa, se entiende que el alcance es local, y los símbolos no se reconocerán desde otros módulos. Ejemplo:

PUBLIC pepe, manolo

La directiva **EXTRN** sirve para indicar una lista de símbolos que deben ser reconocidos en el módulo actual, pero han sido definidos en otro módulo. Se debe indicar el tipo de segmento al que pertenecen los símbolos. Por ejemplo:

EXTRN CODE (vertemp, calculo), DATA (velocidad, estado), NUMBER (recarga)

Para asignar un nombre a un módulo de programa se utiliza la directiva **NAME**. Si esta directiva no está presente en un módulo, se le asigna por defecto el nombre del archivo sin path y sin extensión. En este caso el nombre del archivo no debe empezar por un número.

3.4 Directivas de estado del ensamblador

Las directivas **ORG**, **END** y **USING** se describen en este apartado.

La directiva **ORG** permite definir el valor del contador de posiciones de un segmento. Su principal aplicación es la de fijar la posición a partir de la cual se cargará el código que sigue a la directiva. Si el código es absoluto, **ORG** indica la posición real en la memoria de programa del código que la sigue. Por ejemplo:

```
ORG 0  
ajmp inicio  
ORG 3  
ajmp intext0
```

hace que la instrucción **ajmp inicio** se ubique en la dirección 0 de la memoria de programa y la instrucción **ajmp intext0** en la dirección 3 de la memoria de programa. Se permite el uso de más de una directiva **ORG** en el mismo programa. Ni el ensamblador ni el enlazador comprueban que el uso de **ORG** sea correcto. Si el uso de **ORG** hace que se defina código donde ya se había definido, el nuevo código sustituye al antiguo (lo sobrescribe).

La directiva **END** debe aparecer como la última de todo programa.

La directiva **USING** permite indicar al ensamblador cuál es el banco de registro activo en cada momento. Si se utiliza adecuadamente, se pueden utilizar los símbolos predefinidos **AR0**,...,**AR7** para representar las direcciones de los 8 registros activos. No es una instrucción del microcontrolador para seleccionar el banco de registros. Ejemplo de uso:

```
USING 1  
PUSH AR0
```

pone en la pila el contenido del registro **R0** del banco 1.

3.5 Directivas de selección de segmento

Permiten indicar el segmento en el que se debe incluir el código o datos que la siguen. Se describen brevemente, dado que su aplicación se encuentra en el desarrollo de proyectos de software formados por varios módulos, tema que está fuera del objetivo de este manual. Son las siguientes: **RSEG**, **CSEG**, **XSEG**, **DSEG**, **ISEG**, **BSEG**. Ejemplo de utilización para activar un segmentos reubicable de datos, definido previamente:

```
misegmdatos SEGMENT DATA  
  
RSEG misegmdatos
```

Ejemplo de utilización para activar un segmento absoluto del tipo dirección de bits:

BSEG AT 20H

4. Controles del ensamblador

Existe una serie de palabras clave que permiten determinar la forma en que opera el ensamblador y no se consideran directivas porque no controlan directamente la generación del código, sino aspectos accesorios relacionados casi todos con el formato del archivo de salida LST. Se denominan controles.

Los controles pueden utilizarse indistintamente formando parte del programa fuente o como parámetros en la línea de llamada al ensamblador. Algunos de ellos, denominados primarios (P), no pueden aparecer en cualquier lugar del programa fuente sino exclusivamente en sus primeras líneas. Los restantes se denominan controles generales (G).

A continuación se facilita una tabla resumen de los controles disponibles. Se deben indicar precedidos del símbolo \$.

Nombre	P/G	Por defecto	Abreviatura	Función
DATE(fecha)	P	DATE()	DA	Crea una cabecera de 9 caracteres máximo en el archivo LST
DEBUG	P	NO	DB	Incluye información de depuración en el archivo LST
NODEBUG	P	SI	NODB	Suprime la información
EJECT	G	--	EJ	Inserta salto de página en LST
ERRORPRINT(archivo)	P	NO	EP	Asigna un archivo para enviar los mensajes de error
NOERRORPRINT	P	SI	NOEP	Suprime archivo para errores
GEN	G	NO	GE	Escribe las llamadas a macros y su expansión en el LST
GENONLY	G	SI	GO	Escribe sólo la expansión de las macros en el LST
NOGEN	G	NO	NOGE	Escribe sólo el fuente original en LST (las llamadas a macros)
INCLUDE(archivo)	G	--	IC	Incluye un archivo dentro del programa
LIST	G	SI	LI	Activa la salida en el LST de las líneas que siguen
NOLIST	G	NO	NOLI	Desactiva la salida en el LST de las líneas que siguen
MACRO(porcentaje)	P	MACRO(50)	MR	Expande las macros. Utiliza para ello la fracción de la memoria disponible que se le indica
NOMACRO	P	NO	NOMR	No expande las macros
MOD51	P	SI	MO	Reconoce los SFRs del 8051
NOMOD51	P	NO	NOMO	No reconoce los SFRs del 51
OBJECT(archivo)	P	OJ(fue.obj)	OJ	Nombre del archivo .OBJ
NOOBJECT	P			No crear .OBJ
PAGING	P	SI	PI	Hace que el LST incluya cambios de página y repetición de cabeceras
NOPAGING	P	NO	NOPI	Suprime cambios de página en el archivo LST
PAGELength(n)	P	PL(60)	PL	Ajusta la longitud de página LST
PAGewidth(n)	P	PW(120)	PW	Ajusta el ancho de página LST
PRINTFILE(archivo)	P	PR(fue.lst)	PR	Nombre del archivo .LST

NOPRINT	P	NO	NOPR	No crear .LST
SAVE	G	--	SA	Salva el estado actual de los controles para LST y macros
RESTORE	G	--	RS	Recupera el último estado salvado de los controles
REGISTERBANK(n,...)	P	RB(0)	RB	Indica qué bancos de registros se utilizan en el módulo
NOREGISTERBANK	P	NO	NORB	Indica que no se usa ningún banco de registros
SYMBOLS	P	SI	SB	Crea en el LST una tabla con todos los símbolos utilizados
NOSYMBOLS	P	NO	NOS13	No crea la tabla de símbolos
TITLE(nombre)	G	TT()	TT	Título de hasta 60 caracteres para las siguientes páginas del LST
XREF	P	NO	XR	Crea una tabla de referencia cruzada con todos los símbolos utilizados
NOXREF	P	SI	NOXR	No crea la tabla de referencia cruzada

5. Macroinstrucciones

Se llama macroinstrucción (macro) a un bloque de código que el ensamblador genera de forma automática cada vez que el programa fuente lo requiere. Para ello se debe seguir el siguiente proceso:

- ✓ Definir la macro
- ✓ Utilizar la macro

Para definir una macro se debe incluir en el programa fuente un bloque de definición con la siguiente sintaxis:

```
%*define (nombre)(
; código a ejecutar que se denomina CUERPO DE LA MACRO
)
```

en donde *nombre* representa el nombre que se quiere asignar a la macro. El cuerpo de la macro es el bloque de código que se desea reproducir cada vez que se utilice la macro.

Cada vez que se desea utilizar la macro, se debe escribir una línea de llamada a la macro con la siguiente sintaxis:

```
%nombre
```

en donde *nombre* debe coincidir con el nombre asignado en la definición.

Se denomina expansión de una macro (o macrosustitución) al proceso por el cual el ensamblador sustituye la línea de llamada por el cuerpo de la macro.

En el siguiente ejemplo se define una macro que espera a recibir un byte por el puerto serie y pone el byte recibido en el acumulador.

```
%*define (recibirbyte)(
jnb RI,$
mov a,SBUF
)
```

El siguiente programa invoca la macro:

```
mov r0,#tabla
```

```
%recibirbyte      ; invocación de la macro
mov @r0,a
```

El resultado de la expansión de la macro realizada por el ensamblador es:

```
mov r0,#tabla
jnb, RI,$
mov a,SBUF
mov @r0,a
```

No se debe confundir el concepto de macro con el de subrutina. El ensamblador repite el cuerpo de la macro cada vez que encuentra una invocación. En el momento de la ejecución no existe ningún proceso de salto a subrutina.

Existe la posibilidad de definir macros parametrizadas. Estas macros se expanden de forma diferente según los parámetros que se utilicen en la invocación. El formato de definición es:

```
%*define (nombre (par1, par2, ...)) (
; código a ejecutar que se denomina CUERPO DE LA MACRO
; cada parámetro se utiliza con la sintaxis %pari
)
```

La invocación tiene el formato:

```
%nombre (par1,par2,...)
```

El siguiente ejemplo es una variante parametrizada de la macro del ejemplo anterior.

```
%*define (recibirbyte(x))(
jnb RI,$
mov %x,SBUF
)
```

Los siguientes programas invocan la macro con diferentes parámetros.

Caso 1:

```
%recibirbyte(a)
```

El resultado de la expansión de la macro realizada por el ensamblador es:

```
jnb RI,$
mov a,SBUF
```

Caso 2:

```
%recibirbyte(dato)
```

El resultado de la expansión de la macro realizacia por el ensamblador es:

```
jnb RI,$
mov dato,SBUF
```

La combinación de macros y subrutinas ofrece una gran flexibilidad.

Es habitual reservar el uso de macros a las aplicaciones de cierta complejidad. En la actualidad es frecuente recurrir al empleo de lenguajes de alto nivel (especialmente al lenguaje C) en lugar de emplear macros.

6. Generación del código ejecutable

El ensamblador crea, en general, código reubicable que almacena en un archivo con la extensión OBJ. Para que este código pueda ser ejecutado por el microcontrolador (o en un simulador) debe ser convertido a código absoluto.

Cuando se desarrollan aplicaciones con varios módulos de programa fuente, o se utilizan bibliotecas (libraries), es necesario enlazar los diferentes módulos objeto mediante un programa enlazador (linker).

En el caso de aplicaciones sencillas formadas por un único módulo fuente, y en las que no se utilizan bibliotecas, es posible generar el código ejecutable de forma más sencilla. Para ello basta con escribir el programa fuente utilizando sólo expresiones absolutas (no reubicables) y sustituir el enlazador por una utilidad de conversión directa. Intel facilita esta utilidad con el nombre de OH.EXE.

El programa OH.EXE parte del archivo OBJ (que debe ser absoluto) y genera un archivo HEX (en formato Intel-Hex) que puede ser leído por la mayoría de simuladores, emuladores y grabadores de micros y memorias comerciales.

Los archivos Intel-Hex son de tipo ASCII. Están formados por una serie de líneas que indican el contenido de la memoria de programa con valores en hexadecimal.

7. Estructura típica de un programa en ensamblador

En este apartado se describe una estructura básica de programa en ensamblador para el 8051.

El programa comienza con varias líneas de comentario que se utilizan para describir de forma resumida la finalidad, principales características y forma de utilización de la aplicación.

A continuación se incluye un bloque de definición de símbolos. Se suelen definir constantes (EQU) y variables de varios tipos (DATA, BIT).

Si se utilizan macros, se suele incluir en este punto el bloque que las define.

A continuación, se ubica el bloque de vectores de entrada. Mediante directivas ORG se apunta a la dirección 0 (vector de RESET) y a las direcciones de entradas (vectores) de interrupciones. Es habitual poner instrucciones de salto (absoluto o largo) en la posición de cada vector.

Después de este bloque se puede utilizar una directiva ORG para asignar la dirección de comienzo del bloque de programa principal. También es posible comenzar con el programa principal inmediatamente a continuación del último vector (sin usar ORG). En todo caso, el bloque principal empieza con la misma etiqueta empleada en la instrucción de salto del vector de reset (por ejemplo, inicio:).

El bloque del programa principal suele estar compuesto de una primera parte para inicializaciones, que sólo se ejecuta una vez, y un bucle sin fin de atención a las entradas del sistema (consulta), gestión de una máquina de estados, etc. En algunos casos, este bucle está vacío (jmp \$), con lo que todo el proceso se desarrolla en la o las subrutinas de interrupción que se utilicen.

A continuación se suelen incluir las subrutinas de atención a las interrupciones, que deben finalizar con la instrucción RETI.

Finalmente, habrá un bloque con el resto de las subrutinas utilizadas en el programa.

El archivo del programa fuente finaliza con la directiva END.

Ejemplo de estructura de programa:

```
*****
;
*****
```

;*Nombre, finalidad, uso, fecha inicial y de revisiones, autor, etc.

*

```
*****
;
*****
```

;Definición de constantes y variables

```
estado data 40H
pulsador bit Pl.0
motor bit Pl.1
;
;Vectores de entrada
org 0
    ajmp inicio
org 3
    ajmp intext0
org 0bh
    ajmp inttim0
org 13
    ajmp intextl
org lbh
    ajmp inttiml
org 23h          ; la del p. serie no se usa en este ejemplo
    reti
;
;Programa principal
;
org 100h
inicio:
;
;inicializaciones que se ejecutan una sola vez
;
acall preptemp0
mov ie, #8fh
mov estado,#0
;
; bucle principal
;
bucle:
acall verentradas
acall maqestados
acall actsalidas
sjmp bucle
;
intext0:
;código que corresponda
reti
;
inttim0:
;código que corresponda
reti
;
intextl:
;código que corresponda
reti
;
inttim1:
```

```
;código que corresponda  
reti  
;  
;Otras subrutinas  
;  
preptemp0:  
;código que corresponda  
ret  
;  
verentradas:  
;código que corresponda  
ret  
;  
maqestados:  
;código que corresponda  
ret  
;  
actsalidas:  
;código que corresponda  
ret  
;  
; fin del archivo  
end
```