

INTRODUCCIÓN AL MICROENSAMBLADOR DEL Am29203

Cada microinstrucción está compuesta en varias microoperaciones que son ejecutadas en paralelo. El lenguaje definido por el Microensamblador Am29203 contiene un gran número de microoperaciones expresadas de forma simbólica, cuya intersección, mediante el operando '&' forma una microinstrucción completa. El microensamblador se encarga de traducir cada microinstrucción expresada en lenguaje simbólico a un conjunto de caracteres hexadecimales que pueden ser posteriormente cargados en la tarjeta y ser ejecutados.

Tipos de instrucciones

Por un lado tendremos las directivas que facilitan el trabajo del microcompilador (por ejemplo *EQU*, *ORG*).

Los comentarios deben empezar por ; ignorándose el resto de la línea.

Podremos utilizar etiquetas para definir posiciones de memoria sin necesidad de saber su localización exacta. Las etiquetas deben ir al principio de cada microinstrucción y pueden terminar adicionalmente en :

No se podrán usar ninguna de las palabras reservadas por el microensamblador, como son los mnemónicos o los nombres de registros (.0, .1, .2,..., .F, RQ, IR, RC, PC, IRA, IRB).

Ejemplo:

```
LD RQ,#H10      ; hacemos el registro RQ=16
CLR .5          ; ponemos el registro .5=0
etiqueta1:
INC .5          ; incrementamos .5 hasta que sea igual a RQ
CMP RQ,.5 & LD_UFL
BUNZ etiqueta1
```

Por otro lado, tendremos las microinstrucciones propiamente dichas, que están compuestas por los mnemónicos definidos para la tarjeta y que son los que se compilan y ejecutan (para ver el conjunto de microinstrucciones disponible, mirar la ayuda de los programas, fichero *AMDHELP.HLP*, sección "*Ayuda sobre el Microensamblador | Mnemónicos del Microensamblador*").

Una microinstrucción puede estar compuesta de varias microoperaciones separadas cada una por un carácter '&' debiendo estar todas ellas escritas en la misma línea. Estas microoperaciones deben ser independientes unas de otras para que el procesador pueda ejecutarlas en paralelo.

El microensamblador se encarga de verificar si no hay ninguna incompatibilidad de formato en cada microinstrucción. Debido a que hay campos que se superponen se debe tener cuidado de no utilizar microoperaciones que exijan valores diferentes para campos superpuestos.

Es, por tanto, necesario que el programador conozca relativamente bien el formato de las microoperaciones para no caer en errores de compatibilidad de formato en cada microinstrucción. NOP es una microoperación incompatible con las demás. HALT, LD_UFL y LD_MFL son microoperaciones compatibles con todas las demás.

Mnemónicos del Microensamblador

<u>ADD</u>	<u>ADDMC</u>	<u>ADDUC</u>	<u>AND</u>
<u>BMC</u>	<u>BMN</u>	<u>BMNC</u>	<u>BMNN</u>
<u>BMNV</u>	<u>BMNZ</u>	<u>BMV</u>	<u>BMZ</u>
<u>BR</u>	<u>BUC</u>	<u>BUN</u>	<u>BUNC</u>
<u>BUNN</u>	<u>BUNV</u>	<u>BUNZ</u>	<u>BUY</u>
<u>BUZ</u>	<u>CALL</u>	<u>CALLMC</u>	<u>CALLMN</u>
<u>CALLMNC</u>	<u>CALLMNN</u>	<u>CALLMNV</u>	<u>CALLMNZ</u>
<u>CALLMV</u>	<u>CALLMZ</u>	<u>CALLUC</u>	<u>CALLUN</u>
<u>CALLUNC</u>	<u>CALLUNN</u>	<u>CALLUNV</u>	<u>CALLUNZ</u>
<u>CALLUV</u>	<u>CALLUZ</u>	<u>CLR_FLG</u>	<u>CLR</u>
<u>CMP</u>	<u>DATA</u>	<u>DEC</u>	<u>EQU</u>
<u>HALT</u>	<u>IN</u>	<u>INC</u>	<u>END</u>
<u>JMAP</u>	<u>JMPZ</u>	<u>LD_MFL</u>	<u>LD_UFL</u>
<u>LD</u>	<u>LOOP</u>	<u>MOVE</u>	<u>NAND</u>
<u>NEG</u>	<u>NOP</u>	<u>NOR</u>	<u>NOT</u>
<u>ORG</u>	<u>OR</u>	<u>RETMC</u>	<u>RETMNC</u>
<u>RETMNN</u>	<u>RETMN</u>	<u>RETMNV</u>	<u>RETMNZ</u>
<u>RETMV</u>	<u>RETMZ</u>	<u>RET</u>	<u>RETUC</u>
<u>RETUNC</u>	<u>RETUNN</u>	<u>RETUN</u>	<u>RETUNV</u>
<u>RETUNZ</u>	<u>RETUV</u>	<u>RETUZ</u>	<u>ROL</u>
<u>ROL</u>	<u>RORC</u>	<u>ROR</u>	<u>SET_FLG</u>
<u>SET</u>	<u>SHLA</u>	<u>SHL</u>	<u>SHRA</u>
<u>SHR</u>	<u>ST</u>	<u>SUBMC</u>	<u>SUB</u>
<u>SUBUC</u>	<u>SWP_FLG</u>	<u>TEST</u>	<u>XNOR</u>
<u>XOR</u>			

Las constantes numéricas que se utilicen pueden ser expresadas en hexadecimal (H'3F), en decimal (D63 o simplemente 63) o en octal (Q77).

Registros del procesador Am29203

El procesador Am29203 posee 16 registros de propósito general en su ALU. Estos registros de 16 bits son nombrados según la notación del IEEE por .0, .1, ..., .F. Estos registros se pueden emplear de dos formas distintas: como registro A o como registro B.

Si un registro se utiliza como registro A sólo será accesible en operaciones de lectura y se podrá utilizar:

- como dirección de memoria
- como operando de la ALU.

Si un registro se utiliza como registro B será accesible tanto en lectura como en escritura pero no se podrá utilizar como dirección de memoria. Se podrá utilizar:

- para almacenar el resultado de una operación de la ALU
- para almacenar el contenido de otro registro
- para almacenar el contenido de una posición de memoria
- como operando de la ALU.

Además de estos registros, la ALU tiene otro registro interno de 16 bits que denotamos por RQ (registro Q). Este registro es accesible tanto en lectura como en escritura y se puede utilizar:

- como operando de una microoperación
- para almacenar el resultado de una microoperación
- para almacenar el contenido de otro registro
- para ser cargado directamente por una constante numérica

El procesador además cuenta con un registro de instrucción (IR) de 16 bits que se encarga de almacenar una instrucción procedente de la memoria RAM del procesador.

Esta memoria (que no debe ser confundida con la memoria de control) consta de 1Kpalabras de 16 bits. Cada una de estas palabras puede contener un dato o una instrucción (no confundir con una microinstrucción). Los 8 bits más significativos de IR (que forman el código de operación) se hacen pasar por una ROM interna que 'mapea' esos 8 bits en una cierta dirección de la memoria de control (Ver la sección "**Implementación de la fase de captación**" para una explicación más detallada).

Código de operación	IRA	IRB
---------------------	-----	-----

Los 8 bits menos significativos están divididos en dos campos IRA e IRB. Cada uno de estos campos de 4 bits codifican cada uno de los 16 registros de propósito general de la ALU. Así, por ejemplo, si IR acaba por 4F, IRA sería el registro .4 e IRB sería el registro .F

Ejemplo:

*si el código de operación de la suma (instrucción máquina) es H'03, y en una posición de memoria principal tenemos: 0345
estaríamos ejecutando: SUMA R4,R5
y desde el microprograma de esta instrucción máquina, IRA hará referencia a R4, e IRB hará referencia a R5.*

Debido a que tenemos una memoria principal y una memoria de control, también tenemos registros de estado para cada una de ellas. El micro registro de estado (USR) será utilizado principalmente para los saltos y llamadas condicionales del microprograma mientras que el macro registro de estado (MSR) se utilizará para las que se hagan en el programa situado en la memoria principal. Ambos registros poseen estos 4 bits:

- 1 bit de desbordamiento (V)
- 1 bit de acarreo (C)
- 1 bit de número negativo (N)
- 1 bit de cero (Z)

Por último, en el secuenciador de la tarjeta tenemos una pila hardware de 9 niveles y un registro contador (RC) de 10 bits que se puede cargar con un número inmediato y que se utiliza principalmente para las operaciones de LOOP.

Formato de las instrucciones máquina y uso de constantes

Hemos visto anteriormente que el formato de las instrucciones máquina es el siguiente:

Código de operación	IRA	IRB
---------------------	-----	-----

De forma que sólo hay espacio para el código de operación y especificar dos registros. En ocasiones queremos utilizar datos numéricos directamente en la microinstrucción (para cargar un registro con un valor determinado).

En este caso, el dato numérico se encontrará en la siguiente palabra de memoria principal. Si, por ejemplo, queremos asignar un valor a un registro, y después operar con él (instrucciones a partir de la dirección H'0A de la memoria principal):

CARGAR R3,#0E78

SUMAR R4,R3

supongamos el código de operación de la carga igual a H'07 y el de la suma H'09.

las instrucciones y datos en memoria estarán almacenados de la siguiente forma:

dirección	contenido de la palabra de memoria		
0A	07	0	3
0B	0E78		
0C	09	4	3

Estructura de un microprograma Am29203

Un microprograma en microensamblador Am29203 está compuesto por una serie de microinstrucciones (una por línea), comentarios, declaraciones y directivas de compilación, acabando todo con la palabra reservada END.

Realmente, no tendremos que hacer más que un fichero de texto, tal y como hacíamos los programas ensamblador de la primera práctica, o los programas ensamblador para el CODE-2 (Introducción a los Computadores de 1º).

En este caso, al principio del programa podremos renombrar registros usando la directiva EQU (como definíamos constantes en ensamblador del 8086). De esta forma, la programación será más fácil, ya que podremos acceder a cada uno de los registros por nombres simbólicos:

EQU

Formato: Identificador EQU registro

Acción: su papel es renombrar registro de forma que facilite su utilización en el programa ya que se supone que tendrá un nombre más intuitivo. registro puede ser uno de los 16 registros principales, RQ, IR, RC, PC, IRA ó IRB.

Ejemplos:

CP EQU .D

SP EQU .3

Por otro lado, cada bloque del microprograma podrá ubicarse en la dirección de la memoria de control que nos interese (ya sean bloques de instrucciones o datos), usando la directiva ORG (funciona de forma similar al 8086 o el CODE-2).

ORG

Formato: ORG dirección

Acción: Fija la dirección de la memoria de control donde estará la siguiente microinstrucción. dirección se puede expresar en base decimal, hexadecimal u octal .

Restricciones:

dirección no debe exceder del límite de la memoria de control (H'3FE). Todos los saltos a una zona superior a H'200 tendrán una

advertencia ya que en esa zona se carga la ROM interna de la placa y, si no se tiene cuidado, lo que escribamos ahí puede ser machacado. Si no se va a cargar la ROM no hay ningún problema.

Ejemplos:

```

ORG H'0
LD .0,#H'0 ;al principio del microprograma (dirección H'0
de
LD .1,#H'1 ; la memoria de control) inicializamos registros
....

ORG H'40
;implementación del microprograma de la primera instr. máq.
MOVE .1,.4 ;cargar .4 con el contenido de .1
....

```

A modo de ejemplo, veamos un microprograma completo. Simplemente vamos a renombrar unos registros y vamos a ubicar algún bloque de instrucciones en la memoria de control. No se implementa ni la fase de captación de instrucción. Tampoco se implementa realmente el microprograma correspondiente a una instrucción máquina:

```

CP EQU .E ; indicamos qué registro usaremos como contador de programa
SP EQU .F ; y como puntero de pila en la máquina que estamos definiendo

```

```

ORG H'0 ; inicio de la memoria de control
LD .0,#H'0 ; al principio del microprograma (dirección H'0 de
LD .1,#H'1 ; la memoria de control) inicializamos registros
MOVE .0,CP ; inicializamos el contador de programa

```

```

ORG H'40
;implementación del microprograma de la primera instr. máq.
MOVE .1,.4 ;cargar .4 con el contenido de .1
AND IRB,.A & LD_MFL ; ejemplo de microoperaciones en paralelo
; aquí irían el resto de microinstrucciones del microprograma...

```

```

ORG H'44
;implementación del microprograma de la segunda instr. máq.
ST .0,[SP] ;guardar en la cabecera de la pila el valor del reg .0
BR H'0 ; salto al principio de la memoria de control
; aquí irían el resto de microinstrucciones del microprograma...

```

```

END

```

IMPLEMENTACIÓN DE LA FASE DE CAPTACIÓN Y DECODIFICACIÓN DEL CÓDIGO DE OPERACIÓN

Según vimos en teoría, a partir del código de operación de una instrucción máquina (ensamblador) almacenado en el registro de instrucción (RI), debemos determinar cuál es la primera microinstrucción del microprograma (su implementación usando microinstrucciones) de esa instrucción máquina.

Así pues, en la fase de captación (que será común para todos los microprogramas) deberíamos decodificar, bit a bit ese código de operación, a base de microinstrucciones, y una vez encontrado, saltar a la dirección adecuada.

Pues bien, en el microensamblador del AMD tenemos la alternativa de usar la microinstrucción JMAP, que decodifica el código de operación, usando una ROM de la que obtendrá la dirección a la que saltará (donde empieza el microprograma).

Para entender esto, hay que mirar la tabla (PROM Listing) que contiene el contenido de la PROM (figura *docum.4.jpg* de la documentación escaneada de las figuras referentes a la microarquitectura). En la primera columna de esa tabla está el código de operación y en la tercera la dirección de la memoria de control donde saltaría el JMAP.

La parte que está señalada con "Utilizaremos esta zona" tiene la peculiaridad interesante que $Dirección = opcode * 4$.

Por ej.:

para el código de operación H'20, JMAP salta a H'080.

Es conveniente no utilizar los primeros códigos de operación y dejar las primeras posiciones de la memoria de control para la parte de captación de instrucción, dado que así es posible usar el secuenciamiento:

.. & JMPZ

en la última microinstrucción de la implementación de cada instrucción máquina.

Así pues, se recomienda asignar los códigos de operación a las instrucciones máquina a partir del B'00010000 y en adelante. La posición H'00 de la memoria de control (y hasta la H'3F) se reserva para implementar la fase de captación de instrucción. En cada microprograma, al terminarlo con JMPZ se salta a la posición H'0 de la memoria de control, para ejecutar la fase de captación de la siguiente instrucción máquina.

Un problema típico del JMAP es que no se puede hacer lo siguiente:

INC PC & JMAP

En lugar de ejecutar una sola microinstrucción, habría que sustituirlo por dos:

INC PC

JMAP

debido a que (como puede comprobarse en la Tabla 6.1) la instrucción JMAP activa la señal MAP, pero no la señal PL, y como se ve en la Figura 3-4 la señal PL es necesaria para que la parte de 8 bits del registro de microinstrucción registro PIPELINE) que contiene los números de registros A y B llegue al Am29203. Al no activar PL, esos bits del registro de microinstrucción están en triestado. Como PC (u otro) es uno de **los 16 registros, no puede direccionarse a la vez que se activa en el secuenciador el tipo de salto JMAP.**