



# **ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES I**

**TEMA XIV  
ARQUITECTURA Y PROGRAMACION DE UN  
PROCESADOR DE 16 BITS (II): MC68000**

# 14.1. INTRODUCCIÓN

- Un ***programa escrito en ensamblador***, es un texto escrito siguiendo ciertas ***reglas sintácticas***.
- ***Es el programa fuente***, mientras que al archivo obtenido por ensamblado se le conoce por ***programa objeto***.
- Ciertas palabras que tienen un significado especial se denominan ***palabras reservadas***.
- Los separadores son o espacios en blanco o signos de puntuación como el punto y coma (;)
- La sintaxis de una línea completa del programa fuente tiene la siguiente sintaxis:
  - [***<etiqueta>***] [***<mnemotécnico de instrucción>*** [***<operandos>***];[***<comentarios>***]

- Una **directiva de ensamblador**, no produce código máquina ejecutable.
  - Es utilizada por el programador para dar instrucciones de ensamblado
    - **ORG \$4000 ; dirección de memoria \$4000**
- Los ***símbolos*** son, al igual que las etiquetas, nombres que substituyen a constantes, variables y direcciones de memoria.

## 14.1.1. DIRECTIVAS DE ENSAMBLADOR O PSEUDOINSTRUCCIONES MÁS UTILIZADAS

- *La directiva ORG*

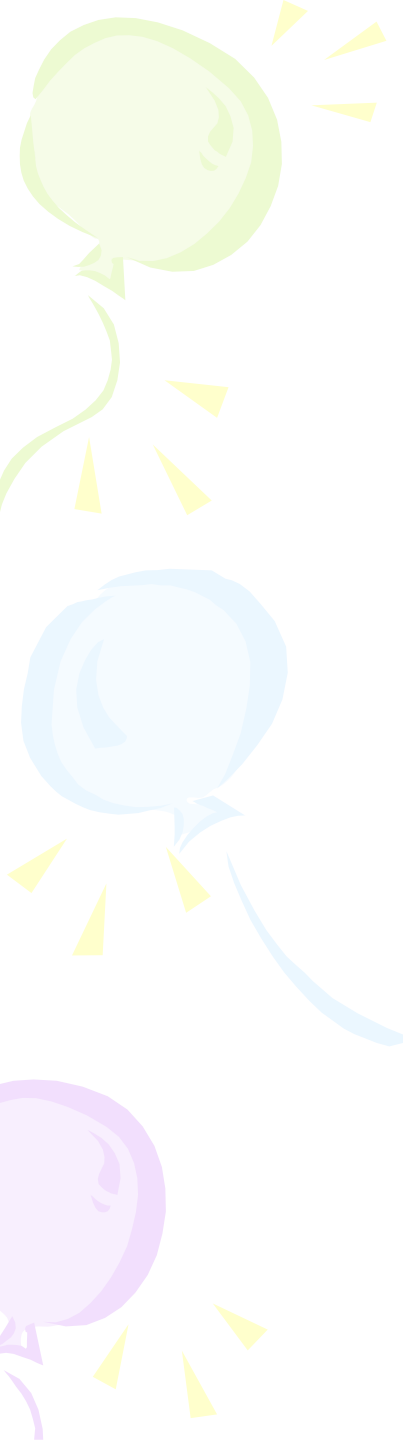
- Indica el origen absoluto (ORiGen) o dirección absoluta de las instrucciones de programa que le sigan.

- ORG <dirección o expresión> [;<comentarios>]

- *La directiva END*

- Sirve para indicar al programa ensamblador que el programa fuente ha finalizado.

- END [;<comentarios>]



; PROGRAMA PRINCIPAL:  
; dirección de inicio = 1024 = \$400  
; Se supone que el programa principal  
; no ocupa más de 2Kbytes.

```
    ORG    $400  
    <instrucciones del programa principal>  
    <...>
```

; ZONA DE SUBROUTINAS:  
; dirección de inicio = 3072 = \$C00  
; Se supone que todas las subrutinas  
; no ocupan más de 1Kbyte.

```
    ORG    $C00  
    <instrucciones de las subrutinas utilizadas por el  
programa>  
    <...>
```

; ZONA DE DATOS:  
; dirección de inicio = 4096 = \$1000  
; Se supone que datos y pila ocupan el resto de la memoria  
disponible.

```
    ORG    $1000  
    <zona de datos>  
    <...>  
END      ; Fin del programa.
```



- *La directiva EQU*

- *<etiqueta> EQU <valor o expresión> [;<comentarios>]*
- Se utiliza para definir un símbolo que se va a utilizar posteriormente
- Esta directiva no utiliza memoria ya que no da lugar a ninguna instrucción en código máquina
- El programa ensamblador crea una ***tabla de símbolos*** donde anota todos los símbolos definidos o encontrados en forma de etiquetas y los asocia a un valor.



- *La directiva DS*

- *<etiqueta> DS.t <número de variables> [;<comentarios>]*
- Se utiliza para reservar posiciones de memoria con vista a utilizarlas como variables
- No define contenido,
  - hay que inicializar las variable
- El programa ensamblador traducirá ese símbolo por esa dirección cada vez que lo encuentre en el programa fuente.
- Esta directiva sí da lugar a un incremento del tamaño de la memoria utilizada por el programa.
- Esta directiva permite utilizarla para definir variables de varios tamaños.
  - DS.B reserva tantos bytes como variables se indiquen a continuación.
  - DS.W reserva palabras y
  - DS.L reserva palabras largas.



- *La directiva DC*

- *<etiqueta> DC.t <valor o valores> [;<comentarios>]*

- Se utiliza para definir datos constantes

- Indica al programa ensamblador que debe fijar una o varias posiciones de memoria como datos y almacena en ellas los valores indicados.

- Esta directiva tiene el mismo efecto que la directiva DS.t con la diferencia de que aquí no sólo se reserva espacio en memoria, también **se le asigna un valor.**



### 14.1.1.6. Ejemplo

El siguiente programa se corresponde con el código máquina, mostrando las direcciones de memoria en las que se encuentra almacenado, que se muestra posteriormente. Ese programa no hace nada, pero sirve de ejemplo de utilización de las directivas vistas anteriormente.

```
Longitud    EQU    4
valor1      EQU    25
valor2      EQU    $A9
```

**; PROGRAMA PRINCIPAL:**

**; dirección de inicio = 1024 = \$400**

**; Se supone que el programa principal no ocupa más de 2Kbytes.**

```
ORG    $400
```

```
NOP
```

**; ZONA DE SUBROUTINAS:**

**; dirección de inicio = 3072 = \$C00**

**; Se supone que todas las subrutinas no ocupan más de 1Kbyte.**

```
ORG    $C00
```

```
NOP
```

**; ZONA DE DATOS:**

**; dirección de inicio = 4096 = \$1000**

**; Se supone que datos y pila ocupan el resto de la memoria disponible.**

```
vector      ORG    $1000
            DS.W   Longitud      ; se reservan cuatro
                                   palabras en memoria
constante1  DC.B   Valor1,Valor2 ; se almacenan dos cons-
tantes                                     en memoria
constante2  DC.L   Valor1,Valor2 ; idem
            END     ; Fin del programa.
```

El programa anterior, al ensamblarlo da lugar al siguiente programa en código máquina.



Dirección:

Contenido:

\$400

\$4E
\$71

\$401

...

...

...

\$C00

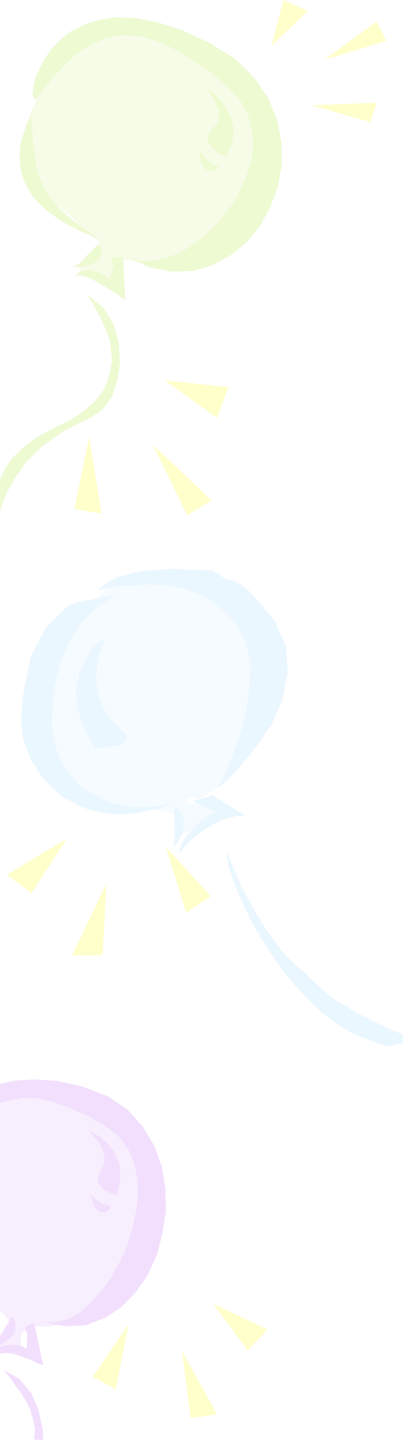
\$4E
\$71

\$C01

...

} Código de instrucción NOP

} Código de instrucción NOP



\$1000	??
\$1001	??
\$1002	??
\$1003	??
\$1004	??
\$1005	??
\$1006	??
\$1007	??
\$1008	\$19
\$1009	\$A9
\$100A	\$00
\$100B	\$00
\$100C	\$00
\$100D	\$19
\$100E	\$00
\$100F	\$00
\$1010	\$00
\$1011	\$A9
\$1012	??
...	...

Dirección del símbolo **vector**  
componente 1 de **vector**

} **vector[2]**

} **vector[3]**

} **vector[4]**

} **constante1**

} **constante2**

```

00000000 =00000004      1 Longitud      EQU      4           ;longitud tiene valor de 4
00000000 =00000019      2 valor1        EQU      25          ;valor1= 25 osea $19
00000000 =000000A9      3 valor2        EQU      $A9         ;valor2 = $A9
00000000
00000000      4
00000000      5 ;aquí empieza el programa principal
00000000      6
00000400      7           ORG      $400      ;el programa empieza a cargarse en la dir $400
00000400      8
00000400 4E71      9           NOP           ;instrucción que no hace nada
00000402      10
00000402      11           ;zona de subrutinas
00000402      12
00000C00      13           ORG      $C00
00000C00 4E71      14           NOP
00000C02      15
00000C02      16 ;zona de datos
00000C02      17
00001000      18           ORG      $1000
00001000      19
00001000      20 vector        DS.W      Longitud      ;a "vector" se la asigna la dirección en la
00001008      21           ;que está colocado y ($1000) además a partir de
00001008      22           ;dicha dirección reserva "Longitud"(4) Words (16 bits) de la
00001008      23           ;$1000 a la $1007
00001008      24
00001008 19 A9      25 cont1        DC.B      valor1,valor2 ;se guarda en las posiciones siguientes $1008
0000100A      26           ;valor1 ($19) y en la $1009 valor2($A9)
0000100A      27
0000100A 00000019 000000A9 28 cont2        DC.L      valor1,valor2 ;igual que la anterior pero en lugar de guardar bytes
00001012      29           ;se guardan palabras largas "L" esto es en la $100A $00
00001012      30           ;$100B $00; $100C $00; $100D $19; $100E $00; $100F $00
00001012      31           ;$1010 $00; $1011 $A9
00001012      32
00001012      33
00001012      34           END

```

## 14.2. EJEMPLOS DE REALIZACIÓN DE ESTRUCTURAS DE DATOS

- **DEFINICIÓN DE CONSTANTES**
  - coef DC.t 34 ; se reservan posiciones de memoria inicializándolas;
- **DEFINICIÓN DE VARIABLES**
  - coches DS.L 1 ; se reserva una palabra larga de memoria
- **DEFINICIÓN DE VECTORES DE DATOS.**
  - MOVE.W vector+4, D0 ; cuatro bytes por delante de vector.
  - vector DS.B 4
    - reservaría espacio para 4 componentes de 8 bits;
  - vector DS.W 4
    - reservaría espacio para 4 componentes de 16 bits,
  - vector DS.L 4
    - reservaría espacio para 4 componentes de 32 bits.
- **DEFINICION DE CADENAS DE CARACTERES**
- **DEFINICION DE PILAS**

1. Una vez ensamblado el siguiente segmento de código en ensamblador del M68000, donde @, % y \$ representan octal, binario y hexadecimal respectivamente, el contenido de la dirección de memoria apuntada por DATO3 es:

		ORG	@100	
	DATO1	EQU	*+%1010	\$40 + \$4A
\$40	ZONA1	DS.B	\$BB	
		EVEN		
40+BB = FB+1	DATO2	DC.L	50	
\$FC →	DATO3	DC.W	DATO3-ZONA1+DATO1	
\$100 →				

Solución:

- a) \$0100
- b) \$010A
- c) \$012E
- d) \$01CA

DATO3 - ZONA1 + DATO1

$$100 - 40 + 4A = 10A$$

@100 = \$40

$$00100000$$


---

4 0

$$\approx 1010 = 10 \text{ $A}$$

## ENSAMBLADOR PARA EL MC68000

Sintaxis      [<etiqueta>] [<mnemonico\_instruc> [<operandos>]] ; [<comentarios>]

Directivas:

ORG	<direcc o expresión>		origen absoluto de las instr. o datos	
END			fin	
EQU	<etiqueta>	EQU	<valor o expresión>	asigna a etiqueta el valor indicado
DS	<etiqueta>	DS.x	<número de variables>	reserva posición de memoria
DC	<etiqueta>	DC.x	<valor o valores>	reserva espacio y mete los valores indicados

Todos los programas deben ser escritos con un editor que no introduzca caracteres de control.  
Todos los programas han de contener la directiva "ORG" al principio del programa y la directiva "END" al final de éste.



# ESTRUCTURA DE PROGRAMAS

- SECUENCIAS DE INSTRUCCIONES
- BIFURCACIONES
- ITERACIONES
- BUCLES FOR
- BUCLE WHILE
- BUCLE REPEAT-UNTIL
- BUCLE LOOP
- SUBROUTINA



	MOVEA.L	NUMBCD, A0	; puntero al vector de resultado
	MOVEI.L	#10000, D0	; divisor
	MOVEQ	#4, D1	; contador
	CLR.L	D2	; D2 (= 0
	MOVE.W	binario, D2	; se copia el dato binario
bucle:	DIVU	D0, D2	; D0 es el divisor ; D2 es el dividendo ; el destino, D2, almacena tanto ; el cociente como el resto ; D2[31:16] es el resto y ; D2[15:0] es el cociente
	MOVE.W	D2, (A0, D1.B)	; guardar resultado <i>! guarda el cociente</i>
	LSR.L	#16, D2	<i>→ desplaza 16 a la derecha (se queda con) el resto</i> <i>en A0 + 4, D0 + 3, D1 + 2.</i>
	DIVU	#10, D0	<i>→</i>
	DBF	D1, bucle	<i>→ dos mensajes en 1 D1</i>
	RTS		; FIN DE LA SUBROUTINA

## 14.4.2. Conversión de un número en código BCD a binario natural

Se parte del ejemplo anterior con un número BCD en memoria, de 5 cifras almacenado en NUMBCD[0:4].

```
MOVEA.L    NUMBCD, A0    ;puntero al vector de resultado
MOVEI.L    #10000, D0     ; multiplicador
MOVEQ      #4, D1        ;contador
CLR.L      D3            ; el resultado
bucle:     MOVE.W      (A0, D1.B), D2 ; D2 =( NUMBCD[4-i]
MULU      D0, D2        ; D2(=(D2*D0
ADD.L      D2, D3       ; calcular resultado
DIVU      #10, D0
DBF       D1, bucle
MOVE.W    D3, binario  ; se guarda el resultado binario
RTS      ; FIN DE LA SUBRUTINA
```

Num. línea	Dirección (HEX)	Código (HEX)	Sentencia fuente		
1	000000		ESCRIPAN	EQU	\$F0432
2	000000		NULO	EQU	0
3	000000		ORG	\$9000	
4	009000	21484F4C4120	CADENA	DC.B	'!HOLA A TODOS!', NULO
5	00900F			EVEN	
6	009010	207C00009000	COMIENZO	MOVEA.L	#CADENA, A0
7	009016	6102		BSR	CADESC
8	009018	60F6		BRA	COMIENZO
9	00901A	0C100000	CADESC	CMPI.B	#NULO, (A0)
10	00901E	670A		BEQ	RETORNO
11	009020	1018		MOVE.B	(A0)+, D0
12	009022	47B9000F0432		JSR	ESCRIPAN
13	009028	60F0		BRA	CADESC
14	00902A	4E75	RETORNO	RTS	
15	00902C		COPIA	DS.B	RETORNO-CADENA+2
16	009058			END	

Febrero 2001, 2ª S, Pregunta 16

Después de ejecutarse el siguiente segmento de código del M68000 el contenido del registro D0.W será:

<b>BUC</b>	<b>MOVE.W</b>	<b>#\$6728, D0</b>	a) \$C128 b) \$7000 c) \$0D28 d) \$5E50
	<b>MOVE.W</b>	<b>#100, D1</b>	
	<b>SUB.W</b>	<b>#\$02A6, D0</b>	
	<b>SUBQ.W</b>	<b>#1, D1</b>	
	<b>BNE</b>	<b>BUC</b>	
	<b>END</b>		

**Solución:**

**MOVE.W**    **#\$6728, D0**

Pone el valor hexadecimal del operando inmediato 6728 en la palabra menos significativa de D0, luego D0 = \$6728

**MOVE.W**    **#100, D1**

Pone el valor decimal del operando inmediato 100 en la palabra menos significativa de D1, luego D1 = \$0064

***EVALUACIÓN DEL BUCLE***

Por una parte en la primera instrucción del bucle lo que se plantea es ir restando al registro D0 el valor hexadecimal \$02A6, y por otra ir restando 1 (decimal) al registro D1 comprobando posteriormente si el valor resultante de ésta última resta es distinto de cero, en cuyo caso se vuelven a repetir las dos operaciones anteriores.

Dicho de otro modo equivale a restar \$02A6 al registro D0, D1 veces:

$D0 - D0 \times D1 = \$6728 - \$6728 \times \$0064 = \text{FFFF } 5E50$ , por lo que el registro D0.W = **5E50**

**Febrero 1999, 2ª S, Pregunta 15**

Sea el siguiente programa escrito en ensamblador de M68000. Supóngase que se ejecutan desde la línea 1 a la línea 9 inclusive del mismo. Después de su ejecución, cual de las siguientes posiciones de memoria NO contiene el byte que se indica (expresado todo en hexadecimal)

<i>PROGRAMA</i>	<i>CLR.L</i>	<i>D0</i>	;Línea 1
	<i>MOVE.B</i>	<i>DAT, D0</i>	;Línea 2
	<i>CLR.L</i>	<i>D1</i>	;Línea 3
	<i>DIVU.W</i>	<i>#32, D0</i>	;Línea 4
	<i>MOVE.B</i>	<i>D0, RE1</i>	;Línea 5
	<i>MOVE.W</i>	<i>#0, D0</i>	;Línea 6
	<i>SWAP</i>	<i>D0</i>	;Línea 7
	<i>BCHG</i>	<i>D0, D1</i>	;Línea 8
	<i>MOVE.L</i>	<i>D1, RE2</i>	;Línea 9
<i>DATOS</i>	<i>ORG</i>	<i>\$1200</i>	;Origen de datos
<i>DAT</i>	<i>DC.B</i>	<i>95</i>	;Dato de entrada
<i>RE1</i>	<i>DS.B</i>	<i>1</i>	;Resultado 1
<i>RE2</i>	<i>DSL</i>	<i>1</i>	;Resultado2

- a) (1202) = 80
- b) (1203) = 08
- c) (1201) = 02
- d) (1204) = 00

**Solución:**

Las instrucciones que aparecen después de la línea 9 nos dan la información sobre los contenidos de memoria.

El contador de ensamblado se pone en la posición \$1200, y es en esa posición que se coloca la constante DAT, luego se deja un espacio de un byte correspondiente a la posición \$1201 para la variable RE1, y un espacio de una palabra larga, es decir cuatro bytes para la variable RE2, luego:

(\$1200) = 95 = \$5F  
 (\$1201) = RE1  
 (\$1202) }  
 (\$1203) } RE2  
 (\$1204) }  
 (\$1205) }

Línea 1: D0 = \$ 0000 0000

Línea 2: MOVE.B DAT, D0 ⇒ D0 = \$ 0000 005F

Línea 3: CLR.L D1 ⇒ D1 = \$ 0000 0000

Línea 4: DIVU.W #32, D0 ⇒ D0/32 = cociente (16 bits menos sig.) + resto (16 bits mas sig)  
 D0 = \$ 001F 0002

Línea 5: MOVE.B D0, RE1 ⇒ almacena en la posición (\$1201) = \$02

Línea 6: MOVE.W #0, D0 ⇒ D0 = \$ 001F 0000

Línea 7: SWAP D0 ⇒ D0 = \$ 0000 001F

Línea 8: BCHG D0, D1 ⇒ comprueba y cambia bit 31 ⇒ D1 = \$8000 0000

Línea 9: MOVE.L D1, RE2 ⇒ coloca en las posiciones reservadas de memoria, luego  
 (\$1202) = 80; (\$1203) = 00; (\$1204) = 00; (\$1205) = 00

Se tiene el siguiente fragmento de código en ensamblador del MC68000, donde % y \$ representan octal, binario y hexadecimal respectivamente.

```
          ORG    @100
DAT01    EQU    *+%1010
ZONA1    DS.B   $BB
          EVEN
DAT02    DC.L   50
DAT03    DC.W   DAT03-ZONA1+DAT01
```

Indique cuál será el contenido de la dirección de memoria apuntada por Dato3, después del ensamblado.

Solución:

Las explicaciones están en los campos de comentarios de este fichero generado por el ensamblador cruzado:

MC68000 Cross Assembler

Copyright (C) Stephen Croll, 1991. Author: Stephen Croll

Version 2.00 beta 1.02

```
00000040          1          ORG    @100          ; El origen está en @100=$40=64
0000004A          2 DAT01    EQU    *+%1010        ; DAT01=(PC)+%1010=64+10=74
00000040          3 ZONA1    DS.B   $BB          ; En ZONA1=$40 se reservan $BB=187
                                ; bytes, sin inicializar
000000FB          4          CNOP    0,2          ; Como $FB es impar, avanza un lugar
000000FC 00000032  5 DAT02    DC.L   50          ; En DAT02=$FC se almacena el número
                                ; 50, el cual ocupa 4 bytes

00000100 010A          6 DAT03    DC.W   DAT03-ZONA1+DAT01 ; En DAT03=$100 se almacena
                                ; $100-$40+74 = 256-64+74=266=$10A
                                7

00000102 307C 0100          8          movea.w #DAT03,A0
00000106 3010          9          move.w  (A0),D0
00000108 1E3C 00E4        10         move.b  #228,D7
0000010C 4E4E          11         trap   #14
0000010E          12         end
```

No errors detected.