LABORATORIO DE
PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR x86-16bits

Programación con MACROS

Objetivo

El objetivo de esta práctica es familiarizarse con la programación usando macros y bibliotecas de macros.

Concepto de macro

En general, una macro (abreviatura de macroinstrucción) es una instrucción compleja, formada por otras instrucciones más sencillas. Una macro se ejecuta de manera secuencial, sin alterar el flujo de control.

En programación ensamblador, una macro es una secuencia de instrucciones a la que se identifica mediante un nombre. Cada vez que aparece el nombre de la macro en el código, se sustituye por la secuencia de instrucciones (se dice que se expande).

Las macros son útiles ya que reducen la cantidad de codificación repetitiva, minimizan las oportunidades de ocasionar un error y simplifican la vista de los programas haciéndolos más legibles.

Declaración de una macro

La declaración de una macro consta de tres partes: la cabecera que contiene el nombre de la macro seguido de la directiva MACRO y, opcionalmente, variables ficticias o parámetros; el cuerpo que contiene el código que será insertado expandiendo el nombre de la macro; y el delimitador de fin con la directiva ENDM.

```
nombre_macro MACRO [parámetro[,parámetro…]]

             //cuerpo de la macro//

             ENDM
```

A continuación se muestra un ejemplo sencillo de programa que usa macros:

```
             DOSSEG
             .MODEL SMALL

@iniciarDS   MACRO
             mov ax,@DATA
             mov ds,ax
             ENDM

@fincodigo   MACRO
             mov ax,4C00h
             int 21h
             ENDM

@mostrarcadena MACRO cadena
             mov ah,09h
             lea dx,cadena
             int 21h
             ENDM

             .STACK 100h
             .DATA
mensaje      DB "Hola a todos",13,10,'$'

             .CODE
inicio:      @iniciarDS
             @mostrarcadena mensaje
             @fincodigo
             END
```

Las macros se deben declaran antes que cualquier segmento. Es habitual colocar algún prefijo al nombre de la macro de manera que no se confunda con etiquetas de datos o de código. Esta convención mejora la legibilidad del código.

En el caso del ejemplo anterior, se habla de macros internas ya que están declaradas en el propio código del programa. Las macros externas se escriben en un fichero de texto y se incluyen al comienzo usando la directiva INCLUDE.

Las macros pueden llevar o no parámetros (variables ficticias). Cada vez aparece un parámetro en el cuerpo de la macro, se sustituye por el valor que se le haya pasado en código. Si se pasan más parámetros de los declarados, no se usan y si se pasan menos, las instrucciones que los procesan no se expanden.

Hay que tener en cuenta que la expansión de las macros hace crecer el tamaño del código estático. Esta circunstancia puede afectar a la longitud de los desplazamientos en los saltos condicionales provocando que superen el rango máximo permitido.

Las macros no hacen salvaguarda previa de los registros que utilizan. Es responsabilidad del programador salvar en la pila los registros sobreescribibles antes de invocar la macro. Por ello, es importante que la documentación de las macros de biblioteca indique qué registros se utilizan en cada una.

Las etiquetas definidas dentro de una macro se expanden también, duplicando los nombres de las mismas (p. ej. los destinos de los saltos). Hay que tener en cuenta esto para evitar repeticiones de etiquetas no deseadas. Cuando queremos que las etiquetas no repitan nombre en sucesivas expansiones, las podemos declarar como locales usando la directiva LOCAL.

```
LOCAL etiqueta[,etiqueta[,…]]
```

Ejemplo:

```
@retardo    MACRO contador
            local seguir

            mov cx,contador
seguir:     loop seguir
            ENDM
```

Bibliotecas de macros

Como se ha dicha más arriba, las macros se pueden agrupar en bibliotecas y se pueden incluir en el código fuente de un programa. La biblioteca se crea reuniendo todas las macros que se desee en un archivo de texto. Es bueno incluir comentarios antes de cada macro que documenten qué hace, qué tipo de parámetros se pasan si es el caso y qué registros utiliza.

Para incluir la biblioteca en el código fuente de un programa basta con usar la directiva **INCLUDE nombre_biblioteca** antes de la declaración de cualquier segmento. El código fuente del programa propuesto como ejemplo quedaría así:

```
            DOSSEG
            .MODEL SMALL

            INCLUDE macros.inc

            .STACK 100h
            .DATA
mensaje     DB "Hola a todos",13,10,'$'

            .CODE
inicio:     @iniciarDS
            @mostrarcadena mensaje
            @fincodigo
            END
```

Y el fichero con la biblioteca **MACROS.INC** sería:

```
;-----------------------------------------------
; @iniciarDS: carga @DATA en DS
; sin parámetros
; no devuelve nada
; registros: AX, DS
;-----------------------------------------------
```

```
@iniciarDS   MACRO
             mov ax,@DATA
             mov ds,ax
             ENDM

;-----------------------------------------------
; @fincodigo: invoca servicio 4Ch-INT21h
; sin parámetros
; devuelve: código de retorno en AL
; registros: AX
;-----------------------------------------------
@fincodigo   MACRO
             mov ax,4C00h
             int 21h
             ENDM

;-----------------------------------------------
; @mostrarcadena: invoca servicio 09h-INT21h
; parámetros:
; cadena -> etiqueta de cadena finalizada en '$'
; devuelve: 24h en AL
; registros: AH, DX
;-----------------------------------------------
@mostrarcadena MACRO cadena
             mov ah,09h
             lea dx,cadena
             int 21h
             ENDM
```

Compilación condicional en macros

A veces una misma macro se puede utilizar con parámetros que pueden ser datos de distinto tipo. Por ejemplo, en el caso de la última macro de la biblioteca anterior (**@mostrarcadena**), el parámetro **cadena** se podría pasar como puntero **near** o **far** en lugar de cómo etiqueta (nombre simbólico). En ese caso, se podría reescribir la macro de esta manera:

```
;-------------------------------------------------
; @mostrarcadena: invoca servicio 09h-INT21h
; parámetros:
; pcadena -> puntero de cadena finalizada en '$'
; devuelve: 24h en AL
; registros: AH, DX
;-------------------------------------------------
@mostrarcadena  MACRO pcadena

                IF TYPE (pcadena) EQ 2      ;; si pcadena word → puntero near
                    mov dx, pcadena
                ELSEIF TYPE (pcadena) EQ 4  ;; si pcadena doble → puntero far
                    lds dx, pcadena
                ELSE
                    .ERR
                    %OUT parametro ilegal
                ENDIF

                mov ah,09h
                int 21h

                ENDM
```

La macro utiliza las directivas de ensamblado condicional (**IFxx**, **ELSEIF**, **ELSE**, **ENDIF**) para adaptarse al tamaño del puntero pasado como parámetro. Para determinar el tamaño del puntero utiliza la directiva **TYPE** (devuelve 1: **byte**, 2: **word**, 4: **doble**, 8: **qword**, 10: **tenbyte**, –1: **near**, –2: **far**). En el ensamblado condicional se ha incluido un mensaje de error que emitirá el programa ensamblador llegado el caso. También se puede observar que los comentarios dentro de la macro se escriben con doble carácter ';'.

Prácticas

A) Ensambla el código propuesto anteriormente.

Escribe el código propuesto anteriormente utilizando las macros de biblioteca o externas. Comprueba el correcto funcionamiento y observa en **`codeview`** cómo se ha expandido el código (vista código ensamblador o mixta).

B) Crea una macro que multiplique por 10 usando desplazamientos.

Sabemos que el producto es una operación lenta. Si hemos de multiplicar por un factor constante conocido, podemos convertir el producto en un conjunto de desplazamientos (que son productos por potencias de 2) y sumas. Por ejemplo, multiplicar por 10 será:

`valor` x 10 = **`valor`** x 8 + **`valor`** x 2

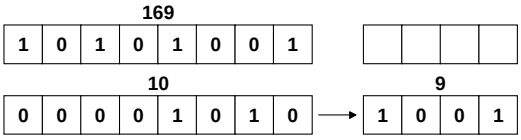
`valor` x 10 = **`valor << 3`** + **`valor << 1`**

Escribe el código de esta macro para un parámetro entero sin signo de tamaño *byte* y un resultado de tamaño *word*. Introduce las sentencias condicionales necesarias para detectar un parámetro ilegal. Comprueba su correcto funcionamiento dentro de un programa usando **`codeview`**.

C) Crea una macro que divida entre 256 usando desplazamientos.

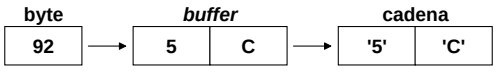
Escribe una macro análoga a la del ejercicio anterior. Ahora el parámetro será un entero sin signo de tamaño *word* representando al dividendo y el resultado se devolverá sobre 2 *bytes*, uno para el cociente y otro para el resto.

Respecto al resto, téngase en cuenta que los bits que salen fuera de la representación en los desplazamientos a derecha representan el resto. La figura siguiente ilustra lo explicado cuando dividimos el entero 169 entre 16.



D) Utiliza macros para obtener los 2 caracteres ASCII de la codificación hexadecimal de un entero sin signo de tamaño *byte*.

El valor de un entero sin signo de tamaño *byte* se puede codificar en hexadecimal en 2 dígitos representables en ASCII. El algoritmo tiene dos fases. En la primera obtenemos los dígitos del valor numérico en base hexadecimal y en la segunda obtenemos el ASCII correspondiente a cada dígito. La primera fase tomará el *byte* de entrada y salvará los 2 dígitos, cada uno de tamaño *byte*, sobre un *buffer* de memoria. La segunda fase tomará cada dígito del *buffer* y lo convertirá a ASCII salvándolo en una cadena.



Cada dígito hexadecimal puede tener un valor entre 0 y 15 (*nibble* ó 4 bits). Si el valor está comprendido entre 0 y 9, el dígito hexadecimal se convierte a ASCII sumando 30h al valor. Si el dígito está comprendido entre 10 y 15, entonces su conversión consiste en sumar a su valor el código ASCII de la 'A' menos 10. Es decir,

el valor 10 se representa como 10 + 'A' – 10 = 'A' + 0 = 'A'
el valor 11 se representa como 11 + 'A' – 10 = 'A' + 1 = 'B'
el valor 12 se representa como 12 + 'A' – 10 = 'A' + 2 = 'C'
el valor 13 se representa como 13 + 'A' – 10 = 'A' + 3 = 'D'
el valor 14 se representa como 14 + 'A' – 10 = 'A' + 4 = 'E'
el valor 15 se representa como 15 + 'A' – 10 = 'A' + 5 = 'F'

Escribiremos una macro para obtener los dígitos y otra para obtener su codificación ASCII. Cada macro tendrá como parámetros la entrada la salida.