Macros

Sistemas Digitales con Microprocesadores.

M.C. Juan Carlos Olguín R.

Macros

Una **macro** es un conjunto de instrucciones asociadas a un identificador: el nombre de la macro. Este conjunto de instrucciones es invocado como una sola instrucción o **macroinstrucción.**

Normalmente las instrucciones de una macro se repiten varias veces en un programa o aparecen frecuentemente en los programas. Para emplear una macro en un programa debemos de hacer dos cosas: Definir la macro e invocar la macro.

La **definición de una macro** establece el nombre al que se asocia la macro, el número y nombre de sus parámetros formales y qué instrucciones contiene la macroinstrucción.

La sintaxis de la definición de una macro es la siguiente:

Definición de una macro

```
macro nomMacro [parForm[, parForm]...]
        proposición
       [proposición]
       ...
endm  [nomMacro]
```

donde las directivas **macro** y **endm** marcan el inicio y el final de la definición de la macro. No generan código.

- *nomMacro es el nombre de la macro y se emplea al invocar la macro.*
- *parForm es cada uno de los parámetros formales de la macro. Los parámetros permiten que una* misma macro opere sobre datos distintos.
- *proposición son cada una de las instrucciones que forman la macroinstrucción.*

Aunque la definición de una macro puede ir en cualquier parte de un programa, el lugar más recomendable para su localización es al principio de un archivo, antes de los segmentos de datos y de código.

Al encontrar una **Invocación de una Macro** el ensamblador, por ejemplo, substituye la línea con la invocación por las proposiciones que contiene la definición de la macro.

Este proceso de substitución se conoce **expansión de la macro.** La sintaxis de la invocación de la macro es similar a cualquier instrucción:

```
nomMacro [parReal[, parReal]...]
```

donde cada *parReal* es conocido como un ***parámetro real*** de la macro.
Al expandirse la macro cada una de las ocurrencias de un parámetro formal en la definición de la macro se substituye por su correspondiente parámetro real.

Los **parámetros reales** pueden ser: símbolos, mnemónicos, directivas, palabras reservadas, expresiones y números.

Ejemplo sobre macros

El siguiente programa ejecuta todas las rotaciones de un bit con una variable de tipo palabra doble.

Las operaciones de rotación se implementan mediante macros. El parámetro formal **dest** de cada macro, representa la localidad de memoria que contiene el dato a rotar.

Al invocar a cada una de las macros el parámetro real de cada macro es el nombre de la variable cuyo dato se va a rotar.

```
;*************************************************************
; ROTACIO2.ASM

; Este programa ejecuta todas las rotaciones de un bit con
; una variable de tipo palabra doble. Las operaciones de
; rotación se implementan mediante macros que rotan el
; contenido de la localidad de memoria dada por el
; parámetro de la macro un bit.
;*************************************************************

;****** CÓDIGO DE INICIO **********************************

        ideal
        dosseg
        model   small
        stack   256
```

```
;****** MACROS ***************************************

;*****************************************************
; RCLDW1
;
; Esta macro rota el contenido de dest un bit a la
; izquierda a través de la bandera de acarreo.
;*****************************************************
macro   rcldw1  dest
        rcl     [word dest], 1          ;; rcl dest, 1
        rcl     [word dest+2], 1
endm    rcldw1

;*****************************************************
; RCRDW1
;
; Esta macro rota el contenido de dest un bit a la
; derecha a través de la bandera de acarreo.
;*****************************************************
macro   rcrdw1  dest
        rcr     [word dest+2], 1        ;; rcr dest, 1
        rcr     [word dest], 1
endm    rcrdw1
```

```
;*************************************************************
; ROLDW1
;
; Esta macro rota el contenido de dest un bit a la izquierda
;*************************************************************
macro   roldw1  dest
        push    bx                          ;; Preserva BX
        mov     bx, [word dest+2]           ;; rol dest, 1
        shl     bx, 1
        rcl     [word dest], 1
        rcl     [word dest+2], 1
        pop     bx                          ;; Restaura BX
endm    roldw1
;*************************************************************
; RORDW1
;
; Esta macro rota el contenido de dest un bit a la derecha.
;*************************************************************
macro   rordw1  dest
        push    bx                          ;; Preserva BX
        mov     bx, [word dest]             ;; ror dest, 1
        shr     bx, 1
        rcr     [word dest+2], 1
        rcr     [word dest], 1
        pop     bx                          ;; Restaura BX
endm    rordw1
```

```
;*** VARIABLES DEL PROGRAMA ********************************

        dataseg
codsal  db      0
r_rcl   dd      ?
r_rcr   dd      ?
r_rol   dd      ?
r_ror   dd      ?

;****** CÓDIGO DEL PROGRAMA ********************************

        codeseg
inicio:
        mov     ax, @data               ; Inicializa el
        mov     ds, ax                  ; segmento de datos

; Rotación a la izquierda a través de la bandera de acarreo.

        rcldw1  r_rcl                   ; rcl r_rcl, 1

; Rotación a la derecha a través de la bandera de acarreo
        rcrdw1  r_rcr                   ; rcr r_rcr, 1

; Rotación a la izquierda
        roldw1  r_rol                   ; rol r_rol, 1

; Rotación a la derecha
        rordw1  r_ror                   ; ror r_ror, 1

salir:
        mov     ah, 04Ch
        mov     al, [codsal]
        int     21h
;****** CÓDIGO DE TERMINACIÓN ********************************
        end     inicio
```

En el ejemplo anterior podemos notar que los comentarios dentro de la definición de una macro empiezan con doble punto y coma. Esto se hace para que en el archivo con el listado del programa generado por el ensamblador no aparezcan los comentarios cada vez que se expande la macro.

También podemos notar que las macros **roldw1** y **rordw1** que implementan las rotaciones a la izquierda y a la derecha, respectivamente, utilizan el registro **BX** y por lo tanto se preserva su valor al inicio de la macro y se restaura al salir.

Esto se hace con el fin de que el uso de la macro sea “transparente” para el usuario.

Macros con etiquetas locales

Al igual que con los procedimientos, si una macro emplea **identificadores** para etiquetar las direcciones de instrucciones que componen la macro, estas **etiquetas** deben ser etiquetas locales, **con el fin de evitar la duplicidad de etiquetas** si la macro se invoca más de una vez en un programa.
Para crear etiquetas locales en el ensamblador se emplea la directiva **local**.
La directiva **local en** una macro **hace** que las etiquetas declaradas en la directiva tengan **un alcance restringido a la macro**.
La sintaxis de esta directiva es:

```
local etiqueta [, etiqueta]...
```

En una macro, la directiva local debe ir **inmediatamente** de la directiva macro.
Pueden tenerse varias directivas local.

Ejemplo: Este programa utiliza la macro **mayor3 para encontrar el mayor de los tres datos. La macro tiene dos etiquetas** locales: **sigcmp y fincmp.**

```
;*************************************************************
; MAYOR3_2.ASM
;
; Este programa encuentra el mayor de tres datos signados
; almacenados en variables de una palabra. El programa
; utiliza una macro para encontrar al mayor de los tres
; datos
;*************************************************************

;****** CÓDIGO DE INICIO ***************************************

        ideal
        dosseg
        model   small
        stack   256
```

```
;*************************************************************
; MAYOR3
;
; Esta macro encuentra el mayor de tres datos signados
; dados por los parámetros dato1, dato2 y dato3. El mayor
; de los tres datos queda en el registro AX.
;*************************************************************
macro   mayor3  dato1, dato2, dato3
        local   sigcmp, fincmp

        mov     ax, [dato1]     ; AX = dato1
        cmp     ax, [dato2]     ; if(AX > dato2)
        jg      sigcmp          ;    goto sigcmp
        mov     ax, [dato2]     ; else AX = dato2
sigcmp:
        cmp     ax, [dato3]     ; if(AX > dato3)
        jg      fincmp          ;    goto fincmp
        mov     ax, [dato3]     ; else AX = dato3
fincmp:
endm    mayor3
```

```
;****** VARIABLES DEL PROGRAMA ****************************

        dataseg
codsal  db      0
dato1   dw      ?
dato2   dw      ?
dato3   dw      ?
mayor   dw      ?

;****** CÓDIGO DEL PROGRAMA *******************************

        codeseg
inicio:
        mov      ax, @data      ; Inicializa el
        mov      ds, ax         ; segmento de datos

        mayor3  dato1, dato2, dato3

        mov     [mayor], ax     ; mayor = AX

salir:
        mov     ah, 04Ch
        mov     al, [codsal]
        int     21h

;****** CÓDIGO DE TERMINACIÓN *****************************

        end     inicio
```

Directiva de repetición rept

Los macroensambladores del 8086 poseen varias directivas que permiten repetir instrucciones. Una de esas es la directiva **rept**. La directiva **rept** permite repetir un grupo de proposiciones varias veces.

Su sintaxis es:

```
rept    expresión
        proposición
        [proposición]
        ...
endm
```

- donde las directivas **rept** y **endm** delimitan la directiva de repetición **rept.**

- *expresión* es una **expresión constante entera** y es el **número de veces** que se repiten las instrucciones dadas por *proposición*.

- La directiva de repetición **rept** puede utilizarse por sí sola o dentro de macros para crear instrucciones que repitan instrucciones.

Ejemplo sobre macros que usan la directiva de repetición rept

El archivo siguiente muestra una modificación de las macros que rotan variables de tipo palabra doble para que en lugar de **rotar** las variables un solo bit, las roten varios bits.

Las macros contienen la directiva de repeticón **rept**.

Las macros se encuentran en un archivo con el nombre ROTA_N.INC el cual puede incluirse en otros programas usando la directiva **include**

La extensión **.INC** no es obligatoria para los archivos de inclusión pero si es una práctica común.

```
;*************************************************************
; ROTA_N.INC
;
; Este archivo contiene una serie de macros que pueden
; incluirse en otros programas con la directiva include e
; implementan las operaciones de rotación del contenido de
; la localidad de memoria dada por el parámetro dest un
; número de bits dado por el parámetro cnta.
;*************************************************************


;*************************************************************
; RCLDWN
;
; Esta macro rota el contenido de dest cnta bits a la
; izquierda a través de la bandera de acarreo.
;*************************************************************
macro   rcldwn  dest, cnta
        rept    cnta
         rcl    [word dest], 1          ;; rcl dest, 1
         rcl    [word dest+2], 1
        endm
endm    rcldwn
```

```
;*************************************************************
; RCRDWN
;
; Esta macro rota el contenido de dest cnta bits a la
; derecha a través de la bandera de acarreo.
;*************************************************************
macro   rcrdwn  dest, cnta
        rept    cnta
         rcr    [word dest+2], 1        ;; rcr dest, 1
         rcr    [word dest], 1
        endm
endm    rcrdwn

;*************************************************************
; ROLDWN
;
; Esta macro rota el contenido de dest cnta bits a la
; izquierda.
;*************************************************************
macro   roldwn  dest, cnta
        push    bx                      ;; Preserva BX
        rept    cnta
         mov    bx, [word dest+2]       ;; rol dest, 1
         shl    bx, 1
         rcl    [word dest], 1
         rcl    [word dest+2], 1
        endm
        pop     bx                      ;; Restaura BX
endm    roldwn
```

```
;*************************************************************
; RORDWN
;
; Esta macro rota el contenido de dest cnta bits a la
; derecha.
;*************************************************************
macro   rordwn  dest, cnta
        push    bx                          ;; Preserva BX
        rept    cnta
         mov    bx, [word dest]             ;; ror dest, 1
         shr    bx, 1
         rcr    [word dest+2], 1
         rcr    [word dest], 1
        endm
        pop     bx                          ;; Restaura BX
endm    rordwn
```

Inclusión de archivos con la directiva include

La directiva **include** permite la inclusión de un archivo texto dentro de otro. La sintaxis de esta directiva es:

include "*nomArch*"

donde *nomArh* es el nombre del archivo cuyo contenido será substituido en lugar de la directiva **include.**

A continuación se muestra un ejemplo que ilustra el uso de la directiva **include** para incluir dentro del código el contenido del archivo ROTA_N.INC del ejemplo anterior.

Este programa ejecuta todas las rotaciones de un número de bits con una variable de tipo palabra doble.

```
;*************************************************************
; ROTACIO3.ASM
;
; Este programa ejecuta todas las rotaciones de un número
; de bits con una variable de tipo palabra doble. Las
; operaciones de rotación están implementadas mediante
; macros y se encuentran en el archivo ROTA N.INC. Este

; programa ilustra el uso de la directiva include.
;*************************************************************

;****** CÓDIGO DE INICIO ***********************************

        ideal
        dosseg
        model   small
        stack   256

;****** INCLUSIÓN DE ARCHIVOS ******************************

include "rota_n.inc"
```

```
;****** VARIABLES DEL PROGRAMA ****************************

        dataseg
codsal  db      0
r_rcl   dd      ?
r_rcr   dd      ?
r_rol   dd      ?
r_ror   dd      ?

;****** CÓDIGO DEL PROGRAMA *******************************

   codeseg
inicio:
        mov     ax, @data                   ; Inicializa el
        mov     ds, ax                      ; segmento de datos

; Rotación a la izquierda a través de la bandera de acarreo
        rcldwn  r_rcl, 2                    ; rcl r_rcl, 2

; Rotación a la derecha a través de la bandera de acarreo
        rcrdwn  r_rcr, 2                    ; rcr r_rcr, 2

; Rotación a la izquierda
        roldwn  r_rol, 2                    ; rol r_rol, 2

; Rotación a la derecha
        rordwn  r_ror, 2                    ; ror r_ror, 2

salir:
        mov     ah, 04Ch
        mov     al, [codsal]
        int     21h
```

Trabajo para Entregar

Proyecto del Segundo Examen.
SDM 2011.

Escribe un programa que implemente las operaciones de suma, resta, multiplicación y división de números fraccionarios. Cada número fraccionario se representará mediante un par de números enteros (de tipo palabra).

El programa constará de dos módulos.

El primer módulo llamado **ARITFRAC.ASM** contiene los siguientes procedimientos:

mcd.- que calcula el máximo común divisor de dos números enteros de tipo palabra. El procedimiento recibe los dos números en los registros AX y DX y regresa el máximo común divisor en el registro AX.

mcm.- que calcula el mínimo común múltiplo de dos números enteros de tipo palabra. El procedimiento recibe los dos números en los registros AX y DX y regresa el mínimo común múltiplo en los registros DX:AX.

redfrac.- que reduce una fracción a su mínima expresión. El procedimiento recibe los dos números que representan el numerador y el denominador en los registros AX y DX, respectivamente y regresa la fracción reducida en los mismos registros.

addfrac.- que suma dos números fraccionarios y regresa la suma como una fracción reducida a su mínima expresión. El procedimiento recibe los dos números fraccionarios en los registros AX y DX para el numerador y denominador del primer número y BX y CX para el numerador, denominador del segundo número. El procedimiento regresa el resultado en los registros AX y DX para el numerador y denominador, respectivamente.

subfrac.- que resta dos números fraccionarios y regresa la resta como una fracción reducida a su mínima expresión. El procedimiento recibe los dos números fraccionarios en los registros AX y DX para el numerador y denominador del primer número y BX y CX para el numerador, denominador del segundo número.

El procedimiento regresa el resultado en los registros AX y DX para el numerador y denominador, respectivamente.

mulfrac.- que multiplica dos números fraccionarios y regresa el producto como una fracción reducida a su mínima expresión. El procedimiento recibe los dos números fraccionarios en los registros AX y DX para el numerador y denominador del primer número y BX y CX para el numerador, denominador del segundo número. El procedimiento regresa el resultado en los registros AX y DX para el numerador y denominador, respectivamente.

divfrac.- que divide dos números fraccionarios y regresa el cociente como una fracción reducida a su mínima expresión. El procedimiento recibe los dos números fraccionarios en los registros AX y DX para el numerador y denominador del primer número y BX y CX para el numerador, denominador del segundo número. El procedimiento regresa el resultado en los registros AX y DX para el numerador y denominador, respectivamente.

El segundo módulo llamado **DEMOFRAC.ASM** contiene un programa para probar los procedimientos del módulo anterior. En este módulo se declaran 8 variables: **numA**, **denA**, **numB**, **denB**, **numC**, **denC**, **numR**, **denR**, para almacenar cuatro números fraccionarios:

A = numA/denA
B = numB/denB
C = numC/denC
R = numR/denR

El programa deberá efectuar la siguiente operación:

R = ((A + B) * (C - A))/(B + C).

El programa ejecutable deberá llamarse **DEMOFRAC.EXE**