PROCEDIMIENTOS

Sistemas Digitales con Microprocesadores.

M.C. Juan Carlos Olguín R.

Un **procedimiento** es una colección de instrucciones relacionadas que realiza una tarea específica. También un procedimiento puede contener un conjunto de instrucciones que deseamos que se ejecuten en varias partes del programa.

Los procedimientos del lenguaje ensamblador tienen su contraparte en los lenguajes de alto nivel, por ejemplo, en el lenguaje C estos procedimientos se llaman funciones. Aunque en los lenguajes de alto nivel, el mecanismo empleado por los procedimientos para implementar su llamada, ejecución y regreso es transparente para el usuario, en ensamblador se requiere entender ese mecanismo.

Un **componente indispensable** empleado por ese mecanismo es la **pila** del programa.

La pila de un programa también se emplea para implementar el paso de parámetros a los procedimientos así como para crear las variables locales al procedimiento.

La Pila De Un Programa

Todo programa escrito en el lenguaje ensamblador del 8086 requiere de una pila. La pila se emplea para almacenar temporalmente direcciones y datos. En los programas hechos para el microprocesador 8086, una pila es un arreglo de palabras.

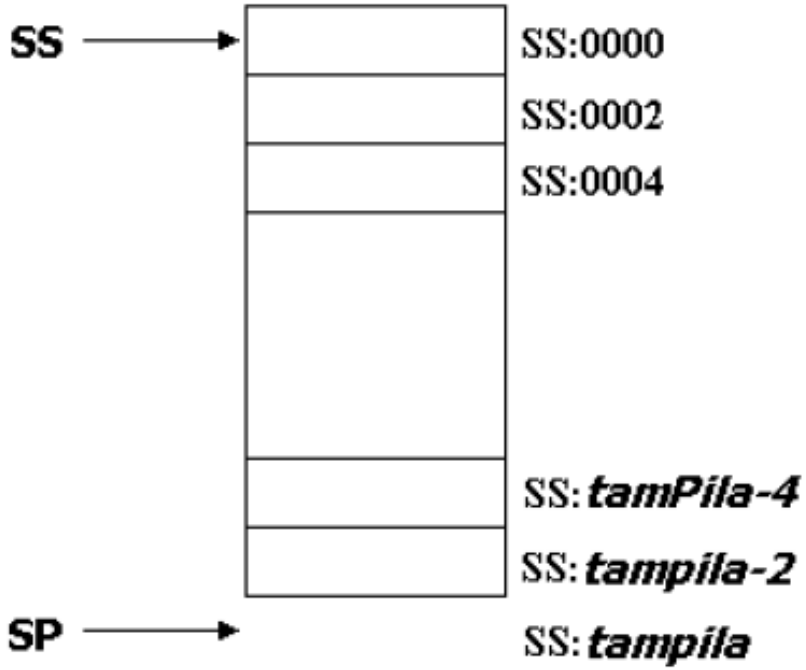
El tamaño de la pila lo establece el programador mediante la directiva **stack** en el código de inicio de un programa**.** Tal como se estudio en el curso anteriormente, con la directiva **stack cuya sintaxis es:**

stack *tamPila*

stack *tamPila*

crea una pila para el programa de tamaño *tampila*. *El cargador del programa inicializa en forma* automática el registro de segmento de pila **SS** a la dirección del segmento en que se creó la pila y el registro apuntador de pila **SP** al valor de ***tamPila*.**

La siguiente figura: *muestra el estado inicial de la pila* de un programa al ser cargado a memoria para su ejecución (la pila se encuentra vacía).



Note que el registro apuntador de pila **SP apunta a la palabra que está pasando el final de la pila.**
La pila difiere de los otros segmentos es la manera en que almacena los datos: **Es decir,** empieza en la localidad más alta (la que tiene la dirección dada por **SS:*tamPila-2***) *y almacena los datos hacia las* localidades más bajas de la pila (hacia **SS:0000**).

Instrucciones para el manejo de la Pila:

El ensamblador del 8086 posee un conjunto de instrucciones especiales para insertar y extraer datos en la pila:

- ✓ **push** *fuente* **;**Transfiere el valor dado por *fuente a la pila del programa.*

Sintaxis:

push *regW|memW*

Donde *regW puede ser cualquiera de los registros de propósito general o de segmento.*

La instrucción **push** transfiere una palabra a la pila del programa. La instrucción **push** primero **decrementa** el valor del registro apuntador de pila **SP** en **dos.** A continuación copia el valor especificado por su operando a la localidad cuya dirección está dada por **SS:SP**.
Note que esto hace que la pila crezca hacia las direcciones más pequeñas de memoria.
La instrucción **push** no afecta a las banderas.

- ✓ **pop** *destino*

Extrae un valor del tope de la pila del programa y lo almacena en *destino*.

Sintaxis:

pop *regW|memW*

Donde *regW puede ser cualquiera de los registros de propósito general o de segmento* ***excepto*** *el* registro de segmento de código **CS,** que no está permitido.

La instrucción **pop** extrae una palabra de la localidad cuya dirección está dada por **SS:SP (el tope de la pila del progra)** y lo almacena en la localidad especificada por el operando. A continuación incrementa el valor del registro apuntador de pila **SP** en **dos**.

La instrucción **pop** no afecta a las banderas.

En la figura 2 se muestran los estados de la pila después de insertar los valores de los registros **AX** y **BX** mediante las instrucción **push** y en la figura 3 se muestran los estados de la **pila** después de extraer esos valores de la pila mediante la instrucción **pop** al ejecutar el siguiente

fragmento de código:

```
mov    ax, 0123h
mov    bx, 4567h
push   ax
push   bx
pop    bx
pop    ax
```

estados de la pila después de insertar los valores mediante Push

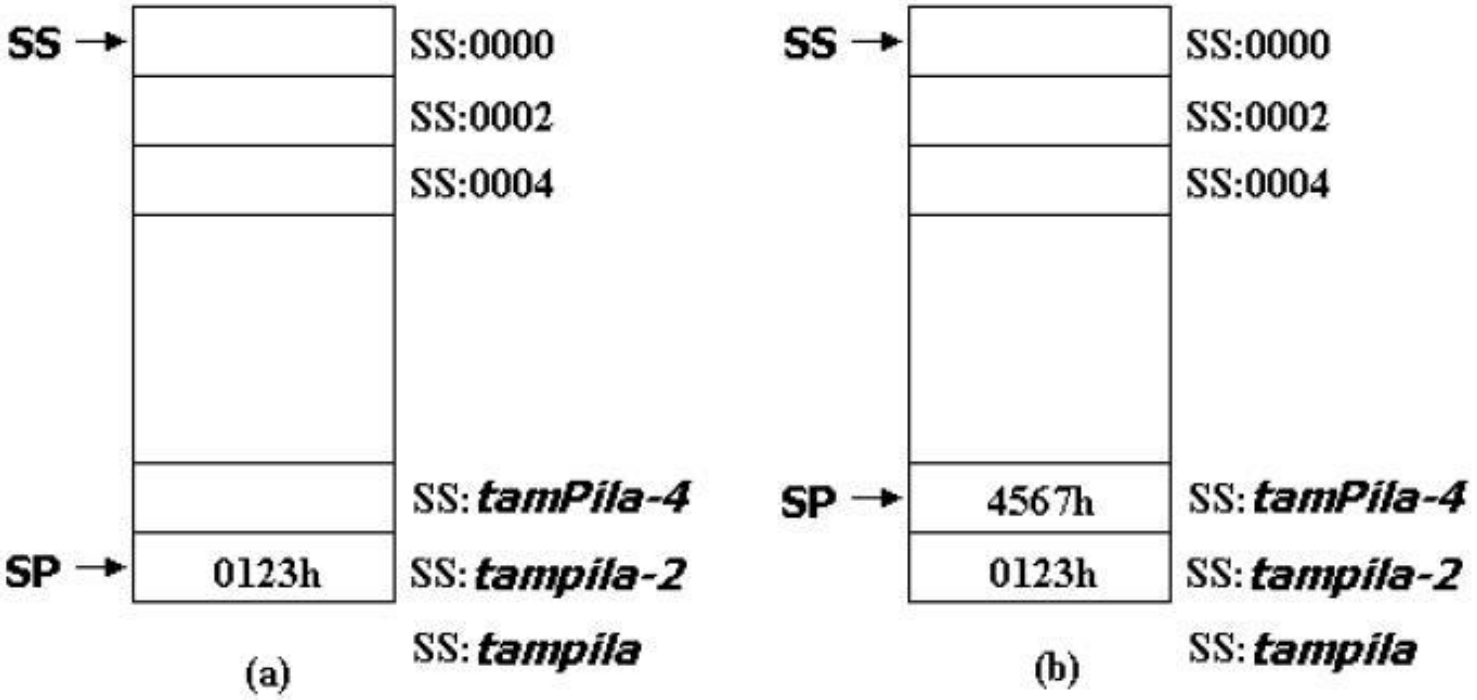


Figura 2

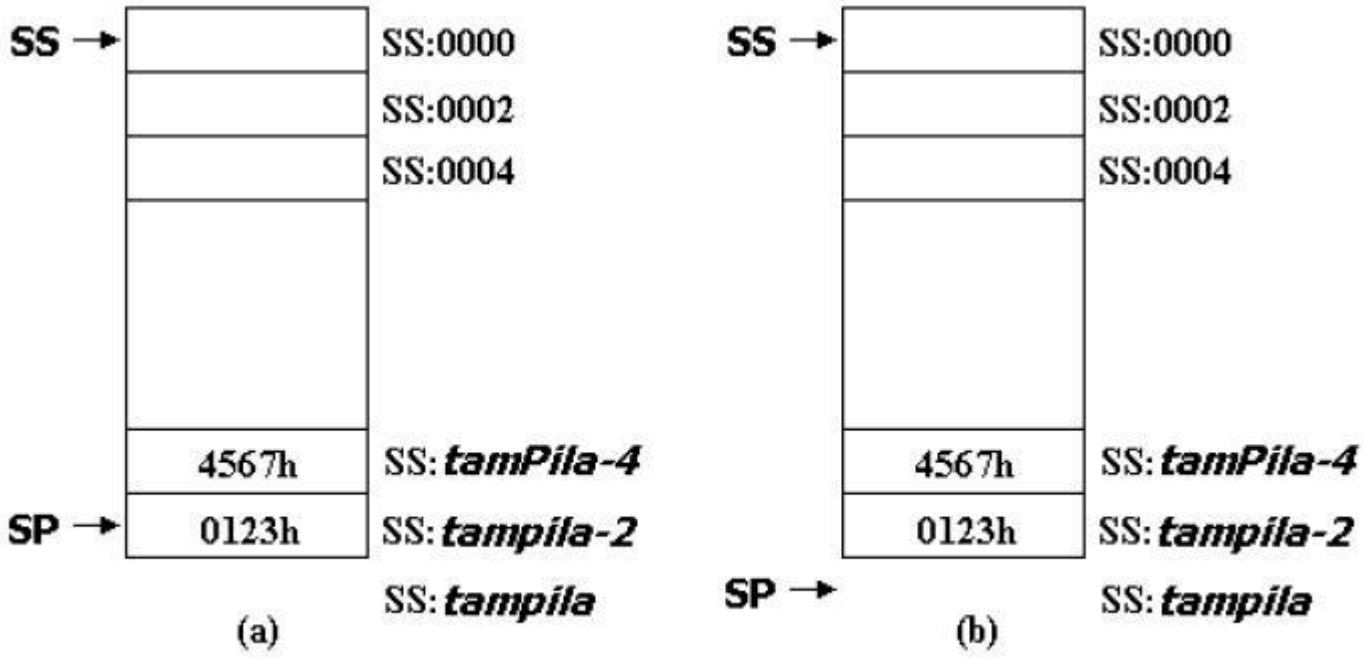


Figura 3

Note en la figura 3a que al extraer el valor 4567h de la pila, éste no se elimina físicamente de la pila. Sin embargo no hay una garantía de que ese valor permanezca en la pila.

Si en ese momento insertamos un nuevo valor en la pila, éste se escribirá en la localidad **SS:*tamPila-4*** *sobrescribiendo* el valor de 4567h.

✓ **pushf**
Transfiere el registro de banderas a la pila del programa.

Sintaxis:

pushf

La instrucción **pushf** transfiere el contenido del registro de banderas a la pila del programa de la misma manera en la que opera la instrucción **push.**
La instrucción **pushf no afecta a las banderas.**

✓ **popf** **;**Extrae un valor del tope de la pila del programa y lo almacena en registro de banderas. Sintaxis:

popf

La instrucción **popf** extrae una palabra del tope de la pila y la inserta en el registro de banderas de la misma manera en que trabaja la instrucción **pop.** Normalmente se hace esto para extraer el estado de las banderas almacenado previamente por una instrucción **pushf,** o para fijar las banderas a nuevos valores.

La instrucción **popf afecta todas las banderas.**

Procedimientos

Como ya se mencionó anteriormente, un procedimiento es una colección de instrucciones que realizan una tarea específica: Sumar dos valores, desplegar un carácter en la pantalla, leer un carácter del teclado, inicializar un puerto, etc.

Dependiendo de su extensión y complejidad, un programa puede contener uno, algunos o inclusive cientos de procedimientos.

Para emplear un procedimiento en un programa se requiere definir el procedimiento y llamarlo.

- Al definir a un procedimiento escribimos las instrucciones que contiene.
- Al llamar al procedimiento transferimos el control del flujo del programa al procedimiento para que sus instrucciones se ejecuten.

Definición de un procedimiento

La sintaxis de la definición de un procedimiento es la siguiente:

proc *nomProc*
proposición
[*proposición*]
...
endp [*nomProc*]

Las directivas **proc** y **endp** marcan el inicio y el final del procedimiento. **No** generan código.

nomProc es el nombre del procedimiento y etiqueta la primera instrucción del procedimiento.

Al menos una de las proposiciones de un procedimiento es la instrucción **ret**, la cual se describe más adelante.

Llamada a un procedimiento

La llamada a un procedimiento normalmente tiene la siguiente forma:

call *nomProc*

La instrucción **call** se describe más adelante.

nomProc es el nombre que se le dio al procedimiento al definirlo.

Ejemplo sobre Procedimientos

```
;*************************************************************
; SUM2DW_P.ASM
;
; Este programa suma dos variables de tipo palabra doble
; y guarda el resultado en una variable de tipo palabra
; doble. Este programa utiliza un procedimiento para
; efectuar la suma.
;
; El pseudocódigo de este programa es:
;
;     DX:AX = dato1
;     CX:BX = dato2
;
;     suma2dw()
;
;     resul = DX:AX
;*************************************************************

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

        ideal
        dosseg
        model   small
        stack   256
```

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

        dataseg
codsal  db      0
dato1   dd      ?
dato2   dd      ?
resul   dd      ?

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

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

        mov     ax, [word dato1]        ; DX:AX = dato1
        mov     dx, [word dato1+2]
        mov     bx, [word dato2]        ; CX:BX = dato2
        mov     cx, [word dato2+2]

; Llama al procedimiento para efectuar la suma
        call    suma2dw                  ; suma2dw()

        mov     [word resul], ax         ; resul = DX:AX
        mov     [word resul+2], dx

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

```
;****** PROCEDIMIENTOS ************************************

;*************************************************************
; SUMA2DW
;
; Este procedimiento suma dos variables de tipo
; palabra doble.
;
; Parámetros:
;
;       DX:AX = Primer sumando
;       CX:BX = Segundo sumando
;
; Regresa:
;
;       DX:AX = Suma
;
; El pseudocódigo de este procedimiento es:
;
;   DX:AX += CX:BX
;*************************************************************
proc    suma2dw
        add     ax, bx                  ; DX:AX += CX:BX
        adc     dx, cx

        ret                             ; Regresa del
                                        ; procedimiento
endp    suma2dw

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

        end     inicio
```

Mecánica de la llamada a un procedimiento

Para implementar la llamada a un procedimiento y el regreso de éste, se tiene el siguiente mecanismo:

Primero.- Cuando el programa lee de memoria y decodifica la instrucción que corresponde a la llamada a un procedimiento, el registro apuntador de instrucciones **IP** tiene el desplazamiento de la siguiente instrucción a ejecutar.

Por ejemplo: en el listado parcial del archivo SUM2DW_P.LST

que se muestra a continuación se puede ver que el desplazamiento de la siguiente instrucción a la llamada al procedimiento **suma2dw** es de 0017h.

```
31
32 0005  A1 0001r              mov     ax, [word dato1]      ; DX:AX = dato1
33 0008  8B 16 0003r           mov     dx, [word dato1+2]
34 000C  8B 1E 0005r           mov     bx, [word dato2]      ; CX:BX = dato2
35 0010  8B 0E 0007r           mov     cx, [word dato2+2]
36
37 0014  E8 000E               call    suma2dw               ; Llama al procedimiento
38                                                           ; para efectuar la suma
39
40 0017  A3 0009r              mov     [word resul], ax      ; resul = DX:AX
41 001A  89 16 000Br           mov     [word resul+2], dx


64 0025                 proc    suma2dw
65 0025  03 C3                  add     ax, bx               ; DX:AX += CX:BX
66 0027  13 D1                  adc     dx, cx
67
68 0029  C3                     ret                          ; Regresa del
69                                                           ; procedimiento
70 002A                 endp    suma2dw
```

Segundo.- La instrucción **call** inserta el valor de este desplazamiento en la pila del programa y carga en el registro apuntador de instrucciones el valor de la dirección etiquetada con el nombre del procedimiento.
En este ejemplo cargaría a **IP** con el valor de **0025h.**
Esto hace que el control de flujo del programa se transfiera a la primera instrucción del procedimiento.

Tercero.- Al ejecutar una instrucción **ret**, ésta extrae del tope de la pila el valor previamente almacenado por la instrucción **call** y lo carga en el registro apuntador de instrucciones.
Como este valor es el desplazamiento de la instrucción que va después de la llamada al procedimiento, ésta será la siguiente instrucción a ejecutarse.
Esto es lo que se conoce como el regreso del procedimiento.
En este ejemplo **IP** se carga con el valor de **0017h**.

- ✓ **call** *destino* **;**Llama a un procedimiento.

Sintaxis:

call *etiqueta*
call *regW|memW*
call *memDW*

Consideraciones sobre Call

Si el procedimiento a llamar se encuentra en el mismo segmento en el que se encuentra su llamada, la instrucción **call** inserta en la pila el desplazamiento de la siguiente instrucción después de la llamada al procedimiento y luego carga en el registro apuntador de instrucciones el desplazamiento dado por *destino*.

Si el procedimiento está en un segmento diferente al en que está su llamada, la instrucción **call** inserta en la pila el segmento:desplazamiento de la siguiente instrucción después de la llamada al procedimiento y luego carga en el registro apuntador de instrucciones y en el registro de segmento de código el segmento:desplazamiento dado por *destino*.

En la mayoría de los programas, *destino es una etiqueta que marca el inicio del procedimiento*. *Sin* embargo también puede ser un registro o localidad de memoria que contenga la dirección del procedimiento.

✓ **ret**

Regresa de un procedimiento

Sintaxis:

ret

Si el procedimiento del que se va a regresar se encuentra en el mismo segmento en el que se encuentra su llamada, la instrucción **ret** extrae de la pila el desplazamiento de la siguiente instrucción después de la llamada al procedimiento, el cual fue insertado en la pila por la llamada al procedimiento y lo carga en el registro apuntador de instrucciones **IP**.

Si el procedimiento está en un segmento diferente al en que está su llamada, la instrucción **ret** extrae de la pila el segmento:desplazamiento de la siguiente instrucción después de la llamada al procedimiento y lo carga en los registros de segmento de código **CS** y apuntador de instrucciones **IP**, respectivamente.

Consideraciones sobre el diseño de un procedimiento

Un procedimiento bien diseñado debe llenar los siguientes requisitos:

- ❖Debe hacer una sola tarea.
- ❖Debe ser tan pequeño como sea posible y tan largo como sea necesario. Su listado no debiera de exceder una o dos páginas.
- ❖Debe contener un comentario describiendo su propósito, sus datos de entrada y de salida.
- ❖Debe entenderse por sí solo sin necesidad de saber que hace el programa completo.
- ❖El comportamiento interno del procedimiento debe ser transparente para el usuario. Esto se llama **encapsulamiento de la información**. El usuario sólo debe saber cómo llamar al procedimiento, esto es, qué datos se le deben suministrar y cómo regresa el resultado. A esto se le llama la **interfase del procedimiento.**

Para lograr esa transparencia se requiere que:

✓Un procedimiento, desde el punto de vista del usuario, funcione como lo haría una instrucción del microprocesador. Por lo tanto un procedimiento sólo puede alterar los registros en los que recibe los datos, los registros en los que regresa el resultado o el registro de banderas.

✓Un procedimiento tampoco debe utilizar variables globales ni para recibir datos o regresar un resultado, ni para almacenar temporalmente resultados intermedios.

Uso de los registros en un procedimiento

Un procedimiento puede utilizar los registros de propósito general de tres formas diferentes:

I. Para recibir valores o direcciones del código que lo llama.
II. Para regresar valores o direcciones al código que lo llama.
III. Como variables locales del procedimiento o como registros de trabajo de las instrucciones del procedimiento.

Anteriormente se mencionó que para que un procedimiento sea transparente al usuario, sólo debe modificar los registros que emplea para recibir datos o regresar resultados. ¿Cómo puede entonces emplear los registros como variables locales o registros de trabajo sin modificar su valor?
La respuesta es que, si un procedimiento emplea un registro como variable local o registro de trabajo, debe guardar el valor original de ese registro en la pila del programa al entrar al procedimiento y recuperar su valor antes de salir del procedimiento.

Ejemplo sobre procedimientos que usan registros como variables locales o registros de trabajo

El siguiente programa intercambia los niveles más significativo y menos significativo de una variable de un byte. El programa utiliza un procedimiento para intercambiar los niveles.

Como el procedimiento **swapnibl** empleado para intercambiar los niveles requiere usar el registro **CX** como registro de trabajo de la instrucción rol el valor inicial de ese registro al entrar al procedimiento se almacena en la pila con una instrucción **push.** Antes de regresar del procedimiento, se restaura el valor original del registro **CX** mediante la instrucción **pop.**

```
;*************************************************************
; SWAPNIBL2.ASM
;
; Este programa intercambia los MSN y LSN de una variable
; de un byte. El resultado queda en la misma variable.
; Este programa utiliza un procedimiento para intercambiar
; los nibles.
;
; El pseudocódigo de este programa es:
;
;    AL = dato
;    swapnibl()
;    dato = AL
;*************************************************************

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

        ideal
        dosseg
        model   small
        stack   256

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

        dataseg
codsal  db      0
dato    db      ?
```

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

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

        mov     al, [dato]      ; AL = dato

; Llama al procedimiento para intercambiar los nibles
        call    swapnibl        ; swapnibl()

        mov     [dato], al      ; dato = AL

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

```
;****** PROCEDIMIENTOS ************************************

;*************************************************************
; SWAPNIBL
;
; Este procedimiento intercambia los MSN y LSN de una
; variable de un byte.
;
; Parámetro:
;
;       AL = dato
;
; Regresa:
;
;       AL = dato con los nibles intercambiados.
;
; El pseudocódigo de este procedimiento es:
;
;     CL = 4
;     rol al, 4
;*************************************************************
proc    swapnibl
        push    cx              ; Preserva CX

        mov     cl, 4
        rol     al, cl

        pop     cx              ; Recupera CX
        ret
endp    swapnibl

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

        end     inicio
```

Etiquetas locales a un procedimiento

Los identificadores que hemos empleado para etiquetar las direcciones de las instrucciones son etiquetas globales. Una **etiqueta global** es conocida en todo el programa, incluyendo en los procedimientos. Si un procedimiento definiera una etiqueta global, ésta también sería conocida por todo el programa.

Como las etiquetas globales deben ser únicas, los nombres de las etiquetas de un procedimiento deben ser diferentes entre sí y diferentes a las de otros procedimientos y de las del resto del programa. Lo anterior viola el principio transparencia mencionado en la sección pasada ya que el programador debería conocer los nombres de las etiquetas empleadas en los diferentes procedimientos para no repetirlos.

Etiquetas locales a un procedimiento

Para evitar este problema el Assembler nos permite definir etiquetas locales. El ámbito de una etiqueta local se extiende desde su definición hacia adelante y hacia atrás hasta la siguiente etiqueta no local. Como las etiquetas definidas con la directiva **proc** son globales, los procedimientos están limitados por etiquetas globales, la del inicio del procedimiento y la de del inicio del siguiente procedimiento.

Por lo tanto, las etiquetas locales son visibles sólo dentro del código del procedimiento y podemos reutilizarlas en otro lado sin problema.

La sintaxis de una etiqueta local es similar a la de las etiquetas globales sólo que empieza con dos arrobas, por ejemplo @@while, @@endwhi, etc.

Ejemplo:proc. con etiquetas locales

```
;*************************************************************
; SERIE3.ASM
;
; Este programa suma los números enteros de 1 hasta nfinal
; Versión que utiliza un procedimiento para obtener la
; suma.
;
; El pseudocódigo de este programa es:
;
;    AX = nfinal
;
;    serie()
;
;    suma = AX
;*************************************************************

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

         ideal
         dosseg
         model   small
         stack   256

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

         dataseg
codsal   db      0
nfinal   dw      ?
suma     dw      ?
```

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

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


        mov     ax, [nfinal]    ; AX = nfinal

        call    serie

        mov     [suma], ax      ; suma = AX
salir:
        mov     ah, 04Ch
        mov     al, [codsal]
        int     21h
```

```
;****** PROCEDIMIENTOS ************************************

;*************************************************************
; SERIE
;
; Este procedimiento suma los números de 1 hasta nfinal
;
; Parámetro:
;
;         AX = nfinal
;
; Regresa:
;
;         AX = suma
;
; El pseudocódigo de este procedimiento es:
;
;    CX = nFinal
;    if(CX == 0) goto enddo
;    AX = 0
;
;    do
;    {
;       AX+= CX
;    }
;    while(--CX > 0)
;*************************************************************
```

```
proc    serie
        push    cx              ; Preserva CX
        mov     cx, ax          ; CX = nfinal

        jcxz    @@enddo         ; if(CX == 0) goto enddo
        xor     ax, ax          ; AX = 0

@@do:                           ; do {
        add     ax, cx          ;    AX += CX
        loop    @@do            ; } while(--CX > 0)
@@enddo:

        pop     cx              ; Recupera CX
        ret
endp    serie


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

        end     inicio
```

Modo de direccionamiento base, Variables locales en la pila, Variables locales en la pila usando la directiva local, Programación Modular.

M.C. Juan Carlos Olguín Rojas