J.T.P. Ing. Steiner Guillermo.
Email:gsteiner@scdt.frc.utn.edu.ar

Trabajo Práctico Nro 5 Pase de Parámetros e Instrucciones de Cadena

Pase de parámetros

Introducción

El envío de parámetros a una función de asembler es un problema que nos enfrentamos comúnmente a la hora de desarrollar un software en este lenguaje, para resolverlo podemos hacerlo de varias maneras. La forma mas fácil es pasar los argumentos en los registros, de hecho es lo que utilizan las interrupciones de software para sus parámetros, otra forma es utilizar variables globales, y por ultimo usar el stack. Los argumentos en el registro tienen el inconveniente de estar acotados a los 4 registros generales que posee el 8086, además hay veces que no podemos utilizarlos por estar siendo usados cuando necesitamos llamar a la función en cuestión. Las variables globales tienen el problema de que deben existir durante la vida del programa, además de necesitar prever cuantas variables debemos usar para todas las llamadas de funciones en el programa. En el stack en cambio, podemos utilizar las funciones de pila que provee (POP y PUSH) para cargar y descargar nuestras variables, es por eso que el uso del stack es lo mas difundido y es en lo que trabajaremos en este apunte.
Lenguajes de más alto nivel como por ejemplo el "**C**" utiliza este método para pasar los parámetros, por lo tanto, una vez conocido este procedimiento, podemos mezclar funciones en assembler dentro de nuestra aplicación en "C" sin muchos inconvenientes.

Ejemplo 1

Este ejemplo utiliza el stack para pasar dos variables, las cuales serán sumadas en la función, retornando el valor en **AX,** además se crea una variable local, con lo cual aplicamos el otro uso que le podemos dar al stack, crear variables para ser utilizadas en nuestra función, y liberar la memoria una vez que salimos.

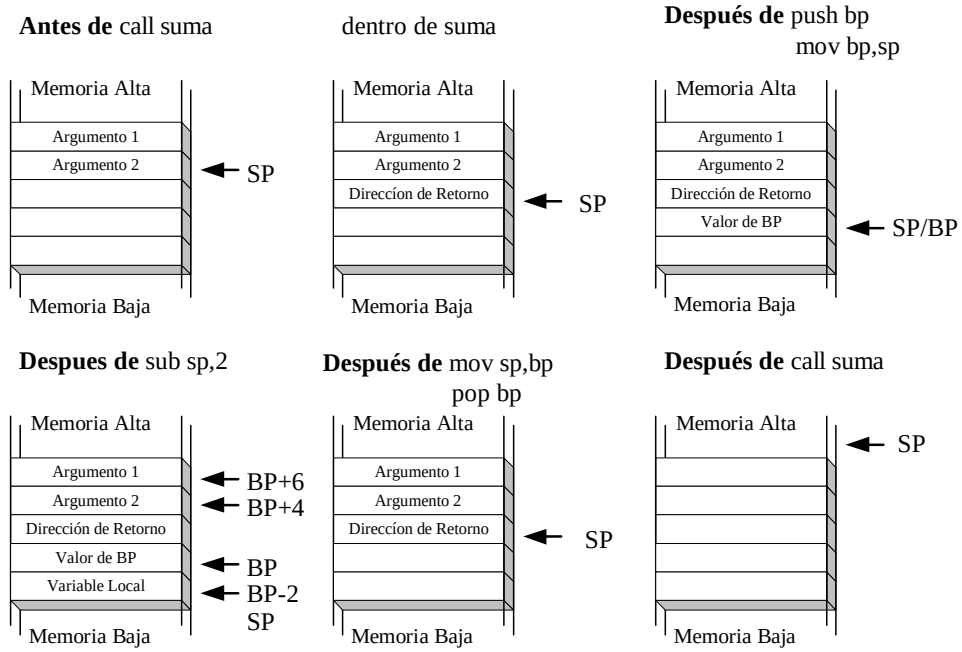
```
name "param"
; Rutina de ejemplo para sumar dos valores y al resultado sumarle un
; valor guardado en una variable local (Los datos son pasados por el stack )
    org 100h
    mov     ax,12234
    mov     operador1,ax
    mov     ax,22345       ; carga en operador1 y operador2 los
    mov     operador2,ax   ; dos números a sumar
    push    operador1      ; los operadores son copiados a la pila
    push    operador2
    call    suma           ; se llama a la función suma
    ret
; --------------------------------
;     Procedimiento de suma
suma PROC NEAR
       push    bp          ; salva el bp (Base Pointer)
       mov     bp,sp       ; copia en bp ==> sp
       sub     sp,2        ; guarda el espacio en el stack para
                           ; la variable local de tipo WORD (2 bytes)

       mov     w.[bp-2],7  ; cargo en la variable local (bp-2) el numero 7
       mov     ax,[bp+6]   ; carga el operador 1
       add     ax,[bp+4]   ; suma el operador 2
       add     ax,[bp-2]   ; suma la variable local

       mov     sp,bp       ; borro la variable local
       pop     bp          ; reestablece el valor de bp
       ret 4               ; retorno y borro el espacio ocupado por los parámetros
suma ENDP
; variables
    operador1 DW (0)
    operador2 DW (0)
       END
```

J.T.P. Ing. Steiner Guillermo.
Email:gsteiner@scdt.frc.utn.edu.ar

Una vez que se utilizó los parámetros enviados, estos deben ser eliminados del stack, para esto, pasamos como argumento a la instrucción **ret** la cantidad de bytes a sumarle al **SP**, realizando la operación de incremento la misma **ret**.

Tabla

La presente tabla muestra los cambios que se producen en el stack mientras corre el programa.



Retorno de Valores

En el ejemplo, se utiliza al acumulador AX para el valor a retornar, esto se debe a que el **C** utiliza este método para devolver valores en sus funciones, por convención entonces utilizaremos este criterio en Assembler.
La siguiente Tabla muestra cual es la convención del **C** con respecto a los valores retornados, el **C** asume que el dato devuelto será algunos de los siguientes, dependiendo de cómo se halla declarado el tipo de retorno de la función.

Tipo de dato	**Registros**
Char	AL
short, puntero near, int	AX
long, puntero far	(Parte alta o Segmento) → DX (Parte baja u Offset) → AX
(Solo p/32 bit) short, puntero near, int	EAX
(Solo p/32 bit) long, puntero far	(Parte alta o Segmento) → EDX (Parte baja u Offset) → EAX

Manejo de Strings

Introducción

Los microprocesadores de la familia 8086, posee instrucciones diseñadas para un rápido procesamiento de los string, cuando se habla de string, se refiere a una secuencia de elementos uno después del otro, estos elementos pueden ser de bytes, words y para el caso de 80386 y superiores, doble word, la diferencia de estas instrucciones en particular con respecto a las demás instrucciones estándar de Assembler, es que, con la suma de un prefijo se puede realizar un determinado procesamiento a un conjunto de posiciones de memoria.

J.T.P. Ing. Steiner Guillermo.
Email:gsteiner@scdt.frc.utn.edu.ar

Instrucciones

MOVS	Copia un string de una locación a otra.
STOS	Graba el contenido de un acumulador a un string.
LODS	Carga el valor de un elemento del string a un acumulador.
CMPS	Compara un string con otro.
SCAS	Busca un valor específico dentro de un string.

Prefijos

REP	Repetir CX veces.
REPE o REPZ	Repetir CX veces, mientras el resultado sea cero.
REPNE o REPNZ	Repetir CX veces, mientras el resultado sea distinto de cero.

Uso

Como se describió antes, estas instrucciones suelen estar acompañadas con un prefijo para su uso, además, dado que utilizan los registros para realizar su función es necesario configurarlos antes.
Es por eso, que se hace necesario seguir una serie de pasos antes de utilizarlas:

1) Configurar el Flag de Dirección, mediante este flag podemos decidir si cada vez que se realiza un "loop" se incrementará o decrementará la dirección a buscar el próximo elemento. La configuración se efectúa por medio de dos instrucciones:
   - **CLD** → pone a cero el flag de dirección (en cada operación de cadena se incrementa la dirección).
   - **STD** → pone en uno el flag de dirección (en cada operación de cadena se decrementa la dirección).
2) Indicar cuántos elementos posee la cadena o mejor dicho cuantos elementos queremos procesar de la cadena, para esto se utiliza el contador **CX**.
3) Se deben ingresar las direcciones de comienzo de la cadena fuente y la de destino, estas se cargan en **DS:SI** (la fuente) y **ES:DI** (el destino), algunas instrucciones usan sólo una de estas direcciones y otras ambas.
4) Por ultimo, estas instrucciones tienen una forma especial para diferenciar si se está trabajando con byte o word, esto es agregando una letra al final de la instrucción, una **b** para operar con byte o una **w** para operar con word.

Las instrucciones con prefijos operan de la siguiente manera (por cada elemento):

1) Chequea el CX, saliendo si llegó a cero.
2) Realiza la operación según sea la instrucción (mover en caso de MOVS, comparar en caso de CMPS, etc).
3) Realiza un incremento o un decremento (dependiendo del flag de dirección) de los índices usados por la instrucción (DI y/o SI). La cantidad del incremento o decremento dependerá si configuramos a la instrucción como byte o word.
4) Decrementa CX sin modificar los flag.
5) Chequear el flag de cero si se están usando prefijos REPE o REPNE, así si la condición del REPn es correcta se realiza el loop al punto 1 sino el loop finaliza.

El prefijo se encarga de los pasos 1,4 y 5 y la operación utilizada realiza los pasos 2 y 3.

Ejemplos de Uso

MOVS

Desde Hasta
[DS:SI] → **[ES:DI]**

El MOVS copia una string fuente a un string destino, esta función puede ser usada en conjunto con el prefijo **REP.**
En el ejemplo copiamos el contenido de un vector de 10 elementos a otro vector de posee también 10 elementos.

```
name "movs"
org 100h
     cld                 ; trabajar en forma ascendente
     mov cx,10           ; cargar la longitud de la cadena en CX
     mov si,offset fuente  ; Cargar dirección fuente a SI
     mov di,offset destino ; Cargar dirección destino a DI (al utilizar
                         ; modelo tiny DS y ES serán iguales)
     rep movsb           ; copiar 10 bytes
     ret
```

```
fuente  db 1 DUP('0123456789')
destino db 10 DUP(?)
       END
```

STOS

Desde Hasta
AL o AX → **[ES:DI]**

Copia el acumulador **AL** o **AX** a una cadena.
Podemos utilizarlo de dos formas:

- **sin prefijo:** cuando deseamos cargar una cadena en la cual cada elemento necesita un procesamiento individual, entonces podemos con un simple **STOS** guardar los elementos de a uno, con la ventaja que nos provee de la función de incremento o decremento de la dirección destino evitando ser implementada por nosotros.
- **con prefijo:** nos sirve para realizar un fill o llenado de una cadena. Ingresamos el dato con que vamos a llenar la cadena en **AL** o **AX** y luego mediante un prefijo **REP** realizamos **CX** loops.

El ejemplo siguiente dada una cadena de 11 elementos, llenar con asteriscos mediante una **STOS** con prefijo **REP**

```
name "stosb"
org 100h
     cld                  ; trabajar en forma ascendente
     mov cx,11            ; cargar long. de la cadena en CX
     mov di,offset destino ; cargar dirección destino en DI
     mov al,'*'           ; byte a llenar el vector destino
     rep stosb            ; lleno con '*' la cadena
     ret

destino db 11 DUP(?)
       END
```

LODS

Desde Hasta
[DS:SI] → **AL o AX**

Carga desde una cadena un elemento a el acumulador **AL** o **AX** según la longitud del elemento.
Esta instrucción no se usan con prefijos puesto que no tendría sentido ya que cada vez que carga un valor al acumulador borra el anterior.
LODS se utiliza cuando queremos procesar cada elemento de la cadena, entonces creamos un bucle mediante una instrucción **LOOP** y dentro de este utilizamos la instrucción **LODS**.
En el ejemplo, utilizamos esta función en conjunto con **STOSB** para transformar una cadena de número a su correspondiente código ASCII.

```
name "lodsstos"
org 100h
      cld                   ; trabajar en forma ascendente
      mov cx,longitud       ; cargar la longitud de la cadena en CX
      mov si,offset fuente  ; cargar fuente para inst LODSB
      mov di,offset destino ; cargar destino para inst STOSB
loop1: lodsb
      add al,'0'
      stosb
      loop loop1
      ret

fuente   db 0,1,2,3,4,5,6,7,8,9
destino  db 10 DUP(?)
longitud equ 10
       END
```

CMPS

Desde Hasta
[DS:SI] con [ES:DI]

Compara dos cadenas, realiza una resta de cada elemento sin modificar ninguno de sus valores pero si actualizando los flags, esta instrucción se puede utilizar en conjunto con el prefijo **REPNZ** o **REPZ**.

- Utilizando REPNZ o REPNE: Compara mientras no sean iguales, saliendo del bucle cuando dos elementos lo sean y dejando el flag de cero en 1, en caso de no encontrar elementos iguales, llegará a CX = 0, y retornará el flag de cero en 0,
- Utilizando REPZ o REPE: Compara mientras sean iguales, saliendo del bucle cuando encuentre dos elementos que no lo sean y dejando el flag de cero en 0, en caso de no encontrar elementos distintos llegara a CX = 0, y retornará el flag de cero en 1.

Luego de realizado el bucle de comparación mediante un salto condicional como el **JZ** o **JNZ** se elige que hacer. En el ejemplo se realiza la comparación de dos cadenas para determinar si son iguales o no, para esto, luego de comparar se verifica el estado de la bandera de cero.

```
name "cmpsb"
org 100h
      mov cx,longitud        ; cargar la longitud de la cadena en CX
      cld                    ; trabajar en forma ascendente
      mov si,offset cadena1  ; Cargar dirección fuente a SI
      mov di,offset cadena2  ; Cargar dirección destino a DI
      repe cmpsb
      je igual
;     en caso que sean distintas
;     .....

igual:
;     en caso que sean iguales
;     .....

final:
      ret

cadena1 db "Cadena a comparar"
cadena2 db "Cadena a comparar"
longitud EQU 17
      END
```

SCAS

Desde Hasta
AL o AX con [ES:DI]

Compara el valor de un elemento de la cadena con el acumulador. La forma en que se usa es similar al CMPS, se utilizara también en conjunto con los prefijos REPNE y REPE.

- Prefijo REPNE o REPNZ: Sale del bucle y retorna el flag cero en 1 cuando encuentra un valor que coincide con el acumulador en caso de no encontrar algún elemento igual que el acumulador llega a CX = 0 retornando el flag cero en 0.
- Prefijo REPE o REPZ: Sale del bucle y retorna el flag cero en 0 cuando encuentra un valor que no coincide con el acumulador en caso de no encontrar algún elemento distinto que el acumulador llega a CX = 0 retornando el flag cero en 1.

El ejemplo busca una letra en el alfabeto, guardando en AX su posición.

```
name "scasb"
org 100h
       cld                    ; trabajar en forma ascendente
       mov cx,longitud        ; cargar la long.de la cadena en CX
       mov di,OFFSET cadena   ; cargar dirección de cadena en DI
       mov al,'F'             ; caracter a buscar
       repne scasb
       jne noencontrado
       mov ax,longitud
       sub ax,cx              ; calcular la posición
noencontrado:
       ret
cadena db "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
longitud EQU 26
       END
```

Trabajo Práctico a Desarrollar

Se debe realizar un programa en assembler, que pida el ingreso de 3 cadenas por medio del teclado.
Además, deberá buscar en la primer cadena las siguientes secuencias `$apellido$` y `$nombre$` remplazando las mismas por la segunda y tercera cadena respectivamente (cada secuencias pueden aparecer varias veces en la primer cadena).
Ejemplo:

```
C:>prac4

Se pide al señor $nombre$ apellido $apellido$ que confirme la aceptación de la beca
Perez
Juan

Salida: se pide al señor Juan apellido Perez que confirme la aceptación de la beca
```

Estructura y forma de programación

El programa deberá estar compuesto por un “main”, donde se encuentre el ingreso de la cadena y el procesamiento de la misma, con la restricción que la única forma de acceder o modificar los datos de las cadenas sea a través de funciones con parámetros enviados por el stack, a excepción del ingreso por teclado y la impresión, los cuales pueden ser resuelto de cualquier forma.
Las funciones de procesamiento de cadena, deberán tener los mismos nombres y realizar las mismas operaciones que las provistas por la librería STRING del C.
La resolución del problema se realizará codificando estas instrucciones y luego utilizándolas en el programa principal.
Funciones de la librería STRING

strcat	strchr	strcmp	strcoll	strcpy	strcspn
strerror	strlen	strncat	strncmp	strncpy	strpbrk
strrchr	strspn	strstr	strtok	strxfrm	