PRÁCTICA 1

Subrutinas y pasaje de parámetros

Objetivos: Comprender la utilidad de las subrutinas y la comunicación con el programa principal a través de una pila. Escribir programas en el lenguaje assembly del simulador VonSim. Ejecutarlos y verificar los resultados, analizando el flujo de información entre los distintos componentes del sistema.

1) Repaso de uso de la pila Si el registro SP vale 8000h al comenzar el programa, indicar el valor del registro SP **luego de ejecutar** cada una de las instrucciones de la tabla, **en el orden en que aparecen**. Indicar, de la misma forma, los valores de los registros AX y BX.

	Instrucción	Valor del registro SP	AX	BX
1	mov ax,5			
2	mov bx,3			
3	push ax			
4	push ax			
5	push bx			
6	pop bx			
7	pop bx			
8	pop ax			

2) Llamadas a subrutinas y la pila Si el registro SP vale 8000h al comenzar el programa, indicar el valor del registro SP **luego** de ejecutar cada instrucción. Considerar que el programa comienza a ejecutarse con el IP en la dirección 2000h, es decir que la primera instrucción que se ejecuta es la de la línea 5 (push ax).
Nota: Las sentencias ORG y END no son instrucciones sino indicaciones al compilador, por lo tanto no se ejecutan.

#	Instrucción	Valor del registro SP
1	**org 3000h**	-----
2	rutina: mov bx,3	
3	ret	
4	**org 2000h**	-----
5	push ax	
6	call rutina	
7	pop bx	
8	hlt	
9	**end**	-----

3) Llamadas a subrutinas y dirección de retorno

Si el registro SP vale 8000h al comenzar el programa, **indicar el valor de SP y el contenido de la pila luego** de ejecutar cada instrucción. Si el contenido es desconocido/basura, indicarlo con el símbolo **?**. Considerar que el programa comienza a ejecutarse con el IP en la dirección 2000h, es decir que la primera instrucción que se ejecuta es la de la línea 5 (call rut). Se provee la ubicación de las instrucciones en memoria, para poder determinar la dirección de retorno de la rutina.

Además, explicar detalladamente:

a) Las acciones que tienen lugar al ejecutarse la instrucción call rut
b) Las acciones que tienen lugar al ejecutarse la instrucción ret

```
        org 3000h
rut:    mov bx,3        ; Dirección 3000h
        ret             ; Dirección 3002h

        org 2000h
        call rut        ; Dirección 2000h
        add cx,5        ; Dirección 2002h
        call rut        ; Dirección 2004h
        hlt             ; Dirección 2006h
        end
```

4) Tipos de Pasajes de Parámetros Indicar con un tilde, para los siguientes ejemplos, si el pasaje del parámetro es por registro o pila, y por valor o referencia;

	Código	A través de		Por	
		Registro	Pila	Valor	Referencia
a)	`mov ax,5` `call subrutina`				
b)	`mov dx, offset A` `call subrutina`				

c)	`mov bx, 5`
	`push bx`
	`call subrutina`
	`pop bx`				

d)	`mov cx, offset A`
	`push cx`
	`call subrutina`
	`pop cx`				

e)	`mov dl, 5` `call subrutina`				
f)	`call subrutina` `mov A, dx`				

5) Pasaje de parámetros a través de registros y la pila

A) Completa las instrucciones del siguiente programa, que envía a una subrutina 3 valores A, B y C a través de registros AL, AH y CL, calcula AL+AH-CL, y devuelve el resultado en DL.

```
org 1000h
A db 8
B db 5
C db 4
D db ?
```

```
org 3000h
CALC: _________
      add DL, AH
      sub DL, CL
      _________
```

```
org 2000h
_________
_________
mov CL, C
_________
mov D, DL
hlt
end
```

B) Ahora lo mismo, pero los valores A, B y C se reciben mediante pasaje de parámetros por valor a través de la pila. El resultado se devuelve de igual forma por el registro **dl** y por valor.

```
org 1000h
A db 8
B db 5
C db 4
D db ?
```

```
org 3000h
CALC: mov bx,sp
      _________
      mov dl, byte ptr [bx]
      sub bx, 2
      _________
      sub bx, 2
      _________
      add DL, AH
      sub DL, CL
      _________
```

```
org 2000h
mov AX, C
push AX
_________
push AX
_________
push AX
call CALC
mov D, DL
_____
_____
_____
hlt
end
```

C) Modificar el programa anterior para enviar los parámetros A, B y C a través de la **pila** pero ahora por **referencia.**

6) Multiplicación de números sin signo.

El simulador no posee una instrucción para multiplicar números. Escribir un programa para multiplicar los números NUM1 y NUM2 de 1 byte, y guardar el resultado en la variable RES, también de 1 byte.

A. Sin usar subrutinas, implementando todo en el programa principal.
B. Llamando a una subrutina **MUL** para efectuar la operación, pasando los parámetros por **valor** desde el programa principal a través de **registros** y devolviendo el resultado a través de un **registro** por **valor**.
C. Llamando a una subrutina **MUL**, pasando los parámetros por **referencia** desde el programa principal a través de **registros**, y devolviendo el resultado a través de un **registro** por **valor**.
D. Llamando a una subrutina **MUL**, pasando los parámetros por **valor** desde el programa principal a través de **la pila**, y devolviendo el resultado a través de un **registro** por **valor**.
E. Llamando a una subrutina **MUL**, pasando los parámetros por **referencia** desde el programa principal a través de **la pila**, y devolviendo el resultado a través de un **registro** por **valor**.

8) Subrutinas para realizar operaciones con cadenas de caracteres Las operaciones con strings son muy comunes. Por eso, vamos a escribir una pequeña colección de subrutinas reutilizables.

a) Escribir una subrutina **LONGITUD** que cuente el número de caracteres de una cadena de caracteres terminada con un carácter que tiene el código ASCII cero (00H) almacenada en la memoria. La cadena se pasa a la subrutina por referencia vía registro, y el resultado se retorna por valor también a través de un registro.
   Ejemplo: la longitud de 'abcd'00h es 4 (el 00h final no se cuenta)

b) Escribir una subrutina **CONTAR_MIN** que cuente el número de letras minúsculas de la 'a' a la 'z' de una cadena de caracteres terminada en cero almacenada en la memoria. La cadena se pasa a la subrutina por referencia vía registro, y el resultado se retorna por valor también a través de un registro.
   Ejemplo: CONTAR_MIN de 'aBcDE1#!' debe retornar 2.

c) * Escriba la subrutina **ES_VOCAL**, que determina si un carácter es vocal o no, ya sea mayúscula o minúscula. La rutina debe recibir el carácter por valor vía registro, y debe retornar, también vía registro, el valor 0FFH si el carácter es una vocal, o 00H en caso contrario.
   Ejemplo: ES_VOCAL de 'a' o 'A' debe retornar 0FFh y ES_VOCAL de 'b' o de '4' debe retornar 00h

d) * Usando la subrutina anterior escribir la subrutina **CONTAR_VOC**, que recibe una cadena terminada en cero por referencia a través de un registro, y devuelve, en un registro, la cantidad de vocales que tiene esa cadena.
   Ejemplo: CONTAR_VOC de 'contar1#!' debe retornar 2

e) Escriba la subrutina **CONTAR_CAR** que cuenta la cantidad de veces que aparece un carácter dado en una cadena terminada en cero. El carácter a buscar se debe pasar por valor mientras que la cadena a analizar por referencia, ambos a través de la pila.
   Ejemplo: CONTAR_CAR de 'abbcde!' y 'b' debe retornar 2, mientras que CONTAR_CAR de 'abbcde!' y 'z' debe retornar 0.

f) Escriba la subrutina **REEMPLAZAR_CAR** que reciba dos caracteres (ORIGINAL y REEMPLAZO) por valor a través de la pila, y una cadena terminada en cero también a través de la pila. La subrutina debe reemplazar el carácter ORIGINAL por el carácter REEMPLAZO.
   Ejemplo: REEMPLAZAR_CAR de un string "Hola buen día", con ORIGINAL = "a" y reemplazo = "o" debe cambiar el string para que sea "Holo buen dío"

g) Utilizando las subrutinas anteriores, escribir un programa que dada una cadena guardada en el

9) Subrutinas para realizar rotaciones Las rotaciones son operaciones que, aunque no parezca, tienen muchas utilidades, como dividir o multiplicar por dos de forma rápida, o manipular los bits de un byte, pero no hay una instrucción que las implemente directamente.

a) **ROTARIZQ:** Escribir una subrutina ROTARIZQ que haga **una** rotación hacia la izquierda de los bits de un byte almacenado en la memoria. Dicho byte debe pasarse por valor desde el programa principal a la subrutina a través de registros y por referencia. No hay valor de retorno, sino que se modifica directamente la memoria.

> **Pista:** Una rotación a izquierda de un byte se obtiene moviendo cada bit a la izquierda, salvo por el último que se mueve a la primera posición. Por ejemplo al rotar a la izquierda el byte **1**0010100 se obtiene 00101001, y al rotar a la izquierda **0**1101011 se obtiene 1101011**0**.
>
> Para rotar a la izquierda un byte, se puede multiplicar el número por 2, o lo que es lo mismo sumarlo a sí mismo. Por ejemplo (verificar):
>
> **0**1101011
> \+ **0**1101011
> 11010110 (CARRY=0)
>
> Entonces, la instrucción add ah, ah permite hacer una rotación a izquierda. No obstante, también hay que tener en cuenta que si el bit más significativo es un 1, el carry debe llevarse al bit menos significativo, es decir, se le debe sumar 1 al resultado de la primera suma.
>
> **1**0010100
> \+ **1**0010100
> 00101000 (CARRY=1)
> \+ 00000001
> 00101001

b) **ROTARIZQ_N**: Usando la subrutina ROTARIZQ del ejercicio anterior, escriba una subrutina ROTARIZQ_N que realice N rotaciones a la izquierda de un byte. La forma de pasaje de parámetros es la misma, pero se agrega el parámetro N que se recibe por valor y registro. Por ejemplo, al rotar a la izquierda 2 veces el byte **10**010100, se obtiene el byte 01010010.

c) **ROTARDER_N**: ***** Usando la subrutina ROTARIZQ_N del ejercicio anterior, escriba una subrutina ROTARDER_N que sea similar pero que realice N rotaciones hacia la **derecha**.

> **Pista**: Una rotación a derecha de N posiciones, para un byte con 8 bits, se obtiene rotando a la izquierda 8 - *N* posiciones. Por ejemplo, al rotar a la derecha 6 veces el byte 10**010100** se obtiene el byte **01010010**, que es equivalente a la rotación a la izquierda de 2 posiciones del ejemplo anterior.

10) SWAP Escribir una subrutina SWAP que intercambie dos datos de 16 bits almacenados en memoria. Los parámetros deben ser pasados por referencia desde el programa principal a través de la pila.

Pista: que los parámetros que se pasan por la pila son las *direcciones* de memoria, por lo tanto para acceder a los datos a intercambiar se requieren accesos indirectos, además de los que ya se deben realizar para acceder a los parámetros de la pila.

11) Subrutinas de cálculo

a) Escriba la subrutina DIV que calcule el resultado de la división entre 2 números positivos A y B. Dichos números deben pasarse por valor desde el programa principal a la subrutina a través de la pila. El resultado debe devolverse también a través de la pila por valor.

b) Escriba la subrutina RESTO que calcule el resto de la división entre 2 números positivos A y B. Dichos números deben pasarse por valor desde el programa principal a la subrutina a través de registros. El resultado debe devolverse a través de un registro por referencia.

Pista: Utilice las subrutinas DIV y MUL, ya que RESTO = A - (A DIV B) MUL B

12) Entender subrutina recursiva Analizar el funcionamiento de la siguiente subrutina **recursiva** y su programa principal:

```
      ORG 3000h          ORG 2000h
MUL: CMP AX, 0           MOV CX, 0
     JZ  FIN             MOV AX, 3
     ADD CX, AX          CALL MUL
     DEC AX              HLT
     CALL MUL            END
FIN: RET
```

Respondé:

a) ¿Qué hace la subrutina?
b) ¿Cuál será el valor final de CX?
c) Dibujar las posiciones de memoria de la pila, anotando qué valores va tomando
d) ¿Cuál será la limitación para determinar el valor más grande que se le puede pasar a la subrutina a través de AX?

Ejercicios optativos

13) Implementar sumatoria recursiva. La sumatoria de los primeros N números naturales es $S_N = 1 + 2 + 3 + \ldots + N$. Si bien hay varias formas de calcular este valor, por ejemplo sumando los N números en un **for** o utilizando la fórmula $S = N * (N-1) / 2$, también se puede calcular de forma recursiva con la fórmula $S_N = S_{N-1} + N$. Escribí una subrutina recursiva (que se llame a sí misma con un call) para computar la sumatoria S. La subrutina debe recibir en CX el número N y devolver en DX la sumatoria.

Pista: La subrutina debe considerar los dos casos: base y recursivo. En el caso base, cuando N = 0, la subrutina simplemente devuelve 0 en DX. En el caso recursivo (N>0), la subrutina apila el valor de N y hace el llamado recursivo con N-1. Cuando recibe el resultado, S_{N-1}, desapila el valor de N para aplicar la fórmula $S_N = S_{N-1} + N$ y devolver el resultado.

14) Pasaje de vectores por la pila Al pasar por parámetro un vector, generalmente enviamos su dirección de memoria inicial y su tamaño. Otra opción posible es apilar el tamaño del vector, y luego todos sus valores. Escribir una subrutina **SUMAR_PILA** que sume todos los valores de un vector pasado de esta forma por la pila. **Pista**: te damos el programa principal de ayuda.

```
org 1000H
vector dw 15,12,-4,5
long dw 5
```

```
org 2000H
mov cx, long
mov bx, offset vector
loop:mov ax, [bx]
     push ax
     add bx, 2
     dec cx
     jnz loop
mov cx, long
push cx
```

```
call SUMAR_PILA
pop cx
loop2:pop ax
      dec cx
      jnz loop2
hlt
end
```