PRACTICA 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 MSX88. 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`	8000h	5	?
2	`mov bx,3`	8000h	5	3
3	`push ax`	7FFEh	5	3
4	`push ax`	7FFCh	5	3
5	`push bx`	7FFAh	5	3
6	`pop bx`	7FFCh	5	3
7	`pop bx`	7FFEh	5	5
8	`pop ax`	8000h	5	5

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`	7FFCh
3	`ret`	7FFEh
4	`org 2000h`	-----
5	`push ax`	7FFEh
6	`call rutina`	7FFCh
7	`pop bx`	8000h
8	`hlt`	8000h
9	`end`	-----

3) *** Llamadas a subrutinas y dirección de retorno**

1. Si el registro SP vale 8000h al comenzar el programa, **indicar 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

3)

1)

```
org 3000h
rut: mov bx,3 ; PILA = 2003h; PILA 200Ah
ret ; PILA = 2003h; PILA 2000Ah
org 2000h
call rut ; PILA = 2003h
add cx,5 ; PILA = 2003h
call rut ; PILA = 200Ah
hlt ; PILA = 200Ah
end
```

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.

Nota: Las sentencias `ORG` y `END` no son instrucciones sino indicaciones al compilador, por lo tanto no se ejecutan ni tienen ubicación en memoria.

2. 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
```

a)

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	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) Cálculo de A+B-C. Pasaje de parámetros a través de registros.

En este ejercicio, programarás tus primeras subrutinas. Las subrutinas recibirán tres parámetros A, B y C, y realizarán un cálculo muy simple, A+B-C, cuyo resultado deben retornar. Si bien en general no tendría sentido escribir una subrutina para una cuenta tan simple que puede implementarse con dos instrucciones, esta simplificación permite concentrarse en los aspectos del pasaje de parámetros.

a) Escribir un programa que dados los valores etiquetados como A, B y C y almacenados en la memoria de datos, calcule A+B-C y guarde el resultado en la memoria con etiqueta D, **sin utilizar subrutinas**.

b) Escribir un programa como en a) pero ahora el cálculo y el almacenamiento del resultado debe realizarse en una subrutina llamada calculo, sin recibir ni devolver parámetros, es decir, utilizando A, B, C y D como variables globales. Si bien esta técnica no está recomendada, en ejercicio sirve para ver sus diferencias con el uso de parámetros.

c) Volver a escribir el programa, pero ahora con una subrutina que reciba A, B y C por valor a través de los registros AX, BX y CX, calcule AX+BX-CX, y devuelva el resultado por valor en el registro DX. El programa principal debe llamar a la subrutina y luego guardar el resultado en la memoria con etiqueta D

d) Si tuviera que realizar el cálculo dos veces con números distintos, por ejemplo, unos guardados en variables A1, B1, C1 y otros guardados en variables A2, B2, C2, ¿podrían reutilizarse las subrutinas del inciso b) sin modificarse? ¿y las del inciso c?

Podría reutilizar la subrutina del inciso c ya que la subrutina requiere de registros pero no la del inciso b ya que esta requiere de variables totales

5) a)

```
ORG 1000H
A DW 4
B DW 3
C DW 2
D DW ?

ORG 2000H
MOV AX, A
MOV BX, B
MOV CX, C
ADD AX, BX
MOV DX, AX
SUB DX, CX
MOV D, DX
HLT
END
```

b)

```
ORG 1000H
A DW 4
B DW 3
C DW 2
D DW ?

ORG 3000H
CALCULO: INC A
DEC B
JNZ CALCULO
RESTA: DEC A
DEC C
JNZ RESTA
RESULTADO: INC D
DEC A
JNZ RESULTADO
RET

ORG 2000H
MOV D, 0
CALL CALCULO

HLT
END
```

c)

```
ORG 1000H
A DW 4
B DW 3
C DW 2
D DW ?

ORG 3000H
CALCULO: ADD AX, BX
SUB AX, CX
MOV DX, AX
RET

ORG 2000H
MOV AX, A
MOV BX, B
MOV CX, C
CALL CALCULO
MOV D, DX
HLT
END
```

```
;Memoria de Datos
      org 1000h
  A   DW 5h
  B   DW 6h
  C   DW 2h
  D   DW ?
;Memoria de programa
      org 2000h
      ….      ; COMPLETAR
end
```

6) * Multiplicación de números sin signo. Pasaje de parámetros a través de registros.
El simulador no posee una instrucción para multiplicar números. Escribir un programa para multiplicar los números NUM1 y NUM2, y guardar el resultado en la variable RES
a) Sin hacer llamados a subrutinas, resolviendo el problema desde 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**.

7) Multiplicación de números sin signo. Pasaje de parámetros a través de Pila.
El programa de abajo utiliza una subrutina para multiplicar dos números, pasando los parámetros por valor para NUM1 y NUM2, y por referencia (RES), en ambos casos a través de la **pila**. Analizar su contenido y contestar.

a) ¿Cuál es el modo de direccionamiento de la instrucción MOV AX, [BX]? ¿Qué se copia en el registro AX en este caso? Direccionamiento por registro por valor
b) ¿Qué función cumple el registro temporal **ri** que aparece al ejecutarse una instrucción como la anterior?
c) ¿Qué se guarda en AX al ejecutarse MOV AX, OFFSET RES? La dirección de memoria de RES
d) ¿Cómo se pasa la variable RES a la pila, por valor o por referencia? ¿Qué ventaja tiene esto?Por referencia
e) ¿Cómo trabajan las instrucciones PUSH y POP?Push apila y Pop desapila siguiendo el principio LIFO (last in, first out)

Observaciones:
- Los contenidos de los registros AX, BX, CX y DX antes y después de ejecutarse la subrutina son iguales, dado que al comienzo se almacenan en la pila para poder utilizarlos sin perder la información que contenían antes del llamado. Al finalizar la subrutina, los contenidos de estos registros son restablecidos desde la pila.
- El programa sólo puede aplicarse al producto de dos números mayores que cero.

```
      ORG  3000H  ; Subrutina MUL
MUL:  PUSH BX     ; preservar registros
      PUSH CX
      PUSH AX
      PUSH DX
      MOV  BX, SP
      ADD  BX, 12 ; corrección por el IP(2),
                  ; RES(2) y los 4 registros(8)
      MOV  CX, [BX]  ; cx = num2
      ADD  BX, 2     ; bx apunta a num1
      MOV  AX, [BX]  ; ax = num1
      SUB  BX, 4     ; bx apunta a la dir de
                     ; resultado
      MOV  BX, [BX]  ; guardo
      MOV  DX, 0
SUMA: ADD  DX, AX
      DEC  CX
      JNZ  SUMA
      MOV  [BX], DX  ; guardar resultado
      POP  DX        ;restaurar registros
      POP  AX
      POP  CX
      POP  BX
      RET
```

```
      ORG 1000H ; Memoria de datos
NUM1  DW    5H
NUM2  DW    3H
RES   DW    ?

      ORG 2000H; Prog principal
          ; parámetros
          MOV   AX, NUM1
          PUSH  AX
          MOV   AX, NUM2
          PUSH  AX
          MOV   AX, OFFSET RES
          PUSH  AX
          CALL  MUL
          ; desapilar parámetros
          POP   AX
          POP   AX
          POP   AX
          HLT
          END
```

Analizar el siguiente diagrama de la pila y verificarlo con el simulador:

6) a)

```
ORG 1000H
NUM1 DW 2
NUM2 DW 3
RES DW ?

ORG 2000H
MOV AX, NUM1
MOV BX, NUM2
MOV DX, 0
LOOP: ADD DX, AX
DEC BX
JNZ LOOP
MOV RES, DX
HLT
END
```

b)

```
ORG 1000H
NUM1 DW 2
NUM2 DW 3
RES DW ?

ORG 3000H
MULT: MOV DX, 0
LOOP: ADD DX, AX
DEC BX
JNZ LOOP
RET

ORG 2000H
MOV AX, NUM1
MOV BX, NUM2

CALL MULT
MOV RES, DX
HLT
END
```

c)

```
ORG 1000H
NUM1 DW 2
NUM2 DW 3
RES DW ?

ORG 3000H
MULT: MOV DX, 0
MOV CX, BX
MOV BX, AX
MOV AX, [BX]
MOV BX, CX
LOOP: ADD DX, [BX]
DEC AX
JNZ LOOP
RET

ORG 2000H

MOV AX, OFFSET NUM1
MOV BX, OFFSET NUM2
CALL MULT
MOV RES, DX
HLT
END
```

7 b) El ir o ri (registro de instrucción) es un registro en la unidad de control de la CPU donde se almacena la instrucción que se está ejecutando actualmente o la próxima instrucción que se va a ejecutar. Este registro es fundamental en la ejecución de programas en una CPU, ya que contiene la información necesaria para decodificar y ejecutar la instrucción actual.

El contenido del registro de instrucción suele consistir en el código de operación (opcode) de la instrucción actual, así como cualquier dato o argumentos que la instrucción pueda requerir. La unidad de control extrae la instrucción del registro de instrucción, la decodifica y coordina las operaciones necesarias en la CPU para llevar a cabo la instrucción.

Por lo tanto, el registro de instrucción RI es un componente crítico de la ejecución de programas en una CPU, ya que guía la secuencia de operaciones que se deben realizar para llevar a cabo el programa. Gracias por la aclaración, y lamento la confusión anterior.

DIRECCIÓN DE MEMORIA	CONTENIDO
7FF0H	DL
	DH
7FF2H	AL
	AH
7FF4H	CL
	CH
7FF6H	BL
	BH

7FF8H	IP RET. L
	IP RET. H
7FFAH	DIR. RES L
	DIR. RES H
7FFCH	NUM2 L
	NUM2 H
7FFEH	NUM1 L
	NUM1 H
8000H	——

8) Subrutinas para realizar operaciones con cadenas de caracteres

a) Escribir una subrutina LONGITUD que cuente el número de caracteres de una cadena de caracteres terminada en 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 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.
Ejemplos: 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.

9) Subrutinas para realizar rotaciones

a) 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.

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**0**, y al rotar a la izquierda **0**1101011 se obtiene 11010110**1**.

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):

01101011
\+ **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.

10010100
\+ **1**0010100
00101000 (CARRY=1)
\+ 00000001
00101001

8 a)

```
ORG 1000H
CAR DB "EXCELENTE"
 DB 00H

ORG 3000H
LONGITUD: MOV DX, 0 ;contador
LOOP: MOV AH, [BX]
      CMP AH, 00H
      JZ FIN
      INC DX
      INC BX
      JMP LOOP
FIN: RET

 ORG 2000h
 MOV BX, offset CAR
 CALL LONGITUD
 HLT
 END
```

b)

```
ORG 1000H
CAR DB "aBcDE1#!"
 DB 00H
RES DW ?

ORG 3000H
CONTAR_MIN: MOV DX, 0 ;contador
LOOP: MOV AH, [BX]
      CMP AH, 00H
      JZ FIN


      CMP AH, 61H
      JZ INCREMENTAR
      CMP AH, 62H
      JZ INCREMENTAR
      CMP AH, 63H
      JZ INCREMENTAR
      CMP AH, 64H
      JZ INCREMENTAR
      CMP AH, 65H
      JZ INCREMENTAR
      CMP AH, 66H
      JZ INCREMENTAR
      CMP AH, 67H
      JZ INCREMENTAR
      CMP AH, 68H
      JZ INCREMENTAR
      CMP AH, 69H
      JZ INCREMENTAR
      CMP AH, 6AH
      JZ INCREMENTAR
      CMP AH, 6BH
      JZ INCREMENTAR
      CMP AH, 6CH
      JZ INCREMENTAR
      CMP AH, 6DH
      JZ INCREMENTAR
      CMP AH, 6EH
      JZ INCREMENTAR
      CMP AH, 6FH
      JZ INCREMENTAR
      CMP AH, 70H
      JZ INCREMENTAR
      CMP AH, 71H
      JZ INCREMENTAR
      CMP AH, 72H
      JZ INCREMENTAR
      CMP AH, 73H
      JZ INCREMENTAR
      CMP AH, 74H
      JZ INCREMENTAR
      CMP AH, 75H
      JZ INCREMENTAR
      CMP AH, 76H
      JZ INCREMENTAR
      CMP AH, 77H
      JZ INCREMENTAR
      CMP AH, 78H
      JZ INCREMENTAR
      CMP AH, 79H
      JZ INCREMENTAR
      CMP AH, 7AH
      JZ INCREMENTAR

MAYUSCULA:  INC BX
  JMP LOOP
FIN: RET

INCREMENTAR: INC DX
             INC BX
             JMP LOOP

 ORG 2000h
 MOV BX, offset CAR
 CALL CONTAR_MIN
 MOV RES, DX
 HLT
 END
```

c)

```
ORG 1000H
CAR DB "B"
RES DW ?


ORG 3000H
ES_VOCAL: MOV CX, 0FFH

      CMP AH, 00H
      JZ FIN
      CMP AH, 41H
      JZ FIN
      CMP AH, 45H
      JZ FIN
      CMP AH, 49H
      JZ FIN
      CMP AH, 4FH
      JZ FIN
      CMP AH, 55H
      JZ FIN

      CMP AH, 61H
      JZ FIN
      CMP AH, 65H
      JZ FIN
      CMP AH, 69H
      JZ FIN
      CMP AH, 6FH
      JZ FIN
      CMP AH, 75H
      JZ FIN
MOV CX, 00H

FIN: RET



 ORG 2000h
 MOV AH, CAR
 CALL ES_VOCAL
 MOV RES, CX
 HLT
 END
```

d)

```
ORG 1000H
CAR DB "aAb"
 DB 0
RES DW ?

ORG 4000H
ES_VOCAL: MOV AL, 0FFH
CMP AH, 41H ;A
JZ FIN
CMP AH, 45H ;E
JZ FIN
CMP AH, 49H ;I
JZ FIN
CMP AH, 4FH ;O
JZ FIN
CMP AH, 55H ;U
JZ FIN
CMP AH, 61H ;a
JZ FIN
CMP AH, 65H ;e
JZ FIN
CMP AH, 69H ;i
JZ FIN
CMP AH, 6FH ;o
JZ FIN
CMP AH, 75H ;u
JZ FIN
MOV AL, 00H
FIN: RET

ORG 3000H
VOCALES: MOV CX, 0
LOOP: MOV AH, [BX]
CMP AH, 0
JZ fin_vocales ; si es 0 es el fin de la cadena
CALL ES_VOCAL; si no es 0 llamo a la subrutina ES_VOCAL
CMP AL, 0FFH ; cuando retorna, comparo AL con 00FH
JNZ NOES ; si comparo y no es 0FFH (o sea que no es vocal, voy a NOES)
INC CX ; si es 0FFH incremento el contador de vocales

NOES: INC BX ; si no es vocal sigo a la siguiente letra
JMP LOOP

fin_vocales: RET; fin de las vocales

ORG 2000h
MOV BX, offset CAR
CALL VOCALES
MOV RES, CX
HLT
END
```

e)

```
ORG 1000H

CAR DB "abbcde!"
db 0
CARBUSC DB 'b'

ORG 3000H

CONTAR_CAR:
    MOV DX, 0
    MOV BX, SP
    ADD BX, 2
    MOV CL, [BX]
    ADD BX, 2
    MOV CH, 0 ; CH a cero


    MOV AX, [BX] ; Cargar el byte de la cadena en AX
    MOV BX, AX
 LOOP:  MOV AX, [BX]
    CMP AL, 0
    JZ FIN
    CMP AL, CL
    JZ INCREMENTO



NO_INCREMENT:
    INC BX          ; Moverse al siguiente byte de la cadena
    JMP LOOP

INCREMENTO:
    INC BX
    INC DX
    JMP LOOP

FIN: RET

ORG 2000H

MOV AX, OFFSET CAR
PUSH AX
MOV CL, CARBUSC
PUSH CX
CALL CONTAR_CAR

HLT
END

```

f)

```
ORG 1000H
CAR DB "abbcde!"
db 0
CARBUSC DB 'b'
CARCAMB DB 'a'

ORG 3000H
REEMPLAZAR_CAR:
    MOV BX, SP
    ADD BX, 2
    MOV DL, [BX]
    MOV DH, 0 ; DH a cero

    ADD BX, 2
    MOV CL, [BX]
    MOV CH, 0 ; CH a cero

    ADD BX, 2
    MOV AX, [BX] ; Cargar el byte de la cadena en AX
    MOV BX, AX
LOOP: MOV AX, [BX]
    CMP AL, 0
    JZ FIN
    CMP AL, CL
    JZ CAMBIAR
NO_CAMBIAR:
    INC BX        ; Moverse al siguiente byte de la cadena
    JMP LOOP
CAMBIAR:
    MOV [BX], DL
    INC BX
    JMP LOOP

FIN: RET

ORG 2000H

MOV AX, OFFSET CAR
PUSH AX
MOV CL, CARBUSC
PUSH CX
MOV DL, CARCAMB
PUSH DX
CALL REEMPLAZAR_CAR
HLT
END

```

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

d) Escriba la subrutina ROTARDER del ejercicio anterior, pero sin usar la subrutina ROTARIZ. Compare qué ventajas tiene cada una de las soluciones.

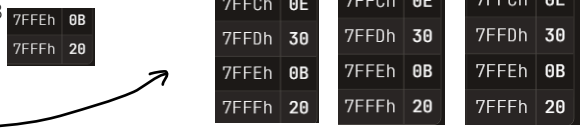
10) SWAP Escribir una subrutina SWAP que intercambie dos datos de 16 bits almacenados en memoria. Los parámetros deben ser pasados por referencia desde el programa principal a través de la pila.
Para hacer este ejercicio, tener en cuenta que los parámetros que se pasan por la pila son las *direcciones* 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. 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. Dichos números deben pasarse por valor desde el programa principal a la subrutina a través de registros. El resultado debe devolverse también a través de un registro por referencia.
c) Escribir un programa que calcule la suma de dos números de 32 bits almacenados en la memoria sin hacer llamados a subrutinas, resolviendo el problema desde el programa principal.
d) Escribir un programa que calcule la suma de dos números de 32 bits almacenados en la memoria llamando a una subrutina SUM32, que reciba los parámetros de entrada por valor a través de la pila, y devuelva el resultado también por referencia a través de la pila.

12) Analizar el funcionamiento de la siguiente subrutina 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
```



a) ¿Qué hace la subrutina? Suma dos números a la vez que uno se decrementa
b) ¿Cuál será el valor final de CX? 6
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?
Que la suma sucesiva de su decrementación no supere los 16 bits

Nota: Los ejercicios marcados con * tienen una solución propuesta.

9a)

```
ORG 1000H
BT DB 00010001b

ORG 3000H
ROTARIZQ:
MOV BX, AX
MOV AX, [BX]
ADD AL, AL
JC HAYCARRY
MOV BT, AL
RET

HAYCARRY: ADD AX, 00000001
MOV BT, AL
RET
ORG 2000H
MOV AX, OFFSET BT
CALL ROTARIZQ
HLT
END
```

b)

```
ORG 1000H
BT DB 10010001b
N DB 2

ORG 3000H
ROTARIZQ:
MOV BX, AX
MOV AX, [BX]
CONDICION: CMP CL, 0
JZ FIN
SUMA: ADD AL, AL
      JC HAYCARRY
      DEC CL
      JMP CONDICION


HAYCARRY: ADD AX, 00000001
DEC CL
JMP CONDICION

FIN: MOV BT, AL
RET

ORG 2000H
MOV AX, OFFSET BT
MOV CL, N
CALL ROTARIZQ
HLT
END
```

c)

```
ORG 1000H
BT DB 10010100b
N DB 2

ORG 3000H
ROTARDER:
MOV BX, AX
MOV AX, [BX]
SUB DL, CL
MOV CL, DL
CONDICION: CMP CL, 0
JZ FIN
SUMA: ADD AL, AL
      JC HAYCARRY
      DEC CL
      JMP CONDICION


HAYCARRY: ADD AX, 00000001
DEC CL
JMP CONDICION

FIN: MOV BT, AL
RET

ORG 2000H
MOV AX, OFFSET BT
MOV CL, N
MOV DL, 8
CALL ROTARDER
HLT
END
```

10)

```
ORG 1000H
VALOR1 DW 45
VALOR2 DW 46

ORG 3000H
SWAP: MOV BX, SP

;LIFO: VALOR 2 SALE PRIMERO
ADD BX, 2
MOV DX, BX
MOV AX, [BX]
MOV BX, AX
MOV AX, [BX]

;DATOS VALOR 1
MOV BX, DX
ADD BX, 2
MOV CX, [BX]
MOV BX, CX
MOV CX, [BX]
;¿DIRECCIONES?
MOV BX, DX
MOV BX, [BX]
MOV [BX], CX

ADD DX, 2
MOV BX, DX
MOV BX, [BX]
MOV [BX], AX

RET


ORG 2000H
MOV AX, OFFSET VALOR1
PUSH AX
MOV AX, OFFSET VALOR2
PUSH AX
CALL SWAP
HLT
END
```