

Trabajo Práctico N° 1:
Subrutinas y Pasaje de Parámetros.

Ejercicio 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
mov ax, 5	8000h	5	---
mov bx, 3	8000h	5	3
push ax	7FFCh	5	3
push bx	7FFAh	5	3
push ax	7FFEh	5	3
pop bx	7FFCh	5	3
pop bx	7FFEh	5	3
pop ax	8000h	5	3

Ejercicio 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
org 3000h	---
rutina: mov bx, 3	7FFCh
ret	7FFEh
org 2000h	---
push ax	7FFEh
call rutina	7FFCh
pop bx	8000h
hlt	8000h
end	8000h

Ejercicio 3: Llamadas a Subrutinas y Dirección de Retorno.

(a) 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 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.

org 3000h		
RUT:	mov bx, 3	Dirección 3000h
	ret	Dirección 3002h
		Pila: 2002h; 2006h
		Pila: 2002h; 2006h
org 2000h		
	call RUT	Dirección 2000h
	add cx, 5	Dirección 2002h
	call rut	Dirección 2004h
	hlt	Dirección 2006h
	end	
		Pila: 2002h
		Pila: 2002h
		Pila: 2002h; 2006h
		Pila: 2002h; 2006h

(b) Explicar detalladamente:

(i) Las acciones que tienen lugar al ejecutarse la instrucción CALL RUT.

Al ejecutarse la instrucción CALL RUT, se guarda el valor de la posición de memoria que está en el puntero de instrucción (IP) en la pila (PUSH del IP), se asigna el valor de la posición de memoria correspondiente a la etiqueta RUT al IP y la CPU comienza a ejecutar las instrucciones de la subrutina RUT.

(ii) Las acciones que tienen lugar al ejecutarse la instrucción RET.

La operación que se realiza con la instrucción ret es retornar al programa principal a partir de la instrucción siguiente a la instrucción CALL RUT. La CPU sabe a qué dirección de memoria debe retornar desde la subrutina al programa principal porque el puntero de instrucción (IP) se carga con el valor de la posición de memoria guardada en la pila (POP del IP) y, por lo tanto, la ejecución del programa sigue a partir de la instrucción siguiente a la instrucción CALL RUT.

Ejercicio 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
mov ax, 5 call subrutina	X		X	
mov dx, offset A call subrutina	X			X
mov bx, 5 push bx call subrutina pop bx		X	X	
mov cx, offset A push cx call subrutina pop cx		X		X
mov dl, 5 call subrutina	X		X	
call subrutina mov A, dx	X		X	

Ejercicio 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.

```
org 1000h
A DW 1h
B DW 2h
C DW 3h
D DW ?
```

```
org 2000h
mov ax, A
add ax, B
sub ax, C
mov D, ax
hlt
end
```

(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 este ejercicio, sirve para ver sus diferencias con el uso de parámetros.

```
org 1000h
A DW 1h
B DW 2h
C DW 3h
D DW ?

CALCULO: org 3000h
          mov ax, A
          add ax, B
          sub ax, C
          mov D, ax
          ret

          org 2000h
```

```
call CALCULO
hlt
end
```

(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.

```
org 1000h
A DW 1h
B DW 2h
C DW 3h
D DW ?

CALCULO: org 3000h
          mov dx, ax
          add dx, bx
          sub dx, cx
          ret
```

```
org 2000h
mov ax, A
mov bx, B
mov cx, C
call CALCULO
mov D, dx
hlt
end
```

(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)?

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, no podría reutilizar la subrutina del inciso (b) sin modificarla, aunque sí la subrutina del inciso (c).

Ejercicio 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.

```
org 1000h
NUM1 DB 1
NUM2 DB 2
RES DW ?

org 2000h
mov dx, 0
mov al, NUM1
cmp al, 0
jz FIN
mov ah, 0
mov cl, NUM2
LAZO: cmp cl, 0
jz FIN
add dx, ax
dec cl
jnz LAZO
FIN: mov RES, dx
hlt
end
```

(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.

```
org 1000h
NUM1 DB 1
NUM2 DB 2
RES DW ?

org 3000h
MUL: mov dx, 0
      cmp cl, 0
      jz FIN
      mov ah, 0
LAZO: add dx, ax
      dec cl
      jnz LAZO
```

```
FIN:      ret  
  
          org 2000h  
          mov al, NUM1  
          mov cl, NUM2  
          call MUL  
          mov RES, dx  
          hlt  
          end
```

(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.

```
org 1000h  
NUM1 DW 1  
NUM2 DW 2  
RES DW ?  
  
MUL:  
    org 3000h  
    mov dx, 0  
    mov bx, ax  
    mov ax, [bx]  
    mov bx, cx  
    mov cx, [bx]  
    cmp cx, 0  
    jz FIN  
LAZO:  
    add dx, ax  
    dec cx  
    jnz LAZO  
FIN:  
    ret  
  
          org 2000h  
          mov ax, offset NUM1  
          mov cx, offset NUM2  
          call MUL  
          mov RES, dx  
          hlt  
          end
```

Ejercicio 7.

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.

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
MUL:
PUSH BX
PUSH CX
PUSH AX
PUSH DX
MOV BX, SP
ADD BX, 12
MOV CX, [BX]
ADD BX, 2
MOV AX, [BX]
SUB BX, 4
MOV BX, [BX]
MOV DX, 0
SUMA:
ADD DX, AX
DEC CX
JNZ SUMA
MOV [BX], DX
POP DX
POP AX
POP CX
POP BX
RET

ORG 1000H
NUM1 DW 5H
NUM2 DW 3H
RES DW ?

ORG 2000H
MOV AX, NUM1
PUSH AX
MOV AX, NUM2
PUSH AX
MOV AX, OFFSET RES
PUSH AX
CALL MUL

*POP AX
POP AX
POP AX
HLT
END*

(a) ¿Cuál es el modo de direccionamiento de la instrucción MOV AX, [BX]? ¿Qué se copia en el registro AX en este caso?

El modo de direccionamiento de la instrucción MOV AX, [BX] es indirecto por registro y el valor que se copia en el registro AX, en este caso, es 5h.

(b) ¿Qué función cumple el registro temporal ri que aparece al ejecutarse una instrucción como la anterior?

El registro temporal denominado “ri” cumple la función de guardar, temporalmente, la dirección contenida en BX para, luego, ir a buscar el contenido de la misma.

(c) ¿Qué se guarda en AX al ejecutarse MOV AX, OFFSET RES?

En AX, al ejecutarse MOV AX, OFFSET RES, se guarda la dirección de la variable RES.

(d) ¿Cómo se pasa la variable RES a la pila, por valor o por referencia? ¿Qué ventaja tiene esto?

La variable RES a la pila se pasa por referencia y la ventaja que tiene esto (versus pasarla a la pila por valor) es poder, luego, en la subrutina SUMA, usar direccionamiento indirecto para guardar el resultado en la dirección de la variable RES.

(e) ¿Cómo trabajan las instrucciones PUSH y POP?

Las instrucciones PUSH y POP trabajan para el pasaje de parámetros y para preservar el contenido de los registros.

Ejercicio 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).

```
org 1000h
CADENA DB "abcde"
DB 0
RES DW ?

org 3000h
LONGITUD: mov dx, 0
LAZO:    cmp byte ptr [bx], 0
          jz FIN
          inc dx
          inc bx
          jmp LAZO
FIN:     ret

org 2000h
mov bx, offset CADENA
call LONGITUD
mov RES, dx
hlt
end
```

(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.

```
org 1000h
CADENA DB "aBcDe"
DB 0
RES DW ?

org 3000h
CONTAR_MIN: mov dx, 0
LAZO:    cmp byte ptr [bx], 0
          jz FIN
          mov al, [bx]
          cmp al, 123
          jns NO_ES_MIN
```

```
        cmp al, 97
        js NO_ES_MIN
        inc dx
NO_ES_MIN:    inc bx
                jmp LAZO
FIN:          ret

        org 2000h
        mov bx, offset CADENA
        call CONTAR_MIN
        mov RES, dx
        hlt
        end
```

(c) Escribir 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.

```
        org 1000h
        VOCALES DB 65,69,73,79,85,97,101,105,111,117
        CHAR DB "A"
        RES DB ?

        org 3000h
ES_VOCAL:    mov ah, 00h
                mov cl, offset CHAR - offset VOCALES
                mov bx, offset VOCALES
LAZO:         cmp al, [bx]
                jz VOCAL
                inc bx
                dec cl
                jz FIN
                jmp LAZO
VOCAL:         mov ah, 0FFh
FIN:          ret

        org 2000h
        mov al, CHAR
        call ES_VOCAL
        mov RES, ah
        hlt
        end
```

(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.

```
org 1000h
VOCALES DB 65,69,73,79,85,97,101,105,111,117
CADENA DB “AbCdE”
DB 0
RES DB ?

ES_VOCAL:
    org 3000h
    push bx
    mov ah, 00h
    mov cl, offset CADENA - offset VOCALES
    mov bx, offset VOCALES
LAZO1:
    cmp al, [bx]
    jz VOCAL
    inc bx
    dec cl
    jz FIN1
    jmp LAZO1
VOCAL:
    mov ah, 0FFh
FIN1:
    pop bx
    ret

CONTAR_VOC:
LAZO2:
    org 4000h
    mov dl, 0
    mov al, [bx]
    cmp al, 0
    jz FIN2
    jz FIN2
    call ES_VOCAL
    cmp ah, 0FFh
    jnz NO_ES_VOCAL
    inc dl
NO_ES_VOCAL:
    inc bx
    jmp LAZO2
FIN2:
    ret

    org 2000h
    mov bx, OFFSET CADENA
    call CONTAR_VOC
    mov RES, dl
    hlt
    end
```

(e) Escribir 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 se pasa 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.

```

CONTAR_CAR:
LAZO:
    org 1000h
    CADENA DB "AbCdE"
    DB 0
    CHAR DB "A"
    RES DB ?

    org 3000h
    mov ah, 0
    cmp byte ptr [bx], 0
    jz FIN
    cmp al, [bx]
    jnz NO_ES_IGUAL
    inc ah
    inc bx
    jmp LAZO
    ret

NO_ES_IGUAL:
    org 2000h
    mov al, CHAR
    mov bx, offset CADENA
    call CONTAR_CAR
    mov RES, ah
    hlt
    end

```

(f) Escribir 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.

```
org 1000h  
ORIGINAL DB "A"  
REEMPLAZO DB "E"  
CADENA DB "AbCdE"  
DB 0  
  
REEMPLAZAR_CAR:  
    org 3000h  
    push ax  
    push bx  
    mov bx, sp  
    add bx, 8  
    mov ax, [bx]  
    mov bx, sp  
    add bx, 6
```

```
        mov bx, [bx]
LAZO:   cmp byte ptr [bx], 0
        jz FIN
        cmp byte ptr [bx], al
        jnz NO_ES_IGUAL
        mov [bx], ah
NO_ES_IGUAL:    inc bx
                  jmp LAZO
FIN:      pop bx
          pop ax
          ret

org 2000h
mov al, ORIGINAL
mov ah, REEMPLAZO
mov cx, offset CADENA
push ax
push cx
call REEMPLAZAR_CAR
pop cx
pop ax
hlt
end
```

Ejercicio 9.

(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. 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 10010100, se obtiene 00101001 y, al rotar a la izquierda 01101011, se obtiene 11010110. Para rotar a la izquierda un byte, se puede multiplicar el número por 2 o, lo que es lo mismo, sumarlo a sí mismo. 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.

```
org 1000h
CADENA DB 10010100b

org 3000h
ROTARIZQ: add al, al
           adc al, 0
           mov CADENA, al
           ret

org 2000h
mov al, CADENA
call ROTARIZQ
hlt
end
```

(b) Usando la subrutina *ROTARIZQ* del ejercicio anterior, escribir 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 10010100, se obtiene el byte 01010010.

```
org 1000h
CADENA DB 10010100b
N DB 2

org 3000h
ROTARIZQ: add al, al
           adc al, 0
           ret

org 4000h
ROTARIZQ_N: cmp ah, 0
              jz FIN
```

```
call ROTARIZQ
dec ah
jmp ROTARIZQ_N
FIN:      mov CADENA, al
          ret
```

```
org 2000h
mov al, CADENA
mov ah, N
call ROTARIZQ_N
hlt
end
```

(c) Usando la subrutina *ROTARIZQ_N* del ejercicio anterior, escribir 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 10010100, se obtiene el byte 01010010, que es equivalente a la rotación a la izquierda de 2 posiciones del ejemplo anterior.

```
org 1000h
CADENA DB 10010100b
N DB 2

org 3000h
ROTARIZQ:    add al, al
              adc al, 0
              ret

org 4000h
ROTARIZQ_N:   cmp ah, 0
              jz FIN
              call ROTARIZQ
              dec ah
              jmp ROTARIZQ_N
FIN:          mov CADENA, al
              ret

org 5000h
ROTARDER_N:   mov cl, 8
              sub cl, ah
              mov ah, cl
              call ROTARIZQ_N
              ret

org 2000h
mov al, CADENA
mov ah, N
```

```
call ROTARDER_N
hlt
end
```

(d) Escribir la subrutina ROTARDER del ejercicio anterior, pero sin usar la subrutina ROTARIZQ. Comparar qué ventajas tiene cada una de las soluciones.

```
org 1000h
CADENA DB 10010100b
N DB 2

org 3000h
DIV:
    cmp al, 0
    jz FIN1
    cmp al, 2
    jc FIN1
    sub al, 2
    jc FIN1
    inc cl
    jmp DIV
FIN1:
    ret

org 4000h
ROTARDER:
    mov cl, 0
    call DIV
    cmp al, 1
    jnz FIN2
    add cl, 80h
FIN2:
    ret

org 5000h
ROTARDER_N:
    cmp ah, 0
    jz FIN3
    call ROTARDER
    dec ah
    mov al, cl
    jmp ROTARDER_N
    mov CADENA, cl
FIN3:
    ret

org 2000h
mov al, CADENA
mov ah, N
call ROTARDER_N
hlt
end
```

Ejercicio 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.

```
org 1000h
NUM1 DW 1234h
NUM2 DW 5678h

SWAP:    org 3000h
          push ax
          push bx
          push cx
          push dx
          mov bx, sp
          add bx, 10
          mov bx, [bx]
          mov cx, [bx]
          mov bx, sp
          add bx, 12
          mov bx, [bx]
          mov dx, [bx]
          mov bx, sp
          add bx, 10
          mov bx, [bx]
          mov [bx], dx
          mov bx, sp
          add bx, 12
          mov bx, [bx]
          mov [bx], cx
          pop dx
          pop cx
          pop bx
          pop ax
          ret

          org 2000h
          mov ax, offset NUM1
          mov cx, offset NUM2
          push ax
          push cx
          call SWAP
          pop cx
          pop ax
          hlt
```

end

Ejercicio 11: Subrutinas de Cálculo.

(a) Escribir 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.

```
org 1000h
NUM1 DB 10
NUM2 DB 5
RES DB ?

DIV:    org 3000h
        push ax
        push bx
        push cx
        mov cx, 0
        mov bx, sp
        add bx, 10
        mov ax, [bx]
LAZO:   sub al, ah
        js FIN
        inc cx
        jmp LAZO
FIN:    mov bx, sp
        add bx, 8
        mov bx, [bx]
        mov [bx], cx
        pop cx
        pop bx
        pop ax
        ret

org 2000h
mov al, NUM1
mov ah, NUM2
mov cx, offset RES
push ax
push cx
call DIV
pop cx
pop ax
hlt
end
```

(b) Escribir 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 valor.

```
org 1000h
NUM1 DB 10
NUM2 DB 5

          org 3000h
RESTO:    mov cl, 0
          mov ch, 0
          cmp ah, 0
          jz FIN
          cmp al, 0
          jz FIN
DIV:      sub al, ah
          js RES
          inc cl
          jmp DIV
RES:      add al, ah
          mov ch, al
FIN:      ret

          org 2000h
          mov al, NUM1
          mov ah, NUM2
          call RESTO
          hlt
          end
```

(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.

```
org 1000h
NUM1 DW 1,2
NUM2 DW 3,4
SUMA DW ?,?
DIR3 DW ?

          org 2000h
          mov ax, offset NUM1 + 2
          mov cx, offset NUM2 + 2
          mov DIR3, offset SUMA + 2
          mov bx, ax
          mov dx, [bx]
          mov bx, cx
          add dx, [bx]
```

```
pushf
mov bx, DIR3
mov [bx], dx
sub ax, 2
sub cx, 2
sub DIR3, 2
mov bx, ax
mov dx, [bx]
mov bx, cx
popf
adc dx, [bx]
mov bx, DIR3
mov [bx], dx
hlt
end
```

(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 referencia a través de la pila y devuelva el resultado también por referencia a través de la pila.

```
org 1000h
NUM1 DW 1,2
NUM2 DW 3,4
SUMA DW ?,?

org 3000h
SUM32:
    push ax
    push bx
    push cx

    mov bx, sp
    add bx, 12
    mov bx, [bx]
    mov ax, [bx]

    mov bx, sp
    add bx, 10
    mov bx, [bx]
    mov cx, [bx]

    add ax, cx
    pushf

    mov bx, sp
    add bx, 10
    mov bx, [bx]
    mov [bx], ax
```

```
mov bx, sp
add bx, 14
mov bx, [bx]
sub bx, 2
mov ax, [bx]

mov bx, sp
add bx, 12
mov bx, [bx]
sub bx, 2
mov cx, [bx]

popf
adc ax, cx

mov bx, sp
add bx, 8
mov bx, [bx]
sub bx, 2
mov [bx], ax

FIN:    pop cx
        pop bx
        pop ax
        ret

org 2000h
mov ax, offset NUM1 + 2
mov cx, offset NUM2 + 2
mov dx, offset SUMA + 2
push ax
push cx
push dx
call SUM32
pop dx
pop cx
pop ax
hlt
end
```

Ejercicio 12.

Analizar el funcionamiento de la siguiente subrutina y su programa principal:

ORG 3000H
MUL: CMP AX, 0
JZ FIN
ADD CX, AX
DEC AX
CALL MUL
FIN: RET

ORG 2000H
MOV CX, 0
MOV AX, 3
CALL MUL
HLT
END

(a) ¿Qué hace la subrutina?

La subrutina suma en CX todos los números comprendidos entre 0 y el valor del registro AX (3).

(b) ¿Cuál será el valor final de CX?

El valor final de CX será 6.

(c) Dibujar las posiciones de memoria de la pila, anotando qué valores va tomando.

SP		call MUL	call MUL	call MUL	call MUL	ret	ret	ret	ret
7FF8h					0E	0E	0E	0E	0E
7FF9h					30	30	30	30	30
7FFAh				0E	0E	0E	0E	0E	0E
7FFBh				30	30	30	30	30	30
7FFCh			0E	0E	0E	0E	0E	0E	0E
7FFDh			30	30	30	30	30	30	30
7FFEh		0B	0B	0B	0B	0B	0B	0B	0B
7FFFh		20	20	20	20	20	20	20	20
8000h	?	?	?	?	?	?	?	?	?

(d) ¿Cuál será la limitación para determinar el valor más grande que se le puede pasar a la subrutina a través de AX?

Para determinar el valor más grande que se le puede pasar a la subrutina a través de AX, se debe calcular el mínimo valor entre 255 (número que corresponde con que la suma entre 0 y ese valor sea igual a 32.767) y el tamaño de la pila (en bits) dividido 16.