

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.

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

(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 ?

                                org 3000h
CALCULO: 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 ?

org 3000h
CALCULO: 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 ?

          org 3000h
MUL:      mov dx, 0
          mov bx, ax
          mov ax, [bx]
          mov bx, cx
          mov cx, [bx]
          cmp cx, 0
          jz FIN
          add dx, ax
          dec cx
          jnz LAZO
LAZO:
          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.*

```

MUL:      ORG 3000H
           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 caracter es vocal o no, ya sea mayúscula o minúscula. La rutina debe recibir el caracter 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

                                org 4000h
CONTAR_VOC:                     mov dl, 0
LAZO2:                          mov al, [bx]
                                cmp al, 0
                                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 caracter dado en una cadena terminada en cero. El caracter 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 'abbcdel' y 'b' debe retornar 2, mientras que CONTAR_CAR de 'abbcdel' y 'z' debe retornar 0.

```

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

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

                                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 caracter ORIGINAL por el caracter 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

```

```
LAZO:      mov bx, [bx]
           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
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

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

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

FIN:

ROTARDER_N:
org 5000h
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

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

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

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

SUM32:          org 3000h
                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.

Trabajo Práctico N° 2: **Interrupciones.**

Ejercicio 1: Escritura de datos en la pantalla de comandos.

Implementar un programa en el lenguaje Assembler del simulador MSX88 que muestre, en la pantalla de comandos, un mensaje previamente almacenado en memoria de datos, aplicando la interrupción por software INT 7.

```
org 1000h
MSJ DB "ARQUITECTURA DE COMPUTADORAS - "
DB "FACULTAD DE INFORMÁTICA - "
DB 55h
DB 4Eh
DB 4Ch
DB 50h
FIN DB ?
```

```
org 2000h
mov bx, offset MSJ
mov al, offset FIN - offset MSJ
int 7
int 0
end
```

Ejercicio 2.

Escribir un programa que muestre, en pantalla, todos los caracteres disponibles en el simulador MSX88, comenzando con el caracter cuyo código es el número 01h.

```
                                org 1000h
                                CHAR DW 01h

                                org 2000h
                                mov al, 1
                                mov bx, offset CHAR
LAZO:                          int 7
                                inc CHAR
                                cmp CHAR, 256
                                jnz LAZO
                                int 0
                                end
```

Ejercicio 3.

Escribir un programa que muestre, en pantalla, las letras del abecedario, sin espacios, intercalando mayúsculas y minúsculas (AaBB...), sin incluir texto en la memoria de datos del programa. Tener en cuenta que el código de “A” es 41h, el de “a” es 61h y que el resto de los códigos son correlativos según el abecedario.

```
                                org 1000h
                                MAY DB 41h
                                MIN DB 61h

                                org 2000h
                                mov bx, offset MAY
                                mov al, 2
LAZO:                          int 7
                                inc MAY
                                inc MIN
                                cmp MAY, 5Bh
                                jnz LAZO
                                int 0
                                end
```

Ejercicio 4: Lectura de datos desde el teclado.

Escribir un programa que solicite el ingreso de un número (de un dígito) por teclado e, inmediatamente, lo muestre en la pantalla de comandos, haciendo uso de las interrupciones por software INT 6 e INT 7.

```
org 1000h
MSJ DB "INGRESAR UN NÚMERO (de un dígito): "
FIN DB ?
NUM DB ?
```

```
org 2000h
mov bx, offset MSJ
mov al, offset FIN - offset MSJ
int 7
mov bx, offset NUM
int 6
mov al, 1
int 7
mov cl, NUM
int 0
end
```

Responder brevemente:

(a) Con referencia a la interrupción INT 7, ¿qué se almacena en los registros BX y AL?

En los registros BX y AL, se almacena la dirección de memoria del carácter inicial de MSJ y el tamaño del mensaje, respectivamente.

(b) Con referencia a la interrupción INT 6, ¿qué se almacena en BX?

En BX, se almacena la dirección de memoria de NUM.

(c) En el programa anterior, ¿qué hace la segunda interrupción INT 7? ¿qué queda almacenado en el registro CL?

La segunda interrupción INT 7 lo que hace es imprimir el número (de un dígito) que se ingresó por teclado. Lo que queda almacenado en registro CL es el código ASCII correspondiente al número (como carácter) ingresado por teclado.

Ejercicio 5.

Modificar el programa anterior agregando una subrutina llamada ES_NUM que verifique si el caracter ingresado es, realmente, un número. De no serlo, el programa debe mostrar el mensaje “CARACTER NO VÁLIDO”. La subrutina debe recibir el código del caracter por referencia desde el programa principal y debe devolver, vía registro, el valor 0FFh, en caso de tratarse de un número, o el valor 00h, en caso contrario. Tener en cuenta que el código del “0” es 30h y el del “9” es 39h.

```

                                org 1000h
                                MSJ1 DB “INGRESAR UN NÚMERO (de un dígito): ”
                                FIN1 DB ?
                                MSJ2 DB “CARACTER NO VÁLIDO”
                                FIN2 DB ?
                                NUM DB ?

ES_NUM:                        org 3000h
                                mov ah, 0FFh
                                cmp byte ptr [bx], 30h
                                js ERROR
                                cmp byte ptr [bx], 3Ah
                                jns ERROR
                                jmp FIN
ERROR:                          mov ah, 00h
                                mov bx, offset MSJ2
                                mov al, offset FIN2 - offset MSJ2
                                int 7
FIN:                            ret

                                org 2000h
                                mov bx, offset MSJ1
                                mov al, offset FIN1 - offset MSJ1
                                int 7
                                mov bx, offset NUM
                                int 6
                                call ES_NUM
                                mov al, 1
                                int 7
                                int 0
                                end

```

Ejercicio 6.

Escribir un programa que solicite el ingreso de un número (de un dígito) por teclado y muestre, en pantalla, dicho número expresado en letras. Luego, que solicite el ingreso de otro y así sucesivamente. Se debe finalizar la ejecución al ingresarse, en dos vueltas consecutivas, el número cero.

```

org 1000h
CERO DB "CERO "
DB "UNO "
DB "DOS "
DB "TRES "
DB "CUATRO"
DB "CINCO "
DB "SEIS "
DB "SIETE "
DB "OCHO "
DB "NUEVE "
MSJ DB "INGRESAR UN NÚMERO (de un dígito): "
FIN DB ?
NUM DB ?

org 2000h
mov cl, 0
OTRO:  mov bx, offset MSJ
      mov al, offset FIN - offset MSJ
      int 7
      mov bx, offset NUM
      int 6
      cmp NUM, 30h
      jnz NO_CERO
      inc cl
      jmp SEGUIR
NO_CERO: mov cl, 0
SEGUIR:  mov bx, offset CERO
      mov al, 6
LAZO:   cmp NUM, 30h
      jz IMPRIME
      add bx, 6
      dec NUM
      jmp LAZO
IMPRIMIR: int 7
      cmp cl, 2
      jnz OTRO
      int 0
      end

```


Ejercicio 7.

Escribir un programa que efectúe la suma de dos números (de un dígito cada uno) ingresados por teclado y muestre el resultado en la pantalla de comandos. Recordar que el código de cada caracter ingresado no coincide con el número que representa y que el resultado puede necesitar ser expresado con 2 dígitos.

```

org 1000h
MSJ1 DB "INGRESAR UN NÚMERO (de un dígito): "
FIN1 DB ?
MSJ2 DB 10, "INGRESAR OTRO NÚMERO (de un dígito): "
FIN2 DB ?
MSJ3 DB 10, "RESULTADO DE LA SUMA DE AMBOS NÚMEROS
INGRESADOS: "
RES_D DB "0"
RES_U DB ?
FIN3 DB ?
NUM1 DB ?
NUM2 DB ?

org 2000h
mov bx, offset MSJ1
mov al, offset FIN1 - offset MSJ1
int 7
mov bx, offset NUM1
int 6
mov al, 1
int 7
mov bx, offset MSJ2
mov al, offset FIN2 - offset MSJ2
int 7
mov bx, offset NUM2
int 6
mov al, 1
int 7
mov al, NUM1
sub al, 30h
add al, NUM2
cmp al, 3Ah
js UNIDAD
sub al, 10
inc RES_D
UNIDAD: mov RES_U, al
mov bx, offset MSJ3
mov al, offset FIN3 - offset MSJ3
int 7
int 0
end

```

Ejercicio 8.

Escribir un programa que efectúe la resta de dos números (de un dígito cada uno) ingresados por teclado y muestre el resultado en la pantalla de comandos. Antes de visualizarlo, el programa debe verificar si el resultado es positivo o negativo y anteponer, al valor, el signo correspondiente.

```

org 1000h
MSJ1 DB "NUM1: "
FIN1 DB ?
MSJ2 DB 10, "NUM2: "
FIN2 DB ?
MSJ3 DB 10, "RESTA: "
SIGNO DB "+"
RES DB ?
FIN3 DB ?
NUM1 DB ?
NUM2 DB ?

org 2000h
mov bx, offset MSJ1
mov al, offset FIN1 - offset MSJ1
int 7
mov bx, offset NUM1
int 6
mov al, 1
int 7
mov bx, offset MSJ2
mov al, offset FIN2 - offset MSJ2
int 7
mov bx, offset NUM2
int 6
mov al, 1
int 7
mov al, NUM1
mov ah, NUM2
cmp al, ah
js NEGATIVO
sub al, ah
mov RES, al
jmp FIN
NEGATIVO: sub ah, al
mov RES, ah
FIN:      add RES, 30h
mov bx, offset MSJ3
mov al, offset FIN3 - offset MSJ3
int 7
int 0
end

```

Escribir un programa que aguarde el ingreso de una clave de cuatro caracteres por teclado sin visualizarla en pantalla. En caso de coincidir con una clave predefinida (y guardada en memoria), que muestre el mensaje “Acceso permitido”, caso contrario el mensaje “Acceso denegado”.

```

org 1000h
CLAVE_PRE DB "1234"
MSJ1 DB "INGRESAR UNA CLAVE (de cuatro caracteres): "
FIN1 DB ?
MSJ2 DB 10, "ACCESO PERMITIDO"
FIN2 DB ?
MSJ3 DB 10, "ACCESO DENEGADO"
FIN3 DB ?
CLAVE DB ?,?,?,?

org 2000h
mov bx, offset MSJ1
mov al, offset FIN1 - offset MSJ1
int 7
mov bx, offset CLAVE
mov ah, 4
LAZO1: int 6
mov al, 1
int 7
inc bx
dec ah
cmp ah, 0
jnz LAZO1
mov al, CLAVE
mov cx, 0
LAZO2: mov bx, offset CLAVE_PRE
add bx, cx
mov dl, [bx]
mov bx, offset CLAVE
add bx, cx
mov dh, [bx]
cmp dl, dh
jnz DENEGADO
inc cx
cmp cx, 4
jnz LAZO2
mov bx, offset MSJ2
mov al, offset FIN2 - offset MSJ2
int 7
jmp FIN
DENEGADO: mov bx, offset MSJ3
mov al, offset FIN3 - offset MSJ3

```

```
int 0
```

end

Ejercicio 10: Interrupción por hardware (Tecla F10).

Escribir un programa que, mientras ejecuta un lazo infinito, cuente el número de veces que se presiona la tecla F10 y acumule este valor en el registro DX.

```

                PIC EQU 20H
                EOI EQU 20H
                N_F10 EQU 10

                ORG 40
                IP_F10 DW RUT_F10

                ORG 3000H
RUT_F10:        PUSH AX
                INC DX
                MOV AL, EOI
                OUT EOI, AL
                POP AX
                IRET

                ORG 2000H
                CLI
                MOV AL, 0FEH
                OUT PIC+1, AL
                MOV AL, N_F10
                OUT PIC+4, AL
                MOV DX, 0
                STI
LAZO:           JMP LAZO
                END

```

Explicar detalladamente:

(a) *La función de los registros del PIC: ISR, IRR, IMR, INT0-INT7, EOI. Indicar la dirección de cada uno.*

La función de los registros del PIC es:

- ISR (23h): Indicar la interrupciones en ejecución.
- IRR (22h): Indicar la interrupciones pedidas.
- IMR (21h): Indicar la interrupciones habilitadas.
- INT0-INT7 (24h-31h): Indicar el ID de interrupción de cada dispositivo.
- EOI (20h): Indicar la finalización de la interrupción.

(b) *Cuáles de estos registros son programables y cómo trabaja la instrucción OUT.*

De estos registros, son programables el IMR, INT0-INT7 y el EOI. La instrucción OUT trabaja moviendo contenido del registro AL al PIC.

(c) Qué hacen y para qué se usan las instrucciones CLI y STI.

Las instrucciones CLI y STI se usan para deshabilitar y habilitar interrupciones, respectivamente.

Ejercicio 11.

Escribir un programa que permita seleccionar una letra del abecedario al azar. El código de la letra debe generarse en un registro que incremente su valor desde el código de A hasta el de Z continuamente. La letra debe quedar seleccionada al presionarse la tecla F10 y debe mostrarse, de inmediato, en la pantalla de comandos.

```

EOI EQU 20h
IMR EQU 21h
INT0 EQU 24h
N_F10 EQU 10

org 40
IP_F10 DW RUT_F10

org 1000h
LETRA DB ?

RUT_F10:    org 3000h
            push ax
            push bx
            inc cl
            mov LETRA, ah
            mov bx, offset LETRA
            mov al, 1
            int 7
            mov al, EOI
            out EOI, al
            pop bx
            pop ax
            iret

ABCDE:      org 4000h
LAZO:       mov ah, 65
            cmp cl, 1
            jz FIN
            inc ah
            cmp ah, 90
            jnz LAZO
            jmp ABCDE
FIN:        ret

org 2000h
cli
mov al, 0FEh
out IMR, al
mov al, N_F10
out INT0, al
mov cl, 0

```

```
sti  
call ABCDE  
int 0  
end
```


Ejercicio 12: Interrupción por hardware (Timer).

Implementar, a través de un programa, un reloj segundero que muestre, en pantalla, los segundos transcurridos (00-59 seg.) desde el inicio de la ejecución.

```

TIMER EQU 10H
PIC EQU 20H
EOI EQU 20H
N_CLK EQU 10

ORG 40
IP_CLK DW RUT_CLK

ORG 1000H
SEG DB 30H
DB 30H
FIN DB ?

RUT_CLK:
ORG 3000H
PUSH AX
INC SEG+1
CMP SEG+1, 3AH
JNZ RESET
MOV SEG+1, 30H
INC SEG
CMP SEG, 36H
JNZ RESET
MOV SEG, 30H
RESET:
INT 7
MOV AL, 0
OUT TIMER, AL
MOV AL, EOI
OUT PIC, AL
POP AX
IRET

ORG 2000H
CLI
MOV AL, 0FDH
OUT PIC+1, AL
MOV AL, N_CLK
OUT PIC+5, AL
MOV AL, 10
OUT TIMER+1, AL
MOV AL, 0
OUT TIMER, AL
MOV BX, OFFSET SEG
MOV AL, OFFSET FIN - OFFSET SEG
STI
LAZO:
JMP LAZO

```

END

Explicar detalladamente:

(a) *Cómo funciona el TIMER y cuándo emite una interrupción a la CPU.*

El Timer es otro dispositivo de ES como el F10. Se utiliza como un reloj despertador para la CPU. Se configura para contar una cantidad determinada de segundos y, cuando finaliza la cuenta, emite una interrupción. El Timer tiene dos registros, CONT (registro contador) y COMP (registro de comparación), con direcciones de la memoria de ES 10h y 11h, respectivamente.

(b) *La función que cumplen sus registros, la dirección de cada uno y cómo se programan.*

La función de los registros del Timer es:

- CONT (10h): Se incrementa, automáticamente, una vez por segundo, para contar tiempo transcurrido.
- COMP (11h): Contiene el tiempo límite del Timer. Cuando CONT vale igual que COMP, se dispara la interrupción.

Ejercicio 13.

Modificar el programa anterior para que también cuente minutos (00:00-59:59), pero que actualice la visualización en pantalla cada 10 segundos.

```

CONT EQU 10h
COMP EQU 11h
EOI EQU 20h
IMR EQU 21h
INT1 EQU 25h
N_CLK EQU 10

org 40
IP_CLK DW RUT_CLK

org 1000h
MIN_D DB 30h
MIN_U DB 30h, 58
SEG_D DB 30h
SEG_U DB 30h, 32
FIN DB ?

RUT_CLK:    org 3000h
            push ax
            inc SEG_D
            cmp SEG_D, 36h
            jnz IMPRIMIR
            mov SEG_D, 30h
            inc MIN_U
            cmp MIN_U, 3Ah
            jnz IMPRIMIR
            mov MIN_U, 30h
            inc MIN_D
            cmp MIN_D, 36h
            jnz IMPRIMIR
            mov MIN_D, 30h

IMPRIMIR:   int 7
            mov al, 0
            out CONT, al
            mov al, EOI
            out EOI, al
            pop ax
            iret

org 2000h
cli
mov al, 0FDh
out IMR, al
mov al, N_CLK

```

```
    out INT1, al
    mov al, 0
    out CONT, al
    mov al, 10
    out COMP, al
    mov bx, offset MIN_D
    mov al, offset FIN - offset MIN_D
    sti
LAZO: jmp LAZO
    int 0
    end
```

Ejercicio 14.

Implementar un reloj similar al utilizado en los partidos de básquet, que arranque y detenga su marcha al presionar sucesivas veces la tecla F10 y que finalice el conteo al alcanzar los 30 segundos.

```

CONT EQU 10h
COMP EQU 11h
EOI EQU 20h
IMR EQU 21h
INT0 EQU 24h
INT1 EQU 25h
N_CLK EQU 10
N_F10 EQU 20

org 40
IP_CLK DW RUT_CLK

org 80
IP_F10 DW RUT_F10

org 1000h
SEG_D DB 30h
SEG_U DB 30h, 32
FIN DB ?

RUT_CLK:  org 3000h
           push ax
           inc SEG_U
           cmp SEG_U, 3Ah
           jnz IMPRIMIR
           mov SEG_U, 30h
           inc SEG_D
           cmp SEG_D, 33h
           jnz IMPRIMIR
           mov cl, 1
           mov al, 0FFh
           out IMR, al
IMPRIMIR: int 7
           mov al, 0
           out CONT, al
           mov al, EOI
           out EOI, al
           pop ax
           iret

RUT_F10:   org 4000h
           push ax
           in al, IMR

```

```
xor al, 00000010b
out IMR, al
mov al, EOI
out EOI, al
pop ax
iret
```

```
org 2000h
cli
mov al, 0FCh
out IMR, al
mov al, N_F10
out INT0, al
mov al, N_CLK
out INT1, al
mov al, 0
out CONT, al
mov al, 10
out COMP, al
mov bx, offset SEG_D
mov al, offset FIN - offset SEG_D
mov cl, 0
sti
LAZO: cmp cl, 0
      jz LAZO
      int 0
      end
```

Ejercicio 15.

Escribir un programa que implemente un conteo regresivo a partir de un valor ingresado desde el teclado. El conteo debe comenzar al presionarse la tecla F10. El tiempo transcurrido debe mostrarse en pantalla, actualizándose el valor cada segundo.

```

CONT EQU 10h
COMP EQU 11h
EOI EQU 20h
IMR EQU 21h
INT0 EQU 24h
INT1 EQU 25h
N_CLK EQU 10
N_F10 EQU 20

org 40
IP_CLK DW RUT_CLK

org 80
IP_F10 DW RUT_F10

org 1000h
NUM DB ?, 32
FIN_NUM DB ?

RUT_CLK:    org 3000h
            push ax
            int 7
            dec cl
            cmp cl, 30h
            jns SEGUIR
            mov al, 0FFh
            out IMR, al
            jmp FIN
SEGUIR:     mov NUM, cl
            mov al, 0
            out CONT, al
FIN:        mov al, EOI
            out EOI, al
            pop ax
            iret

RUT_F10:    org 4000h
            push ax
            in al, IMR
            xor al, 00000010b
            out IMR, al
            mov al, EOI
            out EOI, al

```

```
    pop ax
    iret

    org 2000h
    cli
    mov al, 0FEh
    out IMR, al
    mov al, N_F10
    out INT0, al
    mov al, N_CLK
    out INT1, al
    mov al, 0
    out CONT, al
    mov al, 10
    out COMP, al
    mov bx, offset NUM
    int 6
    mov bx, offset NUM
    mov al, offset FIN_NUM - offset NUM
    mov cl, NUM
    sti
LAZO:  cmp cl, 30h
       jns LAZO
       int 0
       end
```


Trabajo Práctico N° 3: **Entrada/Salida.**

Ejercicio 1: Uso de las luces y las llaves a través del PIO.

Ejecutar los programas con el simulador VonSim utilizando los dispositivos “Llaves y Luces” que conectan las llaves al puerto PA del PIO y a las luces al puerto PB.

(a) Escribir un programa que encienda las luces con el patrón 11000011, o sea sólo las primeras y las últimas dos luces deben prenderse y el resto deben apagarse.

PB EQU 31h

CB EQU 33h

org 1000h

PATRON DB 11000011b

org 2000h

mov al, 0

out CB, al

mov al, PATRON

out PB, al

int 0

end

(b) Escribir un programa que verifique si la llave de más a la izquierda está prendida. Si es así, mostrar en pantalla el mensaje “Llave prendida” y, de lo contrario, mostrar “Llave apagada”. Sólo importa el valor de la llave de más a la izquierda (bit más significativo). Recordar que las llaves se manejan con las teclas 0-7.

PA EQU 30h

CA EQU 32h

org 1000h

MSJ1 DB “Llave prendida”

FIN1 DB ?

MSJ2 DB “Llave apagada”

FIN2 DB ?

org 2000h

mov al, 0FFh

out CA, al

in al, PA

and al, 80h

cmp al, 0

jz APAGADA

```

        mov bx, offset MSJ1
        mov al, offset FIN1 - offset MSJ1
        jmp FIN
APAGADA: mov bx, offset MSJ2
        mov al, offset FIN2 - offset MSJ2
FIN:    int 7
        int 0
        end

```

(c) *Escribir un programa que permita encender y apagar las luces mediante las llaves. El programa no deberá terminar nunca y, continuamente, revisar el estado de las llaves y actualizar, de forma consecuyente, el estado de las luces. La actualización se realiza, simplemente, prendiendo la luz i si la llave i correspondiente está encendida (valor 1) y apagándola en caso contrario. Por ejemplo, si sólo la primera llave está encendida, entonces, sólo la primera luz se debe quedar encendida.*

```

        PA EQU 30h
        PB EQU 31h
        CA EQU 32h
        CB EQU 33h

        org 2000h
        mov al, 0FFh
        out CA, al
        mov al, 0
        out CB, al
POLL:   in al, PA
        out PB, al
        jmp POLL
        int 0
        end

```

(d) *Escribir un programa que implemente un encendido y apagado sincronizado de las luces. Un contador, que inicializa en cero, se incrementa en uno una vez por segundo. Por cada incremento, se muestra a través de las luces, prendiendo sólo aquellas luces donde el valor de las llaves es 1. Entonces, primero, se enciende sólo la luz de más a la derecha, correspondiente al patrón 00000001. Luego, se continúa con los patrones 00000010, 00000011 y así sucesivamente. El programa termina al llegar al patrón 11111111.*

```

        CONT EQU 10h
        COMP EQU 11h
        EOI EQU 20h
        IMR EQU 21h
        INT1 EQU 25h
        PB EQU 31h

```

```

CB EQU 33h
N_CLK EQU 10

org 40
IP_CLK DW RUT_CLK

RUT_CLK:  org 3000h
           push ax
           mov ax, cx
           out PB, ax
           inc cx
           cmp cx, 256
           jnz SEGUIR
           mov al, 0FFh
           out IMR, al
           jmp FIN
SEGUIR:    mov al, 0
           out CONT, al
FIN:       mov al, EOI
           out EOI, al
           pop ax
           iret

           org 2000h
           cli
           mov al, 0FDH
           out IMR, al
           mov al, N_CLK
           out INT1, al
           mov al, 0
           out CONT, al
           mov al, 10
           out COMP, al
           mov al, 0
           out CB, al
           mov cx, 0
           sti
LAZO:      cmp cx, 256
           jnz LAZO
           int 0
           end

```

(e) Escribir un programa que encienda una luz a la vez, de las ocho conectadas al puerto paralelo del microprocesador a través de la PIO, en el siguiente orden de bits: 0-1-2-3-4-5-6-7-6-5-4-3-2-1-0-1-2-3-4-5-6-7-6-5-4-3-2-1-0-1-..., es decir, 00000001, 00000010, 00000100, etc. Cada luz, debe estar encendida durante un segundo. El programa nunca termina.

Opción 1:

```
PB EQU 31h
CB EQU 33h

org 1000h
PATRON DB 0,1,2,4,8,16,32,64,128

org 2000h
mov bx, offset PATRON
mov al, 0
out CB, al
CRECER: inc bx
        mov al, [bx]
        out PB, al
        cmp byte ptr [bx], 128
        jnz CRECER
DECRECER: dec bx
        mov al, [bx]
        out PB, al
        cmp byte ptr [bx], 1
        jnz DECRECER
        jmp CRECER
int 0
end
```

Opción 2:

```
CONT EQU 10h
COMP EQU 11h
EOI EQU 20h
IMR EQU 21h
INT1 EQU 25h
PB EQU 31h
CB EQU 33h
N_CLK EQU 10

org 40
IP_CLK DW RUT_CLK

org 1000h
PATRON DB 0,1,2,4,8,16,32,64,128

org 3000h
RUT_CLK: push ax
        mov al, 0
        out CONT, al
        mov al, EOI
        out EOI, al
        pop ax
```

```
    ired

    org 2000h
    cli
    mov al, 0FDh
    out IMR, al
    mov al, N_CLK
    out INT1, al
    mov al, 0
    out CONT, al
    mov al, 10
    out COMP, al
    mov al, 0
    out CB, al
    mov bx, offset PATRON
    sti
CRECER:  cli
        inc bx
        mov al, [bx]
        out PB, al
        sti
        cmp byte ptr [bx], 128
        jnz CRECER
DECRECER: cli
        dec bx
        mov al, [bx]
        out PB, al
        sti
        cmp byte ptr [bx], 1
        jnz DECRECER
        jmp CRECER
    int 0
    end
```

Ejercicio 2: Uso de la impresora a través del PIO.

Ejecutar los programas configurando el simulador VonSim con los dispositivos “Impresora (PIO)”. En esta configuración, el puerto de datos de la impresora se conecta al puerto PB del PIO y los bits de busy y strobe de la misma se conectan a los bits 0 y 1, respectivamente, del puerto PA. Presionar F5 para mostrar la salida en papel. El papel se puede blanquear ingresando el comando BI.

(a) *Escribir un programa para imprimir la letra “A” utilizando la impresora a través de la PIO.*

```

PA EQU 30h
PB EQU 31h
CA EQU 32h
CB EQU 33h

org 1000h
CHAR DB "A"

PIO:    org 3000h
        push ax
        mov al, 1
        out CA, al
        mov al, 0
        out CB, al
        pop ax
        ret

STROBE0: org 4000h
        push ax
        in al, PA
        and al, 11111101b
        out PA, al
        pop ax
        ret

STROBE1: org 5000h
        push ax
        in al, PA
        or al, 00000010b
        out PA, al
        pop ax
        ret

POLL:   org 6000h
        push ax
        in al, PA
        and al, 1
        jnz POLL

```

```

pop ax
ret

org 2000h
call PIO
call STROBE0
call POLL
mov al, CHAR
out PB, al
call STROBE1
nop
nop
nop
nop
nop
int 0
end

```

(b) Escribir un programa para imprimir el mensaje “ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORAS” utilizando la impresora a través de la PIO.

```

PA EQU 30h
PB EQU 31h
CA EQU 32h
CB EQU 33h

org 1000h
MSJ DB “ORGANIZACIÓN Y ARQUITECTURA DE
COMPUTADORAS”
FIN DB ?

PIO:
org 3000h
push ax
mov al, 1
out CA, al
mov al, 0
out CB, al
pop ax
ret

STROBE0:
org 4000h
push ax
in al, PA
and al, 1111101b
out PA, al
pop ax
ret

```

```

STROBE1:    org 5000h
            push ax
            in al, PA
            or al, 00000010b
            out PA, al
            pop ax
            ret

POLL:       org 6000h
            push ax
            in al, PA
            and al, 1
            jnz POLL
            pop ax
            ret

LAZO:       org 2000h
            call PIO
            call STROBE0
            mov bx, offset MSJ
            mov cl, offset FIN - offset MSJ
            call POLL
            mov al, [bx]
            out PB, al
            call STROBE1
            call STROBE0
            inc bx
            dec cl
            jnz LAZO
            int 0
            end

```

(c) *Escribir un programa que solicita el ingreso de cinco caracteres por teclado y los envía, de a uno por vez, a la impresora a través de la PIO a medida que se van ingresando. No es necesario mostrar los caracteres en la pantalla.*

```

            PA EQU 30h
            PB EQU 31h
            CA EQU 32h
            CB EQU 33h

            org 1000h
            CHAR DB ?
            CHARS DB 5

PIO:        org 3000h
            push ax
            mov al, 1

```



```
        out CA, al
        mov al, 0
        out CB, al
        pop ax
        ret

STROBE0:    org 4000h
            push ax
            in al, PA
            and al, 11111101b
            out PA, al
            pop ax
            ret

STROBE1:    org 5000h
            push ax
            in al, PA
            or al, 00000010b
            out PA, al
            pop ax
            ret

POLL:       org 6000h
            push ax
            in al, PA
            and al, 1
            jnz POLL
            pop ax
            ret

LAZO:       org 2000h
            call PIO
            call STROBE0
            mov bx, offset CHAR
            mov cl, CHARS
            int 6
            call POLL
            mov al, [bx]
            out PB, al
            call STROBE1
            call STROBE0
            dec cl
            jnz LAZO
            int 0
            end
```

(d) *Escribir un programa que solicite ingresar caracteres por teclado y que, recién al presionar la tecla F10, los envíe a la impresora a través de la PIO. No es necesario mostrar los caracteres en la pantalla.*

```
EOI EQU 20h
IMR EQU 21h
INT0 EQU 24h
PA EQU 30h
PB EQU 31h
CA EQU 32h
CB EQU 33h
N_F10 EQU 10

org 40
ID_F10 DW RUT_F10

org 1000h
CADENA DB ?

RUT_F10:    org 3000h
            push ax
            mov ch, 1
            mov al, 0FFh
            out IMR, al
            mov al, EOI
            out EOI, al
            pop ax
            iret

PIO:        org 4000h
            push ax
            mov al, 1
            out CA, al
            mov al, 0
            out CB, al
            pop ax
            ret

STROBE0:    org 4500h
            push ax
            in al, PA
            and al, 11111101b
            out PA, al
            pop ax
            ret

STROBE1:    org 5000h
            push ax
            in al, PA
            or al, 00000010b
            out PA, al
            pop ax
            ret
```

```
POLL:    org 5500h
          push ax
          in al, PA
          and al, 1
          jnz POLL
          pop ax
          ret

PIC:      org 6000h
          push ax
          mov al, 0FEh
          out IMR, al
          mov al, N_F10
          out INT0, al
          pop ax
          ret

          org 2000h
          cli
          call PIO
          call STROBE0
          call PIC
          sti
          mov bx, offset CADENA
          mov cl, 0
          mov ch, 0
LAZO1:    int 6
          inc bx
          inc cl
          cmp ch, 1
          jnz LAZO1
          mov bx, offset CADENA
LAZO2:    call POLL
          mov al, [bx]
          out PB, al
          call STROBE1
          call STROBE0
          inc bx
          dec cl
          jnz LAZO2
          int 0
          end
```

Ejercicio 3: Uso de la impresora a través del HAND-SHAKE.

Ejecutar los programas configurando el simulador VonSim con los dispositivos “Impresora (Handshake)”.

(a) *Escribir un programa que imprima “INGENIERÍA E INFORMÁTICA” en la impresora a través del HAND-SHAKE. La comunicación se establece por consulta de estado (polling). ¿Qué diferencias se encuentran con el Ejercicio 2b?*

```

DATO EQU 40h
ESTADO EQU 41h

org 1000h
MSJ DB “INGENIERÍA E INFORMÁTICA”
FIN DB ?

org 2000h
mov bx, offset MSJ
mov cl, offset FIN - offset MSJ
POLL: in al, ESTADO
      and al, 1
      jnz POLL
      mov al, [bx]
      out DATO, al
      inc bx
      dec cl
      jnz POLL
      int 0
      end

```

Las diferencias que se encuentran con el Ejercicio 2b son que no es necesario configurar el PIO ni tampoco es necesario configurar las señales de strobe.

(b) *¿Cuál es la ventaja en utilizar el HAND-SHAKE con respecto al PIO para comunicarse con la impresora? Sacando eso de lado, ¿qué ventajas tiene el PIO, en general, con respecto al HAND-SHAKE?*

La ventaja en utilizar el HAND-SHAKE con respecto al PIO para comunicarse con la impresora es que manda señal de strobe automáticamente. Sacando eso de lado, las ventajas que tiene el PIO, en general, con respecto al HAND-SHAKE, es que sirve para comunicarse con otros dispositivos.

(c) *Escribir un programa que imprime “UNIVERSIDAD NACIONAL DE LA PLATA” en la impresora a través del HAND-SHAKE. La comunicación se establece por interrupciones emitidas desde el HAND-SHAKE cada vez que la impresora se desocupa.*

```
EOI EQU 20h
IMR EQU 21h
INT2 EQU 26h
DATO EQU 40h
ESTADO EQU 41h
N_HSK EQU 10

org 40
IP_HSK DW RUT_HSK

org 1000h
MSJ DB "UNIVERSIDAD NACIONAL DE LA PLATA"
FIN_MSJ DB ?

RUT_HSK: org 3000h
          push ax
          mov al, [bx]
          out DATO, al
          inc bx
          dec cl
          cmp cl, 0
          jnz FIN
          mov al, 0FFh
          out IMR, al
          in al, ESTADO
          and al, 0111111b
          out ESTADO, al
FIN:      mov al, EOI
          out EOI, al
          pop ax
          iret

org 2000h
cli
mov al, 0FBh
out IMR, al
mov al, N_HSK
out INT2, al
in al, ESTADO
or al, 10000000b
out ESTADO, al
mov bx, offset MSJ
mov cl, offset FIN_MSJ - offset MSJ
sti
LAZO:    cmp cl, 0
          jnz LAZO
          int 0
          end
```

(d) *Escribir un programa que solicite el ingreso de cinco caracteres por teclado y los almacene en memoria. Una vez ingresados, que los envíe a la impresora a través del HAND-SHAKE, en primer lugar tal cual fueron ingresados y, a continuación, en sentido inverso. Utilizar el HAND-SHAKE en modo consulta de estado. ¿Qué diferencias se encuentran con el Ejercicio 2c?*

```

DATO EQU 40h
ESTADO EQU 41h

org 1000h
CHARS DB 5
CADENA DB ?,?,?,?,?
FIN DB ?

org 2000h
mov bx, offset CADENA
mov cl, CHARS
LAZO:  int 6
      inc bx
      dec cl
      cmp cl, 0
      jnz LAZO
      mov bx, offset CADENA
      mov cl, offset FIN - offset CADENA
POLL1: in al, ESTADO
      and al, 1
      jnz POLL1
      mov al, [bx]
      out DATO, al
      inc bx
      dec cl
      jnz POLL1
      mov bx, offset CADENA+4
      mov cl, offset FIN - offset CADENA
POLL2: in al, ESTADO
      and al, 1
      jnz POLL2
      mov al, [bx]
      out DATO, al
      dec bx
      dec cl
      jnz POLL2
      int 0
      end

```

Las diferencias que se encuentran con el Ejercicio 2c son que los cinco caracteres son enviados a la impresora todos a la vez y no de a uno por vez.

(e) *Idem (d), pero, ahora, utilizar el HAND-SHAKE en modo interrupciones.*

```

EOI EQU 20h
IMR EQU 21h
INT2 EQU 26h
DATO EQU 40h
ESTADO EQU 41h
N_HSK EQU 10

org 40
IP_HSK DW RUT_HSK

org 1000h
CADENA DB ?,?,?,?,?
FIN_CADENA DB ?

RUT_HSK:  org 3000h
          push ax
          mov al, [bx]
          out DATO, al
          dec ch
          cmp ch, 6
          js DESC
          inc bx
          jmp SEGUIR
DESC:     cmp ch, 5
          jz SEGUIR
          dec bx
SEGUIR:   dec cl
          cmp cl, 0
          jnz FIN
          mov al, 0FFh
          out IMR, al
          in al, ESTADO
          and al, 0111111b
          out ESTADO, al
FIN:      mov al, EOI
          out EOI, al
          pop ax
          iret

org 2000h
cli
mov al, 0FBh
out IMR, al
mov al, N_HSK
out INT2, al
in al, ESTADO

```

```
    or al, 10000000b
    out ESTADO, al
    mov bx, offset CADENA
    mov cl, offset FIN_CADENA - offset CADENA
LAZO1: int 6
       inc bx
       dec cl
       cmp cl, 0
       jnz LAZO1
       mov bx, offset CADENA
       mov cl, 10
       mov ch, 10
       sti
LAZO2: cmp cl, 0
       jnz LAZO2
       int 0
       end
```


Ejercicio 4: Uso de la impresora a través del dispositivo USART por consulta de estado.

Ejecutar utilizando el simulador MSX88 (versión antigua del VonSim) en configuración P1 C4 y utilizar el comando PI que corresponda en cada caso (ver uso de Comando PI en el simulador).

(a) *Escribir un programa que imprima el carácter “A” en la impresora a través de la USART usando el protocolo DTR . La comunicación es por consulta de estado.*

(b) *Escribir un programa que imprima la cadena “USART DTR POLLING” en la impresora a través de la USART usando el protocolo DTR. La comunicación es por consulta de estado.*

(c) *Escribir un programa que imprima la cadena “USART XON/XOFF POLLING” en la impresora a través de la USART usando el protocolo XON/XOFF realizando la comunicación entre CPU y USART por consulta de estado.*

Ejercicio 5: DMA (Transferencia de datos memoria-memoria).

Programa que copia una cadena de caracteres almacenada a partir de la dirección 1000H en otra parte de la memoria, utilizando el CDMA en modo de transferencia por bloque. La cadena original se debe mostrar en la pantalla de comandos antes de la transferencia. Una vez finalizada, se debe visualizar en la pantalla la cadena copiada para verificar el resultado de la operación. Ejecutar el programa en la configuración P1 C3.

- (a) Analizar, minuciosamente, cada línea del programa anterior.*
- (b) Explicar qué función cumple cada registro del CDMA e indicar su dirección.*
- (c) Describir el significado de los bits del registro CTRL.*
- (d) ¿Qué diferencia hay entre transferencia de datos por bloque y bajo demanda?*
- (e) ¿Cómo se le indica al CDMA desde el programa que debe arrancar la transferencia de datos?*
- (f) ¿Qué le indica el CDMA a la CPU a través de la línea hrq? ¿Qué significa la respuesta que le envía la CPU a través de la línea hlra?*

(g) *Explicar, detalladamente, cada paso de la operación de transferencia de un byte desde una celda a otra de la memoria. Verificar que, en esta operación, intervienen el bus de direcciones, el bus de datos y las líneas mrd y mwr.*

(h) *¿Qué sucede con los registros RF, CONT y RD del CDMA después de transferido un byte?*

(i) *¿Qué evento hace que el CDMA emita una interrupción y a través de qué línea de control lo hace?*

(j) *¿Cómo se configura el PIC para atender la interrupción del CDMA?*

(k) *¿Qué hace la rutina de interrupción del CDMA del programa anterior?*

Ejercicio 6: DMA (Transferencia de datos memoria-periférico).

Programa que transfiere datos desde la memoria hacia la impresora sin intervención de la CPU, utilizando el CDMA en modo de transferencia bajo demanda.

(a) *Analizar, minuciosamente, cada línea del programa anterior.*

(b) *¿Qué debe suceder para que el HAND-SHAKE emita una interrupción al CDMA?*

(c) *¿Cómo demanda el periférico, en este caso el HAND-SHAKE, la transferencia de datos desde memoria? ¿A través de qué líneas se comunican con el CDMA ante cada pedido?*

(d) *Explicar, detalladamente, cada paso de la operación de transferencia de un byte desde una celda de memoria hacia el HAND-SHAKE y la impresora.*

(e) *¿Qué evento hace que el CDMA emita una interrupción al PIC?*

(f) *¿Cuándo finaliza la ejecución del LAZO?*

Ejercicio 7: Configuración del CDMA.

Indicar cómo configurar el registro Control del CDMA para las siguientes transferencias:

(a) *Transferencia Memoria \rightarrow Memoria, por robo de ciclo.*

(b) *Transferencia Periférico \rightarrow Memoria, por ráfagas.*

(c) *Transferencia Memoria \rightarrow Periférico, por robo de ciclo.*