

# PRÁCTICA 1 - RESOLUCIÓN

## Parte 1: Repaso de VonSim

1. **Contar letras** ★ Escribir un programa que dado un string llamado MENSAJE, almacenado en la memoria de datos, cuente la cantidad de veces que la letra 'a' (minúscula) aparece en MENSAJE y lo almacene en la variable CANT. Por ejemplo, si MENSAJE contiene "Hola, Buenas Tardes", entonces CANT debe valer 3.

```
; Ejercicio 1

MENSAJE      ORG 1000H
              DB  "Hola, Buen Dia"
LETRA        DB  "a"
CANT         DB  ?

              ORG 2000H
              MOV AL, LETRA
              MOV CH, 0
              MOV CL, OFFSET LETRA - OFFSET MENSAJE
              MOV BX, OFFSET MENSAJE

SIGO:        MOV AH, [BX]
              CMP AH, AL
              JNZ SALTAR
              INC CH

SALTAR:      INC BX
              DEC CL
              JNZ SIGO
              MOV CANT, CH
              HLT

END
```

2. **Es mayúscula** ★ Escribir un programa que determine si un carácter (un string de longitud 1) es una letra mayúscula. El carácter está almacenado en la variable C, y el resultado se guarda en la variable RES de 8 bits. Si C es mayúscula, debe almacenarse el valor 0FFh en RES; de lo contrario, debe almacenarse 0. **Pista:** Los códigos de las mayúsculas son todos consecutivos. Buscar en la tabla ASCII los caracteres mayúscula, y observar qué valores ASCII tienen la 'A' y la 'Z'.

```
; Ejercicio 2

C            ORG 1000H
              DB  "G"
RES         DB  ?

              ORG 2000H
              MOV AL, 41h ; "A"
              MOV AH, 5Ah ; "Z"
              CMP C, AL
              JS NO_ES
              CMP AH, C
              JS NO_ES
              MOV RES, 0FFh
              JMP FIN

NO_ES:      MOV RES, 0
FIN:        HLT
END
```

3. **Convertir carácter a minúscula** ★ Escribir un programa que convierta un carácter de mayúsculas a minúsculas. El carácter está almacenado en la variable C. Asumir que el carácter es una mayúscula. **Pista:** Las mayúsculas y las minúsculas están en el mismo orden en el ASCII, y por ende la distancia entre, por ejemplo, la “A” y la “a” es la misma que la distancia entre la “Z” y la “z”.

```
; Ejercicio 3
C          ORG 1000H
           DB  "G"          ; Tiene una letra mayuscula
RES        DB  ?
           ORG 2000H
           MOV AL, C
           ADD AL, 20h      ; Distancia entre may. y min.
           MOV C, AL
           HLT
END
```

4. **Convertir string a minúscula** ★ Escribir un programa que convierta todos los caracteres de un string MENSAJE a minúscula. Por ejemplo, si MENSAJE contiene “Hola, Buenas Tardes”, luego de ejecutar el programa debe contener “hola, buenas tardes”.

```
; Ejercicio 4
MENSAJE    ORG 1000H
           DB  "Hola, Buen Dia"
FIN        DB  ?
           ORG 2000H
           MOV BX, OFFSET MENSAJE
           MOV CL, OFFSET FIN - OFFSET MENSAJE
           MOV AL, 41h      ; "A"
           MOV AH, 5Ah      ; "Z"
SIGO:      MOV CH, [BX]
           CMP CH, AL
           JS NO_ES
           CMP AH, CH
           JS NO_ES
           ADD BYTE PTR [BX], 20h
NO_ES:     INC BX
           DEC CL
           JNZ SIGO
           HLT
END
```

## Parte 2: Entrada/Salida con Interrupciones por Software

### 1) Mostrar mensajes en la pantalla ★

- Imprime: ARQUITECTURA DE COMPUTADORAS-FACULTAD DE INFORMATICA-UNLP
- Porque en memoria están esos cuatro caracteres: están inicializados con su código ASCII en lugar de con el carácter correspondiente.
- BX: dirección de comienzo del mensaje a mostrar. AL: cantidad de caracteres.

### 2) Lectura de datos desde el teclado ★

- BX: dirección donde se almacenará el carácter leído desde teclado (recordar que se lee de a un solo carácter)
- La segunda interrupción INT muestra el carácter recientemente leído desde teclado.
- En CL queda almacenado el código ASCII del carácter leído. Se trata del carácter correspondiente al dígito ingresado desde teclado, por ej., si se ingresó "5" queda el valor 35h (código ASCII del "5").

### 3) Errores comunes al mostrar y leer caracteres ★

De izquierda a derecha, y de arriba abajo:

- Debería ser mov al, 5 → el espacio cuenta
- Debería ser mov bx, offset A → INT 7 necesita la dirección
- Debería ser mov al, offset B - offset A → debe quedar la cantidad de caracteres
- Debería ser mov bx, offset A → para que el carácter leído quede en la dirección correcta
- Debería poner mov bx, offset A antes que el INT 6 → para que el carácter leído quede en la dirección correcta
- Debería ser mov bx, offset A → para que el carácter leído quede en la dirección correcta y luego se pueda mostrar en pantalla

<b>ORG 1000H</b>	<b>ORG 2000H</b>
A DB "HO LA"	mov bx, offset A
B DB ?	mov al, 4
	int 7
	END

<b>ORG 1000H</b>	<b>ORG 2000H</b>
A DB "ARQ"	mov al, 3
B DB ?	mov bx, A
	int 7
	END

<b>ORG 1000H</b>	<b>ORG 2000H</b>
A DB "HOLA"	mov al, offset A - offset B
B DB ?	mov bx, offset A
	int 7
	END

<b>ORG 1000H</b>	<b>ORG 2000H</b>
A DB ?	mov al, 3
	mov bx, A
	int 6
	END

<b>ORG 1000H</b>	<b>ORG 2000H</b>
A DB ?	int 6
	mov bx, offset A
	END

<b>ORG 1000H</b>	<b>ORG 2000H</b>
A DB ?	mov bx, A
	int 6
	mov al, 1
	int 7
	END

### 4) Mostrar caracteres individuales ★

- Escribir un programa que muestre en pantalla las letras mayúsculas ("A" a la "Z").  
**Pista:** Podés buscar los códigos de la "A" y la "Z" en una [tabla de códigos ascii](#). No utilizar un vector. Usar una sola variable de tipo **db**, e incrementar el valor de esa variable antes de imprimir.
- ¿Qué deberías modificar en a) para mostrar los dígitos ("0" al "9")? ¿Y para mostrar todos los caracteres disponibles en el código ASCII? Probar en el simulador.
- Modificar el ejercicio b) que muestra los dígitos, para que cada dígito se muestre en una línea separada.  
**Pista:** El código ASCII del carácter de *nueva línea* es el 10, comúnmente llamado "\n" o LF ("line feed" por sus siglas en inglés y porque se usaba en impresoras donde había que "alimentar" una nueva línea).

```
; Ejercicio 4-a
LETRA      ORG 1000H
           DB  "A"
           ORG 2000H
           MOV BX, OFFSET LETRA
           MOV AL, 1
MOSTRAR:   INT 7
           CMP LETRA, 5Ah      ; "Z"
           JZ  FIN
           INC LETRA
           JMP MOSTRAR
FIN:       HLT
END
```

```
; Ejercicio 4-b1
LETRA      ORG 1000H
           DB  "0"
           ORG 2000H
           MOV BX, OFFSET LETRA
           MOV AL, 1
MOSTRAR:   INT 7
           CMP LETRA, 39h      ; "9"
           JZ  FIN
           INC LETRA
           JMP MOSTRAR
FIN:       HLT
END
```

```
; Ejercicio 4-b2
LETRA      ORG 1000H
           DB  1
           ORG 2000H
           MOV BX, OFFSET LETRA
           MOV AL, 1
MOSTRAR:   INT 7
           CMP LETRA, 255      ; ultimo codigo
           JZ  FIN
           INC LETRA
           JMP MOSTRAR
FIN:       HLT
END
```

```
; Ejercicio 4-c
LETRA      ORG 1000H
           DB  "0"
ENTER      DB  10
           ORG 2000H
           MOV AL, 1
MOSTRAR:   MOV BX, OFFSET LETRA
           INT 7
           MOV BX, OFFSET ENTER
           INT 7
           CMP LETRA, 39h      ; "9"
           JZ  FIN
           INC LETRA
           JMP MOSTRAR
FIN:       HLT
END
```

## 5) Acceso con contraseña ★★

Escribir un programa que solicite el ingreso de una contraseña de 4 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 mostrar el mensaje "Acceso denegado", y volver a pedir que se ingrese una contraseña. Al 5to intento fallido, debe mostrarse el mensaje "Acceso BLOQUEADO" y terminar el programa.

```

; Ejercicio 5

INGRESO      DB "Ingrese clave:"
ACC_OK       DB "Acceso permitido"
ACC_FAIL     DB "Acceso denegado"
ACC_OUT      DB "Acceso BLOQUEADO"
ERRORES      DB 0
CLAVE        DB "pass"
CLAVEING     DB ?

ORG 2000H
INICIO:      MOV BX, OFFSET INGRESO
              MOV AL, OFFSET ACC_OK - OFFSET INGRESO
              INT 7
              MOV CL, 4
              MOV BX, OFFSET CLAVEING
LEER:        INT 6
              INC BX
              DEC CL
              JNZ LEER
              MOV DX, 3
COMP:        MOV BX, OFFSET CLAVE
              ADD BX, DX
              MOV AL, [BX]
              MOV BX, OFFSET CLAVEING
              ADD BX, DX
              CMP [BX], AL
              JNZ ERROR
              DEC DX
              JNZ COMP
              MOV BX, OFFSET ACC_OK
              MOV AL, OFFSET ACC_FAIL - OFFSET ACC_OK
              INT 7
              JMP FINAL
ERROR:       INC ERRORES
              CMP ERRORES, 5
              JZ ECHAR
              MOV BX, OFFSET ACC_FAIL
              MOV AL, OFFSET ACC_OUT - OFFSET ACC_FAIL
              INT 7
              JMP INICIO
ECHAR:       MOV BX, OFFSET ACC_OUT
              MOV AL, OFFSET ERRORES - OFFSET ACC_OUT
              INT 7
FINAL:      HLT
END

```

## Parte 3: Pila, subrutinas y dirección de retorno.

### 1) Repaso de Conceptos de Pila y Subrutinas

A) **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

B) **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	<b>org 3000h</b>	-----
2	rutina: mov bx,3	7FFCh
3	ret	7FFEh
4	<b>org 2000h</b>	-----
5	push ax	7FFEh
6	call rutina	7FFCh
7	pop bx	8000h
8	hlt	8000h
9	<b>end</b>	-----

### C) Llamadas a subrutinas y dirección de retorno ★

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

- Al ejecutarse la instrucción **call rut**, se guarda el IP de retorno (el de la siguiente instrucción al call) en la pila y se pone como IP actual la dirección de comienzo de la rutina llamada.
- Acciones al ejecutarse la instrucción **ret**, se desapila un elemento y se pone como IP actual. Por eso se debe estar seguro de que se esta apuntando al IP de retorno almacenado al hacer el CALL.

<b>org 3000h</b>	<b>org 2000h</b>
rut: mov bx,3 ; Dirección 3000h	call rut ; Dirección 2000h
<b>ret</b> ; Dirección 3002h	add cx,5 ; Dirección 2002h
	call rut ; Dirección 2004h
	hlt ; Dirección 2006h
	<b>end</b>

#	Instrucción	Valor SP	Contenido Pila
1	<b>org 3000h</b>	-----	
2	rutina: mov bx,3	7FFEh	2002h
3	ret	8000h	?
4	<b>org 2000h</b>	-----	
5	call rut	7FFEh	2002h
6	add cx, 5	8000h	?
7	call rut	7FFEh	2006h
8	hlt	8000h	?
9	<b>end</b>	-----	

## Parte 4: Pasaje de parámetros

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

	Código	A través de		Por	
		Registro	Pila	Valor	Referencia
a)	mov ax, 5 call subrutina	✓		✓	
b)	mov dx, offset A call subrutina	✓			✓
c)	mov bx, 5 push bx call subrutina pop bx		✓	✓	
d)	mov cx, offset A push cx call subrutina pop cx		✓		✓
e)	mov dl, 5 call subrutina	✓		✓	
f)	call subrutina mov A, dx	✓		✓	

## 2) Pasaje de parámetros a través de registros y la pila ★

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

<b>org 1000h</b> A db 8 B db 5 C db 4 D db ?	<b>org 3000h</b> CALC: mov DL, AL add DL, AH sub DL, CL ret	<b>org 2000h</b> mov AL, A mov AH, B mov CL, C call CALC mov D, DL hlt <b>end</b>
--	---	--

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

<b>org 1000h</b> A db 8 B db 5 C db 4 D db ?	<b>org 3000h</b> CALC: push bx mov bx, sp add bx, 8 mov dl, [bx] sub bx, 2 add dl, [bx] sub bx, 2 sub dl, [bx] pop bx ret	<b>org 2000h</b> mov AL, A push AX mov AL, B push AX mov AL, C push AX call CALC mov D, DL pop AX pop AX pop AX hlt <b>end</b>
--	---	---





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

<b>org 1000h</b> A db 8 B db 5 C db 4 D db ?	<b>org 3000h</b> CALC: push bx mov bx, sp add bx, 8 mov bx, [bx] mov dl, [bx] mov bx, sp add bx, 6 mov bx, [bx] add dl, [bx] mov bx, sp add bx, 4 mov bx, [bx] sub dl, [bx] pop bx ret	<b>org 2000h</b> mov AX, offset A push AX mov AX, offset B push AX mov AX, offset C push AX call CALC mov D, DL pop AX pop AX pop AX hlt <b>end</b>
--	---	--

### 3) Primeras subrutinas

Reimplementar los programas del **Ejercicio 1 - Parte 1**, pero ahora implementando las siguientes subrutinas. En todos los casos, recibir los valores por parámetros pasados por registro, y devolver el resultado también **por valor y por registro**.

1.  **CONTAR\_CAR**  Recibe la dirección de comienzo de un string en BX, su longitud en AL, y el carácter a contar en AH. Retorna en CL la cantidad de veces que aparece el carácter.

; Ejercicio 3-1	
MENSAJE	ORG 1000H DB "Hola, Buen Dia"
LETRA	DB "a"
CANT	DB ?
CONTAR_CAR:	ORG 3000H MOV CL, 0
SIGO:	MOV CH, [BX] CMP CH, AH JNZ SALTAR INC CL
SALTAR:	INC BX DEC AL JNZ SIGO RET
	ORG 2000H MOV BX, OFFSET MENSAJE MOV AL, OFFSET LETRA - OFFSET MENSAJE MOV AH, LETRA CALL CONTAR_CAR MOV CANT, CL HLT
END	

2.  **ES\_MAYUS** ★ Recibe un carácter en el registro AL y retorna en AH el valor 0FFh si es mayúscula y 0 de lo contrario.

```
; Ejercicio 3-2
                                ORG 1000H
C                               DB  "G"
RES                             DB  ?
                                ORG 3000H
ES_MAYUS:                      MOV AH, 0
                                MOV CL, 41h ; "A"
                                MOV CH, 5Ah ; "Z"
                                CMP AL, CL
                                JS NO_ES
                                CMP CH, AL
                                JS NO_ES
                                MOV AH, 0FFh
NO_ES:                          RET
                                ORG 2000H
                                MOV AL, C
                                CALL ES_MAYUS
                                MOV RES, AH
                                HLT

END
```

3.  **A\_MINUS** ★ Recibe un carácter mayúscula en AL y lo devuelve como minúscula.

```
; Ejercicio 3-3
                                ORG 1000H
C                               DB  "G"                                ; Tiene una letra mayuscula
RES                             DB  ?
                                ORG 3000H
A_MINUS:                      ADD AL, 20h ; Distancia entre may. y min.
                                RET
                                ORG 2000H
                                MOV AL, C
                                CALL A_MINUS
                                MOV RES, AL
                                HLT

END
```

4. 📄 **STRING\_A\_MINUS** ★★ Recibe la dirección de comienzo de un string en BX, su longitud en AL. Recorre el string, cambiando a minúscula las letras que sean mayúsculas. No retorna nada, sino que modifica el string directamente en la memoria.

```
; Ejercicio 3-4
                                ORG 1000H
MENSAJE                        DB "Hola, Buen Dia"
FIN                            DB ?
                                ORG 3000H
STRING_A_MINUS:               MOV CL, 41h  ; "A"
                                MOV CH, 5Ah  ; "Z"
SIGO:                          MOV AH, [BX]
                                CMP AH, CL
                                JS NO_ES
                                CMP CH, AH
                                JS NO_ES
                                ADD BYTE PTR [BX], 20h
NO_ES:                          INC BX
                                DEC AL
                                JNZ SIGO
                                RET
                                ORG 2000H
                                MOV BX, OFFSET MENSAJE
                                MOV AL, OFFSET FIN - OFFSET MENSAJE
                                CALL STRING_A_MINUS
                                HLT
END
```

#### 4) Multiplicación de números sin signo con parámetros

Escribir un programa que tenga dos valores de 8 bits A y B y realice la multiplicación de A y B. El resultado se debe guardar en la variable RES de 16 bits, o sea que  $RES = A \times B$ . Para hacerlo, implementar una subrutina MUL:

:

- A. ★ Pasando los parámetros por **valor** desde el programa principal a través de los **registros AL y AH**, y devolviendo el resultado a través del **registro AX** por **valor**.

```
; Ejercicio 4-A
                                ORG 1000H
A                                DB 100
B                                DB 3
RES                              DW ?
                                ORG 3000H
MUL:                             PUSH CX
                                PUSH DX
                                CMP AL, 0
                                JZ VOLVER
                                CMP AH, 0
                                JZ VOLVER
                                MOV CH, 0
                                MOV CL, AL
                                MOV DX, 0
LOOP:                            ADD DX, CX
                                DEC AH
                                JNZ LOOP
VOLVER:                          MOV AX, DX
                                POP DX
                                POP CX
                                RET
                                ORG 2000H
                                MOV AL, A
                                MOV AH, B
                                CALL MUL
                                MOV RES, AX
                                HLT

END
```

- B. ★★ Pasando los parámetros por **referencia** desde el programa principal a través de **registros**, y devolviendo el resultado a través de un **registro** por **valor**.

```
; Ejercicio 4-B
                                ORG 1000H
A                                DB 100
B                                DB 3
RES                              DW ?
                                ORG 3000H
MUL:                             PUSH BX
                                PUSH CX
                                PUSH DX
                                MOV BX, SP
                                ADD BX, 10
                                MOV BX, [BX]
                                MOV AL, [BX]
                                MOV BX, SP
                                ADD BX, 8
                                MOV BX, [BX]
                                MOV AH, [BX]
                                MOV DX, 0
                                CMP AL, 0
                                JZ VOLVER
                                CMP AH, 0
                                JZ VOLVER
                                MOV CH, 0
                                MOV CL, AL
LOOP:                             ADD DX, CX
                                DEC AH
                                JNZ LOOP
VOLVER:                           MOV AX, DX
                                POP DX
                                POP CX
                                POP BX
                                RET
                                ORG 2000H
                                MOV BX, OFFSET A
                                PUSH BX
                                MOV BX, OFFSET B
                                PUSH BX
                                CALL MUL
                                POP BX
                                POP BX
                                MOV RES, AX
                                HLT
END
```

- C. ★★ Pasando los parámetros por **valor** desde el programa principal a través de **registros**, y devolviendo el resultado a través de un **registro** por **referencia**.

```
; Ejercicio 4-C
                                ORG 1000H
A                                DB 100
B                                DB 3
RES                              DW ?
                                ORG 3000H
MUL:                            PUSH CX
                                PUSH DX
                                CMP AL, 0
                                JZ VOLVER
                                CMP AH, 0
                                JZ VOLVER
                                MOV CH, 0
                                MOV CL, AL
                                MOV DX, 0
LOOP:                          ADD DX, CX
                                DEC AH
                                JNZ LOOP
VOLVER:                        MOV RES, DX
                                MOV AX, OFFSET RES ; en AX queda la dir. de donde esta el
                                resultado
                                POP DX
                                POP CX
                                RET
                                ORG 2000H
                                MOV AL, A
                                MOV AH, B
                                CALL MUL
                                HLT
END
```

- D. ★★ Pasando los parámetros por **valor** desde el programa principal a través de **la pila**, y devolviendo el resultado a través de un **registro** por **valor**.

```
; Ejercicio 4-D
                                ORG 1000H
A                                DB 100
B                                DB 3
RES                              DW ?
                                ORG 3000H
MUL:                             PUSH BX
                                PUSH CX
                                PUSH DX
                                MOV BX, SP
                                ADD BX, 10
                                MOV AL, [BX]
                                SUB BX, 2
                                MOV AH, [BX]
                                CMP AL, 0
                                JZ VOLVER
                                CMP AH, 0
                                JZ VOLVER
                                MOV CH, 0
                                MOV CL, AL
                                MOV DX, 0
LOOP:                            ADD DX, CX
                                DEC AH
                                JNZ LOOP
VOLVER:                          MOV AX, DX
                                POP DX
                                POP CX
                                POP BX
                                RET
                                ORG 2000H
                                MOV BL, A
                                PUSH BX
                                MOV BL, B
                                PUSH BX
                                CALL MUL
                                POP BX
                                POP BX
                                MOV RES, AX
                                HLT
END
```

- E. ★★ ★ Pasando los parámetros por **referencia** desde el programa principal a través de **la pila**, y devolviendo el resultado a través de un **registro** por **valor**.

```
; Ejercicio 4-E
                                ORG 1000H
A                               DB  100
B                               DB   3
RES                             DW   ?
                                ORG 3000H
MUL:                            PUSH BX
                                PUSH CX
                                PUSH DX
                                MOV BX, SP
                                ADD BX, 10
                                MOV BX, [BX]
                                MOV AL, [BX]
                                MOV BX, SP
                                ADD BX, 8
                                MOV BX, [BX]
                                MOV AH, [BX]
                                CMP AL, 0
                                JZ  VOLVER
                                CMP AH, 0
                                JZ  VOLVER
                                MOV CH, 0
                                MOV CL, AL
                                MOV DX, 0
LOOP:                          ADD DX, CX
                                DEC AH
                                JNZ LOOP
VOLVER:                        MOV AX, DX
                                POP DX
                                POP CX
                                POP BX
                                RET
                                ORG 2000H
                                MOV BX, OFFSET A
                                PUSH BX
                                MOV BX, OFFSET B
                                PUSH BX
                                CALL MUL
                                POP BX
                                POP BX
                                MOV RES, AX
                                HLT

END
```



## Parte 5: Ejercicios integradores o tipo parcial

### 1) Ahorcado secuencial ★★

Escribir un programa que permita a una persona desafiar a otra jugando al **ahorcado secuencial**. En el **ahorcado secuencial**, a diferencia del tradicional, hay que adivinar las letras en orden. Por ejemplo, si la palabra a adivinar es “alma”, la persona que adivina debe ingresar primero la “a”, luego la “l”, luego la “m” y finalmente debe ingresar nuevamente la “a”.

El programa tiene dos fases: primero, una persona carga la palabra a adivinar, y luego la otra persona adivina la palabra.

- **Fase 1:** Se debe mostrar el mensaje “Ingresa la palabra a adivinar: ”. Luego, se debe leer un string hasta que llegue el carácter “.”, y al terminar de leer, se debe mostrar el mensaje “Comenzá a adivinar!”.
- **Fase 2:** se deben leer caracteres hasta que la persona termine de adivinar todo el string, o se le acaben los intentos.

Si la persona ingresa un carácter que coincide con el que tenía que adivinar, se muestra ese carácter en pantalla, y se avanza al carácter siguiente del string a adivinar. De lo contrario, no se muestra nada, y la persona debe seguir intentando. Si adivinó todo el string, debe mostrarse el mensaje “Ganaste!”.

La persona tiene 50 intentos de letras para adivinar el string. Si se acaba la cantidad de intentos y no adivinó todo el string, debe mostrarse el mensaje “Perdiste, el string era S”, donde S es el string a adivinar completo.

```
; Ejercicio 1
                org 1000h
intentos        db 50
msjIngresar     db "Ingresa la palabra a adivinar: "
msjAdivinar     db "Comenzá a adivinar!"
msjGanaste      db "Ganaste!"
msjPerdiste     db "Perdiste, el string era "
string          db ?
                org 1200h
stringEval      db ?
```

```
org 2000h
MOV BX, OFFSET msjIngresar ; Mensaje Inicial
MOV AL, OFFSET msjAdivinar - OFFSET msjIngresar
INT 7
MOV DH, 0
MOV CL, 0
MOV BX, OFFSET string
ingreso: INT 6
MOV AL, BYTE PTR [BX]
INC BX
INC CL
CMP AL, '.'
JNZ ingreso
DEC CL ; Decremento caracter de corte
MOV BX, OFFSET msjAdivinar ; Mensaje de Adivinar
MOV AL, OFFSET msjGanaste - OFFSET msjAdivinar
INT 7
MOV CH, intentos ; Set de Intentos y string a Adivinar
MOV AX, OFFSET string
volver: MOV BX, OFFSET stringEval
INT 6
MOV DL, BYTE PTR [BX] ; Muevo car. a DL p/ evitar evaluar Mem a Mem
MOV BX, AX
CMP DL, BYTE PTR [BX] ; evalua car. p/ avanzar o descontar Intento
JZ seguir
DEC CH
JNZ volver
perdiste: MOV BX, OFFSET msjPerdiste ; Mensaje de Perdiste
MOV AL, OFFSET string - OFFSET msjPerdiste
ADD AL, CL
INT 7
JMP FIN
seguir: DEC CL ; Acierto Valido, sig. caracter a acertar e Impresion
MOV BX, AX
MOV AL, 1
INT 7
MOV AX, BX ; Recuperar puntero a String
INC AX
CMP CL, 0
JNZ volver
MOV BX, OFFSET msjGanaste ; Mensaje de Ganaste
MOV AL, OFFSET msjPerdiste - OFFSET msjGanaste
INT 7
FIN: HLT

END
```

## 2) Estadísticas de notas ★★ ★

Escribir un **programa** que permite calcular estadísticas de las notas de los exámenes de una materia. Las notas son valores entre 0 y 9, donde 4 es el valor mínimo para aprobar. El programa debe leer de teclado las notas y almacenarlas en un vector, convertidas a números; la lectura termina con el carácter “.”. Luego, el programa debe informar el promedio de las notas y almacenar en memoria el porcentaje de exámenes aprobados.

Para desarrollar el programa, implementar las subrutinas:

- **CANT\_APROBADOS:** Recibe un vector de números y su longitud, y retorna la cantidad de números iguales o mayores a 4.
- **DIV:** calcula el resultado de la división entre 2 números positivos A y B de 16 bits. Pasaje de parámetros por valor y por registro. Retorna el cociente y el resto en dos registros respectivamente.
- **MUL:** calcula el resultado de la multiplicación entre 2 números positivos A y B de 16 bits. Pasaje de parámetros por valor y por registro. Retorna el resultado en un registro.
- **PORCENTAJE:** Recibe la cantidad de notas aprobadas, y la cantidad total de notas, y retorna el porcentaje de aprobadas.

**Pista:** Como VonSim no tiene soporte para números en punto flotante, el porcentaje debe calcularse con enteros utilizando las subrutinas DIV y MUL. Es decir, si se leen 3 notas y 2 son aprobadas, el porcentaje de aprobados sería 66%, o sea  $(2 * 100)/3$ . Como son números enteros, es importante primero hacer la multiplicación y luego la división (¿por qué?).

```
; Ejercicio 2
                                ORG 1000H
MSJ_PRO    DB "El promedio de las notas es: "
PROMEDIO   DB ?
PORCEN     DB ?
APROBADOS  DB ?
VECTOR     DB ?

                                ORG 3000H
CANT_APROBADOS: PUSH AX ; resguardo valor
                                PUSH BX           ; resguardo valor
                                PUSH CX           ; resguardo valor
                                MOV DL, 0         ; Cant. de aprobados
LOOP:      MOV AL, [BX]
                                CMP AL, 4
                                JS SALTO
                                INC DL
SALTO:     INC BX
                                DEC CL
                                JNZ LOOP
                                POP CX
                                POP BX
                                POP AX
                                RET
```

```
; la subrutina DIV realiza: DX/CX
; se retorna el COCIENTE en BX y el RESTO en AX
                ORG 3200H
DIV:            PUSH CX
                PUSH DX
                MOV AX, 0    ; inicializo el resto en 0
                MOV BX, 0    ; inicializo el cociente de la división
                CMP CX, 0    ; CX tiene num B
                JZ FIN
                CMP DX, 0    ; DX tiene num A
                JZ FIN
OTRO:           SUB DX, CX
                JS RES        ; si negativo, voy a calcular el resto
                INC BX        ; sumo al cociente, es resultado de la DIV
                JMP OTRO
RES:            ADD CX, DX    ; sumo de vuelta CX para determinar el resto
                MOV AX, CX    ; devuelvo el resto en AX
FIN:            POP DX
                POP CX
                RET

; la subrutina MUL realiza: DX * AX
; se retorna el RESULTADO en DX
                ORG 3300H
MUL:            PUSH CX        ; resguardo valor
                CMP AX, 0
                JZ SALIR
                CMP DX, 0
                JZ SALIR
                MOV CX, 0
VUELVO:         ADD CX, AX
                DEC DX
                JNZ VUELVO
                MOV DX, CX    ; en DX tengo el resultado de la
multiplicacion
SALIR:          POP CX
                RET

; la subrutina realiza: (DX * 100)/CX y lo retorna en BX
                ORG 3500H
PORCENTAJE:     PUSH CX
                PUSH DX
                MOV AX, 100 ; para llamar a MUL
                CALL MUL
                CALL DIV
                POP DX
                POP CX
                RET
```

```

                                ORG 2000H
                                MOV DX, 0           ;Inicializo para ir sumando lo
elementos del vector
                                MOV CX, 0           ;Inicializo en 0 para llevar la
cantidad de elementos
                                MOV AX, 0           ; en AL voy a manejar cada caracter
leido
                                MOV BX, OFFSET VECTOR
LAZO:
                                INT 6
                                MOV AL, [BX]
                                CMP AL, '.'        ; comparo elem ingresado con '.'
                                JZ SUB1
                                SUB AL, 30H
                                MOV [BX], AL
                                ADD DX, AX          ; Lleva la suma para el promedio
                                INC CX              ; Cuenta cant elemento del vector
                                INC BX              ; Prox dir para el sgte elemento
                                JMP LAZO
SUB1:
                                CALL DIV            ; hago DX/CX
                                ADD BL, 30h         ; promedio:1 byte, lo paso a caracter
                                MOV PROMEDIO, BL
                                MOV BX, OFFSET MSJ_PRO
                                MOV AL, OFFSET PORCEN - OFFSET MSJ_PRO
                                INT 7               ; informo promedio
                                MOV BX, OFFSET VECTOR ; BX = dir. comienzo vector
                                CALL CANT_APROBADOS
                                MOV APROBADOS, DL
                                CALL PORCENTAJE      ; DX = cant aprobados, CX
= cant elementos
                                MOV PORCEN, BL
                                HLT
END
```

### 3) Estadísticas de texto ★★

Escribir un programa que permita calcular estadísticas básicas de texto, como su longitud, cantidad de vocales, etc. El programa debe leer un string de teclado. El string se almacena en la memoria principal con la etiqueta **CADENA**. La lectura termina cuando se lee el carácter “.”

Luego, calcular y almacenar en variables distintas: la cantidad de caracteres totales (sin contar “.”), la cantidad de letras, la cantidad de vocales y la cantidad de consonantes. Por último, verificar si la cadena contiene el carácter ‘x’.

Para ello, implementar las subrutinas:

- **ES\_LETRA** que recibe un carácter C por valor y retorna 0FFh si C es una letra o 00h de lo contrario. Para implementar la subrutina, tener en cuenta un carácter es una letra si es una minúscula o mayúscula.
- **ES\_VOCAL** que recibe un carácter C por valor y retorna 0FFh si C es vocal o 00h de lo contrario. Para implementar la subrutina, utilizar un string auxiliar que contiene las vocales, como **vocales db** “aeiouAEIOU”, y utilizar la subrutina **CONTIENE**.
- **CONTAR\_VOC** Usando la subrutina **ES\_VOCAL**, escribir la subrutina **CONTAR\_VOC**, que recibe una cadena terminada por referencia a través de un registro, su longitud en otro registro y devuelve, en un registro, la cantidad de vocales que tiene esa cadena.  
**Ejemplo:** CONTAR\_VOC de ‘contar!#!’ debe retornar 2
- **ES\_CONSONANTE** que recibe un carácter C por valor y retorna 0FFh si C es una letra consonante o 00h de lo contrario. Para implementar la subrutina, tener en cuenta que un carácter es consonante si es una letra pero no es una vocal.
- **CONTAR\_CONSONANTES** Idem **CONTAR\_VOC** pero para consonantes.
- **CONTIENE** que recibe un string A por referencia, y un carácter C por valor, ambos por registro, y debe retornar, también vía registro, el valor 0FFh si el string contiene a C o 00h en caso contrario.  
**Ejemplo:** CONTIENE de ‘a’ y “Hola” debe retornar 0FFh y CONTIENE de ‘b’ y “Hola” debe retornar 00h.

; Ejercicio 3