



PRÁCTICA 1

E/S con Interrupciones por Software Subrutinas con Pasaje de Parámetros

Objetivos: Realizar operaciones de Entrada/Salida de caracteres mediante Interrupciones por Software con el teclado y la pantalla. Comprender la utilidad de las subrutinas con parámetros y la comunicación con el programa principal a través de una pila. Escribir programas en el lenguaje assembly del simulador [VonSim](#). Ejecutarlos y verificar los resultados, analizando el flujo de información entre los distintos componentes del sistema.

Parte 1: Entrada/Salida con Interrupciones por Software

Instrucción	Operación
INT 0	Detiene el programa. Similar al HLT, pero permite que se terminen de ejecutar las interrupciones.
INT 3	Pausa la ejecución del simulador, y lo pone en modo de <i>depuración</i> para poder inspeccionar el programa.
INT 6	Lee un carácter de teclado. Para usarla previamente se debe guardar en BX la dirección de memoria en donde se almacenará el carácter leído. Luego de ejecutar INT 6, el carácter leído se encontrará en esa dirección de memoria.
INT 7	Imprime un string en pantalla. Previo a llamar a INT 7, se debe guardar en BX la dirección de comienzo del string a mostrar, y en AL la longitud o cantidad de caracteres del string.

1) Repaso de VonSim: Contar Letras ★

Escribir un programa que declare 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.

2) Mostrar mensajes en la pantalla de comandos ★

El siguiente programa del lenguaje assembler del simulador VonSim muestra en la pantalla de comandos un mensaje previamente almacenado en memoria de datos, aplicando la interrupción por software INT 7. Probar en el simulador.

<pre> ORG 1000H MSJ DB "ARQUITECTURA DE COMPUTADORAS-" DB "FACULTAD DE INFORMATICA-" DB 55H DB 4EH DB 4CH DB 50H FIN DB ? </pre>	<pre> ORG 2000H MOV BX, OFFSET MSJ MOV AL, OFFSET FIN - OFFSET MSJ INT 7 INT 0 END </pre>
--	---

- Ejecutar en el simulador ¿qué imprime?
- ¿Por qué imprime "UNLP" al final?
- Con referencia a la interrupción INT 7, ¿qué se almacena en los registros BX y AL?

- d) Modifique el programa para que solo imprima la primera parte del mensaje: "ARQUITECTURA DE COMPUTADORAS-"
- e) Modifique el programa para que solo imprima la primera letra, es decir, la "A"

3) Lectura de datos desde el teclado ★

El siguiente programa solicita el ingreso de un número (de un dígito) por teclado, lo almacena en la variable NUM e inmediatamente lo muestra en la pantalla de comandos, haciendo uso de las interrupciones por software INT 6 e INT 7. Probar en el simulador.

ORG 1000H MSJ DB "INGRESE UN NUMERO:" FIN DB ? ORG 1500H 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
--	---	---

- a) Con referencia a la interrupción INT 6, ¿qué se almacena en BX?
- b) En el programa anterior, ¿qué hace la segunda interrupción INT 7?
- c) ¿Qué valor queda almacenado en el registro CL? ¿Es el mismo que el valor numérico ingresado?
- d) Modificar el programa anterior para que lea dos dígitos en lugar de uno, con dos variables: NUM1 y NUM2.

4) Errores comunes al mostrar y leer caracteres ★

Los siguientes programas leen e imprimen caracteres. Indicar cuáles tienen errores y cómo solucionarlos.

ORG 1000H A DB "HO LA" B DB ?	ORG 2000H mov bx, offset A mov al, 4 int 7 int 0 END	ORG 1000H A DB "ARQ" B DB ?	ORG 2000H mov al, 3 mov bx, A int 7 int 0 END
ORG 1000H A DB "HOLA" B DB ?	ORG 2000H mov al, offset A - offset B mov bx, offset A int 7 int 0 END	ORG 1000H A DB ?	ORG 2000H mov al, 3 mov bx, A int 6 int 0 END
ORG 1000H A DB ?	ORG 2000H int 6 mov bx, offset A int 0 END	ORG 1000H A DB ?	ORG 2000H mov bx, A int 6 mov al, 1 int 7 int 0 END

5) Mostrar caracteres individuales ★

- a) 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 mostrar en pantalla.
- b) ¿Qué deberías modificar en a) para mostrar solamente los dígitos ("0" al "9")?
- c) ¿Y para mostrar todos los caracteres disponibles en el código ASCII? Probar en el simulador.
- d) 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). Entonces,

se puede imprimir el string “{c}\n”, que tiene longitud 2, donde {c} es el carácter a imprimir y \n es el carácter de nueva línea.

6) Lectura e impresión de strings ★★

- Escribir un programa que lea caracteres hasta que se ingrese el carácter “.” (punto). Contar la cantidad de caracteres ingresados y guardarla en la variable CANT.
- Modificar a) para contar solamente la cantidad de letras “a” que se leen
- Modificar a) para almacenar los caracteres leídos en memoria como un string (sin incluir el “.”).
- Modificar c) para mostrar en pantalla el string leído luego de leerlo.

Parte 2: 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			
2	mov bx,3			
3	push ax			
4	push ax			
5	push bx			
6	pop bx			
7	pop bx			
8	pop ax			

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.

#	Dirección	Instrucción	Valor del registro SP
1	———	org 3000h	———
2	3000h	rutina: mov bx,3	
3	3004h	ret	
4	———	org 2000h	———
5	2000h	push ax	
6	2001h	call rutina	
7	2004h	pop bx	
8	2005h	hlt	
9	———	end	———

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.

Además, explicar detalladamente:

- Las acciones que tienen lugar al ejecutarse la instrucción **call rut**
- Las acciones que tienen lugar al ejecutarse la instrucción **ret**

org 3000h	org 2000h
rut: mov bx,3 ; Dirección 3000h	call rut ; Dirección 2000h
ret ; Dirección 3004h	add cx,5 ; Dirección 2003h
	call rut ; Dirección 2007h
	int 0 ; Dirección 200Ah
	end

Parte 3: 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 por valor a través de registros AL, AH y CL, calcula $AL+AH-CL$, y devuelve el resultado por valor en DL.

org 1000h	org 3000h	org 2000h
A db 8	CALC: _____	_____
B db 5	add DL, AH	_____
C db 4	sub DL, CL	mov CL, C
D db ?	_____	_____
		mov D, DL
		int 0
		end

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.

org 1000h A db 8 B db 5 C db 4 D db ?	org 3000h CALC: push bx mov bx, sp _____ mov dl, [bx] sub bx, 2 _____ sub bx, 2 _____ pop bx _____ 	org 2000h mov AL, A push AX _____ push AX call CALC mov D, DL _____ _____ _____ int 0 end
--	---	--

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

3) Operaciones con Carácteres

Escribir un programa que lea un string, lo almacene en MENSAJE y convierta todos sus caracteres a minúscula. Por ejemplo, si MENSAJE contiene “Hola, Buenas Tardes”, luego de la conversión debe contener “hola, buenas tardes”. Mostrar en pantalla el mensaje luego de la conversión. Para ello, debe implementar y utilizar las siguientes subrutinas:

1. **ES_MAYUS** ★ Recibe un carácter en el registro AL y retorna en AH el valor 0FFh si es mayúscula y 0 de lo contrario. **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’.
2. **A_MINUS** ★ Recibe un carácter mayúscula en AL y lo devuelve como minúscula. **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”.
3. **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.

4) Multiplicación de números sin signo con parámetros

El pasaje de parámetros más usual suele ser por valor y por registro. No obstante, en algunas ocasiones también se utilizan pasajes de parámetros más avanzados que permiten más flexibilidad o eficiencia.

Escribir un programa que tenga dos valores de 8 bits A y B almacenados en su memoria 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**.
- 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**.
- 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**.
- 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**.
- 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**.

Parte 4: Ejercicios integradores o tipo parcial

1) 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. Para implementar el programa, deberá implementar y llamar a las siguientes subrutinas:

- LEER_CONTRA:** recibe una dirección de memoria y una longitud N, y lee una contraseña de N caracteres de teclado y la almacena en la dirección recibida
- COMPARAR_STRING:** recibe 2 direcciones de memoria, y una longitud N, y devuelve 0FFh si los strings son iguales, y 0 si son distintos
- MOSTRAR_MENSAJE:** recibe el resultado de COMPARAR_STRING, y la cantidad de intentos restantes. Imprime el mensaje correspondiente (Acceso permitido/denegado/BLOQUEADO)

2) 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 "Ingresá 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.

3) 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?).







4) 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 ‘contar1#!’ 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.