

UNIVERSIDAD DE GUADALAJARA
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍA
DIVISION DE ELECTRONICA Y COMPUTACION

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

SEMINARIO DE SOLUCIÓN DE PROBLEMAS DE TRADUCTORES DE LENGUAJE I (I7026)

Reporte de Actividad Práctica
(reporte de investigación, síntesis de lectura o cuestionario)

Práctica 5: "Funciones"

Alumno: Torres Rivera Rodrigo
Código: 397423431

Sección: D01

Profesor: Valentín Martínez López

Fecha 11 de Noviembre de 2025
firma de revisado

Nombre de la Práctica: Cálculo de potencia

Objetivo de la Práctica: El alumno desarrollará habilidades en el manejo de funciones de biblioteca proporcionadas por el entorno de programación emu8086, específicamente las funciones GET_STRING y PRINT_STRING, para la implementación de un programa en lenguaje ensamblador..

Antecedentes:

Se requiere contar con el siguiente material:

- Computadora con sistema operativo compatible.
- Emulador EMU8086 instalado.
- Incluir la librería “emu8086.inc”, para las macro funciones

Desarrollo:

Características y Requisitos:

- 1. Entrada de datos:**
 - El programa recibirá un número de 16 bits en formato hexadecimal, ingresado por el usuario a través del teclado utilizando la función `GET_STRING`.
 - 3.
- 2. Conversión y salida:**
 - El número hexadecimal será convertido a su equivalente en formato BCD 8421.
 - El resultado en formato decimal será impreso en pantalla utilizando la función `PRINT_STRING`.
 - 7.
- 3. Requisitos adicionales:**
 - El programa debe mostrar el nombre del alumno en pantalla.
 - Se deben proporcionar instrucciones claras en pantalla para guiar al usuario en la introducción de los datos.
 - Se debe utilizar la librería contenida en el archivo `emu8086.inc`, cuya documentación se encuentra en la sección 5 de los tutoriales del emu8086.
 - El programa debe finalizar utilizando la interrupción `20h`.
 - El registro `SP` (Stack Pointer) debe tener el valor `FFF8` al finalizar el programa, lo que indica que los procedimientos y el retorno al sistema operativo se han realizado correctamente.
 - No se permite el uso de la interrupción `21h`.
 - 15.
- 4. Entrega:**
 - Sube un archivo comprimido que contenga todos los archivos necesarios para ejecutar el programa, incluyendo el proyecto completo.

Resultados Obtenidos:

- Entrada:
 - Número de 16 bits escrito en Hexadecimal Big Endian
 - Dicho número debe de estar en mayúsculas
 - Nota el programa no puede manejar números en minúsculas
 - Resultado: dígitos decimales del número en formato LittleEndian

Entrada de los datos en el formato como en el ejemplo mostrado en el texto:

```

SFR emulator screen (80x25 chars)
Alumno: Rodrigo Torres Rivera
Escribe un numero de 16 bits en HEX usando mayusculas (ejemp FFF) FEDC_
  MSB   LSB

```

En la siguiente imagen veremos el flujo del programa para la impresión de los mensajes:

```

SFR emulator: practica.com...
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
Registers H L
AX 00 00
BX 00 00
CX 01 F8
DX 00 05
CS 0700
IP 0188
SS 0700
SP FFFC
BP 0000
SI 017E
DI 0181
DS 0700
ES 0700
0700:0188 50 080 P
07188: 50 080 V
07189: 56 086 V
0718A: 8A 130 V
0718B: 04 004 V
0718C: 3C 060 V
0718D: 00 000 NI
0718E: 74 116 V
0718F: 07 007 BI
07190: 46 070 F
07191: B4 180 I
07192: 00 014 C
07193: CD 205 =>
07194: 10 016 >
07195: EB 235 U
07196: F3 243 U
07197: SE 094 ...
0700:0188
PUSH AX
PUSH SI
MOV AL, [SI]
CMP AL, 00h
JZ 0197h
INC SI
MOV AH, 0Eh
INT 10h
JMP 018Ah
POP SI
POP AX
RET
...
07192: 00 014 C
07193: CD 205 =>
07194: 10 016 >
07195: EB 235 U
07196: F3 243 U
07197: SE 094 ...
0700:0188
DEFINITION STRING
DEFINITION GET STRING
;Subrutinas
Principal:
push ss
pop ds
Preguntar_datos msj1
lea si, msj0
call print_string
salto_de_linea
lea si, nuevlin
call print_string
Preguntar_datos msj1
lea si, msj1
call print_string
leer_los_datos_en_buffer
lea di, buffer
mov dx, tammax
call get_string
salto_de_linea
nuevlin
string
  
```

En el inciso (A) vemos que se ha impreso tres mensajes denotados en (C) msj0, nuevlin, msj1 antes de llamar a la Macro Instrucción get_string tras obtener el dato como muestra (A), vemos que es necesario imprimir un salto de línea como muestra (C), para eso llamamos en (B) la Macro Instrucción print_string la cual está definida en la linea: DEFINE_PRINT_STRING, podemos ver en (B) que esto lo que hace realmente es expandir un conjunto de instrucciones para imprimir un mensaje carácter a carácter utilizando INT 10H como muestra (B), la ventaja es que nos permite evitar el manejo de rutinas para tareas comunes como imprimir mensajes o pedir datos al usuario, el precio es que el código se hace más grande, ya que por cada uso de las Macro Funciones se está insertando código, a diferencia del uso de una Función (Subrutina), la cual mantiene el código pequeño pero se ejecuta más lento.

```

048 Convertir:
049 mov al, [si]
050
051 ;mov ah, 0Eh ;funcion 14 de int 10h
052 ;mov al, [si] ;hecho antes
053 sub al, 30h;
054 ;int 10h ;int imprimir un caracter
055 mov [si], al ;guardamos el valor convertido
056
057 inc si ;siguiente caracter
058 loop convertir ;CX decrementa y repite lazo
  
```

Con la cadena de caracteres a convertir ya en la variable buffer, nos centramos en la conversión de cada carácter a su formato binario, restando 30h en la línea 53, lo cual convierte un ASCII a su valor binario correspondiente, para acomodar el resultado del Formato Big Endian usamos la subrutina:

```

068 inc si
069 loop Invertir ;
070
071 mov si, offset buffer ;inicio del buffer
072 mov cx, bufferlong    ;inicializamos contador
073 sub ax, ax            ;AX = 0000_0000
074 Rellenar:             ;recuperaremos de la
075 pop ax                ;pila el buffer
076 mov [si], al          ;invertido ahora
077 inc si
078 loop Rellenar
079 ;-----Termina Convertir el dato del buffer a fc
080

```

La cual utiliza la pila para invertir el arreglo:

The screenshot shows two windows. On the left is a memory dump window titled 'stack' with 'size: byte' and 'elements: 4'. It displays the following data:

	hex	dec
MSJ0	41h	65
MSJ1	45h	69
MSJ2	0Dh	13
NUEVLIN	0Dh	13
BUFFER	16h, 15h, 14h, 13h	26, 25, 24, 13

On the right is a stack window titled 'stack' with 'size: byte' and 'elements: 4'. It displays the following data:

	hex	dec
0700	FFFE	0000
0700	FFFC	0016
0700	FFFA	0015
0700	FFF8	0014
0700	FFF6	0013 <
0700	FFF4	0700
0700	FFF2	0195
0700	FFF0	0700
0700	FFEE	01AD

Después de invertir el arreglo:

The screenshot shows two windows. On the left is a memory dump window titled 'stack' with 'size: byte' and 'elements: 4'. It displays the following data:

	hex	dec
MSJ0	41h	65
MSJ1	45h	69
MSJ2	0Dh	13
NUEVLIN	0Dh	13
BUFFER	13h, 14h, 15h, 16h	13, 14, 15, 16

On the right is a stack window titled 'stack' with 'size: byte' and 'elements: 4'. It displays the following data:

	hex	dec
0700	FFFE	0000 <
0700	FFFC	0016
0700	FFFA	0015
0700	FFF8	0014
0700	FFF6	0013
0700	FFF4	0700
0700	FFF2	0195
0700	FFF0	0700

Para mostrar los dígitos del número en decimal debemos convertir el valor binario de cada elemento del arreglo a su representación en ASCII:

```

file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
registers H L
AX 0E 35
BX 00 00
CX 00 01
DX 01 64
CS 0700
IP 025B
SS 0700
SP FFFE
BP 0000
SI 0184
DI 0181
DS 0700
ES 0700
0700:025F INT 010h
0725C: 0E 014 J 0725D: B0 176 JMP 002EBh
0725D: 00 002 SI CMP AX, 00045h
0725E: 20 032 SI JNE 0027Dh
0725F: CD 205 = MOV AH, 0Eh
07260: 10 016 > INT 010h
07261: E9 233 U MOV AH, 0Eh
07262: 87 135 C MOV AL, 031h
07263: 00 000 NI MOV AH, 0Eh
07264: 3D 061 E INT 010h
07265: 45 069 E MOV AH, 0Eh
07266: 00 000 NI MOV AL, 020h
07267: 75 117 U INT 010h
07268: 14 020 I JMP 002EBh
07269: B4 180 J CMP AX, 00044h
0726A: 0E 014 J ... 093 sub ax, ax
094 mov al, [si]
;mov al, [si] ;hecho antes
095 add al, 30h
096 num15:
097 cmp ax, 46h
098 jne num14: ← 1 } output
099 mov ah, 0Eh
100 mov al, 31h
101 int 10h ← 5
102 mov ah, 0Eh
103 mov al, 35h
104 int 10h
105 mov ah, 0Eh
106 mov al, 32h ;espacio en blanco
107 int 10h
108 jmp siguiente
109 num14:
110 cmp ax, 45h
111 jne num13:
112 mov ah, 0Eh
113 mov al, 31h
114 int 10h
115 mov ah, 0Eh
116 mov al, 34h
117 int 10h
118 mov ah, 0Eh
119 mov al, 32h ;espacio en blanco
120 int 10h
121 jmp siguiente
122 num13:
123 cmp ax, 44h
124 jne num12:
125 mov ah, 0Eh
126 mov al, 31h
127 int 10h ...
128

```

Alumno: Rodrigo Torres Rivera
Escribe un numero de 16 bits en HEX usando mayusculas Ejemp F

El numero escrito fue: 12 13 14 15 (A)

clear screen change font 0/16

BUFFER 13h, 14h, 15h, 16h

Para esto en (A) debemos leer el valor binario guardado en el Buffer y sumarle 30H, para así saber a cual carácter Hexadecimal nos estamos refiriendo ya que en Hexadecimal los dígitos ABCDEF son adicionales, y necesitan dos dígitos decimales para su representación, para estos 6 casos especiales, creamos una instrucción equivalente al SWICHT () del lenguaje C, verificamos si el dígito leído corresponde a alguno de estos dígitos hexadecimales y en caso de ser así imprimimos cada dígito decimal correspondiente seguido del espacio en blanco para separarlos.

Conclusiones:

Aunque muy cómodas las Macro Instrucciones aumentan el tamaño del código de nuestro programa, dependiendo de los requerimientos de la implementación del problema a resolver, este puede ser un factor a tomar en cuenta, por otro lado si no es problema el tamaño del código es una buena opción utilizarlas porque además de facilitar la legibilidad del código también se ejecutan más rápido que las subrutinas. Académicamente su uso puede ser controversial.