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 libreria "emu8086.inc", para las macro funciones

Desarrollo:

Características y Requisitos:

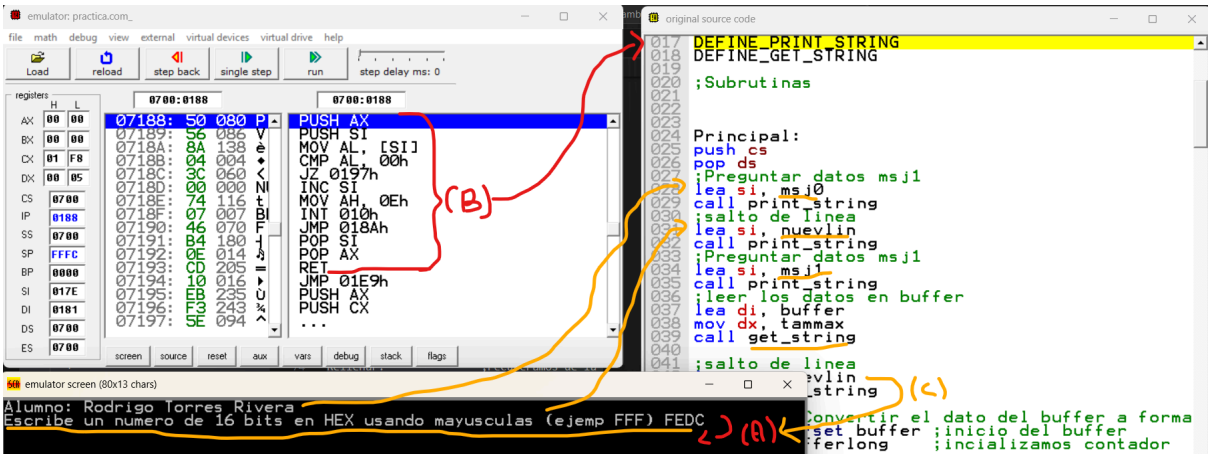
1. **1. Entrada de datos:**
2. - El programa recibirá un número de 16 bits en formato hexadecimal, ingresado por el usuario a través del teclado utilizando la función `GET_STRING`.
3.
4. **2. Conversión y salida:**
5. - El número hexadecimal será convertido a su equivalente en formato BCD 8421.
6. - El resultado en formato decimal será impreso en pantalla utilizando la función `PRINT_STRING`.
7.
8. **3. Requisitos adicionales:**
9. - El programa debe mostrar el nombre del alumno en pantalla.
10. - Se deben proporcionar instrucciones claras en pantalla para guiar al usuario en la introducción de los datos.
11. - Se debe utilizar la librería contenida en el archivo `emu8086.inc`, cuya documentación se encuentra en la sección 5 de los tutoriales del emu8086.
12. - El programa debe finalizar utilizando la interrupción `20h`.
13. - 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.
14. - No se permite el uso de la interrupción `21h`.
15.
16. **4. Entrega:**
17. - Sube un archivo comprimido que contenga todos los archivos necesarios para ejecutar el programa, incluyendo el proyecto completo.

Resultados Obtenidos:

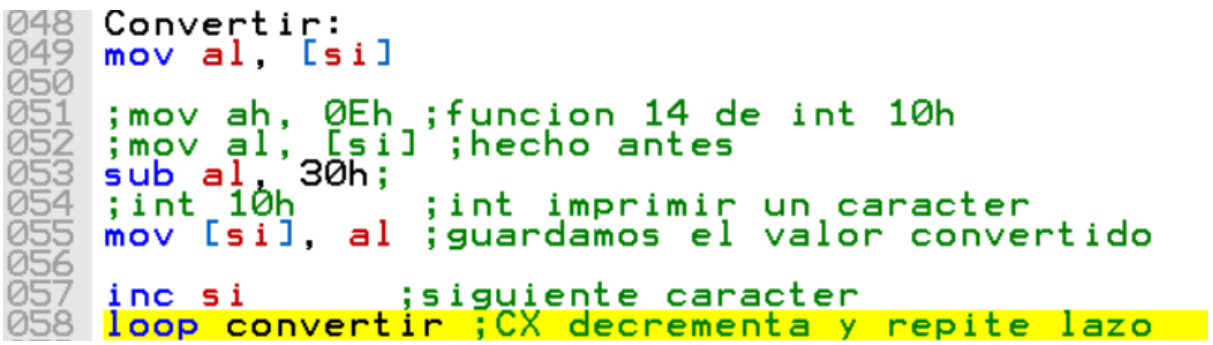
- **Entrada:**
  - Número de 16 bits escrito en Hexadecimal Big Endian
  - Dicho número debe de estar en mayúsculas
  - Nota el programa no puede manejar números en minúsculas
  - Resultado: dígitos decimales del número en formato Little Endian

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

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



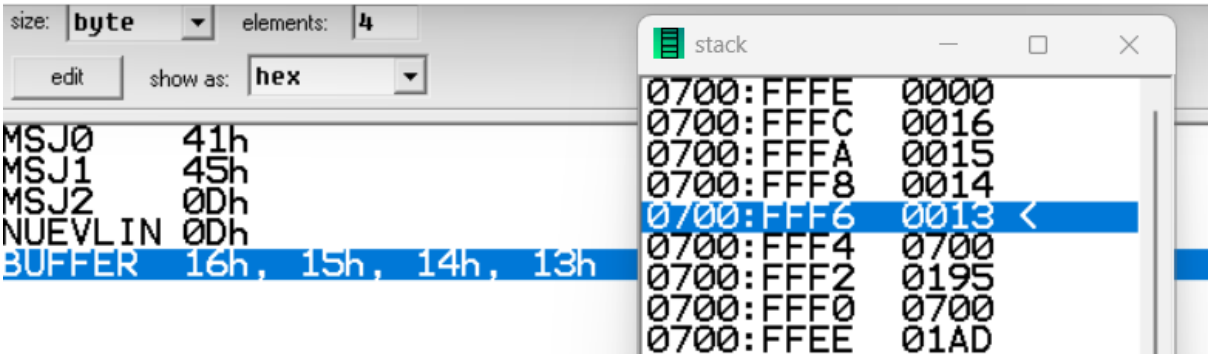
En el inciso (A) vemos que se ha impreso tres mensajes denotados en (C) msj0, nuevlin, msj1 antes de llamar a la Macro Instrucción get_string tras obtener el dato como muestra (A), vemos que es necesario imprimir un salto de línea como muestra (C), para eso llamamos en (B) la Macro Instrucción print_string 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 caracter a caracter 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.



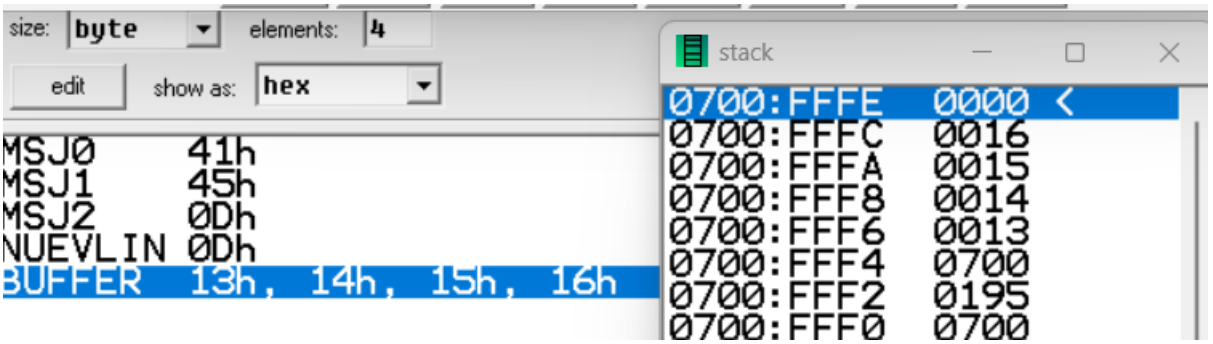
Con la cadena de caracteres a convertir ya en la variable buffer, nos centramos en la conversión de cada carácter a su formato binario, restando 30h en la línea 53, lo cual convierte un ASCII a su valor binario correspondiente, para acomodar el resultado del Formato Big Endian usamos la subrutina:

```
068 inc si
069 loop Invertir ;
070
071 mov si, offset buffer ;inicio del buffer
072 mov cx, bufferlong     ;incializamos contador
073 sub ax, ax             ;AX = 0000_0000
074 Rellenar:                 ;recuperamos 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
```

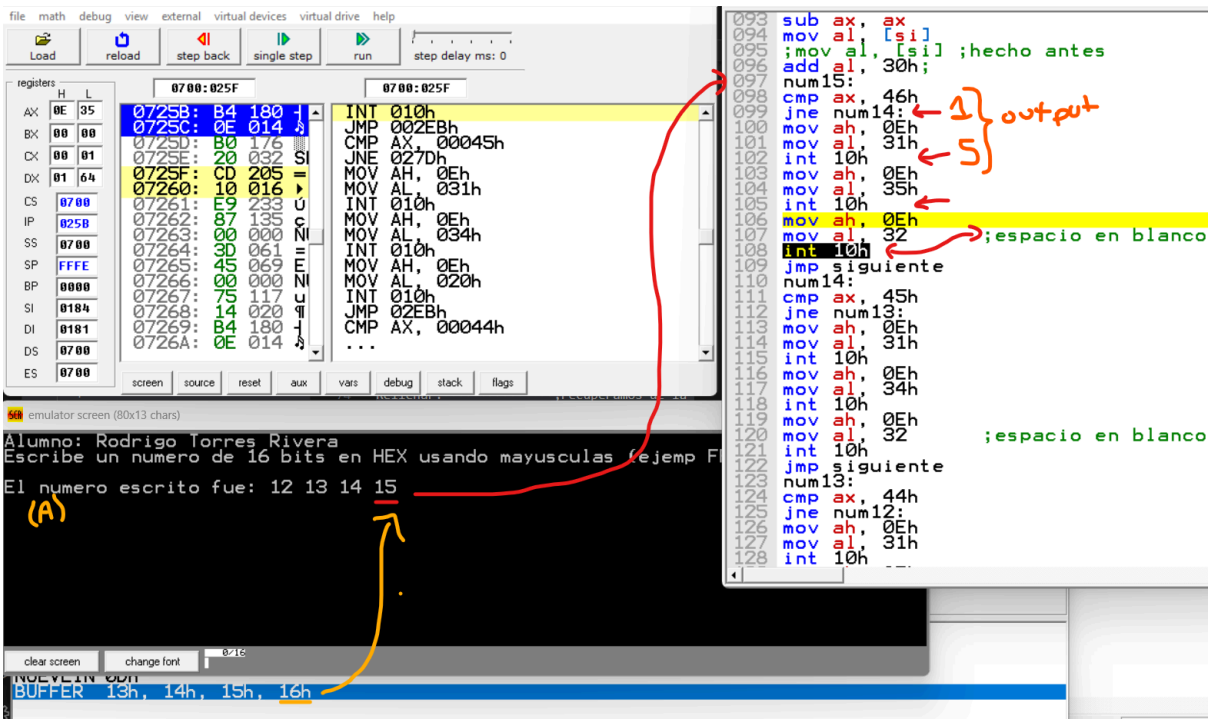
La cual utiliza la pila para invertir el arreglo:



Después de invertir el arreglo:



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



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 SWICTH () 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.