INSTITUTO TECNOLÓGICO DE COSTA RICA

SEDE SAN CARLOS

ESCUELA DE INGENIERÍA EN COMPUTACIÓN

CURSO DE FUNDAMENTOS DE LA ADMINISTRACIÓN DE COMPUTADORAS

PROYECTO PROGRAMADO ENSAMBLADOR

PROFESOR:

CARLOS CASTILLO VARGAS

ESTUDIANTES:

JONATHAN CALETH GUILLÉN CAMPOS

HÉCTOR GERARDO MEJÍAS CASCANTE

FECHA:

09/06/2013

INTRODUCCIÓN

El presente proyecto fue realizado con el objetivo de poner a prueba los conocimientos adquiridos en el curso de Fundamentos de la Administración de Computadoras sobre *lenguaje ensamblador*, así como para fomentar la investigación más detallada de los estudiantes a cargo.

DESCRIPCIÓN DEL PROBLEMA

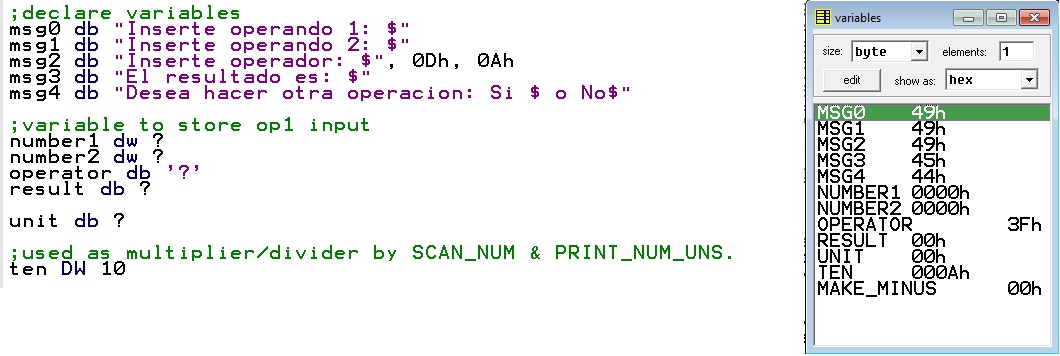
Para la elaboración de este proyecto, se ha asignado la elaboración de una aplicación que sirva como una calculadora básica con las siguientes especificaciones:

1. La aplicación será de tipo COM
2. Al ser ejecutada deberá indicar un mensaje solicitando el operando 1, luego el usuario ingresa el operando 1 y pulsa ENTER; seguidamente la aplicación le mostrará otro mensaje solicitando el operando 2, y el usuario ingresa el operando 2 y pulse ENTER. Luego la aplicación mostrar un mensaje solicitando el tipo de operador a aplicar a los 2 operandos recibidos, el usuario ingresa 1 de los posibles signos de los operadores (+, -, \*, /) seguido de la tecla ENTER y aplicará el operador ingresado al operando 1 contra el operando 2. El resultado es desplegado en la pantalla con un mensaje seguido del valor resultante. Por ejemplo, si el usuario ingresa:
3. 8
4. 3
5. -
6. El resultado de la operación 8 - 3 = 5.
7. Vamos a ignorar operaciones que den resultados negativos, por ejemplo 3 - 8 = -5. Este tipos de operaciones no se van a probar.
8. El programa debe validar que si se esta esperando un operando, solamente se acepte dicho operando si es un número (positivo) de un máximo de 4 dígitos. Si el usuario ingresa una letra o un carácter especial y presiona ENTER la aplicación debe desplegar un mensaje de error y volver a pedir el operando.
9. El programa debe validar que si se esta esperando un operador, solamente se acepta dicho operador si es uno de los 4 operadores permitidos (+, -, \*, /). Si el usuario ingrese algo diferente a eso y presiona ENTER debe desplegar un mensaje de error y volver a pedir el operando.
10. Nombres de variables, etiquetas debe estar en inglés.
11. Instrucciones, Registros deben ir en mayúscula como lo es el estándar de ASM.
12. Los operandos se leerán como números decimales.
13. El código debe estar debidamente documentado y en inglés (explicar aquellas instrucciones que realizan alguna función específica)
14. El programa se debe finalizar en cualquier momento que el usuario pulse la tecla ESC.

DESCRIPCIÓN DE LA SOLUCIÓN

Con el fin de elaborar la aplicación requerida, se utilizó el emulador de procesador EMU8086. Se realizaron una serie de códigos y variables que se explican a continuación:

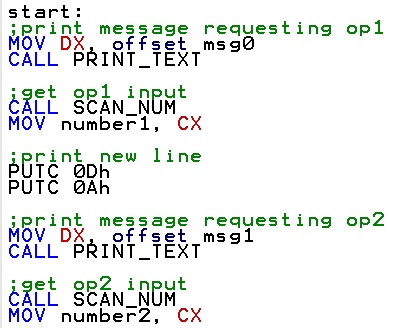
1. Variables:



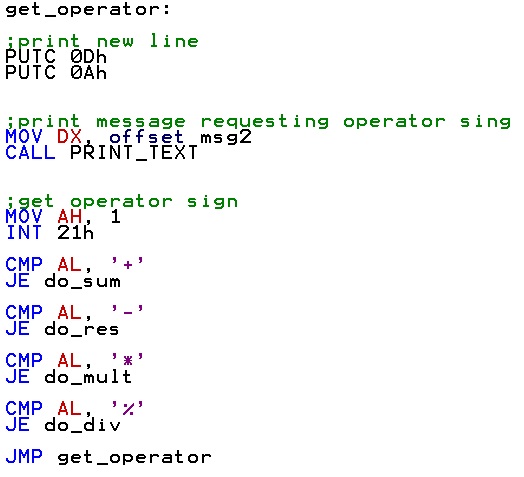
1. msg0:

Contiene el mensaje que solicita el primer operando de la operación.

1. msg1:  
   Contiene el mensaje que solicita el primer operando de la operación.
2. msg2:  
   Contiene el mensaje que solicita el operador.
3. msg3:  
   Contiene el mensaje que indica que el valor en esa línea es el resultado de la operación.
4. number1:  
   Variable no inicializada que es destinada a guardar el primer valor en una operación.
5. number2:  
   Variable no inicializada destinada a guardar el segundo valor en una operación.
6. operator:  
   Variable no inicializada donde se guardará el operando a utilizar.
7. result:  
   Variable no inicializada que almacenará el resultado a mostrar luego de efectuar una operación.
8. unit:  
   Variable no inicializada que es usada para almacenar las unidades si el valor de la respuesta es mayor a 10.
9. ten:  
   Variable usada en el macro SCAN\_NUM.
10. Etiquetas:
11. start:

  
Esta parte del código se encarga de la obtención de los operandos y del símbolo para la operación a realizar.  
Primero, imprime un mensaje mediante la llamada al macro PRINT\_TEXT solicitando el primer operando, luego se deberá presionar la tecla ENTER para salvarlo, el cual es guardado en la variable number1.  
Luego, imprime el siguiente mensaje utilizando el mismo macro solicitando el segundo operando y de la misma forma se deberá presionar ENTER para salvarlo, que se guardará en la variable number2.  
Después, el emulador continúa con la siguiente etiqueta.

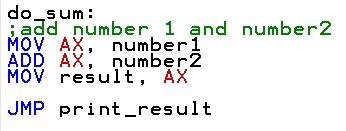
1. get\_operator:

  
Al ser ejecutado el código de esta etiqueta, se imprimirá en consola con el macro PRINT\_TEXT el mensaje para solicitar el operando, que determinará la operación a realizar entre los operandos.  
Después, se recurrirá a la interrupción 21H (AH=1) para capturar el símbolo u operando, que dependiendo de cuál sea, hará lo siguiente:

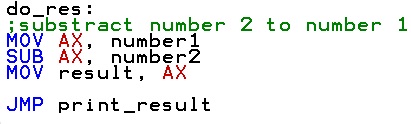
1. Si el operando es = ‘+’:  
   Se saltará a la etiqueta do\_sum.
2. Si el operando es = ‘-’:  
   Se saltará a la etiqueta do\_rest.
3. Si el operando es = ‘\*’:  
   Se saltará a la etiqueta do\_multiply.
4. Si el operando es = ‘%’:  
   Se saltará a la etiqueta do\_divide. (NOTA: Aunque en el enunciado del proyecto se indicaba que para efectuar una división era necesario digitar el símbolo ‘/’, en consola no se podía digitar debido a un asunto del teclado, por lo que se cambió al símbolo ‘%’, no obstante, modificando esa parte del código, se puede cambiar para que permita el símbolo ‘/’ para efectuar una división.)

Si el símbolo tecleado no equivale a ninguno de éstos, se ejecuta un salto a ésta misma etiqueta, es decir, se vuelve a pedir el signo.

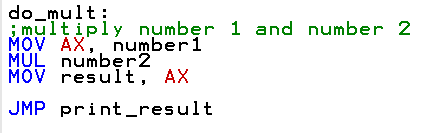
1. do\_sum:

  
Efectúa la suma de los operandos con la instrucción SUM y guarda el resultado en la variable result. Luego salta a la etiqueta print\_result.

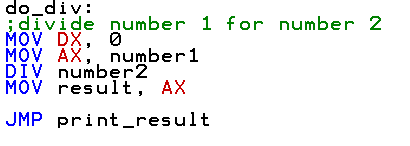
1. do\_res:

  
Efectúa la resta del operando 2 al operando 1 con la instrucción RES y después mueve el resultado a la variable result. Después salta a la etiqueta print\_result.

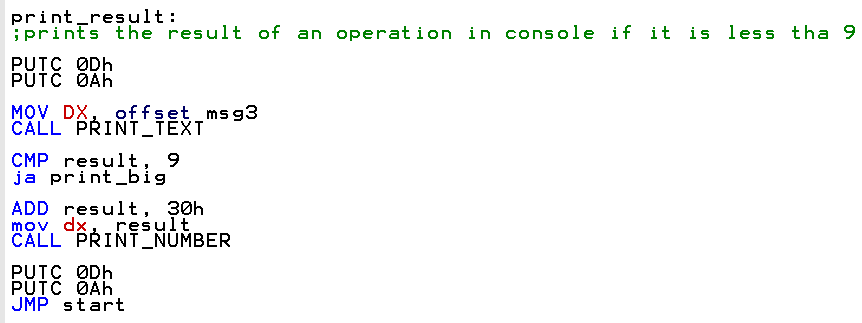
1. do\_mult:

  
Efectúa la multiplicación de los operandos con la instrucción MUL y guarda el resultado en la variable result. Luego salta a la etiqueta print\_result.

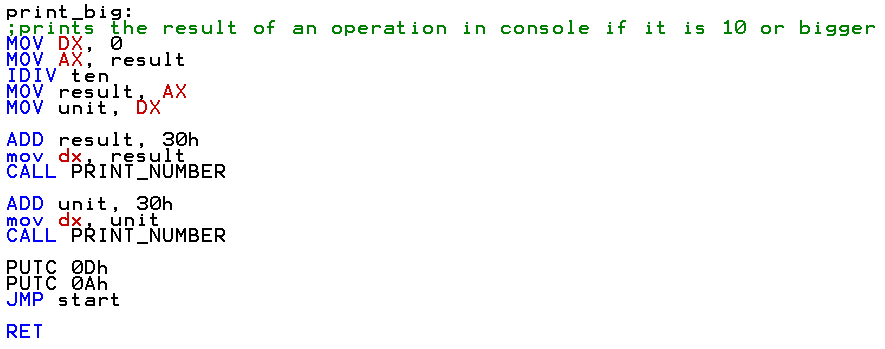
1. do\_div:

  
Efectúa la división del operando 1 entre el operando 2 con la instrucción DIV y guarda el resultado en la variable result. Luego salta a la etiqueta print\_result.

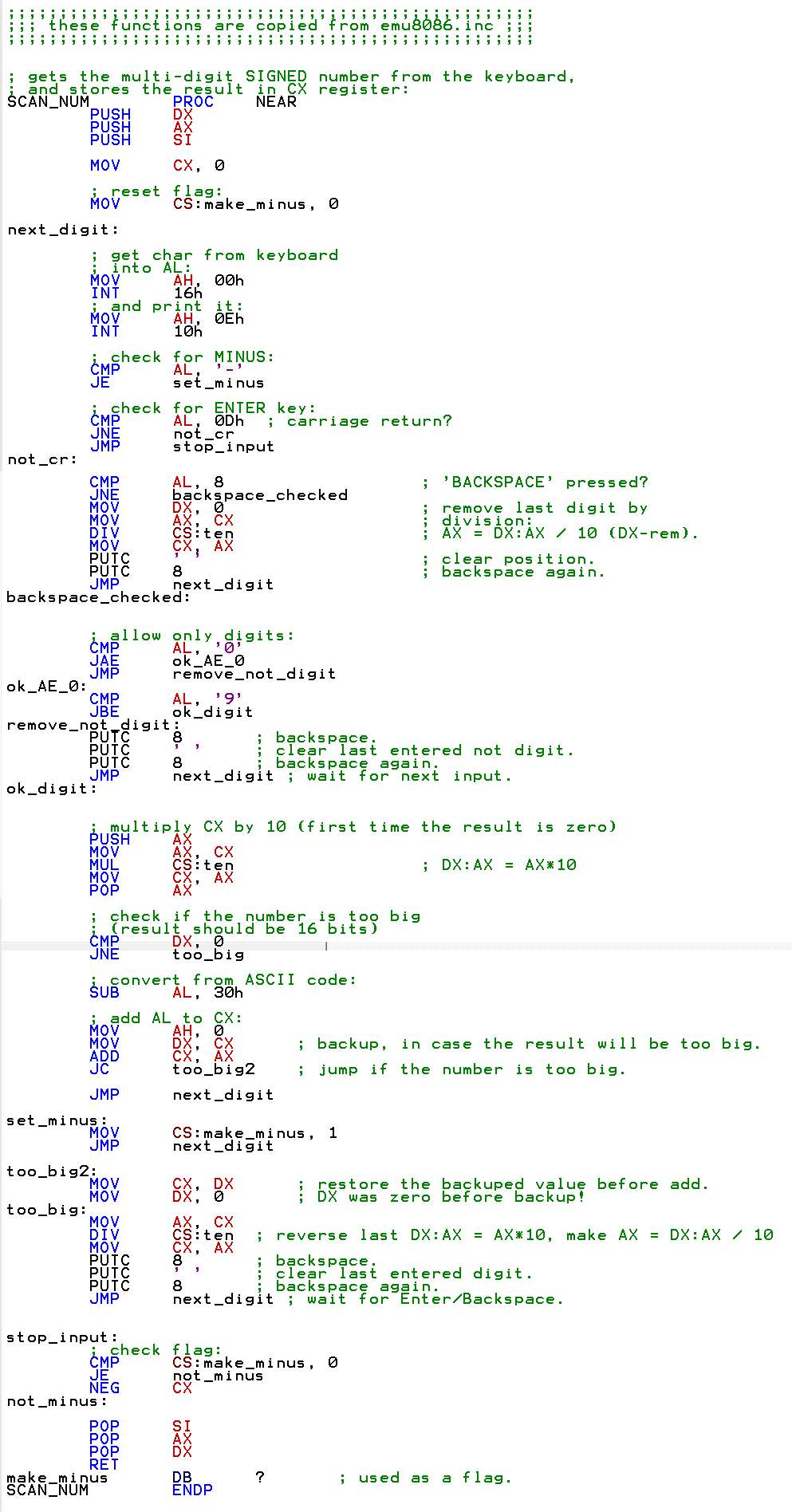
1. print\_result:

  
El código en esta etiqueta ejecuta primero un salto de línea, luego imprime el mensaje indicando que se va a mostrar el resultado de la operación efectuada, llamando el macro PRINT\_TEXT. Luego se compara el resultado con el número 9; si es mayor, se salta a la etiqueta print\_big; sino, se procede a imprimir el caracter equivalente al número que es el resultado. Para ésto, al resultado se le suma el número 48 (30h) para que coincida con el caracter de la tabla ASCII y después se llama a el macro PRINT\_NUM. Finalmente se ejecuta un salto a la etiqueta start para realizar otra operación.

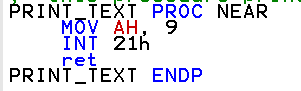
1. print\_big:

  
Si el resultado de una operación es 10 o mayor, se ejecuta el código en esta etiqueta. Primero se divide la variable result entre 10, se guarda el resultado de la división en la misma variable result y el residuo se guarda en la variable unit. Luego se imprime la variable result con la llamada del macro PRINT\_RESULT que equivaldría a las decenas, de la misma forma, se imprime la variable unit que serían las unidades. Finalmente se ejecuta un salto a la etiqueta start para realizar otra operación.

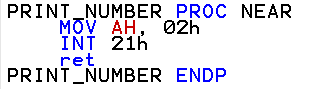
1. Macros:
2. SCAN\_NUM:

  
Éste es un macro que viene incluido en el EMU8086 que se encarga de capturar un número y asegurarse de que éste no incluya letras u otros símbolos.

1. PRINT\_TEXT:

  
Es creado para imprimir una cadena de texto mediante la interrupción 21H (AH=9)

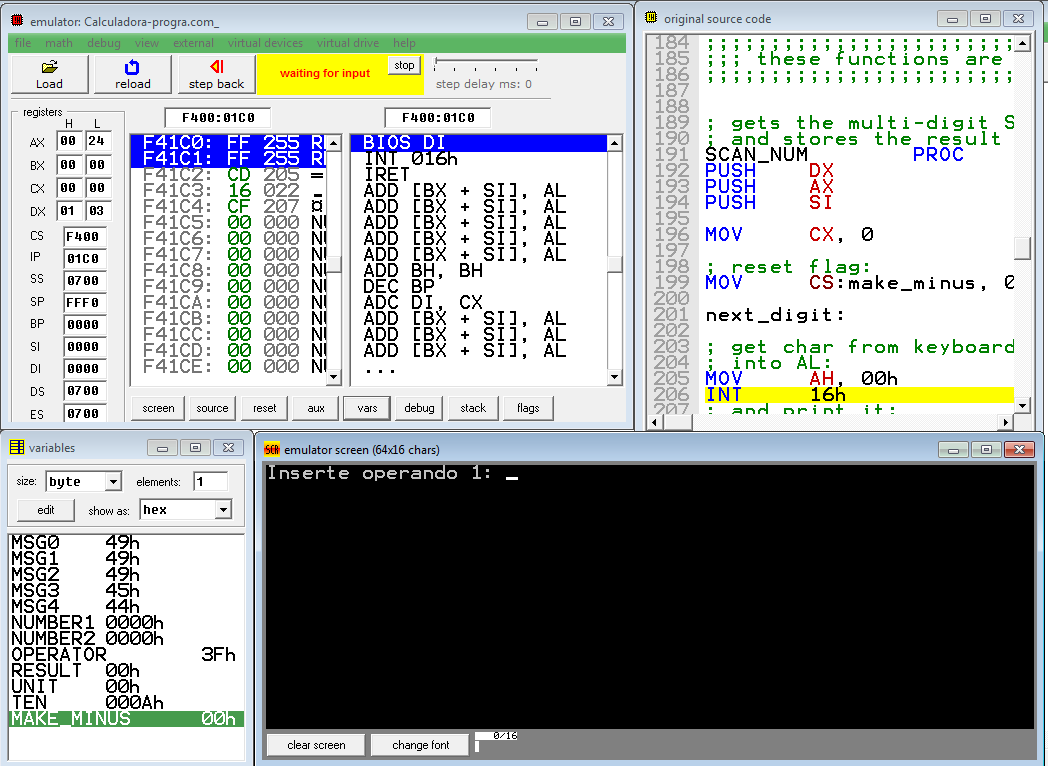
1. PRINT\_NUMBER:

  
Es creado para imprimir en consola un caracter mediante la interrupción 21H (AH=02h)

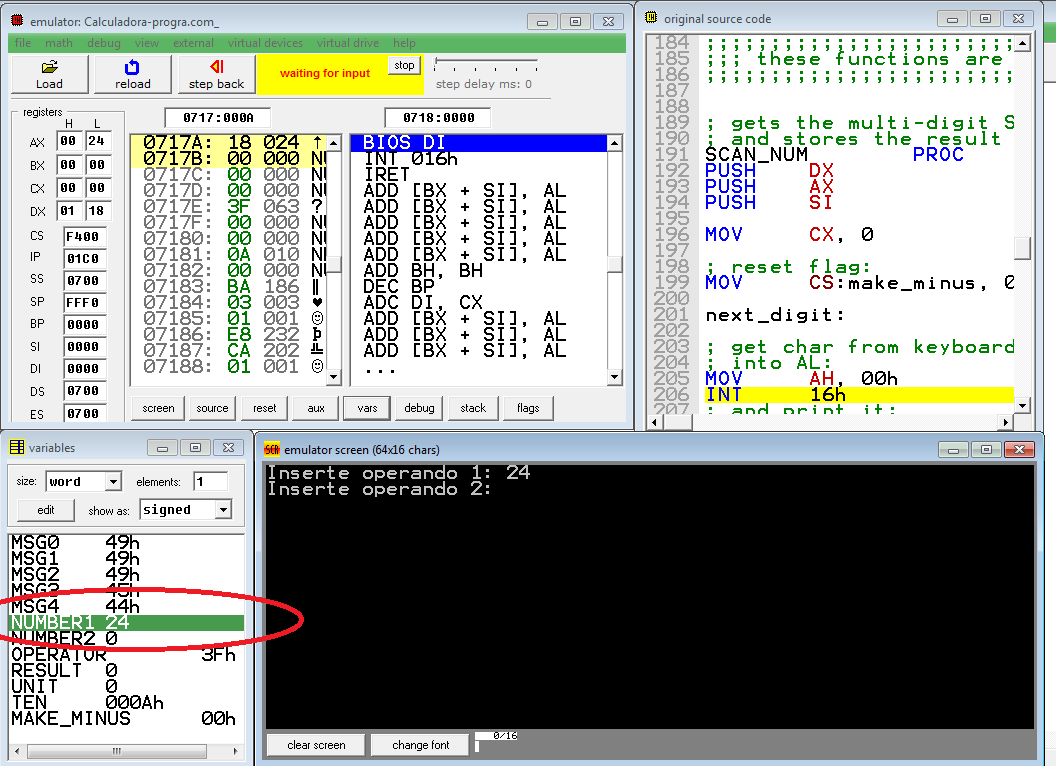
RESULTADOS

Luego de la elaboración del código anteriormente explicado, la aplicación fue capaz de capturar los operandos, de capturar el operador y de efectuar la operación e imprimir el resultado en consola.

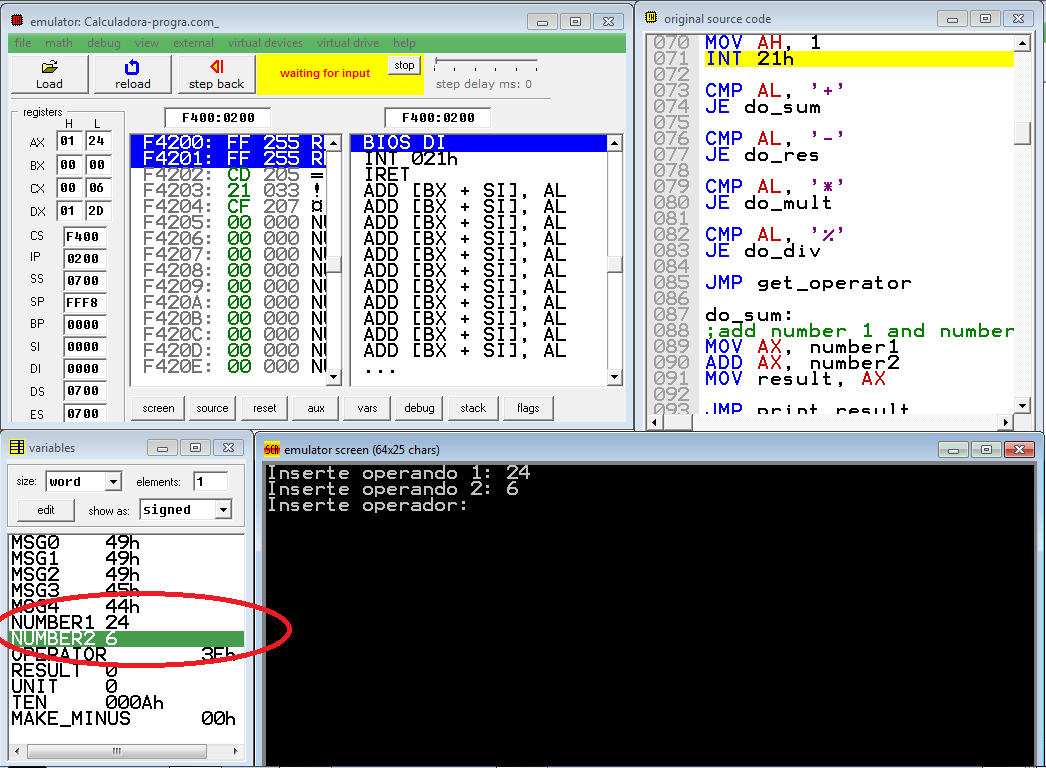
Inicio de la aplicación. Digitar primer operando:



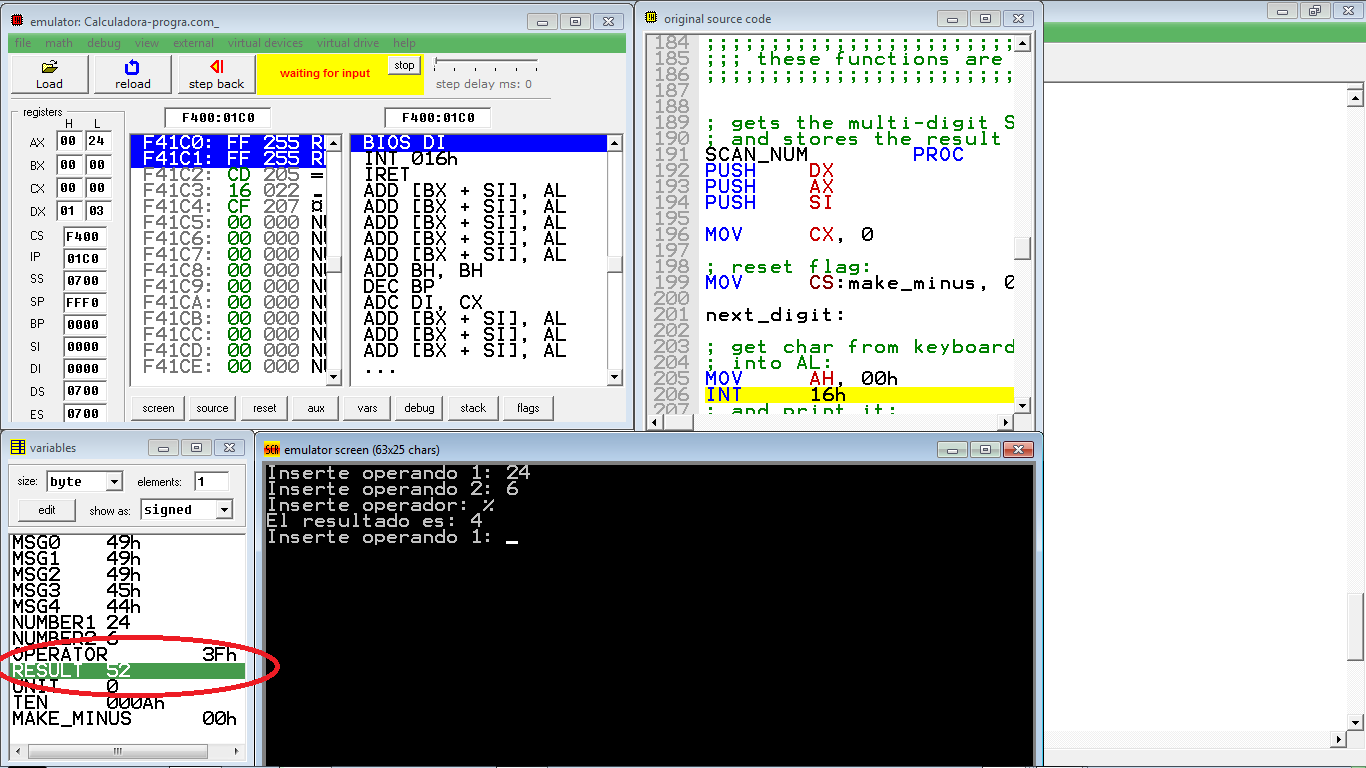
Primer operando capturado. Digitar segundo operando:



Segundo operando capturado. Digitar operador:

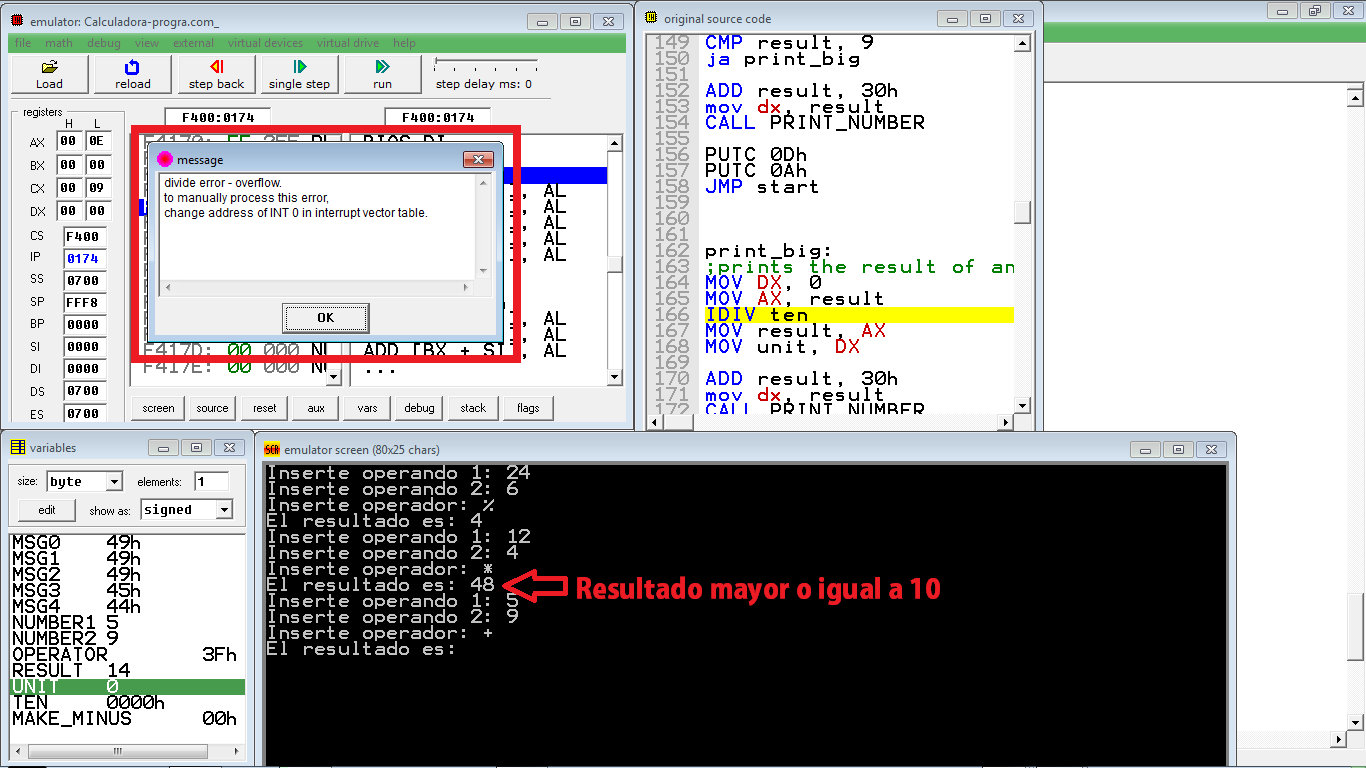


Operador digitado. Operación efectuada. Resultado mostrado en consola. Se solicita de nuevo el primer operando para una nueva operación:



NOTA: véase que la variable result contiene el número decimal 52 aunque la respuesta a la operación sea 4. Ésto es porque al imprimir el resultado, se le suman 48 al resultado para que concuerde con el símbolo adecuado de la tabla ASCII, en éste caso el 4, cuyo valor en la tabla es de 52.

Por otra parte, se descubrió algo particular. Luego de efectuar una operación cuyo resultado fuera mayor o igual a 10, al efectuar la siguiente operación el programa mostraba un error de OVERFLOW, lo cual se descubrió era causado en la etiqueta de impresión.



Luego de varios intentos para solucionarlo, se descubrió que tenía que ver con la pila, sin embargo no fue posible solucionarlo en el tiempo establecido.

No obstante, el código sí funciona para realizar operaciones cuyo resultado sea menor a 100.

No fue posible conseguir que la aplicación se cerrara una vez que se presionara la tecla ESC, por lo que esa función no está disponible.

CONCLUSIONES

Luego de la elaboración de la tarea solicitada, fue posible la mejor comprensión de cómo funciona el lenguaje ensamblador, así como la forma en que trabajan los registros y el procesador. Además se amplió el conocimiento relacionado a este lenguaje debido a la investigación de ciertas funciones. Por otra parte, se mejoró la forma en cómo resolver problemas relacionados con la programación en general.

RECOMENDACIONES

La tarea en sí no era demasiado complicada de realizar, por lo que entre las recomendaciones estarían aprender más sobre las instrucciones disponibles en el lenguaje ensamblador y para el EMU8086. También practicar más el funcionamiento del emulador para estar más familiarizado con éste.