PRÁCTICA N°1 – USO DEL SET DE INSTRUCCIONES INTEL X86 EN EMU8086

Universidad Pedagógica y Tecnológica de Colombia

María Camila Cepeda Sáenz cód.: 201111557, Santiago Andrés Gélvez Camargo cód.: 201111163

Marzo, 2017

***Resumen:* La presente práctica de laboratorio consiste en adquirir habilidad en el manejo del set de instrucciones Intel x86 en EMU8086, así como en los tipos de direccionamiento de datos que se pueden realizar. Esto será posible mediante la realización de 3 programas básicos que consisten en realizar operaciones básicas sencillas entre 6 números, suma y multiplicación entre dos matrices 4X4 y hallar el determinante de una matriz 3X3.**

Índice de términos: Set de instrucciones, emu8086, direccionamiento de datos.

1. **INTRODUCCION**

Los Intel 8086 e Intel 8088 (i8086, llamado oficialmente iAPX 86, e i8088) son dos microprocesadores de 16 bits diseñados por Intel en 1978, iniciadores de la arquitectura x86. La diferencia entre el i8086 y el i8088 es que este último utiliza un bus externo de 8 bits, para poder emplear circuitos de soporte al microprocesador más económicos, en contraposición al bus de 16 bits del i8086.

EMU8086 es un emulador que sirve para simular el microprocesador 8086, posee una interfaz de usuario amistosa, la cual permite familiarizarse con los fundamentos de la programación en lenguaje ensamblador de forma muy intuitiva, además de brindar una serie de recursos para ejecutar y depurar los programas. Entre las ventajas de este emulador se encuentra su facilidad para manipularlo, interfaz amigable con el usuario y barras de herramientas que permiten realizar programas más fácilmente. Entre las desventajas se encuentran que no puede soportar algunas interrupciones que posee el sistema operativo y no puede acceder a puertos físicos (reales), sino que los emula usando otros programas

1. **OBJETIVOS**

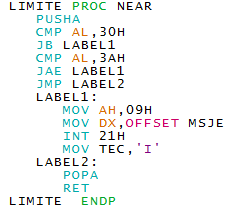
* Conocer el entorno para el desarrollo de aplicaciones en ensamblador para Intel x86 denominado Emu8086.
* Identificar las características de funcionamiento del funcionamiento de las instrucciones lógicas y aritméticas para Intel x86.
* Realizar un programa en ensamblador aplicando las instrucciones lógicas y aritméticas para Intel x86, sobre el emulador Emu8086.

1. **ELEMENTOS Y EQUIPO**

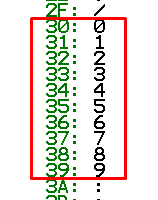
* Software Emu8086.
* Computador personal.

1. **PROCEDIMIENTO**
2. *Diseñe un programa en ensamblador que calcule el resultado de una operación matemática básica haciendo uso de las instrucciones aritméticas. Dicha operación debe tener como mínimo seis operandos y debe ser ingresada por teclado y su resultado debe ser visualizado por pantalla*.

En primer lugar, el programa diseñado muestra mensajes para ingresar cada uno de los operandos (de máximo 3 dígitos) mediante la interrupción 21H/09H, la captura de cada uno de los valores se realiza mediante la interrupción 21H/01H, y luego se evalúa que el valor ingresado por teclado efectivamente corresponda a un número mediante el process mostrado en la figura 1. Éste consiste en realizar una comparación entre el valor ASCII del número ingresado y el numero 30H (correspondiente al número 0), en caso de que el número ingresado sea menor que 30H, el programa emitirá un mensaje que informe al usuario que la tecla ingresada es errónea. En caso contrario, se compara nuevamente la tecla de ingreso, pero ahora con el numero 3AH (ya que 39H corresponde al número 9, cosa que se puede corroborar en la figura 2), si el resultado de la comparación es que el número ingresado es igual o mayor a 3AH, también será emitido el mensaje de tecla errónea, en caso contrario se continúa con ejecución normal del programa.

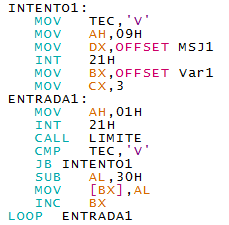


*Figura 1. Process para comprobar que el valor ingresado es una tecla correspondiente a número.*



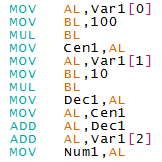
*Figura 2. Valores ASCII (hexadecimales) para los dígitos.*

Se debe crear una variable de 3 posiciones en la cual se guardará el primer número ingresado llamada **VAR1**, y se ubica la dirección de inicio de esta variable en BX. Conforme se van ingresando y evaluando (con el process **LIMITE** mostrado en la figura 1) cada uno de los 3 dígitos correspondientes al primer número, se resta 30H a los dígitos para para tomar el valor decimal de los mismos, y la dirección BX aumenta. El código correspondiente a la descripción anterior se muestra en la figura 3.



*Figura 3. Ingreso de dígitos correspondientes al primer operando.*

Como los dígitos ingresados para el primer número quedan ingresados en 3 posiciones diferentes del mismo vector, se debe realizar una serie de operaciones para convertir ese vector en un único número ubicado en una sola dirección de memoria. Para esto, se toma la posición 0 de **VAR1** (correspondiente a las centenas del número ingresado) y se multiplica por 100, este dato se guarda en una variable llamada **CEN1**, ahora se toma la posición 2 de **VAR1** (correspondiente a las decenas) y se multiplica por 10, guardando el resultado correspondiente en una variable llamada **DEC1.** Finalmente, se suman los valores almacenados en las variables **DEC1** y **CEN1**, y a este valor se le suma el valor contenido en la posición 2 de **VAR1**, obteniendo como resultado final el número ingresado por teclado en una sola variable de una posición de memoria llamada **NUM1**. Lo descrito anteriormente se muestra en la figura 4.



*Figura 4. Conversión de número de 3 posiciones de memoria a número de una posición de memoria.*

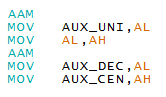
Todos los pasos descritos anteriormente (ilustrados en las figuras 3 y 4), se realizan de igual manera para el ingreso de los otros 5 operandos, cambiando los nombres de las variables dependiendo del operando en cuestión (**VARX, CENX, DECX** y **NUMX**, siendo X el número del operando ingresado).

Tendiendo todos los operandos guardados en sus correspondientes variables, se procede a realizar la operación entre ellos, que en este caso es:

***(NUM1+NUM2-NUM3)NUM4-NUM5+NUM6***

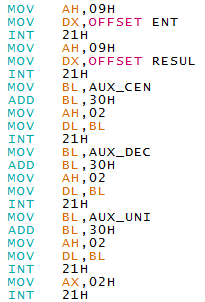
Estas operaciones se realizan haciendo uso de los comandos **ADD**, **SUB** y **MUL**, guardando el resultado final en una variable llamada **RES.**

Para mostrar el resultado en pantalla es necesario utilizar la instrucción **AAM**, la cual toma el valor almacenado en AL (en este caso la variable **RES**) y de la parte correspondiente a las unidades del número es ubicada en **AL** (valor decimal), mientras que el resto del número se ubica en **AH**. El contenido de **AL** es guardado en una variable llamada **AUX\_UNI**. Nuevamente se utiliza la instrucción **AAM**, ubicando ahora el valor correspondiente a las decenas de la variable **RES** en **AL**, y las centenas de dicha variable en **AH.** El contenido que ahora se encuentra en AL será guardado en una variable llamada **AUX\_DEC**, mientras que el de **AH** será almacenado en **AUX\_CEN**, esto se puede observar en la figura 5.



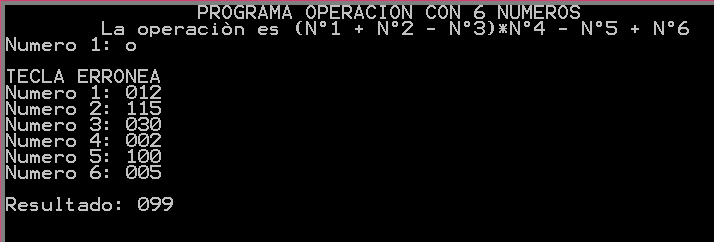
*Figura 5. Sección de código para separar un número en unidades, decenas y centenas.*

Finalmente se procede a mostrar de resultado final por pantalla. Después de enviar el mensaje “el resultado es: ” (**RESUL**), se envían mediante la instrucción 21H/02H el valor contenido en **AUX\_CENT**, seguido de **AUX\_DEC** y **AUX\_UNI**. Es importante destacar que es necesario sumar 30H a cada una de estas variables para que el número sea mostrado en formato ASCII. La parte final del código se muestra en la figura 6.



*Figura 6. Muestra de resultado por pantalla.*

En la figura 7 se muestra un ejemplo del programa realizado.

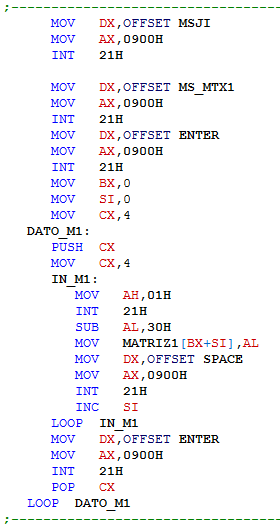


*Figura 7. Ejemplo programa de operación con 6 números.*

1. *Diseñe un programa en ensamblador que calcule la suma y la multiplicación de dos matrices 4 x 4. Los datos de entrada deben ser dispuestos en dos arreglos de memoria, así como las matrices suma y multiplicación. Presente las matrices de entrada y los resultados en pantalla*.

Al igual que en el primer punto, primero se muestra un mensaje indicando la actividad a realizar y las indicaciones para ingresar los números. Cabe aclarar que los números ingresados deben ser de un solo digito por comodidad para realizar las operaciones ya que un número de más de dos dígitos pueden dar como resultado valores demasiado grandes para ser guardados en una posición de memoria (1 Byte). Estos números ingresados se guardan en una variable de 16 bytes llamada **MATRIZ\_1.**

Con el fin de que los datos se muestren visualmente en forma matricial, se realiza un ciclo llamado **DATO\_M1** que se repetirá 4 veces e internamente otro ciclo llamado **IN\_M1** el cual también se repite 4 veces, esto con el fin de mostrar la matriz en filas y columnas (Ver figura 8). En el ciclo **IN\_M1** mediante la interrupción 21H/01H se ingresa el número por teclado, se resta 30H para obtener el valor decimal a cambio del valor ASCII y se guarda el valor en la posición 0 (determinada por el puntero SI), de la variable **MATRIZ1**. Luego se incrementa el valor de SI y se realiza de nuevo el ciclo mediante la instrucción LOOP para seguir ingresando los valores de la primera columna. Una vez terminado el ciclo se indica mediante un mensaje que se debe saltar a la siguiente línea de pantalla y se reinicia el contador del ciclo **IN\_M1** para ingresar los valores de la segunda columna. Cuando se termine el ciclo **DATO\_M1**, se ingresa la segunda matriz en la variable **MATRIZ2 (**también de 16 posiciones), de la misma manera que se realizó para la primera matriz.

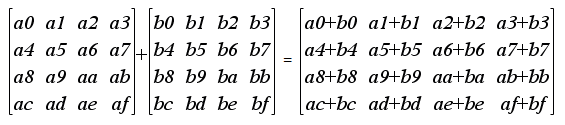


*Figura 8. Ingreso de números de datos para matriz.*

Una vez ingresados los datos por teclado, debe dividir la tarea a realizar en dos partes: suma y multiplicación.

***Suma de matrices:***

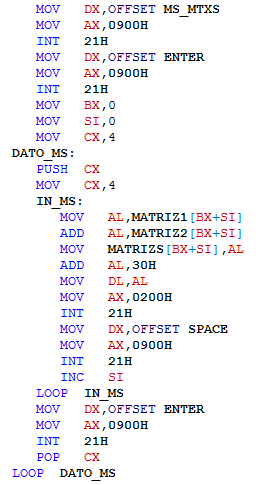
Es necesario recordar el proceso de suma de matrices. La suma se realiza de manera directa, es decir, al sumar dos matrices, cada término de la primera matriz se suma con el término de la segunda matriz teniendo en cuenta la posición en la que se encuentran. Esto se ilustra en la figura 9 a continuación:



*Figura 9. Algoritmo para suma de matrices*

Para esta operación se crea una nueva variable llamada **MATRIZS** la cual servirá para guardar el resultado de la suma. Al igual que el algoritmo de los dos ciclos para ingresar matrices, se realizan dos ciclos para ingresar el valor de la suma en cada una de las posiciones de la matriz suma resultante.

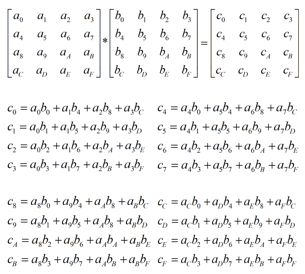
El proceso de suma inicia tomando el dato en la primera posición (posición 0 determinado por **SI**), y se guarda en el registro **AL** para inmediatamente ser sumado con la primera posición de la segunda matriz. Como el dato queda guardado en **AL**, se puede escribir en la primera posición de la variable **MATRIZS.** Luego de ser guardado, se muestra por pantalla sumando 30H para mostrar el carácter ASCII correspondiente al número resultante. Seguido de esto, se incrementa el valor de **SI** con el fin de apuntar a la segunda posición de las matrices. Se puede observar en la figura 10 el uso de los ciclos para calcular y mostrar en pantalla el resultado de las operaciones.



*Figura 10. Sección de código correspondiente al proceso se suma de matrices.*

***Multiplicación de matrices:***

En la figura 11 se detalla claramente las operaciones que utilizan para la multiplicación de matrices.

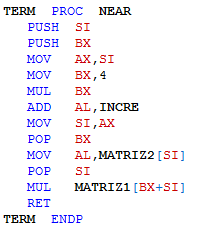


*Figura 11. Método grafico para la multiplicación de matrices.*

Según la figura 11, los términos de la matriz A cada cuatro términos de C se deben sumar 4 a los subíndices de A.

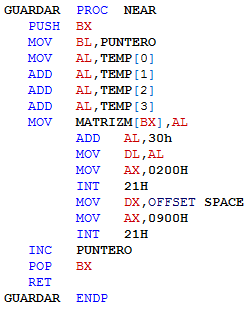
En los términos de B, para cada operación del mismo término C se le debe agregar 4 al subíndice de B. Para el siguiente termino de C, los subíndices de B se reinician, pero esta vez empiezan en 1 y se sigue sumando 4 a partir de este subíndice para cada operación. Al llegar al cuarto termino de C se debe reiniciar el subíndice de C a 0 y ejecutar la operación de la misma manera mencionada inmediatamente anteriormente.

Dicho lo anterior es necesario crear una variable de 4 posiciones llamada **TEMP** la cual guardará los resultados de cada uno de los productos para luego sumarlos y obtener cada uno de los términos de matriz resultante la cual será una variable de 16 posiciones llamada **MATRIZM.** Cada uno de los productos se realiza dentro de un proces llamado **TERM** mostrado en la figura 12, en el cual se manejan los subíndices como se mencionan en la descripción de la figura 11.El proces primero almacena en pila mediante la instrucción PUSH el valor de **SI.** Este valor de **SI**, se multiplica por 4 y se le adiciona el valor de **INCRE (INCRE** es una variable creada para llevar la cuenta de la posición del dato que se debe tomar de la matriz B), ya que debe para cada operación este valor resultante es la posición del dato que se debe tomar de la variable **MATRIZ2.** Luego de tomar este valor, se recupera el valor de SI con la instrucción POP, y mediante el direccionamiento base más índice se multiplica el valor del dato adquirido de **MATRIZ2** con el valor que se adquiere de la variable **MATRIZ1** en la posición BX+SI.



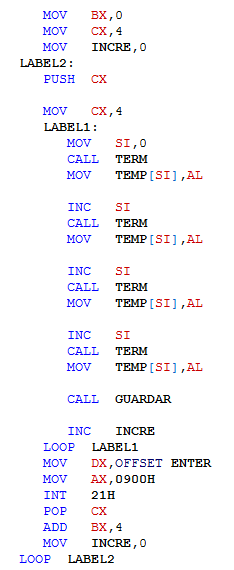
*Figura 12. Proces para tomar cada uno de los factores de cada término de la matriz C.*

Una vez terminado el proces, este regresa con un valor almacenado en el registro **AL** el cual se guarda en la variable **TEMP** en la posición **SI**. Luego este valor de **SI** se incrementa en 1 y luego se realiza de nuevo el proces **TERM.** Cuando se han realizado las cuatro operaciones correspondientes al primer valor del primer término de C, se llama a un nuevo proces llamado **GUARDAR** mostrado en la figura 13, en el cual se realiza la suma de cada una de las posiciones de **TEMP** y este se almacena en la posición **PUNTERO (PUNTERO** es una variable creada con el fin de ser quien lleve la cuenta de la posición de la variable **MATRIZM).** Cuando se almacena el valor resultante de la suma, inmediatamente se suma a este valor 30H para luego ser mostrado por pantalla mediante la interrupción 21H/09H y se incremente el valor de **PUNTERO** en 1.



*Figura 13. Proces para almacenar el dato en la matriz* ***MATRIZM*** *y mostrar el resultado por pantalla.*

Luego de guardar el resultado en la matriz, se incrementa el valor de **INCRE** y se repite nuevamente todo el proceso mediante el LOOP reiniciando el valor de **SI** a cero. Cuando el proceso se ha repetido 4 veces (finalización del LOOP), se realiza un salto de línea y se incrementa el valor de **BX** para aumentar en 4 para tomar las siguientes posiciones de la matriz A (**MATRIZ1**) y se repite el proceso desde **LABEL2** 4 veces más. Los ciclos **LABEL1** y **LABEL2** semuestran en la figura 14.



*Figura 14. Seccion de código encargado de los ciclos requeridos para la operación multiplicación de matrices.*

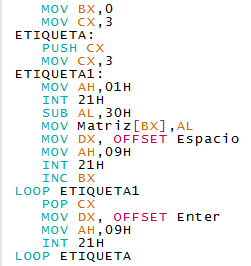
En la figura 15 se muestra un ejemplo del programa utilizado. Para el caso de la multiplicación se utiliza la propiedad **AI=A**, es decir, una matriz A multiplicada por la matriz identidad da como resultado la matriz A.



*Figura 15. Ejemplo de programa para suma y multiplicación de matrices 4X4.*

1. *Desarrolle un programa en ensamblador que evalúe el determinante de matrices 3 x 3. Presente las matrices de entrada y los resultados en pantalla*.

Para el caso de la matriz 3X3, al igual que en el punto anterior, los valores de la matriz son de solo un digito, por lo cual la captura de los valores es sencilla, iniciando en 0 el registro **BX**, el cual se usará como un puntero. En primer lugar, se crea un ciclo (**ETIQUETA**) que se repetirá 3 veces, y dentro de éste, otro ciclo que se repite de igual manera (**ETIQUETA1**), en el segundo ciclo, se toma el número ingresado y se le resta 30H, correspondiente a la conversión del número ASCII a decimal, luego se ubica dicho número en la posición 0 (**BX**) de una variable **MATRIZ** creada anteriormente, se añade un espacio por cuestiones de visualización en pantalla, y se incrementa **BX**. Cuando termine este ciclo (**ETIQUETA1**), estarán ingresados los 3 números correspondientes a la primera fila de la matriz, y luego de esto se añade un enter por cuestiones de visualización. Esto se repite 3 veces (**ETIQUETA**), obteniendo finalmente la matriz en su totalidad. Lo anterior se puede observar con más claridad en la figura 16.



*Figura 16. Ingreso de valores matriz 3X3*

Una vez ingresada la matriz, es necesario tener en cuenta como se realiza el determinante de la misma, esto es:

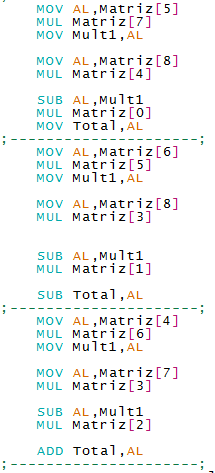
 

Teniendo claro esto, y observando que no se sigue una secuencia de operaciones en específico (con respecto a la ubicación de cada uno de los valores de la matriz), se realizan las operaciones utilizando direccionamiento indirecto de la manera que se describe a continuación:

Se toma **MATRIZ** en la posición 5 () y su contenido se guarda en el registro **AL**, y a este registro se multiplica el contenido de MATRIZ en la posición 7 (), el valor de esta multiplicación se guarda en la variable **MULT1**. Ahora se mueve el contenido de **MATRIZ** en la posición 8 a **AL** y se realiza la multiplicación de este registro y el valor de **MATRIZ** en la posición 4. Ahora se resta el resultado de esta operación con la variable **MULT1**, y este valor es multiplicado por el contenido de **MATRIZ** en la posición 0, guardando el resultado final en una variable llamada **TOTAL**.

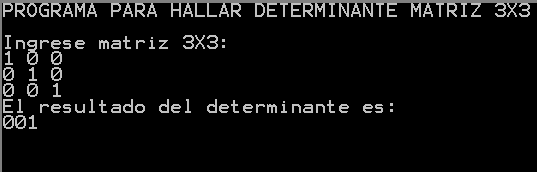
Este proceso se repite 2 veces más con las direcciones de **MATRIZ** correspondientes a las operaciones para hallar el determinante, realizando la suma entre la variable TOTAL y el resultado final de la última multiplicación para cada caso (**MULT1** por **MATRIZ** en las posiciones 1 y 2). Al finalizar este proceso, la variable **TOTAL** tendrá el valor final del determinante.

Las operaciones realizadas para hallar el determinante mediante código se pueden observar en la figura 17.



*Figura 17. Calculo de determinante matriz 3X3 mediante código.*

Para mostrar el resultado en pantalla, se realiza el mismo proceso utilizado para el primer punto (ver figuras 5 y 6). La figura 18 muestra un ejemplo del programa desarrollado en funcionamiento.



*Figura 18. Programa para hallar el determinante de una matriz 3X3 en funcionamiento.*

1. **CONCLUSIONES**

* Desde el primer procesador (Intel 4004), los microprocesadores se encuentran que cada objeto que maneje datos en el mercado. Por esto es importante conocer cuál es su diseño interno y los métodos que usan para realizar las operaciones que lo llevan a ser tan útil. El lenguaje ensamblador ayuda demasiado en cuanto a la programación de dichos microprocesadores y es el programa EMU8086 quien nos facilita el entendimiento de la realización de las operaciones en un microprocesador utilizando el direccionamiento en modo real.
* Aunque hoy en día las calculadoras realizan operaciones difíciles a gran velocidad, anteriormente era una tarea lenta comparada con la actual velocidad, aunque depende del tipo de operación a realizar como es el caso de la multiplicación de matrices donde la realización de la practica demuestra lo demorado que es realizar la operación en un microprocesador como el Intel 8086.
* Es necesario tener en cuenta que existe un tipo de direccionamiento de datos que se ajusta de mejor manera para cada uno de los trabajos a realizar, por lo cual es importante identificar y entender cómo funciona cada uno de dichos tipos, de tal manera que los programas realizados sean más eficientes.

1. **BIBLIOGRAFIA**

* Documentación EMU8086 (Set de instrucciones e interrupciones)
* Barry B. Brey. Microprocesadores Intel. Séptima edición. Editorial Pearson Prentice Hall. 2006.
* Ensamblador para 8086. ITSL. https://es.slideshare.net/ece79/ensamblador-8086?next\_slideshow=1. Publicado el 21 septiembre de 2010. Consultado el 14 de marzo de 2017.
* Emulador 8086. http://electronicautil. blogspot.com.co/2008/08/emu8086-v405.html Publicado el 19 noviembre de 2014. Consultado el 14 de marzo de 2017.