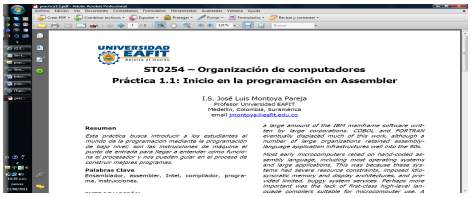


ST0254 – Organización de Computadores

Práctica 1.1: Inicio en la Programación en Assembler

Ernesto Quintero, Luisa Fda. Querubín, Jorman Bustos

Universidad EAFIT

Medellín, Colombia, Suramérica

equinte5@eafit.edu.co

lquerubi@eafit.edu.co

jbustos@eafit.edu.co

Resumen

El documento contiene la descripción de un programa desarrollado en assembler que lee de pantalla dos vectores de N posiciones cada uno y le calcula la suma y la resta vectorial según lo decida el usuario mediante un sencillo menú.

Introducción

Este documento presenta una descripción detallada del proceso de desarrollo de un programa que tomando dos vectores de N posiciones realiza o la suma o la resta de los mismos, todo esto mediante: El algoritmo, las instrucciones utilizadas del lenguaje y finalmente una explicación de cómo funciona la aplicación, con el objetivo de que el lector pueda comprender fácilmente la solución del problema planteado.

I. Objetivos

- Iniciar el proceso de aprendizaje de la programación en Assembler, mediante el uso de algunas de sus instrucciones.
- Identificar el algoritmo como el primer paso para comprender el problema y posteriormente encontrar una solución.

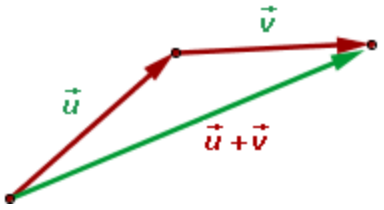
II. Suma y Resta Vectorial

Para realizar la suma y la resta vectorial básicamente se debe:

1. Definir los vectores con los que se realizaran las operaciones.

$\vec{U}=(U_1,U_2)$ $\vec{V}=(V_1,V_2)$, $\vec{U}=(U_1,U_2)$ $\vec{V}=(V_1,V_2)$

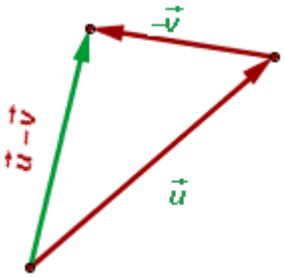
2. Como conocemos los valores de las componentes podemos aplicar la siguiente formula.



$$\vec{U}+\vec{V}=(U_1+V_1,\ U_2+V_2)$$

Y así obtenemos el vector solución.

3. Para la resta se lleva a cabo el mismo proceso pero con la siguiente formula.



$$\vec{u} - \vec{v} = (u_1 - v_1,\ u_2 - v_2)$$

III. Funciones del Lenguaje

Para desarrollar la primera versión del programa utilizamos las siguientes instrucciones que ofrece Simuproc:

- MSG: Muestra un mensaje en pantalla.
- LDT: Lee un valor del Teclado y lo lleva al registro AX.
- STA: Guarda el contenido de AX en la dirección de Memoria especificada.
- EAP: Escribe en Pantalla el contenido del registro AX

De esta forma almacenamos los datos uno por uno especificando una posición en memoria.
Luego utilizando las instrucciones:

- LDB: La instrucción carga en AX el contenido de memoria almacenado en [mem] + BX.
- ADD: AX = AX + el contenido de la dirección de memoria.
- SUB: AX = AX - el contenido de la dirección de memoria.
- EAP: Escribe en Pantalla el contenido del registro AX.
- HLT: Termina el programa.
  (Debe ir al final de cada programa).

Obteníamos los resultados.
Para la segunda versión consideramos la posibilidad de trabajar con flotantes, así que modificamos todo el programa para que funcionara con este tipo de dato mediante las instrucciones designadas por simuproc, algunas de ellas son:

- LDF: Carga en BX y AX un número de 32 bits (IEEE) que esta almacenado en la dir [mem] y mem+1. En BX quedan los digitos mas Significativos.
- ADDF: Suma números de 32 bits: En BX y AX queda el resultado de la suma de estos mas el contenido de [mem] y mem+1.
- SUBF: Resta el numero de 32 bits: BX y AX = BX y AX - [mem]y mem+1.

Pero según indicaciones posteriores se estableció que el programa debería manejar únicamente números enteros.
Para la tercera y última versión tuvimos en cuenta la primera versión que habíamos realizado pero mejorando varios aspectos como:

- Poder determinar el tamaño del vector. (Que inicialmente se había fijado con únicamente 5 posiciones).
- Determinar solo una posición inicial de memoria para cada vector y a partir de esta ubicar los demás valores en las posiciones siguientes “automáticamente”.
- Sumar y restar mediante ciclos, sin tener que especificar las posiciones para cada una de las operaciones.
- Un menú con la posibilidad de almacenar los vectores y permitirle al usuario elegir la operación a realizar (por ejemplo: sumar) y posteriormente retomar los datos para realizar una nueva operación (restar).
- Manejo de excepciones.

Debido a los cambios incorporamos nuevas instrucciones:

- CLA: Hace AX = 0.
- MOV: Copia el valor almacenado en el origen al destino. El destino y/o origen pueden ser registros o direcciones de memoria o combinación de estos.
- INC: Incrementa en 1 el destino especificado, el parámetro puede ser una dirección de memoria o un registro.
- CMP: Compara AX con [mem], si AX es mayor, Z=0 N=0, si es igual Z=1 N=0, si es menor Z=0 N=1.

- JMA: Saltar si es Mayor.
  Si Z = 0 y N = 0, PC = contenido de memoria.
- STB: guarda el contenido de AX en la dirección [mem] + BX
- LOOP: Decrementa CX y salta a la Pos de memoria si CX no es cero.
- JME: Saltar si es Menor.
  Si N = 1, PC = contenido de la memoria.

IV. Funcionamiento de la Aplicación

- El programa inicia y le presenta al usuario 4 opciones: "0 - PARA INGRESAR VECTORES", "1 - PARA SUMAR VECTORES", "2 - PARA RESTAR VECTORES" y "3 - PARA SALIR".
  En primer lugar el usuario deberá ingresar los vectores o salir, pues sin los datos para operar tanto la segunda como la tercera opción lanzaran excepciones.
- Si el usuario decide ingresar los vectores aparecerá un mensaje que le pregunta por el tamaño del vector, después de definirlo se le pide que ingrese los valores para el vector uno, luego para el dos y finalmente regresa al menú inicial en donde ya están habilitadas todas las opciones.
- Si el usuario oprime 1 el programa retornara el resultado de la suma de los vectores e inmediatamente volverá al menú inicial, lo mismo ocurre para el caso de oprimir 2 solo que devuelve la resta de los vectores.
- Después de realizar las operaciones con los vectores el usuario puede elegir 3 para salir o volver a realizar las operaciones, obteniendo el mismo resultado.

V. Problemas encontrados durante el desarrollo

Posiciones de memoria que se modificaban sobreescribian durante la ejecución.

No haber leído todos los comandos y no darnos cuenta de que entre mas instrucciones conocíamos mas fácil resultaba.

VI. Algoritmo

```
inicio
    var a = tam = 0
    mientras( a != 3 )haga
        imprima "0 - para llenar vectores"
        imprima "1 - para sumar vectores"
        imprima "2 - para restar vectores"
        imprima "3 - para salir"

        lea a
        si ( a == 0 ) ent.
            lea tam
            var bx = 0
            var cx = tam
            while( cx > 0 )haga
                cx = cx - 1
                lea num
                vec1[FDA + bx] = num
                bx = bx + 1
            fin-mientras
            var bx
            var cx = tam
            while( cx > 0 )haga
                cx = cx - 1
                lea num
                vec1[FDA + bx] = num
                bx = bx + 1
            fin-mientras
        fin-si
        si ( a == 1 ) ent.
            si( tam == 0 )end
                imprima "no se han ingresado vectores"
                continue // salta a la proxima iteracion
del ciclo
            fin-si
            bx = 0
            cx = tam
            imprima "la suma de vectores es: "
```

```
            while( cx > 0 )haga
                cx = cx - 1
                var temp = vec1[FDA+bx];
                temp = vect2[FDD + bx] + temp
                imprima temp
                bx = bx + 1
            fin-mientras
        fin-si
        si ( a == 2 ) ent.
            si( tam == 0 )end
                imprima "no se han ingresado vectores"
                continue // salta a la proxima iteracion
del ciclo
            fin-si
            bx = 0
            cx = tam
            imprima "la resta de vectores es: "
            while( cx > 0 )haga
                cx = cx - 1
                var temp = vec1[FDA+bx];
                temp = vect2[FDD + bx] - temp
                imprima temp
                bx = bx + 1
            fin-mientras
        fin-si
    fin-mientras
fin
```

VII. Referencias

[1]http://www.vitutor.com/geo/vec/a_6.html Visitado el 1 de agosto de 2011.

[2] http://simuproc.cjb.net/ ó http://simuproc.tk/ Visitado en Agosto de 2011.