Sistema de resolución de ecuaciones cuadráticas (Agosto de 2013)

Héctor C. Alzate, *Mateo Carvajal, Juan D. Castillo, UNIVERSIDAD EAFIT.*

Resumen

En este documento se describe la forma en que la primera práctiva de la materia Organización de Computadores fue realizada, se desarrolló un programa en el lenguaje Assembler que permite encontrar las raíces de una ecuación cuadrática que se ejecuta sobre la plataforma SimuProc.

I. Introducción

La práctica consistía en desarrollar un programa que permitiese desarrollar la siguiente serie de pasos:

1. Leer datos para conformar una ecuación desde el teclado.
2. Leer datos para definir un rango en los números enteros, en el cual se buscarán soluciones para la ecuación.
3. Procesar dicha información e imprimir las raíces que anulan el valor de “y” en el intervalo.

Este trabajo debía ser desarrollado en el lenguaje Assembler utilizando como herramienta el simulador “SimuProc” que sirve para emular el funcionamiento de un procesador simple y el uso de sus operaciones.

II. Problema

A. Definición

“Una ecuación de segundo grado o ecuación cuadrática es una ecuación que tiene la forma de una suma de términos cuyo grado máximo es dos, es decir, una ecuación cuadrática puede ser representada por un polinomio de segundo grado o polinomio cuadrático. La expresión canónica general de una ecuación cuadrática es:

$$ax^2+bx+c=0,\ para\ a\neq 0$$

donde x representa la variable y a, b y c son constantes; a es un coeficiente cuadrático (distinto de 0), b el coeficiente lineal y c es el término independiente. Este polinomio se puede representar mediante una gráfica de una función cuadrática o parábola. Esta representación gráfica es útil, porque la intersección de esta gráfica con el eje horizontal coinciden con las soluciones de la ecuación (y dado que pueden existir dos, una o ninguna intersección, esos pueden ser los números de soluciones de la ecuación).”

B. Datos

El programa debe leer desde el teclado los datos necesarios para conformar la ecuación y a continuación encontrar las raíces que anulan la ecuación; estos datos incluyen:

1. Los coeficientes de la ecuación.
2. La definición del intervalo a revisar.

III. Algoritmo

Nuestra solución utiliza una aproximación simple al problema:

1. Se leen los datos conformando la ecuación.
2. Se leen los datos conformando el intervalo.
3. Se procede a iterar sobre el intervalo con dx = 1.
4. Se evalúa la ecuación con el valor de “x” determinado.
5. El resultado de “y” se compara con 0, en caso de que esta comparación sea verdadera se imprime el valor de “x”.

A. Pseudocódigo

```
Inicio
  leer a, b, c, min, max;
  max := max + 1;
  mientras (min < max) haga
    y := a * (min * min) + (b * min) + c
    si (y = 0) entonces
      imprimir min
    finsi
    min := min + 1
  finmientras
Fin
```

B. Restricciones

Los datos ingresados necesitan cumplir unas características específicas:

- Los coeficientes deben ser números enteros.
- Las raíces deben ser números enteros, de lo contrario no las encontrará.

- El número ingresado como “mayor” debe ser mayor que el “menor”.
- El coeficiente “a” debe ser diferente de 0.

IV. Pruebas

Para verificar los resultados de la ejecución del programa utilizamos casos de prueba básicos, con números que podíamos calcular simplemente y utilizando la calculadora de wolphram alpha.

Luego nos dimos cuenta de que este método era tedioso y decidimos desarrollar un programa en Java que se acercara al funcionamiento del algoritmo propuesto y comparamos los resultados.

V. Dificultades y Soluciones

1. Desconocimiento del lenguaje Assembler, para solucionarlo nos referenciamos de la documentación de SimuProc.
2. Manejo de números negativos, se solucionó utilizando números de punto flotante de 32 bits que permitían almacenar el signo.
3. Dificultad verificando los resultados, se desarrolló un programa simple en Java para realizar comparaciones.
4. Dificultad utilizando las instrucciones GOTO (JMP), se solucionó mediante revisión extensiva del código.
5. La incapacidad para comparar números flotantes, se solucionó restandolos y comparando su resultado a 0.

VI. Conclusiones

1. El lenguaje Assembler es complejo de manejar, ya que se han de cuidar muchos aspectos de la programación que damos por hecho en otros lenguajes.
2. El lenguaje que compila SimuProc es una aproximación a Assembler.
3. Se pudó apreciar el flujo básico de operación en lenguaje Assembler.
4. El Debugger de SimuProc es extremadament útil para encontrar errores en el código.
5. Las instrucciones y datos en el Editor se representan mediante números binarios y hexadecimales.