Implementación de algoritmo optimizado de multiplicación en 8 bits para lenguaje ensamblador

1st Joseph Valenciano Madrigal
Tecnológico de Costa Rica
jvalencianom08@yahoo.com

2nd Erick Blanco Fonseca
Tecnológico de Costa Rica
erickbf22@gmail.com

3rd Emmanuel Murillo Sánches
Tecnológico de Costa Rica
e-mursa@live.com

***Abstract*—El primer proyecto programado de arquitectura de computadores básicamente consta de dos secciones, las cuales son implementar una función o método de multiplicación entera sin signo de ocho bits, pero mediante la técnica de corrimientos lógicos a la izquierda, pero inicialmente debe ser programada en un lenguaje de prototipado rápido, seguidamente debe pasarse a código de ensamblador(MASM), pero a diferencia que este debe tomar los datos del usuario en la terminal y validar que sea un numero que únicamente contenga ocho bits.**

I. Introducción

Para este primer proyecto se buscará el aprendizaje y puesta en práctica de los distintos operandos de ensamblador como ”MOV”, ”ADD”, ”SUB”, etc. Además de la implementación de directivas como ”PROC”, ”PROTO”, e ”INVOKE”, junto con el uso de la biblioteca Irvine, que ayudará con la disposición de estas directivas en el código.

Primero se implementará el algoritmo en Python, ya que es un lenguaje muy sencillo y cómodo para el programador. Con esta base será un poco más fácil, pasar o implementar el código en ensamblador, porque se tiene la base en un lenguaje de alto nivel y además ya se conoce el funcionamiento del algoritmo.

No obstante, el algoritmo consiste en la multiplicación de dos números enteros, pero con una gran diferencia, que se efectuará con corrimientos de bits lógicos a la izquierda, para aprender y experimentar el uso del operando ”SHL” en ensamblador.

II. Algoritmo en Python

Se escogió Python como lenguaje de prototipado rápido para este algoritmo de multiplicación binaria mediante corrimiento de bits, porque es un lenguaje de programación el cual conocemos y nos sentimos muy cómodos con él.

Básicamente la función recibe como parámetros, dos números enteros los cuales se multiplicarán, seguidamente de esto, se convierten de decimal a binario, para poder trabajar con el corrimiento de bits.

Python toma los números binarios como un ”string”, por lo que es necesario revertir el orden del binario del multiplicador(segundo numero binario), para poder tomar sus índices en una posición correcta, para posteriormente hacer el corrimiento hacia el izquierda con el índice tomado.

Posteriormente se realiza un ciclo para que recorra el multiplicador y verifique en que posiciones se encuentran los bits activos, es decir los ”uno”. Con dichas posiciones(índices), luego se realiza el ”Shift left” al multiplicando(primer numero binario). Se suman todos estos resultados y se guardan en una variable, finalmente retornando el resultado de la multiplicación.

Este algoritmo hizo una ”simulación” de lo que haría el ensamblador, conociendo que en ensamblador no habría que hacer el proceso de pasar un numero de decimal a binario, porque ya lo guarda como tal.

III. Diagrama de lenguaje prototipado

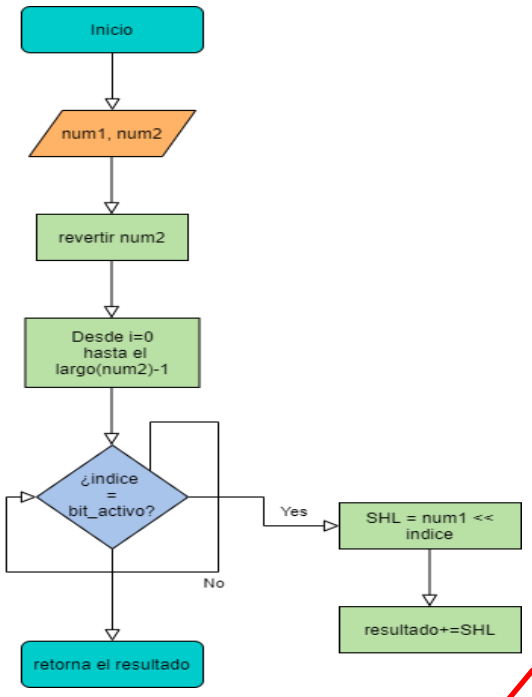

Fig. 1. Diagrama de flujo del algoritmo de python.

IV. Algoritmo en MASM

Partiendo del algoritmo previamente señalado y teniendo claro el objetivo que se tiene planteado alcanzar a través del mismo. Se procedió a la construcción de un algoritmo similar haciendo uso de un lenguaje de bajo nivel como lo es Microsoft Macro Assembler (MASM). En relación con el algoritmo desarrollado, como ya es conocido, el objetivo principal del mismo se encuentra centralizado en multiplicar

cualquier par de números enteros sin signo de 8 bits (1 byte) mediante la técnica de corrimientos lógicos a la izquierda. La experiencia de construir este tipo de programa haciendo uso de dicho lenguaje, no cabe duda que ha sido muy enriquecedora y es por eso que a continuación se procederá con una detallada explicación acerca del algoritmo desarrollado.

Comenzando con los aspectos más generales, relacionados con el programa desarrollado se debe tener en cuenta que el mismo hace uso de cinco diferentes procedimientos y de algunos prototipos de dichos procedimientos para llevar a cabo un adecuado funcionamiento. En primera instancia tenemos el procedimiento llamado **multiplicacion-main**, este en esencia es el encargado de conducir el hilo de la aplicación dado que se encarga de llamar o dicho de otra manera invocar los distintos procedimientos acordes al funcionamiento y orden lógico requerido para el correcto funcionamiento del algoritmo.

Seguidamente, en concordancia con la secuencia de funcionamiento del programa se cuenta con el procedimiento llamado **Pedir-Numeros**, este como su nombre lo indica es el encargado de recibir una entrada de dos números enteros distintos denominados dentro del flujo del programa como multiplicador y multiplicando. Posteriormente, habiendo recibido los valores, se lleva a cabo el procedimiento denominado **validar-Entero**, este haciendo uso de una estructura de tipo "if", se encarga de verificar que efectivamente la información recibida corresponda a numero entero ubicado dentro del rango de validez.

Suponiendo que se cuenta con valores válidos, el algoritmo desarrollado se encargaría de ejecutar en este caso el procedimiento llamado **multiplicacion-binaria**, el cual a su vez se encargará de realizar la multiplicación de los valores recibidos. Para alcanzar dicho objetivo, este procedimiento se fundamenta en la utilización de corrimientos a la derecha para determinar a través de un ciclo "while", la ubicación (posición) de los bits activos en el número binario correspondiente al multiplicador. Posteriormente haciendo uso en esta ocasión del corrimiento a la izquierda dicho algoritmo realizará la multiplicación de los diferentes productos.

Finalmente, con la meta de mostrar o imprimir el resultado de dicha operación, se ejecuta el procedimiento denominado como **mostrar-resultado**, el cual es el encargado de realizar dicha tarea. Cabe destacar que como se indicó anteriormente, adicionalmente se cuenta con un archivo que incluye algunos prototipos los cual son esenciales para el flujo correcto del programa.

V. Conclusión

En conclusión, queda evidenciado que si bien el modo de operación de un lenguaje ensamblador como lo es MASM, es en gran medida distinto a un lenguaje de alto nivel como Python, existe la posibilidad de llevar a cabo aplicaciones, u operaciones que brindaran resultados de forma similar. A su vez, queda más que clara la gran utilidad que proporcionan los corrimientos a nivel del lenguaje ensamblador en relación con el manejo de operaciones de multiplicación o división. En ese sentido, el presente programa es un ejemplo de ello. Finalmente, se destaca también en relación con la elaboración en general del código, la documentación, comentarios y guías con las que cuenta el mismo, como un punto alto y uno de los aspectos más importantes en al momento de retomar el código, o que este sea analizado u utilizado por un tercero.

VI. Diagrama de flujo ensamblador

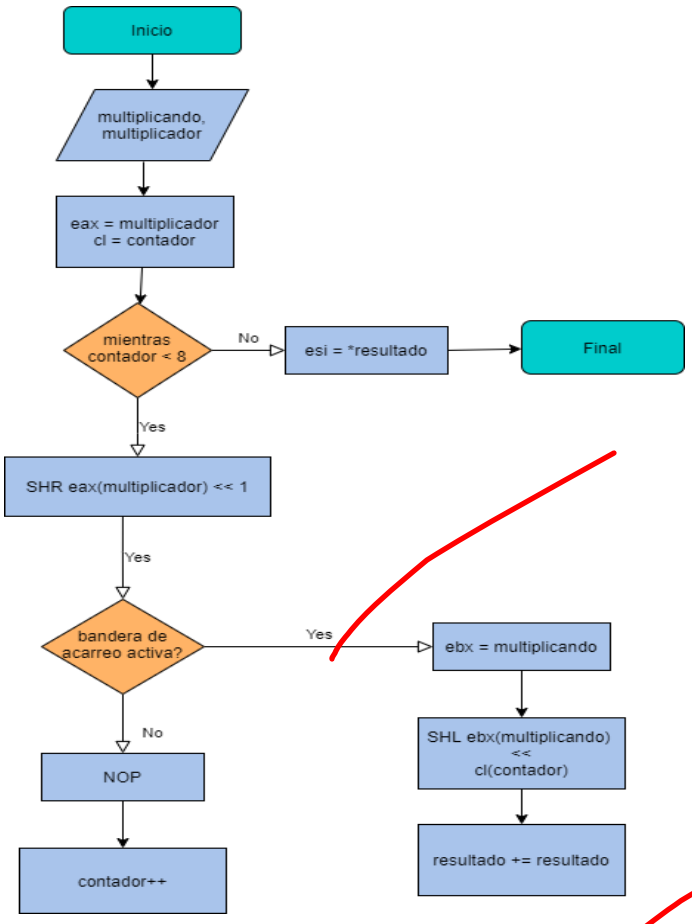

Fig. 2. Diagrama de flujo del algoritmo en ensamblador.

References

[1] Irvine, Kip R. (2007). LENGUAJE ENSAMBLADOR PARA COMPUTADORAS BASADAS EN INTEL. Florida: PEARSON Educación.

Falta un poco de claridad en la explicación del algoritmo.