Proyecto Programado 1

1st Eric Alpizar Prendas
Carné 2020426169

2nd Rodrigo Espinach Hernández
Carné 2014055978

3rd Jimmy Salas Hernández
Carné 2019031383

***Abstract*—A continuación se desarrollará la documentación correspondiente para la solución del primer poryecto progrmado del curso Arquitectura de Computadores de la carrera Ingeniería en Computación impartida en el Instituto Tecnológico de Costa Rica. La meta de este proyecto fue desarrollar un algoritmo en el lenguaje de programación MASM y que este fuera capaz de realizar multiplicaciones enteras, sin signo y de 8 bits. Esto mediante la técnica de de corrimientos lógicos a la izquierda.**

I. Introducción

Para la realización de este proyecto, se utilizó específicamente el lenguaje ensamblador Microsoft Macro Assembler (MASM), este emplea la sintaxis de Intel para operar en Windows. Además, se utilizó el lenguaje de programación Octave, cuyo principal propósto es realizar cálculos numéricos, para diseñar un prototipo del algoritmo previo a la programación en ensamblador.

El prototipado en Octave fue una herramienta necesaria para idear el algoritmo y que este se apegara a los requisitos del mismo, además, el hecho de ser de alto nivel y que su propósito sea representar cálculos numéricos sirvió de guía a la hora de laborar en ensamblador.

```
res = 0
cont = 0

num_1 = 7
num_2 = 4

while(num_2)
  if(bitand(num_2,1) == 1)
    res += bitshift(num_1, cont)
  endif

  cont += 1
  num_2 = bitshift(num_2, -1)
endwhile

disp('The result is: '), disp(res)
```

Un archivo con el código comentado se facilitará en GitHub.

II. Desarrollo

El programa hace uso de tres archivos para completar su funcionamiento cuyos nombres son: uint8_mult.asm, cli.inc y por último main.asm.

- El archivo **main.asm** es el encargado de invocar a ejecución a los demás archivos de código. Aquí es donde se analizan los datos de entrada que el usuario previamente ingresó y a partir de ellos determinar lo que el usuario desea hacer con el programa, además, maneja el resultado de la operación de multiplicación, retornada de *uint8_mult.inc*, para después, mediante el código escrito en el archivo *cli.inc* mostrarlo en la interfaz del usuario.
- En el archivo **cli.inc** se encuentra el código responsable de desplegar la interfaz de usuario, en el que este puede ingresar los números correspondientes al multiplicando y multiplicador para que estos sean operados. El mismo es el encargado de mostrar el producto de la operación en sistema decimal, para esto, hace un llamado a *WriteDec* de la librería Irvine. Por último, se encarga de preguntar si se desea realizar otra operación. Si la respuesta es equivalente a ”Y" o ”y", se correrá una nueva ejecición del programa, por el contrario, si es ”N" o ”n", se dará por finaliazada la ejecución del programa.
- Por último, el archivo **uint8_mult.asm** es el encargado de operar los números ingresados por el usuario. Este algoritmo consiste en la ejecución de un ciclo mientras que el valor multiplicador sea mayor que cero.

  Primeramente su valor original es movido a una variable temporal y sobre este se aplicará la operación módulo para comprobar si es un número par, si esto es verdadero, el programara ejecturá el código contenido en la etiqueta *_shift_left* en la que, al valor multiplicando se le correrán una cantidad de bits representado por el registro *cl* y el resultado es guardado en la variable *res*.

  Si el resultado de la operación modulo es cero, el registro *cl* es incrementado en 1 unidad y el multiplicador es divido entre 2, para esto, se corren los bits una única vez hacia la derecha. A partir de este punto se verificará si el multiplicador es cero para decidir si ejecuta la etiqueta *_while* (se sigue en el ciclo) o el algoritmo ha terminado.

El siguiente diagrama es una repodrucción del diagrama presente en el libro *Assembly Language for Intel-Based Computers* en el que se ejemplifica la estrutura del programa solución para el presente proyecto, los cuadros: WriteString, CrLf, WriteChar, ReadChar y ReadInt corresponden a procedimientos tomados de la librería de Irvine. Mientras que los cuadros en color celeste representan prototipos de funciones.

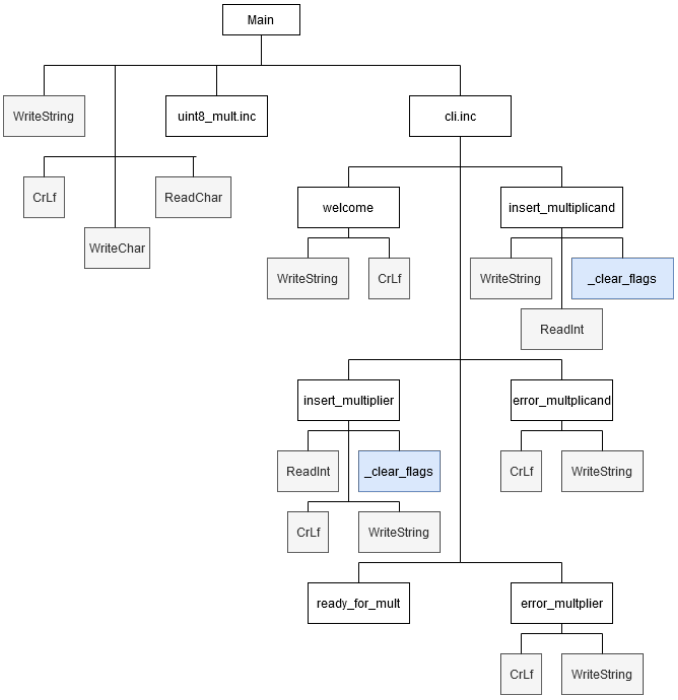

Fig. 1. Diagrama de estructura del programa.

III. Conclusiones

- Los archivos con código de un programa escrito en lenguaje ensamblador requieren una mayor cantidad de instrucciones para llevar a cabo una tarea que de ser desarrollada en un lenguaje de alto nivel, necesitaría un número muy reducido de líneas con instrucciones.
  Su relación directa con el hardware requiere de indicaciones más específicas y como consecuencia escribir código en ensamblador requiere más tiempo y limita la productividad.
- El lenguaje ensamblador presenta una mayor velocidad a la hora de ejecución, esto cuando se compara con otros lenguajes de alto nivel, su aproximación con el hardware y los recortes en tiempo que implica que el código no sea compilador ni interpretado juegan a favor del rendimiento en ensamblador.
- El lenguaje ensamblador permite un manejo directo con la memoria y los registros que el procesador pone a disposición, lo que implica un mejor y más eficiente uso de estos.
- Entender sistemas numéricos binario, octal (base 8) y hexadecimal (base 16) resulta de gran importancia a la hora de estudiar el funcionamiento de un computador en en sus niveles más bajos y primitivos.

References

[1] Kip R. Irvine, "Assembly Language for Intel-Based Computers," 4th ed. Upper Saddle River, New Jersey, Prentice Hall, 2002.

[2] "About," The GNU Operating System and the Free Software Movement. [Online]. Available: https://www.gnu.org/software/octave/about. [Accessed: 15-Nov-2020].

[3]