**Laboratorio 3 Electrónica Digital II**

**Alexander Passo**

**Víctor Manuel Jiménez García**

**John Byron Buitrago**

**Docente**

****

**Universidad de Antioquia**

**Departamento de Ingeniería electrónica y telecomunicaciones Ingeniería electrónica**

**Medellín 2022**

**Abstract**

Para esta práctica se usará ‘assembler’ con el fin de llevar el concepto que se tiene de alto nivel a uno de bajo nivel, esto brindará un mayor conocimiento acerca de cómo funcionan internamente los programas que se desarrollan regularmente, para este caso en particular, se realizarán algunas operaciones como las de ordenar datos en un orden específico y hallar los múltiplos de un número en una lista.

Con el objetivo de entender correctamente todo el procedimiento que se llevará a cabo en el lenguaje ensamblador, es indispensable desarrollar una serie de ideas en alto nivel que conduzcan a una solución óptima, es por esta razón que se optó primeramente por plantear el problema de una forma más familiar, es decir, en lenguaje de alto nivel y pseudocódigos, donde luego se pudo extraer una posible solución y llevarla a un lenguaje de bajo nivel.

Luego de tener el algoritmo, se implemento a partir de la herramienta de desarrollo ARM de bajo nivel, que a partir de registros y direccionamiento de memoria es capaz de realizar el procedimiento que se necesita, ordenamiento de datos y encontrar múltiplos de un numero en base de una lista dada en memoria.

Para el ordenamiento de los datos, se basó en el método de la burbuja que permite comparar datos en posiciones adyacentes e intercambiar los datos si cumplían una condición, mayor o menor según fuese el caso. Mientras que, para encontrar los múltiplos de un número, se procedió a partir de restas sucesivas hasta que el resultado llegara a cero se concluía que el dato era múltiplo del número dado.

**Pseudocódigo**

Como ya se mencionó anteriormente los métodos utilizados para la realización de la práctica, se mostrarán a continuación en forma de algoritmos, en donde se utilizo el lenguaje de Python para lograr entender su funcionamiento y lograr extraer un posible pseudocódigo.

|  |
| --- |
| # DESCENDENTE  v=[100,5,2,7,1,4,-100,8,13,15,20,22,24]  band = False  while band == False:  band = True  for i in range(len(v)-1):  a=v[i]  b=v[i+1]  if a<b:  v[i] = b  v[i+1] = a  band = False  print(v) |

**Tabla 1 Código en Python para organizar datos de forma ascendente.**

|  |
| --- |
| # ASCENDENTE  v= [100,5,2,7,1,4,-100,8,13,15,20,22,24]  band = False  while band == False:  band = True  for i in range(len(v)-1):  a=v[i]  b=v[i+1]  if a>b:  v[i] = b  v[i+1] = a  band = False  print(v) |

**Tabla 2 Código en Python para organizar datos de forma ascendente.**

|  |
| --- |
| # MULTIPLO  v = [100,-35,2,7,1,4,-100,8,13,15,21,22,24]  d=7  mult = []  for i in range(len(v)):  a=v[i]  s=a  a=abs(a)  for j in range (max(v)):  a=a-d  if a == 0:  mult.append(s)  print(mult) |

**Tabla 3 Código en Python para encontrar múltiplos de un numero a partir de un vector de datos.**

A partir de los códigos realizados en Python, fue posible realizar el algoritmo para cada uno de ellos.

**Pseudocódigo para ordenamiento de datos descendente y ascendente.**

1. Se asigna la primera dirección del vector de Datos
2. Se carga el valor máximo de N
3. Se define una condición para el for: Este caso, un for NxN más una constante. Para garantizar que se hagan todos los ciclos necesarios
4. Dentro del for: Se carga el primer y segundo valor de la lista de Datos. (Dos registros)
5. Definir una condición, que me permita comparar dos valores: Realizar una resta entre los dos valores, si es negativo o positivo se intercambian las posiciones según fuese el caso.
   1. Si se quieren organizar los datos de forma ascendente, y si la comparación da positivo se intercambian los valores, y si da negativo, se dejan los valores tal cual.
   2. Si se quieren organizar los datos de forma descendente, y si la comparación da negativo se intercambian los valores, y si da positivo, los valores se dejan tal cual en la posición actual.
6. Para garantizar que se recorra todo el vector, se hace una resta de tal forma que el índice llegue a cero.
7. Y para terminar el for general, se decrementa el valor NxN hasta llegar a cero.
8. Y por último se visualizará el vector de Datos ordenado de forma ascendente o descendente.

**Pseudocódigo para encontrar múltiplos de un numero M.**

1. Cargar la primera dirección del vector Datos que permite recorrer el vector, y cargar la dirección de memoria del valor M
2. Luego, cargar los valores desde la dirección de memoria (Para M y para el primer dato de la lista)
3. Se define un for, que garantice recorrer todo el vector de la lista y compararlo con el numero M.
4. La comparación entre el numero M y el dato, se hace en restas sucesivas, ya que, si la resta llega a cero, se concluye que el dato es múltiplo de M
5. Para los números negativo, se tiene en cuenta que están en complemento a2, entonces se convierten a número positivo, invirtiendo los bits y sumando un bit.
6. Si el dato es múltiplo, entonces el valor se guarda en una lista.

Por último, se planteó un pseudocódigo que permite guardar los datos ordenados o los múltiplos del numero M, el cual se partió desde el vector principal de los datos y se apuntaba a una nueva dirección de memoria dada por DatosOut, y se cargaba cada valor a su posición correspondiente, variando las posiciones de 4 en 4.

# Conclusiones

* En esta práctica se encontraron instrucción que ayudaron a la simplificación del código aplicado, por ejemplo, STRPL, STRMI, que comparaban un resultado de signos y guardaban el resultado si se cumplía una condición.
* Cabe resaltar que para la implementación en ARM de los problemas planteados, fue posible resolverlos a partir de operaciones aritméticas como la resta y suma que ayudaban a controlar una condición.
* Para encontrar los múltiplos de un número, se tuvo en cuenta el signo, ya que, si el dato era negativo, se debía pasar a positivo para poder realizar la resta. Para ello, se utilizó la instrucción MVNMI, que permite guardar números negativos y realizar una OR exclusiva de tal manera que cambiara los bits, y luego con la instrucción ADDMI, se le suma 1 para llevar el numero a positivo.
* Es necesario ser recursivos a la hora de solucionar un problema, ya que a falta de la operación de la división en cpulator se acudió a resolverlo mediante el uso de las restas sucesivas y algunas condiciones más.
* Estar bien documentado sobre el lenguaje que se está trabajando ayudó a dar facilidad al desarrollo de la práctica pues permitió conocer ciertos comandos que son muy útiles para ciertas circunstancias donde no parecía posible seguir trabajando.