



UNIVERSIDAD DE CARABOBO
FACULTAD EXPERIMENTAL DE CIENCIAS Y
TECNOLOGÍA
DEPARTAMENTO DE COMPUTACIÓN
ARQUITECTURA DEL COMPUTADOR



Practica 2

Icler Anaya — C.I. 24.450.482
Wilsen Hernandez — C.I. 24.993.998

Introducción

En este informe podrá ser visualizado un contenido breve de la **Práctica n°3** de la materia Arquitectura del Computador, donde serán explicados algunos elementos que son utilizados al momento de implementar el método de ordenamiento Burbuja bajo el lenguaje de ensamblador; se podrán observar algunos aspectos importantes como el uso de memoria o las direcciones de memoria a utilizar, escribir y leer en estos dichos espacios definidos (es decir que los elementos tengan un límite superior e inferior) entre otras características de este lenguaje.

Algo importante para poder comprender la implementación realizada es tener claro cómo funciona el método de búsqueda de burbuja el cual revisa cada elemento de la lista que va a ser ordenada con el siguiente elemento, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que ***“La lista está ordenada”***.

Adicionalmente en esta práctica se maneja el uso de la pila (**Stack**), procedimientos por la cual el código tiene una complejidad mayor que en la **Práctica n°2** debido a la implementación de nuevas instrucciones.

La forma metodológica de abordar el problema es la experimental, ya que de esta forma es posible evaluar los procesos, observar los criterios para el armado del programa y conocer el código que hace posible la resolución del problema, además conduce a lograr un mayor dominio en cuanto al lenguaje ensamblador del microprocesador **Intel 8051** mediante la programación de este método de ordenamiento.

Marco Teórico

El problema

El problema consiste en realizar en lenguaje ensamblador para **micro-controlador 8051** un programa que implemente en memoria, dado un vector compuesto de **n** caracteres **ASCII** en el rango **A-Z**, el algoritmo de ordenamiento de burbuja haciendo uso de la pila (**Stack**) y realizando llamadas a procedimientos. El primer elemento del vector a ordenar estará ubicado en la dirección **30H** de la memoria **RAM**, validando que la cantidad de elementos a ordenar (**n**) no supera la dirección **7FH**, es decir la cantidad máxima de elementos aceptados es **4FH (hexadecimal)** lo cual equivale a una cantidad de 79 elementos (**enteros**). Además, se tiene que validar que los elementos del vector se encuentren en el rango de letras mayúsculas **A- Z**. A su vez, se debe tener en cuenta que el parámetro **n** estará localizado en el registro **R5** y que el programa intercambie posiciones de los elementos del vector original en el mismo lugar donde el vector reside. adicionalmente se implementará el **Sub-programa Random** para conseguir los elementos que ocupan desde la posición **30H** hasta la **7FH** de manera aleatoria y el algoritmo de ordenamiento deberá llamar a los siguientes procedimientos:

```
void intercambiar(int a[], k)
{
    int aux = a[k];
    a[k] = a[k+1];
    a[k+1] = aux;
}
```

```
void incrementar_BCD(posicion_memoria)
{
    M[posicion_memoria] = M[posicion_memoria] + 1;
}
```

Objetivos

- Conocer el uso de procedimientos en el **8051**
- Identificar los modos de direccionamiento de los registros **R0** y **R1**, y utilizar dichos modos de direccionamiento
- Comprender el funcionamiento de la pila (**Stack**).
- Entender la relación de la pila con las instrucciones de llamadas a procedimientos.
- Realizar un programa con el uso intensivo de la pila y llamadas a procedimientos.

Aspectos teóricos importantes

Para el desarrollo de esta practica, se hace uso del lenguaje ensamblador para el micro-controlador Intel 8051, como se menciono anteriormente, el cual es un lenguaje de programación de bajo nivel que implementa una representación simbólica de los códigos de máquina binarios y otras constantes necesarias para programar una arquitectura dada de **CPU**. Constituye la representación más directa del código máquina específico para cada arquitectura legible por un programador.

También se hace uso del algoritmo de ordenamiento Burbuja, el cual funciona revisando cada

elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "*burbujas*".

Estas dos herramienta también las hemos acompañado con el uso de la memoria **RAM** (*La memoria de acceso aleatorio, por su nombre en ingles random-access memory*) se utiliza como memoria de trabajo para el sistema operativo, los programas y la mayor parte del software. Es allí donde se cargan todas las instrucciones que ejecutan el procesador y otras unidades de cómputo. Se denominan «*de acceso aleatorio*» porque se puede leer o escribir en una posición de memoria con un tiempo de espera igual para cualquier posición, no siendo necesario seguir un orden para acceder a la información de la manera más rápida posible.

Específicamente en esta memoria estaremos trabajando en la dirección **30H**, que se visualiza en el área 2 (**marco azul**) del simulador del **micro-controlador DOSBox 8051**. Y la cual representa la primera posición del vector de caracteres.

También esta la dirección **7FH**, la cual también es una dirección específica de la memoria **RAM** y la cual se puede visualizar en el simulador del **micro-controlador DOSBox 8051**. Esta dirección representa el límite superior del vector por el cual la cantidad de elementos a ordenar no debe superar dicha dirección.

Adicional a estas dos direcciones específicas de la memoria **RAM**, también trabajaremos con las direcciones **2EH/2FH**, las cuales se encuentran ubicadas en el área 2 (marco azul) del simulador del **micro-controlador DOSBox 8051**. En las direcciones **2EH** y **2FH** se almacenara el número total de intercambios realizados durante el proceso de ordenamiento, como un número de **16 bits**.

En la memoria **RAM** encontraremos una cantidad de registros, entre ellos encontramos el **Registro R5**. El cual es un registro específico de la memoria **RAM**, se encuentra ubicado en el área 1 (**marco amarillo**) del simulador del **micro-controlador DOSBox 8051**. En dicho registro se almacena el valor de **N** el cual puede ser validado también en la dirección **05H** ya que está asociada **R5**.

Pila (Stack)

Es una lista ordenada o **estructura de datos** en la que el modo de acceso a sus elementos es de tipo **LIFO** (*del inglés Last In First Out, último en entrar, primero en salir*) que permite almacenar y recuperar datos.

Procedimientos (subrutina)

En computación, una subrutina o subprograma (*también llamada procedimiento, función o rutina*), como idea general, se presenta como un bloque de código definido externamente pero que forma parte del algoritmo principal invocando a la función, el cual permite resolver una tarea específica.

Grupo de Comandos utilizados

Instrucción: PUSH

Instrucción: PUSH
Función: Apilar un byte de RAM interna
Sintaxis: PUSH direcc

Instrucción	Código de Operación	2º Byte	Bytes	Ciclos	Flags
PUSH direcc	0xC0	direcc	2	2	-

Operación: PUSH direcc

$(SP) \leq (SP) + 1$
 $((SP)) \leq (\text{direcc})$

Descripción: PUSH almacena el valor de una dirección de **RAM** interna en la pila. En primer lugar incrementa en 1 el valor de **SP (Stack Pointer)**, y después guarda el valor de la **dirección de RAM** interna implicada, en la posición apuntada por el **SP** ya incrementado.

En resumen esta operación toma como argumento un registro general, de segmento o una posición de memoria y la introduce (**copia**) al Stack.

Ejemplo: PUSH A

Instrucción: POP

Instrucción: POP
Función: Desapilar un byte hacia RAM interna
Sintaxis: POP direcc

Instrucción	Código de Operación	2º Byte	Bytes	Ciclos	Flags
POP direcc	0xD0	direcc	2	2	-

Operación: POP direcc

$(\text{direct}) \leq ((SP))$
 $(SP) \leq (SP) - 1$

Descripción: **POP** copia el contenido de la posición de **RAM** interna direccionado por el **SP (Stack Pointer)**, en la **dirección de RAM** interna que indica el segundo byte de la instrucción. El valor del **SP** se decrementa en 1.

POP al igual que la anterior, toma como argumento un registro o una celda de memoria. Pero lo que hace es que toma el valor almacenado en el tope de la pila y lo guarda en dicho lugar.

Ejemplo: **POP** 34h

Instrucción: LCALL

Instrucción: LCALL

Función: Llamada larga

Sintaxis: LCALL *dir_16*

Instrucción	Código de Operación	Byte 2º	Byte 3º	Bytes	Ciclos	Flags
LCALL <i>dir_16</i>	0x12	dir 15-8	dir 7-0	3	2	-

Operación: LCALL

(PC) <= (PC) + 3
(SP) <= (SP) + 1
((SP)) <= (PC.7-0)
(SP) <= (SP) + 1
((SP)) <= (PC.15-8)
(PC) <= addr15-0

Descripción: **LCALL** realiza una llamada incondicional a la subrutina situada en la dirección indicada. **LCALL** incrementa el **PC (Program Counter)** tres veces para obtener la dirección de la siguiente instrucción, luego guarda dicha dirección en la pila (**el byte de menor peso en primer lugar**). En consecuencia el apuntador de pila (**SP o Stack Pointer**) incrementa su valor en 2. Posteriormente el control del programa se transfiere a la dirección indicada en la instrucción.

Ejemplo: **LCALL** SUB1

Instrucción: ACALL

Instrucción: ACALL

Función: Llamada absoluta dentro de un bloque de 2K

Sintaxis: ACALL *dir_11*

Instrucción	Código de Operación	2º Byte	Bytes	Ciclos	Flags
ACALL <i>dir_11</i> a10a9a810000		dir 7-0	2	2	-

Operación: ACALL

$(PC) \leq (PC) + 2$
 $(SP) \leq (SP) + 1$
 $((SP)) \leq (PC7-0)$
 $(SP) \leq (SP) + 1$
 $((SP)) \leq (PC15-8)$
 $(PC10-0) \leq \text{dir_11}$

Descripción: **ACALL** realiza una llamada incondicional a la subrutina situada en la dirección indicada. **ACALL** incrementa el **PC (Program Counter)** dos veces para obtener la dirección de la siguiente instrucción, luego guarda dicha dirección en la pila (**el byte de menor peso en primer lugar**). En consecuencia el apuntador de pila (**SP o Stack Pointer**) incrementa su valor en 2. Posteriormente el control del programa se transfiere a la dirección indicada en la instrucción.

La dirección de salto, o nuevo valor para el **PC** se obtiene uniendo a los 5 bits de mayor peso del **PC** ya incrementado, los bits **7-5** del código de operación y el segundo byte de la instrucción.

Como la instrucción **ACALL** sólo afecta a los 11 bits de menor peso del **PC**, la llamada siempre se produce a una dirección de memoria de código situada dentro del bloque de **2K** al que pertenece el primer byte de la instrucción que sigue al **ACALL**.

Ejemplo: ACALL LABEL

Instrucción: RET

Instrucción: RET
Función: Retorno desde subrutina
Sintaxis: RET

Instrucción	Código de Operación	Bytes	Ciclos	Flags
RET	0x22	1	2	-

Operación: RET

$(PC15-8) \leq ((SP))$
 $(SP) \leq (SP) - 1$
 $(PC7-0) \leq ((SP))$
 $(SP) \leq (SP) - 1$

Descripción: **RET** se utiliza para retornar desde una subrutina llamada previamente con **LCALL** o **ACALL**. La ejecución del programa continúa desde la dirección formada al extraer 2 bytes de la pila. En primer lugar de la pila se saca el byte más significativo.

Ejemplo: RET

Procedimientos

En esta oportunidad se tomó como base la **práctica n°2**, en la cual se implementó el algoritmo de ordenamiento de burbuja en lenguaje **ensamblador para el micro-controlador 8051**; Primeramente se esbozó en pseudocódigo del algoritmo de ordenamiento (**practica 2**) para manejar la dinámica de cómo trabaja el algoritmo.

Previamente se estudió detalladamente el conjunto de entradas en el programa de simulación de ordenamiento de burbuja, como lo son el tamaño del vector (**N**) ubicado en el registro **R5**, los registros **R2** y **R3** para simular los índices '**i**' y '**j**' respectivamente que sirven para ir iterando y desplazándose por todo el vector, los registros **2EH/2FH** para almacenar el número de intercambios realizados en el proceso de ordenamiento. Se trabajó con llamadas a procedimientos y se implementó el uso de la pila de memoria.

Resultados y Análisis

El algoritmo principal cuenta con el llenado del vector haciendo uso del procedimiento random, que genera cada una de las letras del vector. Luego se valida si cada una de las direcciones del vector se encuentran en el rango **A-Z**. Luego se llama al procedimiento de ordenamiento que a su vez llama al procedimiento de intercambio de elementos en el caso que ambos elementos que se estén comparando se encuentren en las posiciones equivocadas.

Al finalizar el programa, si el vector contiene elementos válidos, genera el vector ordenado. En caso contrario, se muestra en pantalla un mensaje de error. Los resultados obtenidos en esta practica se obtuvieron realizando los siguientes pasos:

Se asigna valores a los registros R2, R3, R4, R5 para iniciar programa.

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: AVSIM51
LABEL OPERATION 8051/8751 AUSIM 8051 Simulator/Debugger U1.31
PRINC MOV R5,#2BH CPU REGISTERS FLAGS SCL SPD DSP SKP CURSOR
0103H MOV R6,#2AH C Accumulator AC FO OV P OFF HI ON OFF MENU
0106H MOV R2,#00H 0 00000000:00: 0 0 0 0 Cycles:
0109H MOV R3,#00H addr data
010CH MOV 2EH,#00H PC:0109 > 75 02 00 75 Timers TH/TL TF/TR G/T/M1/M0
010FH MOV 2FH,#00H SP: 07 > 00 00 00 00 TO: 00 00 0 0 0 0 0 0
0112H MOV A,R5 00 2B 2A 00 T1: 00 00 0 0 0 0 0 0
0113H MOV R4,A DP:0000 > 00 FF FF FF
0115H DEC R5 R0:00: > 00: RB:00 Ints A S T1 X1 T0 X0 Edg IT IE
0116H MOV R0,#30H R1:00: > 00: B:00 En 0 0 0 0 0 0 0 0 X0: 0 0
0118H MOV A,R5 R2:00 R4:00 R6:00 Pr 0 0 0 0 0 0 0 0 X1: 0 0
0119H MOV 2DH,A R3:00 R5:2B R7:00 SBUF: In Out PCON:0.00000000
LLENAR LCALL RANDOM Data Space 00: 00: SCON:00000000
011EH MOV A,2DH 0000 00 00 00 00 00 2B 2A 00 ** Ports
0120H MOV B,#1AH 0008 00 00 00 00 00 00 00 00 P0 11111111
0123H DIV AB 0010 00 00 00 00 00 00 00 00 FF: 11111111
0124H MOV A,B 0018 00 00 00 00 00 00 00 00 P1 11111111
0126H ADD A,#41H Data Space FF: 11111111
0128H MOV @R0,A 0020 00 00 00 00 00 00 00 00 P2 11111111
0129H INC R0 0028 00 00 00 00 00 00 00 00 FF: 11111111
FLENAR DJNZ R4,LLENAR 0030 00 00 00 00 00 00 00 00 P3 11111111
VALI CLR A 0038 00 00 00 00 00 00 00 00 FF: 11111111
>Select Command - or use arrow keys
Dump Expression commandFile Help IO Load --space-- ESC to screen

```

Se llena el vector con la semilla generada

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: AVSIM51
LABEL OPERATION 8051/8751 AUSIM 8051 Simulator/Debugger U1.31
RANDOM PUSH R0 CPU REGISTERS FLAGS SCL SPD DSP SKP CURSOR
01CEH PUSH R1 C Accumulator AC FO OV P OFF HI ON OFF MENU
RAND8 MOV A,2DH 0 01001100:4C:L 0 0 0 1 Cycles:
01D2H JNZ RAND8B addr data
01D4H CPL A PC:01CE > C0 00 00 01 Timers TH/TL TF/TR G/T/M1/M0
01D5H MOV 2DH,A SP: 0A > 2A 2A 00 1E TO: 00 00 0 0 0 0 0 0
RAND8B ANL A,#B8H 01 34 00 00 T1: 00 00 0 0 0 0 0 0
01D9H MOV C,P DP:0000 > 00 FF FF FF
01DBH MOV A,2DH R0:00: > 00: RB:00 Ints A S T1 X1 T0 X0 Edg IT IE
01DDH RLC A R1:00: > 34: B:00 En 0 0 0 0 0 0 0 0 X0: 0 0
01DEH MOV 2DH,A R2:00 R4:27 R6:20 Pr 0 0 0 0 0 0 0 0 X1: 0 0
01E0H POP R1 R3:00 R5:2A R7:00 SBUF: In Out PCON:0.00000000
01E2H POP R0 Data Space 00: 00: SCON:00000000
01E4H RET 0000 34 00 00 00 27 2A 2A 00 4 ** Ports
FIN NOP 0008 1E 01 34 00 00 00 00 00 00 P0 11111111
01E6H no memory 0010 00 00 00 00 00 00 00 00 FF: 11111111
01E7H no memory 0018 00 00 00 00 00 00 00 00 P1 11111111
01E8H no memory Data Space FF: 11111111
01E9H no memory 0020 00 00 00 00 00 00 00 00 P2 11111111
01EAH no memory 0028 00 00 00 00 00 A7 00 00 FF: 11111111
01EBH no memory 0030 47 4E 46 4C 00 00 00 00 GNFL P3 11111111
01ECH no memory 0038 00 00 00 00 00 00 00 00 FF: 11111111
>Select Command - or use arrow keys
Dump Expression commandFile Help IO Load --space-- ESC to screen

```

Se hacen las comparaciones e intercambio de las posiciones

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: AVSIM51
LABEL OPERATION 8051/8751 AUSIM 8051 Simulator/Debugger U1.31
INTER PUSH R0 CPU REGISTERS FLAGS SCL SPD DSP SKP CURSOR
01BDH PUSH R1 C Accumulator AC FO OV P OFF HI ON OFF HEX
01BFH MOV A,0R0 1 11111110:FE: 0 0 0 1 Cycles:
01C0H MOV R7,A addr data
01C2H MOV A,0R1 PC:01BB > 00 00 C0 01 Timers TH/TL TF/TR G/T/M1/M0
01C3H MOV 0R0,A SP: 0D > 47 2A 76 01 T0: 00 00 0 0 0 0 0 0
01C4H MOV A,R7 31 32 00 00 T1: 00 00 0 0 0 0 0 0
01C6H MOV 0R1,A DP:0000 > FF FF FF FF
01C7H POP R1 R0:32:2 > 4E:0 RB:00 Ints A S T1 X1 T0 X0 Edg IT IE
01C9H POP R0 R1:31:1 > 4C:1 B:00 En 0 0 0 0 0 0 0 0 X0: 0 0
01CBH RET R2:00 R4:00 R6:20 Pr 0 0 0 0 0 0 0 0 X1: 0 0
RANDOM PUSH R0 R3:02 R5:20 R7:41 SBUF: In Out PCON:0.0000000
01CEH PUSH R1 Data Space 00: 00: SCON:00000000
RAND8 MOV A,2DH 0000 32 33 00 02 00 2A 2A 4E 23 0 **N Ports
01D2H JNZ RAND8B 0008 38 01 47 2A 76 01 31 32 80G**012 P0 11111111
01D4H CPL A 0010 00 00 00 00 00 00 00 00 FF: 11111111
01D5H MOV 2DH,A 0018 00 00 00 00 00 00 00 00 P1 11111111
RAND8B ANL A,#B8H Data Space:R7\+19H FF: 11111111
01D9H MOV C,P 0020 00 00 00 00 00 00 00 00 P2 11111111
01DBH MOV A,2DH 0028 00 00 00 00 00 A3 00 01 ú 0 FF: 11111111
01DDH RLC A 0030 47 46 4E 4C 41 42 48 50 GFNLABHP P3 11111111
01DEH MOV 2DH,A 0038 45 4E 46 4F 43 4A 58 5A ENFOCJXZ FF: 11111111
>Configure Memory Dump Windows ESC to menu

```

Incrementa el contador de 2EH/2FH para saber el número de intercambios realizados

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: AVSIM51
LABEL OPERATION 8051/8751 AUSIM 8051 Simulator/Debugger U1.31
INCON1 MOV A,2FH CPU REGISTERS FLAGS SCL SPD DSP SKP CURSOR
0178H INC A C Accumulator AC FO OV P OFF HI ON OFF HEX
0179H CLR C 1 00010000:10: 0 0 0 1 Cycles:
017AH CLR AC addr data
017CH DA A PC:0178 > E5 2F 04 C3 Timers TH/TL TF/TR G/T/M1/M0
017DH MOV 2FH,A SP: 0B > 47 2A 76 01 T0: 00 00 0 0 0 0 0 0
017FH CJNE A,#99H,INCRJ 3C 3D 00 00 T1: 00 00 0 0 0 0 0 0
0182H MOV 2FH,#00H DP:0000 > FF FF FF FF
0185H CLR A R0:3C: > 40:J RB:00 Ints A S T1 X1 T0 X0 Edg IT IE
0186H MOV A,2EH R1:3D: > 50:F B:00 En 0 0 0 0 0 0 0 0 X0: 0 0
0188H INC A R2:00 R4:00 R6:20 Pr 0 0 0 0 0 0 0 0 X1: 0 0
0189H CLR C R3:0C R5:20 R7:50 SBUF: In Out PCON:0.0000000
018AH CLR AC Data Space 00: 00: SCON:00000000
018CH DA A 0000 3C 3D 00 0C 00 2A 2A 50 <= 0 **P Ports
018DH MOV 2EH,A 0008 38 01 47 2A 76 01 3C 3D 80G**0<= P0 11111111
018FH LJMP INCRJ 0010 00 00 00 00 00 00 00 00 FF: 11111111
INCRJ INC R3 0018 00 00 00 00 00 00 00 00 P1 11111111
0194H CLR C Data Space:R7\+19H FF: 11111111
0195H MOV A,R3 0020 00 00 00 00 00 00 00 00 P2 11111111
0197H CJNE A,R6,COND 0028 00 00 00 00 00 A3 00 10 ú 0 FF: 11111111
INCRJ INC R2 0030 47 46 4C 41 42 48 4E 45 GFLABHNE P3 11111111
019CH CLR C 0038 4E 46 4F 43 4A 50 58 5A NFOCJPXZ FF: 11111111
>Configure Memory Dump Windows ESC to menu

```

Programa completado burbuja hecho con su número de intercambios realizados

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: AVSIM51
LABEL OPERATION 8051/8751 AUSIM 8051 Simulator/Debugger V1.31
2A47H no memory CPU REGISTERS FLAGS SCL SPD DSP SKP CURSOR
2A48H no memory C Accumulator AC F0 00 P OFF HI ON OFF HEX
2A49H no memory 0 00101010:2A:* 0 0 0 1 Cycles:
2A4AH no memory addr data
2A4BH no memory PC:2A47 > FF FF FF FF Timers TH/TL TF/TR G/T/M1/M0
2A4CH no memory SP: 09 > 2A 43 38 01 T0: 00 00 0 0 0 0 0 0
2A4DH no memory 47 2A 76 01 T1: 00 00 0 0 0 0 0 0
2A4EH no memory DP:0000 > FF FF FF FF
2A4FH no memory R0:00:0 > 00:0 RB:00 Ints A S T1 X1 T0 X0 Edg IT IE
2A50H no memory R1:0A:2 > 0A:2 B:00 En 0 0 0 0 0 0 X0: 0 0
2A51H no memory R2:2A R4:00 R6:2A Pr 0 0 0 0 0 X1: 0 0
2A52H no memory R3:2A R5:2A R7:4 SBUF: In Out PCON:0.0000000
2A53H no memory Data Space 00: 00: SC0N:00000000
2A54H no memory 0000 59 5A 2A 2A 00 2A 2A 43 YZ** **C Ports
2A55H no memory 0008 38 01 47 2A 76 01 32 33 80G*UE23 P0 11111111
2A56H no memory 0010 00 00 00 00 00 00 00 00 FF: 11111111
2A57H no memory 0018 00 00 00 00 00 00 00 00 P1 11111111
2A58H no memory Data Space:R7\+19H FF: 11111111
2A59H no memory 0020 00 00 00 00 00 00 00 00 P2 11111111
2A5AH no memory 0028 00 00 00 00 00 A3 03 73 ús FF: 11111111
2A5BH no memory 0030 41 42 42 43 43 43 44 44 ABBCCCDD P3 11111111
2A5CH no memory 0038 45 45 46 46 46 47 48 48 EEEFFGHH FF: 11111111
>Configure Memory Dump Windows
Trap: Undefined Address: C:2A47H

```

Conclusión

Para concluir, en esta ocasión con la implementación del método de ordenamiento de burbuja en el lenguaje **ensamblador para el micro-controlador Intel 8051**, podemos ver cómo se trabaja y como nos podemos mover por la memoria, con dicho método. Al tratar de implementar la el método de ordenamiento de burbuja se investigaron algunas instrucciones que utiliza este lenguaje para poder obtener el resultado deseado, y nos permitió observar detalles muy importantes.

Por ejemplo; las conversiones de hexadecimal a decimal con el ajuste a **BCD** implementando la instrucción **DA** para realizar la transformación, sin embargo este ajuste trabaja con el acumulador **(A)** y el auxiliar de acumulador **(AC)** por tal motivo para cada iteración al realizar este ajuste se debe limpiar estos acumuladores con las instrucciones **(CLR A)** y **(CLR AC)**.

Por otra parte, al momento de trabajar con la pila de memoria fue posible utilizar las instrucciones **POP (restaura)** **PUSH (preserva)**, **LCALL (llamada a la subrutina)**, **RET (retorna a la dirección de memoria siguiente a la que fue llamada la subrutina)** con esta práctica, y también se implementó llamadas a procedimientos que tienen ciertas semejanzas al **lenguaje C** en el cual se puede trabajar con procedimientos mediante llamadas a estos fragmentos de códigos, con el lenguaje ensamblador se logro hacer las llamadas a estos fragmentos de códigos, llamados procedimientos, con las instrucción **LCALL (Etiqueta o nombre del fragmento de código)**.

Finalmente la implementación de esta práctica se tuvo en cuenta que el proceso de ordenamiento se realizará solo si son validados ciertos aspectos, como por ejemplo; que los elementos del vector se encuentren en el rango del código **ASCII A-Z**, que el número de elementos se encuentre en el rango de posiciones **30H a 7FH**, es decir, un total de **4FH** elementos (79 elementos en base decimal), y que en la direcciones **2EH/2FH** se almacenen los intercambios de l método burbuja.

Lista de Referencias

Ehu. 8051 Set de instrucciones: PUSH.
http://www.sc.ehu.es/sbweb/webcentro/automatica/web_8051/Contenido/set_8051/Instrucciones/51push.htm

Ehu. 8051 Set de instrucciones: POP.
http://www.sc.ehu.es/sbweb/webcentro/automatica/web_8051/Contenido/set_8051/Instrucciones/51pop.htm

Ehu. 8051 Set de instrucciones: LCALL.
http://www.sc.ehu.es/sbweb/webcentro/automatica/web_8051/Contenido/set_8051/Instrucciones/51lcall.htm

Ehu. 8051 Set de instrucciones: ACALL.
http://www.sc.ehu.es/sbweb/webcentro/automatica/web_8051/Contenido/set_8051/Instrucciones/51acall.htm

Ehu. 8051 Set de instrucciones: RET.
http://www.sc.ehu.es/sbweb/webcentro/automatica/web_8051/Contenido/set_8051/Instrucciones/51ret.htm

KEIL. 8051 Set de instrucciones: PUSH. http://www.keil.com/support/man/docs/is51/is51_push.htm

KEIL. 8051 Set de instrucciones: POP. http://www.keil.com/support/man/docs/is51/is51_pop.htm

KEIL. 8051 Set de instrucciones: LCALL. http://www.keil.com/support/man/docs/is51/is51_lcall.htm

KEIL. 8051 Set de instrucciones: ACALL. http://www.keil.com/support/man/docs/is51/is51_acall.htm

KEIL. 8051 Set de instrucciones: RET. http://www.keil.com/support/man/docs/is51/is51_ret.htm