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 especificas 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**](http://es.wikipedia.org/wiki/Estructura_de_datos) en la que el modo de acceso a sus elementos es de tipo [**LIFO**](http://es.wikipedia.org/wiki/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)<= (SP) + 1   
((SP))<= (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**

(direct)<= ((SP))   
(SP)<= (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 a**l 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 *dir\_16* | 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 *dir\_11*   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | a10 | a9 | a8 | 1 | 0 | 0 | 0 | 0 | |  | dir 7-0 | 2 | 2 | - |

***Operación:* ACALL**

(PC) <= (PC) + 2   
(SP) <= (SP) + 1   
((SP)) <= (PC7-0)   
(SP)<= (SP) + 1   
((SP)) <= (PC15-8)   
(PC10-0) <= 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)<= ((SP))   
(SP)<= (SP) - 1   
(PC7-0)<= ((SP))   
(SP)<= (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.

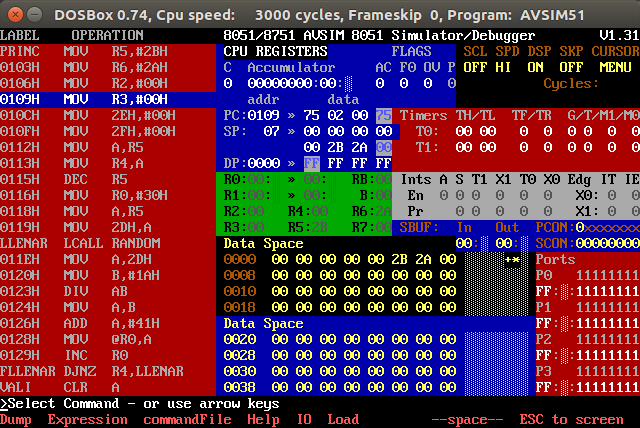
Previamente se estudió detalladamente el conjunto de entradas en el programa de simulación de ordenamiento de burbuja, como lo son el tamaño del vector **(N)** ubicado en el registro **R5**, los registros **R2** y **R3** para simular los índices **‘i’ y ‘j’** respectivamente que sirven para ir iterando 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**

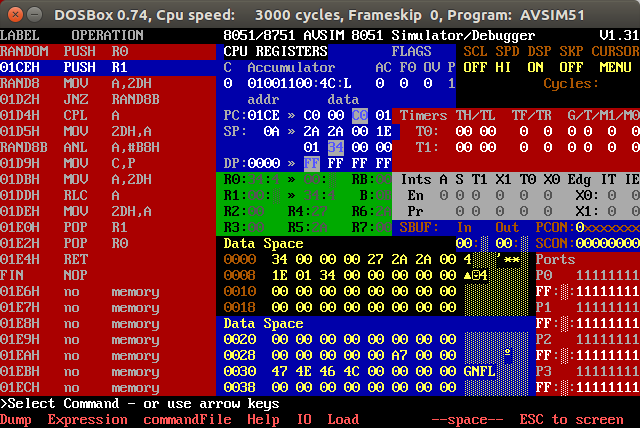
El algoritmo principal cuenta con el llenado del vector haciendo uso del procedimiento random, que genera cada una de las letras del vector. Luego se valida si cada una de las direcciones del vector se encuentran en el rango **A-Z**. Luego se llama al procedimiento de ordenamiento que a su vez llama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

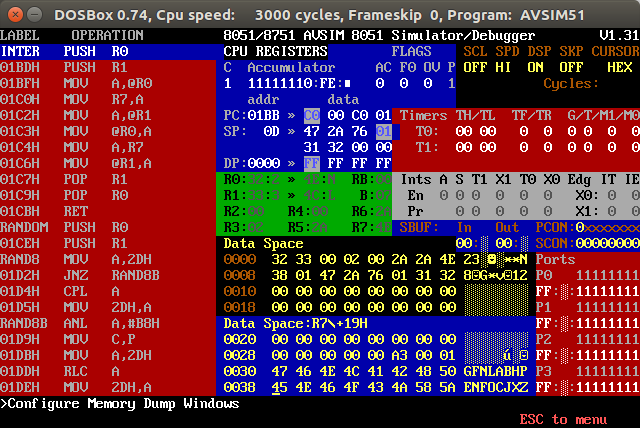
Se asigna valores a los registros **R2, R3, R4, R5** para iniciar programa.



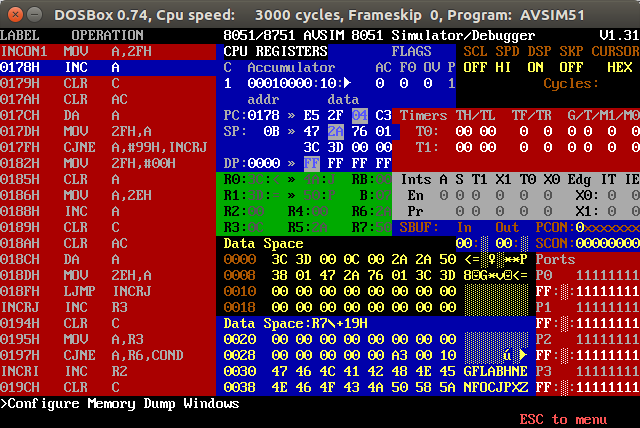
Se llena el vector con la semilla generada



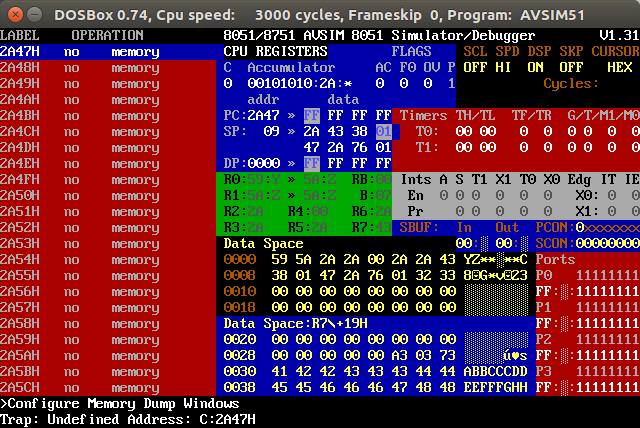
Se hacen las comparaciones e intercambio de las posiciones



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



Programa completado burbuja hecho con su número de intercambios realizados



**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