



Unidad de aprendizaje: Lenguaje Ensamblador

Docente: M. en C. C. Juan Carlos Ambriz Polo

Evaluación: Primer parcial

Fecha de asignación: 7/03/2019

Fecha límite de entrega: 26/03/2019

Alumno: Blas Duran Gustavo

Nota₁: Recuerde que las primeras entregas obtiene el 100% del valor, posteriormente por cada clase que transcurra reduce el 10% esto hasta llegar a la fecha límite, ya que después de dicha fecha no se aceptan entregas. Lea detenidamente las instrucciones y evite omitir pasos, trabaje sobre este formato de lo contrario no se recibirá la práctica.

Practica 1 Estructura del lenguaje ensamblador y desplegado de mensajes en pantalla

Objetivo:

Analizar la estructura del lenguaje ensamblador, para la generación de programas que permitan desplegar mensajes con formato en pantalla, así como familiarizarse con el ensamblador que se utilizará durante el curso (TASM).

Actividad 1: Resumir conceptos principales del tema.

Un ensamblador es un programa que traduce los programas escritos en lenguaje ensamblador a lenguaje máquina. El ensamblador es una herramienta practica para comprender el funcionamiento de una computadora, es por eso por lo que se necesita tener un conocimiento acerca del hardware de una computadora.

El lenguaje ensamblador se puede denominar lenguaje de bajo nivel, se diseñó para ejecutarse en poca memoria y consiste en operaciones de bajo nivel. Por lo que consiste en una serie de instrucciones que la parte física de cualquier hardware es capaz de interpretar.

La unidad central de procesamiento (CPU), en donde se realizan los cálculos y las operaciones lógicas, contiene un número limitado de lugares de almacenamiento, conocidas como registros; además de un reloj de alta frecuencia, una unidad de control y una unidad aritmética lógica. El *reloj* sincroniza las operaciones internas del CPU, la *unidad de control* coordina la secuencia de los pasos involucrados en la ejecución de instrucciones de máquina, la *unidad aritmética lógica* realiza operaciones aritméticas como la suma y la resta y operaciones lógicas como AND, OR y NOT. [1]

La *unidad de almacenamiento de memoria* es en donde se mantienen las instrucciones y los datos mientras se ejecuta un programa de computadora. Un *bus* es un grupo de cables que transfieren datos de una parte de la computadora a otra parte de esta. El bus de la computadora consta de tres tipos, el bus de datos: por el cual se transfieren instrucciones y datos entre la CPU y la memoria, el bus de control: utiliza las señales binarias para

sincronizar las acciones de todos los dispositivos conectados al bus del sistema, el bus de direcciones almacena las direcciones y los datos.

La ejecución de una sola instrucción de máquina puede dividirse en una secuencia de operaciones individuales conocidas como *ciclo de ejecución de instrucciones*. Las tres operaciones primarias son *búsqueda*: en la cual la unidad de control busca la instrucción en cola de instrucciones e incrementa el apuntador de instrucciones; *decodificación*: la unidad de control decodifica la función de una instrucción para determinar lo que esta debe de hacer. Los operandos de entrada de la instrucción se pasan a la unidad aritmética-lógica (ALU) y esta envía señales para indicar la operación de los usuarios, *ejecución*: la ALU ejecuta la instrucción, utilizando los registros con nombres y los registros internos como operandos, y envía el resultado a los registros con nombre y a la memoria.

El ensamblador trabaja directamente con los registros, los registros son ubicaciones de almacenamiento de alta velocidad que se encuentran dentro del CPU, por lo cual se tienen los registros de propósito general y de propósito específico. Los *registros de propósito general* se utilizan principalmente para las operaciones aritméticas y el movimiento de datos, algunos registros de propósito general tienen usos especializados tal es el caso de Ax, el cual es el primer registro que utilizan de manera automática las instrucciones de multiplicación y división. En pocas palabras es un acumulador. El registro Bx tiene el propósito de base, Cx como contador, Dx como los datos. En los *registros de propósito específico* se tienen registros de *segmento*: indican las direcciones base de las áreas preasignadas de memoria, en los cuales se dividen en segmentos de código, segmentos de pila y segmentos de datos. También se tiene el registro de apuntador, y el registro de memoria. Es así como conforma la memoria, a través de registros.

Como se menciono anteriormente, los registros de propósito general realizan movimientos de datos. Es por eso por lo que, al escribir las instrucciones, se tienen modos de direccionamiento. Como se menciono anteriormente en los nemónicos, para realizar un movimiento se hace uso del nemónico mov, en la cual seguidamente de la palabra se escriben el destino seguido de coma y el origen. Esto es si se tiene la siguiente instrucción *mov Ax,Bx*, se refiere que la instrucción transfiere el contenido de Bx hacia Ax. Al origen y el destino se les denomina operandos. En los cuales se tienen diferentes modos;

direccionamiento de registro (mov Ax,Bx): transfiere una copia de un byte o palabra del registro al origen.

Direccionamiento inmediato (mov Ax, S): transfiere el origen (datos tipo byte) al registro o posición de memoria al registro de destino.

Direccionamiento directo (mov mov Ax,[100]): mueve un byte o palabra entre una posición de memoria y un registro.

Direccionamiento de registro indirecto (mov Ax, [1]): transfiere un byte o una palabra entre un registro y una posición de memoria direccionados por un registro de índice o base.

Direccionamiento de base más índice (mov indexado mov [Bx +Si],Ax): transfiere un byte o una palabra entre un registro y la posición de memoria direccionada por un registro base más un registro índice.

Direccionamiento de registro relativo (mov Ax,[Bx+2]): mueve un byte o una palabra entre un registro y la posición de memoria direccionada por un registro índice o base mas un desplazamiento.

Direccionamiento de base relativa más índice (mov array[Bx+Si],Dx): transfiere un byte o una palabra entre un registro y la posición de memoria direccionada por un registro base y un registro índice más un desplazamiento.

El ciclo de ensamblado-enlazado-ejecución consta de cuatro pasos: [1]

Paso 1: un programador utiliza un editor de texto para crear un archivo de texto, conocido como archivo de código fuente.

Paso 2: el ensamblador lee el archivo de código fuente y produce el archivo de código objeto, una traducción del programa a lenguaje máquina. De manera opcional, produce un archivo de listado. Si ocurre un error, el programador debe regresar al paso 1 y corregir el programa.

Paso 3: el enlazador lee el archivo de código objeto y verifica si el programa contiene alguna llamada a los procedimientos en una biblioteca de enlace. El enlazador copia cualquier procedimiento requerido de la biblioteca de enlace, lo combina con el archivo de código objeto y produce el archivo ejecutable. De manera opcional, el enlazador puede producir un archivo de mapa.

Paso 4: la herramienta cargador (loader) del sistema operativo lee el archivo ejecutable y lo carga en memoria, y bifurca la CPU hacia la dirección inicial del programa, para que este empiece a ejecutarse.

La estructura del lenguaje ensamblador consta de cuatro partes: directivas, instrucciones, etiquetas y comentarios.

Una **directiva** es un comando incrustado en el código fuente, que el ensamblador reconoce y actúa en base a esta. Las directivas que se emplean en el lenguaje ensamblador son las siguientes:

La directiva `.data` identifica el área de un programa que contiene variables.

La directiva `.code` identifica el área de un programa que contiene instrucciones.

La directiva `.stack` identifica al área de un programa que guarda la pila en tiempo de ejecución y establece su tamaño. [1]

Una **instrucción** es un conjunto que se vuelve ejecutable cuando se ensambla un programa. El ensamblador traduce las instrucciones en bytes de lenguaje máquina, para que la CPU los cargue y los lleve a cabo en tiempo de ejecución. En la instrucción se contempla el nemónico de instrucción; es una palabra corta que identifica a una instrucción. [1] cada uno de los nemónicos proporcionan una sugerencia acerca del tipo de operación que realizan, por ejemplo el nemónico *mov*: mueve un valor a otro, *add*: suma dos valores, *sub*: resta un valor de otro, entre otros más.

Una **etiqueta** es un identificador que actúa como marcador de posición para las instrucciones y los datos. Una etiqueta se coloca justo antes de una instrucción, representa la directiva de esta instrucción. [1]

Los **comentarios** son un medio de importe para que el escrito de un programa comunique información acerca de su funcionamiento a la persona que lee el código fuente. [1] como en todo programa, los comentarios son muy importantes, pues no solo ayuda a comunicar o ayudar a entender al lector acerca del funcionamiento del programa, si no que al programador, le sirve para que el mismo también comprenda el programa después de algún tiempo.

Para utilizar las llamadas funciones del DOS, se ubican el número de función en el registro AH y se carga el resto de la información pertinente en los registros. Después de esto se continua con un 21H para ejecutar la función del Dos, en la posición actual del cursor. Como regla general las llamadas a las funciones del Dos guardan a todos los registros no utilizados como datos de salida.

Una interrupción del software es una llamada a un procedimiento del sistema operativo. La mayoría de estos procedimientos, conocidos con manejadores de interrupciones, proporcionan la capacidad de entrada-salida a los programas de aplicaciones. Se utilizan para diferentes tareas como mostrar caracteres o cadenas, leer caracteres del teclado, hora del sistema, etc.

Actividad 2: Codificación de programas en ensamblador

Instrucciones: Codifique los siguientes programas en ensamblador

1. Programa en ensamblador que permita desplegar tres mensajes diferentes aplicando un formato en cascada, haciendo usos para genera el formato de saltos de línea, retornos de carro y sangrías generadas con el (TAB).

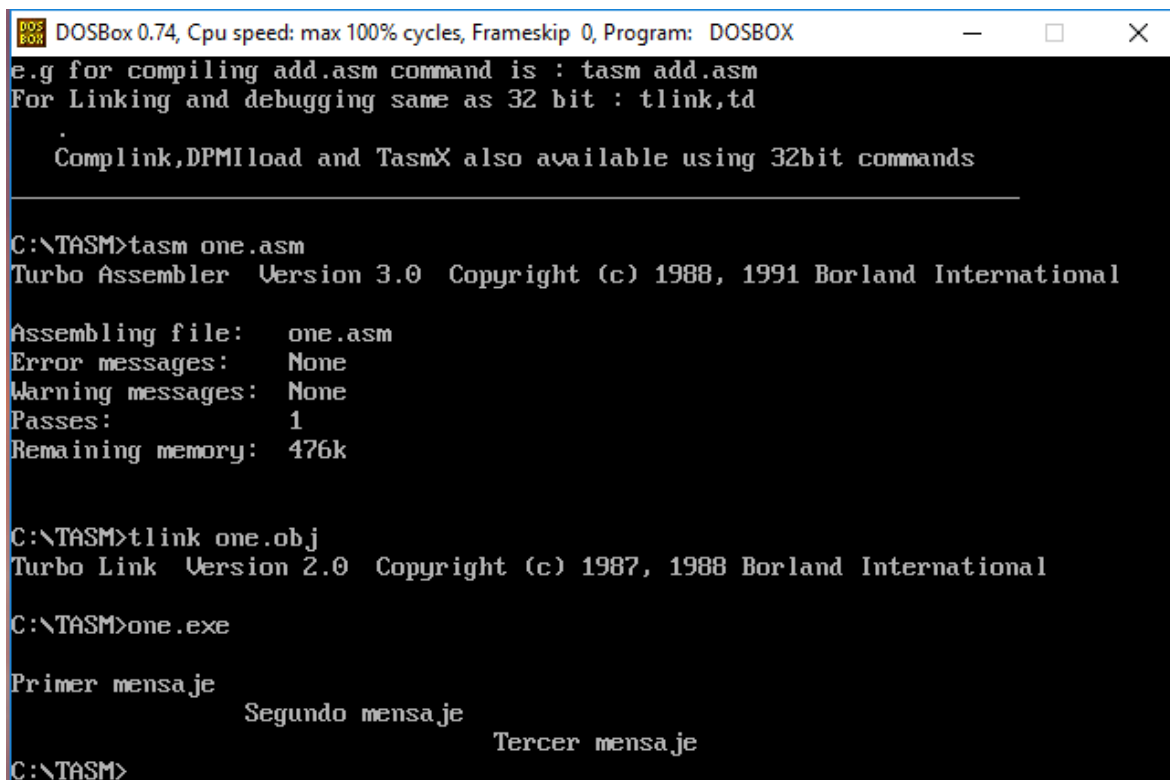
Código fuente del ejercicio 1:

```

1  .model small           ; tamaño del modelo
2  .stack                 ; tamaño de la pila predeterminado
3  .data                  ; área de las variables
4
5      v1 db 0dh,0Ah,0Dh,"Primer mensaje",0dh,0Ah,09h,09h,"Segundo
mensaje",0dh,0Ah,09h,09h,09h,09h,,"Tercer mensaje$"
6
7  .code                  ; área de instrucciones
8
9      mov Ax,@data        ; se guarda el contenido en Ax
10     mov Ds,Ax           ; el contenido de Ax se guarda en Ds
11     mov Ah,09h          ; 09h despliega una cadena de caracteres
12     mov Dx,offset v1    ; la cadena v1 entra a Dx
13     int 21h             ; llamara a la interrupcion 21h con la funcion
09h
14     mov ah,4ch          ; se solicita la opcion FINALIZAR UN PROCESO
de 21h
15     int 21h             ; se devuelve el control al sistema
16
17 end

```

A continuación, se presenta en la *ilustración 1*, la captura de la ejecución del código anterior. Se muestran las frases en cascada. Se hace uso del retorno de carro (0Dh), salto de línea (0Ah) y tabulador horizontal en cada una de las frases.



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: DOSBOX
e.g for compiling add.asm command is : tasm add.asm
For Linking and debugging same as 32 bit : tlink,td

Complink,DPMIload and TasmX also available using 32bit commands

C:\TASM>tasm one.asm
Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International

Assembling file:   one.asm
Error messages:   None
Warning messages: None
Passes:           1
Remaining memory: 476k

C:\TASM>tlink one.obj
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

C:\TASM>one.exe

Primer mensaje           Segundo mensaje           Tercer mensaje
C:\TASM>_
```

Ilustración 1 Ejecución del código en tasm. se Produce el archivo de código objeto y el loader del OS.

2. Programa en ensamblador que permita desplegar una cadena de caracteres especiales a partir de la concatenación del código ASCII de cada carácter.

Se presenta el código fuente del ejercicio 2, la cadena de caracteres a desplegar en el programa a partir de la concatenación de código ASCII es: GUSTAVO. En la ilustración 2, se presenta la captura de pantalla del ensamblado.

```
1  .model small           ; tamaño del modelo
2  .stack                 ; tamaño de la pila predeterminado
3  .data                 ; área de las variables
4
5      array db 9,71,85,83,84,65,86,79,"$"
6
7  .code                 ; área de instrucciones
8
9      mov Ax,@data       ; se guarda el contenido en Ax
10     mov Ds,Ax          ; el contenido de Ax se guarda en Ds
11     mov Ah,09h         ; 09h despliega una cadena de caracteres
12     mov Dx,offset array ; la cadena v1 entra a Dx
13     int 21h            ; interrupción 21h con la función 09h
14     mov ah,4ch         ; FINALIZAR UN PROCESO 21h
15     int 21h            ; se devuelve el control al sistema
16
17 end
```

```

C:\TASM>tasm two.asm
Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International

Assembling file:   two.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k

C:\TASM>tlink two.obj
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

C:\TASM>two.exe
GUSTAVO
C:\TASM>

```

Ilustración 2 Se llama al programa TASM para ejecutar el código fuente. Se crea el archivo de código objeto y el loader del SO. Se aprecia la impresión de la cadena.

3. Programa en ensamblador que permita limpiar pantalla y posteriormente despliegue un mensaje.

Se presenta el código fuente del ejercicio 3. En este ejercicio se hace uso de la interrupción 10H del BIOS, también conocido como interrupción de servicios de video. En la línea 9, se establece el modo de operación del despliegue del video, esto se logra colocando un 00h en AH. Como salida se obtiene modo cambiado y pantalla borrada, en la *ilustración 3*, se muestra el ensamblado y la generación del código objeto. En la *ilustración 4* se muestra la ejecución del archivo los resultados.

```

1  .model small           ; tamaño del modelo
2  .stack                ; tamaño de la pila predeterminado
3  .data                 ; área de las variables
4
5      v1 db 9,"Se ha borrado la pantalla...$"
6
7  .code                 ; área de instrucciones
8
9      mov ah,00h         ; (00h) modo de video 3
10     mov al,3           ; modo de video 3 (texto a color)
11     int 10h            ; interrupción de servicios de video
12                                ; una vez borrada la pantalla se muestra el
mensaje
13     mov Ax,@data       ; se guarda el contenido en Ax
14     mov Ds,Ax          ; el contenido de Ax se guarda en Ds
15     mov Ah,09h         ; 09h despliega una cadena de caracteres
16     mov Dx,offset v1   ; la cadena v1 entra a Dx
17     int 21h            ; llamara a la interrupción 21h con la función
09h
18     mov ah,4ch         ; se solicita la opción FINALIZAR UN PROCESO
19     int 21h            ; se devuelve el control al sistema
20
21 end

```

```
C:\TASM>tasm three.asm
Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International

Assembling file:   three.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k

C:\TASM>tlink three.obj
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

C:\TASM>three.exe
```

Ilustración 3 se muestra la ejecución del programa TASM con el archivo de código fuente. Se produce el archivo de código objeto.

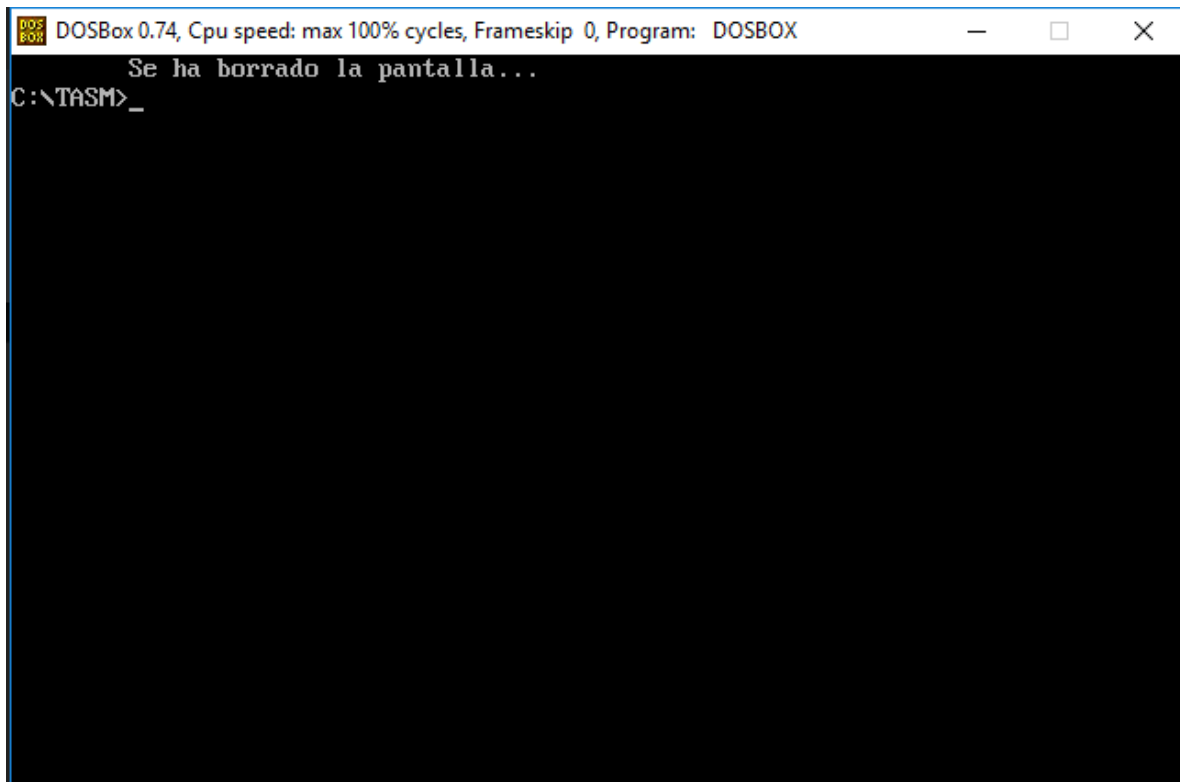


Ilustración 4 Se muestra el resultado después de ejecutar el archivo .exe. la pantalla se borra de manera automática, posteriormente se muestra el mensaje.

4. Programa en ensamblador que permita leer un carácter del teclado y posteriormente imprima el carácter ingresado por el usuario

En el siguiente programa se hace el uso de llamados a función DOS 01H LEE EL TECLADO, 02H ESCRIBE A DISPOSITIVO ESTANDAR DE SALIDA en este caso la pantalla de video. Primero se hace uso de 01H, se tiene como salida AL=carácter ASCII, el cual tiene como destino DL = carácter ASCII a desplegar, posteriormente se despliega en pantalla.

```
1  .model small          ; tamaño del modelo
2  .stack                ; tamaño de la pila predeterminado
3  .data                 ; área de las variables
4
5      v1 db 9,"carcater insertado:$"
6
7  .code                 ; área de instrucciones
8
9      mov Ah,01h        ; 01h lee el teclado
10     int 21h            ; interrupcion 21h con la funcion 01h
11     mov dl,al
    ;guarda en: (al)caracter ASCII y (dl)caracter ASCII a desplegar
12     int 21h            ;
13     mov ah,02h         ;escribe a dispositivo estandar de salida
14     int 21h            ;
15     mov ah,4ch         ;solicita la opcion FINALIZAR UN PROCESO 21h
16     int 21h            ;se devuelve el control al sistema
17
18 end
```



```
C:\TASM>tasm four.asm
Turbo Assembler Version 3.0 Copyright (c) 1988, 1991 Borland International

Assembling file:   four.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  476k

C:\TASM>tlink four.obj
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

C:\TASM>four.exe
a
a
C:\TASM>_
```

Ilustración 5 una vez insertado el carácter 'a' y presionar ENTER, se despliega en la parte de abajo el carácter.

Actividad 3: Análisis de archivo *.MAP

Instrucciones: una vez concluida la actividad 2 analice cada uno de los archivos de mapeo de los programas y realice las conversiones de las direcciones de inicio, fin y longitud.

Como se había mencionado anteriormente, el en ciclo de ensamblado-enlazado-ejecución, el paso número dos, nos menciona que se produce un archivo de mapeo. Este archivo de mapeo. Un archivo que en donde se nos presenta la traducción de lenguaje ensamblador a lenguaje máquina. Pero este archivo se encuentra en hexadecimal. Por lo que a continuación se presenta la conversión de los datos, se presenta en hexadecimal, Decimal y Binario.

EJERCICIO 1:

HEXADECIMAL

Start	Stop	Length	Name	Class
00000H		0000FH		
			00010H _TEXT	CODE
00010H		00049H		
			0003AH _DATA	DATA
00050H		0044FH		
			00400H STACK	STACK

DECIMAL:

Start	Stop	Length	Name
00000D		00015D	
		00016D	_TEXT
00016D		00073D	
		00058D	_DATA
00080D		01103D	
		01024D	STACK

El inicio de _Text empieza en 0D y termina en 15D, con una longitud de 16D, pues se contempla la posición cero. Como _TEXT termina en 15D, _DATA empieza en 16D, pues empieza en una posición mas que el anterior. El segmento de STACK empieza 12 posiciones después de donde termina _DATA, termina en 1103D con una longitud de 1024D.

EJERCICO 2:

HEXADECIMAL:

Start	Stop	Length	Name
00000H		0000FH	
		00010H	_TEXT
00010H		00018H	
		00009H	_DATA
00020H		0041FH	
		00400H	STACK

DECIMAL:

Start	Stop	Length	Name
00000B		00015D	
		00016D	_TEXT
00016D		00024D	
		00099D	_DATA
00032D		01055D	
		01024D	STACK

En este programa el segmento de STACK igualmente tiene una longitud de 1024D, a diferencia del anterior, este empieza en 32D, y termina en 1055D. existe un espacio entre _DATA y STACK de 8D. en el segmento de _DATA tiene una longitud mayor que a la del ejercicio 1, pues en este ejercicio se colocan únicamente los caracteres en código ASCII.

EJERCICIO 3:**HEXADECIMAL:**

Start	Stop	Length	Name
00000H	00015H	00016H	_TEXT
00016H	00033H	0001EH	_DATA
00040H	0043FH	00400H	STACK

DECIMAL:

Start	Stop	Length	Name
00000D	00021D	00022D	_TEXT
00022D	00051D	00030D	_DATA
00064D	01087D	01024D	STACK

en este ejercicio _TEXT empieza en 0D y termina en 21D, con una longitud de 22D, pues se contempla la posición 0. De igual manera existe un espacio entre _DATA y STACK de 14D.

EJERCICIO 4:**HEXADECIMAL:**

Start	Stop	Length	Name
00000H	0000FH	00010H	_TEXT
00010H	00024H	00015H	_DATA
00030H	0042FH	00400H	STACK

DECIMAL:

Start	Stop	Length	Name
00000D	00015D	00016D	_TEXT
00016D	00036D	00021D	_DATA
00048D	01071D	01024D	STACK

En el segmento _DATA se tiene una longitud de 21D, pues se contempla el mensaje de la línea número 5 del código fuente del ejercicio 4, pero si este mensaje es puesto como comentario y de nuevo se ensambla el programa, en el archivo de mapa, el valor de la longitud del segmento _DATA cambia a 0D, pues no se tiene ninguna variable, y el inicio y fin de este segmento es en el mismo. Se presenta a continuación el archivo de código objeto sin la variable en el segmento de _DATA. Se puede apreciar el inicio y fin del segmento _DATA-

Start	Stop	Length	Name
00000H	0000FH	00010H	_TEXT
00010H	00010H	00000H	_DATA
00010H	0040FH	00400H	STACK

Analizando cada uno de los archivos (.MAP), se percato de que todos tienen alrededor de 1KB (1024bytes), pues cuando se realiza la conversión de hexadecimal a decimal, en el segmento de pila terminan en 1024D y finalizan alrededor de los 1024.

Otro aspecto que se percato es en un espacio que existe entre _DATA y STACK, pues después de haber realizado las conversiones, como ya se sabe, una acción empieza un espacio después de donde se termina una sección y así sucesivamente, pero entre estos dos registros, hay un espacio en el que se empieza después. Por ejemplo, en el archivo de mapeo del primer ejercicio, _DATA empieza en 26D y finaliza en 83D, con una longitud de 58D. el segmento de pila STACK se tiene entendido empieza o empezaría en 84D, pero empieza en 96D. un ultimo detalle es que la longitud del segmento de pila de los cuatro ejercicios tiene una longitud de 1024. Esto pasa con cada uno de los programas.

Conclusiones: Redacte las conclusiones de su práctica (extensión mínima 20 renglones).

Una computadora esta compuesta por un CPU, el cual esta compuesto por registros: numero limitado de ubicaciones (estos a la vez se dividen en registros de propósito general y específico); una unidad aritmética lógica (ALU); un reloj: de alta frecuencia con el cual se sincronizan las operaciones; unidad de control (CU), también se conforma por una unidad de almacenamiento de memoria: en donde se guardan los datos mientras se carga un programa, buses: por los cuales se transmite los datos, dispositivos de E/S.

La ejecución de una instrucción de maquina se puede dividir en varias secuencias de operaciones: búsqueda, decodificación y ejecución. En la búsqueda la CU, busca las instrucciones en pila, en la decodificación, la unidad de control decodifica para obtener la instrucción y después la envía a la unidad aritmética lógica, en la ejecución la unidad aritmética lógica se encarga de ejecutar la instrucción que le han mandado para devolver un resultado de esa instrucción.

El ensamblador es un programa con el cual se trabajan directamente con los registros de la memoria, también se ejecuta con muy poca memoria. Para poder comprender como es que esto funciona, se debe tener en cuenta como es el hardware de una computadora, cada uno de sus componentes, pero principalmente como es que trabajan los registros, pues a través de ellos se realizan movimientos para ejecutar las instrucciones.

El lenguaje ensamblador tiene una estructura que consta de cuatro partes: directivas, por la cual el ensamblador actúa en base a esta. El .model se refiere al tamaño del modelo, .stack se refiere al tamaño de la pila, si en este no se indica algún tamaño, utiliza el tamaño predeterminado de 1024 bytes, .data en donde se escriben las variables a utilizar, .code se refiere al espacio en el cual se escriben las instrucciones. También consta de instrucciones, las cuales el ensamblador traduce a lenguaje máquina, en esta parte los componentes realizan sus procesos correspondientes. También se compone por etiquetas, el cual funciona como un marcador de posición para instrucciones y datos. Y por último los comentarios, los cuales ayudan al programador a identificar la acción de cada una de las líneas, estos comentarios como todo programa son ignorados por el ensamblador.

De acuerdo con los ejercicios realizados en la práctica, son programas que ocupan muy poco espacio en memoria.

Referencias

- [1] K. R. IRVINE, LENGUAJE ENSAMBLADOR PARA COMPUTADORAS BASADAS EN INTEL, Quinta Edicion ed., México: Prentice Hall, 2007.
- [2] P. Abel, LENGUAJE ENSAMBLADOR Y PROGRAMACION PARA IBM PC Y COMPATIBLES, Mexico: PEARSON Educacion, 1996.
- [3] B. B. Brey, MICROPROCESADORFES INTEL, aquitectura, programacion e interfaz, Septima Edicion ed., PEARSON Prentice Hall.