

TECNOLÓGICO NACIONAL DE MEXICO INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

Carrera: Sistemas Computacionales

Unidad 1. Introducción al Lenguaje Ensamblador.

Alumno:

Reyes Villar Luis Ricardo | 21070343

Profesora: Guadalupe Martínez Jauregui

Materia: Lenguajes de Interfaz

Hora: 09:00 – 10:00 hrs

Grupo: 6502-C

Semestre: Agosto 2023 – Diciembre 2023

1.1 Importancia de la programación en lenguaje ensamblador.

Definición del lenguaje ensamblador.

El lenguaje ensamblador es un tipo de lenguaje de bajo nivel utilizado

para escribir programas informáticos, y constituye la representación más directa

del código máquina específico para cada arquitectura de microprocesador.

La principal característica del lenguaje ensamblador es principalmente que se trabaja

directamente con el microprocesador; por lo cual se debe de conocer el

funcionamiento interno de este, tiene la ventaja de que en el se puede realizar

cualquier tipo de programas que en los lenguajes de alto nivel no lo pueden

realizar. Otro punto sería que los programas en ensamblador ocupan menos

espacio en memoria.

```
MONITOR FOR 6802 1.4          9-14-80   TSC ASSEMBLER   PAGE    2

C000          ORG      ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START  LDS      #STACK

*****
* FUNCTION: INITA - Initialize ACIA
* INPUT: none
* OUTPUT: none
* CALLS: none
* DESTROYS: acc A

0013          RESETA  EQU     %00010011
0011          CTLREG  EQU     %00010001

C003 86 13          INITA  LDA  A  #RESETA   RESET ACIA
C005 B7 80 04          STA  A  ACIA
C008 86 11          LDA  A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04          STA  A  ACIA

C00D 7E C0 F1          JMP   SIGNON   GO TO START OF MONITOR

*****
* FUNCTION: INCH - Input character
* INPUT: none
* OUTPUT: char in acc A
* DESTROYS: acc A
* CALLS: none
* DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04  INCH   LDA  A  ACIA      GET STATUS
C013 47          ASR  A                SHIFT RDRF FLAG INTO CARRY
C014 24 FA      BCC  INCH              RECIEVE NOT READY
C016 B6 80 05      LDA  A  ACIA+1      GET CHAR
C019 84 7F      AND  A  #$7F          MASK PARITY
C01B 7E C0 79      JMP   OUTCH        ECHO & RTS

*****
* FUNCTION: INHEX - INPUT HEX DIGIT
* INPUT: none
* OUTPUT: Digit in acc A
* CALLS: INCH
* DESTROYS: acc A
* Returns to monitor if not HEX input

C01E 8D F0  INHEX   BSR   INCH        GET A CHAR
C020 81 30      CMP  A  #'0           ZERO
C022 2B 11      BMI  HEXERR          NOT HEX
C024 81 39      CMP  A  #'9           NINE
C026 2F 0A      BLE  HEXRTS          GOOD HEX
C028 81 41      CMP  A  #'A           'A
C02A 2B 09      BMI  HEXERR          NOT HEX
C02C 81 46      CMP  A  #'F           'F
C02E 2E 05      BGT  HEXERR          'F
C030 80 07      SUB  A  #7            FIX A-F
C032 84 0F      HEXRTS AND  A  #$0F   CONVERT ASCII TO DIGIT
C034 39          RTS

C035 7E C0 AF  HEXERR JMP   CTRL      RETURN TO CONTROL LOOP
```

Figura 1.1.1. Ejemplo de código en lenguaje ensamblador

Características del lenguaje ensamblador.

- El único lenguaje que entienden los microcontroladores es el código máquina formado por ceros y unos del sistema binario.
- El lenguaje ensamblador expresa las instrucciones de una forma más natural al hombre a la vez que muy cercana al microcontrolador, ya que cada una de esas instrucciones se corresponde con otra en código máquina.

Figura 1.1.2. Comparación de código en lenguaje C contra lenguaje ensamblador.

Lenguaje de Alto Nivel C	Lenguaje Ensamblador
SWITCH TYPE) (case 'a': type=type+10; break; case 'b': type= type+20; break; default: break;)	MOV1 R1 = Type LD4 R2 = [R1] ;; cmp.eq P1, P2 = 'a', R2 cmp.eq P3, P4 = 'b', R2 ;; (P1) Add R2 = 10, R2 (P3) Add R2 = 20, R2 ;; st4 (R1) = R2 default::

- El lenguaje ensamblador trabaja con nemónicos, que son grupos de caracteres alfanuméricos que simbolizan las órdenes o tareas a realizar.
- La traducción de los nemónicos a código máquina entendible por el microcontrolador la lleva a cabo un programa ensamblador.
- El programa escrito en lenguaje ensamblador se denomina código fuente (*.asm). El programa ensamblador proporciona a partir de este fichero el correspondiente código máquina, que suele tener la extensión *.hex

Ventajas y Desventajas del lenguaje ensamblador.

Ventajas.

1. Como trabaja directamente con el microprocesador al ejecutar un programa, pues es el mas cercano a la máquina y debido a esto la computadora lo procesa más rápido.
2. Eficiencia de tamaño. - Un programa en ensamblador no ocupa mucho espacio en memoria porque no tiene que cargar librerías y demás como son los lenguajes de alto nivel
3. Flexibilidad. - Es flexible porque todo lo que puede hacerse con una máquina, puede hacerse en el lenguaje ensamblador de esta máquina; los lenguajes de alto nivel tienen en una u otra forma, limitantes para explotar al máximo los recursos de la máquina. O sea, que en lenguaje ensamblador se pueden hacer tareas específicas que en un lenguaje de alto nivel no se pueden llevar a cabo porque tienen ciertas limitantes que no se lo permiten.

Desventajas.

1. Tiempo de programación. - Como es un lenguaje de bajo nivel requiere más instrucciones para realizar el mismo proceso, en comparación con un lenguaje de alto nivel. Por otro lado, requiere de más cuidado por parte del programador, pues es propenso a que los errores de lógica se reflejen más fuertemente en la ejecución.
2. Programas fuente grandes. - Por las mismas razones que aumenta el tiempo, crecen los programas fuentes; simplemente requerimos más instrucciones primitivas para describir procesos equivalentes. Esto es una desventaja porque dificulta el mantenimiento de los programas, y nuevamente reduce la productividad de los programadores.
3. Peligro de afectar recursos inesperadamente. - Que todo error que podamos cometer, o todo riesgo que podamos tener, podemos afectar los recursos de la máquina. Al programar en este lenguaje lo más común que pueda pasar es que la máquina se bloquee o se reinicie, porque con este lenguaje es perfectamente posible (y sencillo) realizar secuencias de instrucciones inválidas, que normalmente no aparecen al usar un lenguaje de alto nivel.
4. Falta de portabilidad. - Porque para cada máquina existe un lenguaje ensamblador; por ello, evidentemente no es una selección apropiada de lenguaje cuando deseamos codificar en una máquina y luego llevar los programas a otros sistemas operativos o modelos de computadoras.

1.2 El procesador y sus registros internos.

El procesador o CPU (Unidad Central de Procesamiento por sus siglas en inglés) tiene 14 registros de 16 bits cada uno, los cuales son los siguientes:

Registros generales.

AX- Registro acumulador: Se utiliza principalmente para definir los servicios a utilizar en las interrupciones y para los números usados al realizar operaciones aritméticas.

BX- Registro base: Se utiliza para indicar un desplazamiento (offset).

CX- Registro contador: Se utiliza para contar durante los ciclos.

DX- Registro de datos: Se utiliza para el manejo de datos en operaciones aritméticas.

Los registros generales, también se conocen como registros de datos, debido a que estos son utilizados frecuentemente durante los programas de ensamblador para que el procesador no necesite acceder constantemente a la memoria. Cada uno de estos conformado por 16 bits o 2 bytes, los cuales pueden separarse en 2 registros de 8 bits cada uno.

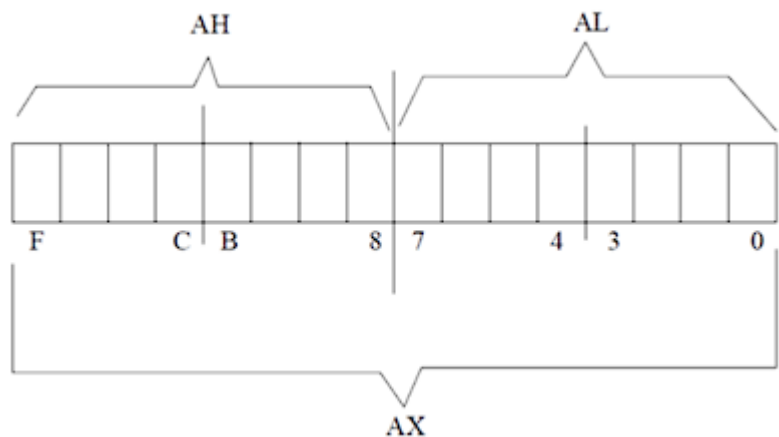


Figura 1.2.1. Distribución de los registros generales.

Cuando se hace referencia al registro completo se utiliza la letra del registro seguida por X, cuando se quiere utilizar una de las 2 partes que conforman el registro se utiliza la letra del registro seguida por H para acceder al byte superior (más significativo) o L para acceder al byte inferior (menos significativo), esto sucede con los registros 4 registros generales.

Registros índices.

SI- Índice fuente: Indica la dirección desde donde inician los datos a leer.

DI- Índice destino: Indica la dirección donde terminan los datos a leer.

Registros de segmentos.

CS- Segmento de código: Indica el segmento donde se encuentran las instrucciones del programa que se encuentra en ejecución.

DS- Segmento de datos: Indica el segmento de memoria donde se leen o almacenan datos que usa el programa en ejecución.

SS- Segmento de pila: Indica la dirección de comienzo del segmento de pila.

ES- Segmento Extra: Se utiliza generalmente como una expansión del segmento de datos.

Registros de punteros.

BP- Puntero base: Se utiliza al acceder a datos contenidos en la pila.

SP- Puntero de pila: Contiene la dirección relativa al segmento de la pila.

Registros especiales.

IP- Puntero de instrucción: Indica la dirección de la siguiente instrucción a ejecutar por el programa.

Banderas: En este registro se guardan indicadores de condiciones los cuales tienen un valor de 0 o 1, dependiendo de si su estado es activo o inactivo, estas banderas son las siguientes:

AF: Bandera Auxiliar.

CF: Bandera de acarreo.

DF: Bandera de dirección.

IF: Bandera de interrupciones.

OF: Bandera de desbordamiento.

PF: Bandera de paridad.

SF: Bandera de signo o desigualdad.

TF: Bandera de paso a paso.

ZF: Bandera de resultado 0 o de igualdad.

1.3 La memoria principal (RAM).

La memoria de acceso aleatorio (RAM por sus siglas en inglés) es utilizada por la mayor parte del software para llevar a cabo sus funciones.



Figura 1.3.1. Memoria RAM DDR4

Todas las instrucciones que ejecuta una computadora necesitan ser previamente cargadas a esta memoria, para posteriormente ser ejecutadas por el procesador. La razón de esto es que la RAM es la tercera memoria más rápida a la que puede acceder el procesador, siendo los primeros los registros del procesador y las memorias cache, las cuales, aunque más rápidas que la memoria RAM, cuentan con una capacidad de almacenamiento muy baja.

Existen 2 tipos de RAM según la forma en que mantienen la información, la RAM estática y la dinámica.

La RAM estática mantiene su contenido inalterado mientras que esta esté alimentada, mientras que en el caso de la RAM dinámica la información contenida se va degradando con el tiempo, llegando esta a desaparecer a pesar de contar con corriente eléctrica.

Según los tipos de conectores que lleven los módulos se pueden clasificar en Módulos SIMM (Single In-line Memory Module), con 30 ó 72 contactos, módulos DIMM (Dual In-line Memory Module), con 168 contactos y RIMM (RAMBUS In-line Memory) con 184 contactos.

1.4 El concepto de Interrupciones.

Una interrupción es una instrucción que detiene la ejecución de un programa para darle tiempo de procesador a otro proceso más importante. El utilizar una interrupción en ensamblador permite al procesador lleva a cabo funciones especiales predefinidas por ensamblador llamadas servicios, las cuales permiten entre otras cosas el desplegar información. Las interrupciones se separan en dos

tipos, que son de DOS (dependientes del sistema operativo) y de BIOS (dependientes del hardware).

Cuando un periférico desea acceder a un recurso, envía un pedido de interrupción al procesador para llamar su atención, los periféricos cuentan con un número de interrupción que se denomina IRQ (Interrupt Request).



Figura 1.4.1. Funciones especiales predefinidas.

Para utilizar una interrupción en ensamblador se utiliza la instrucción `int` seguida del número de la interrupción, por ejemplo: `int 10h`.

Ejercicio: Imprimir en pantalla el símbolo @

Resolucion:

```
.model small
.stack 64
.data
.code
;Limpiamos la pantalla
mov ax, 02h
int 10h
;Imprimimos @
mov dx, 64
mov ah, 02
int 21h
.exit
end
```

Figura 1.4.2. Programa que imprime en pantalla el símbolo @.

1.5 Llamadas a servicios del sistema.

Los servicios del sistema son funciones predefinidas, que pueden ser utilizadas en los programas. Por medio de las llamadas a estos servicios es posible que exista la comunicación entre un programa en ejecución y el sistema operativo, las llamadas a servicios se encuentran en manuales de ensamblador y dependen de la arquitectura en la que se esté programando.

Cuando una llamada al sistema es invocada, la ejecución del programa que invoca es interrumpida y sus datos son guardados, en el bloque de control de proceso o PCB (Process Control Block) para poder continuar ejecutándose posteriormente.

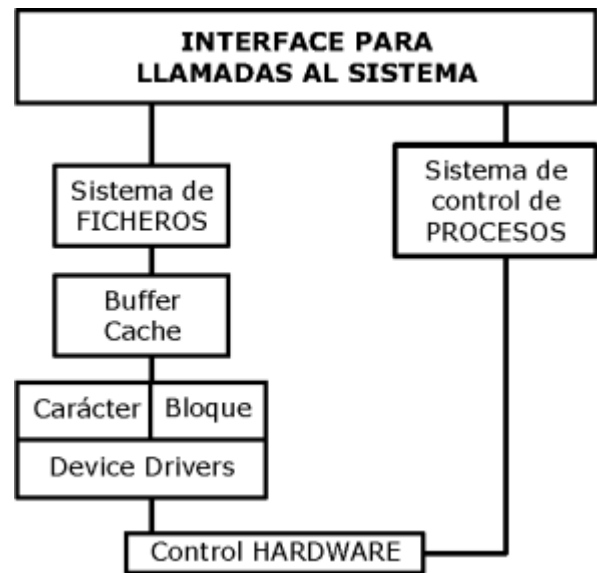


Figura 1.5.1. Esquema explicativo para llamadas al sistema.

Las categorías de las llamadas a servicios son las siguientes:

Comunicaciones: Esta categoría consta de las acciones relacionadas con mensajes, como es crear la conexión, enviar y recibir mensajes, transferir información de estado y eliminar la conexión.

Control de proceso: Consta de crear, cargar, ejecutar, abortar y terminar un proceso, así como obtener y establecer atributos, liberar memoria o esperar un suceso.

Manipulación de archivos: Esta categoría consiste de crear, abrir, leer, obtener atributos, establecer atributos, cerrar y eliminar archivos.

Manipulación de información: Acciones que requieren obtener datos del sistema, como por ejemplo la fecha y hora pertenecen a esta categoría.

Manipulación de periféricos: Cualquier acción que requiera utilizar un periférico, como solicitar, escribir, leer, obtener y establecer atributos y liberar periférico pertenece a esta categoría.

Ejemplo.

Para utilizarlas en ensamblador es necesario cargar datos a los registros adecuados según el servicio a utilizar y después llamar a la interrupción correspondiente, como se puede ver en el siguiente ejemplo:

mov dx,82 -> Asigna el valor 82 ASCII (Correspondiente a “R”) al registro dx

mov ah,02h -> Carga el valor 02h (servicio a utilizar) en el registro ah

int 21h -> Inicia la interrupción 21h, revisa el valor en ah y utiliza el servicio correspondiente, en este caso desplegar el carácter R.

1.6 Modos de direccionamiento.

Los modos de direccionamiento son las diferentes formas de definir la ubicación de un operando en memoria, para que de esta forma el programa sea capaz de encontrarlo durante el tiempo de ejecución. Es necesario mencionar que, al realizar una operación, es posible realizarla entre registros o entre registros y memoria, pero no es posible realizarla entre memoria y memoria.

Los modos de direccionamiento son los siguientes:

Implícito: Este modo de direccionamiento se utiliza sin declarar ninguna dirección, ya que las operaciones que lo utilizan ya conocen la dirección en la cual se encuentra el operando que se va a utilizar.

Inmediato: En este modo de direccionamiento se declara directamente el valor del operando en la instrucción.

Ejemplo:

mov ah,02h -> El valor del operando (02h) se especifica directamente.

Directo: En el direccionamiento directo, el operando hace referencia a un dato almacenado en la dirección especificada en la instrucción, la cual puede estar escrita de dos formas, como un registro o como una posición en memoria.

Para utilizar un registro basta con escribir el nombre del registro como operando para la operación, para utilizar una posición en memoria es necesario escribir el valor de la posición entre corchetes [], de esta forma TASM lo interpretara como una dirección y no como un valor.

Ejemplos:

mov ax, bx -> Se establece el dato en la dirección de bx como operando.

mov ax, [021BH] -> Se establece el dato en la dirección 021BH como operando.

Indirecto: Al utilizar direccionamiento indirecto, el operando se utiliza como una referencia a un dato que se encuentra almacenado en una posición de memoria, para esto el operando debe especificar un registro entre corchetes, el cual contiene la dirección de memoria a la que se desea acceder.

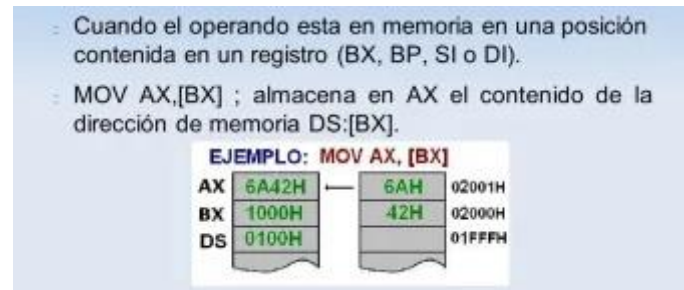


Figura 1.6.1. Ilustración del direccionamiento indirecto

Ejemplo:

`mov bx,[ax]` -> ax contiene la dirección a la que se desea acceder y este valor se asigna a bx.

Indexado: En el direccionamiento indexado el operando hace referencia a una posición en memoria, la cual puede ser expresada utilizando un numero el cual es sumado a un registro que funciona como un índice respecto a la dirección de memoria.

`mov ax, [1324h+12]` -> En este caso 1324h es la dirección base y 12 es el registro utilizado como índice.

Relativo: En el direccionamiento relativo se declara el operando como el valor de un registro entre corchetes, al cual se le aplica un desplazamiento, es decir, se le suma un valor que indicara el desplazamiento a partir de la dirección indicada por el registro.

Ejemplo:

`Mov ax,[bx+2]` -> En este ejemplo bx contiene la dirección base y 2 es el desplazamiento que se utilizará.

Estos son los únicos registros que pueden usarse de modo dual (en 8 o 16 bits) Los registros de la CPU son conocidos por sus nombres propios, que son:

- AX (acumulador)
- BX (registro base)
- CX (registro contador)

- DX (registro de datos)
- DS (registro del segmento de datos)
- ES (registro del segmento extra)
- SS (registro del segmento de pila)
- CS (registro del segmento de código)
- BP (registro de apuntadores base)
- SI (registro índice fuente)
- DI (registro índice destino)
- SP (registro del apuntador de pila)
- IP (registro del apuntador de siguiente instrucción)
- F (registro de banderas)

El registro AX se usa para almacenar resultados, lectura o escritura desde o hacia los puertos. El BX sirve como apuntador base o índice. El CX se utiliza en operaciones de iteración, como un contador que automáticamente se incrementa o decrementa de acuerdo con el tipo de instrucción usada. El DX se usa como puente para el acceso de datos.

1.7 Proceso de ensamblado y ligado

El proceso de ensamblado es el mismo sin importar el ensamblador que se está utilizando, aunque para las explicaciones se utilizara la sintaxis del ensamblador Turbo Assembler el cual fue desarrollado por Borland y el enlazador Turbo Linker.



Figura 1.7.1 Ensamblado, ligado y ejecución de un programa.

Al escribir un programa en lenguaje ensamblador es necesario utilizar editor de texto y guardar el archivo con la extensión asm para que sea reconocida por el ensamblador.

Una vez contando con el programa con extensión asm es necesario acceder con la consola a la ubicación del archivo y escribir “TASM Archivo.asm” siendo Archivo el nombre del archivo creado anteriormente, el nombre del archivo no debe superar los 8 caracteres, debe empezar con letra y no debe contener caracteres especiales.





Si el archivo cumple los requisitos entonces el ensamblador producirá un código objeto, el cual es una traducción a código máquina del programa, en caso de que el ensamblador encuentre un error en el programa lo indicará, así como la línea en la que se encuentra ese error.

Después de que se cree exitosamente el código objeto es necesario utilizar el enlazador, para el cual se escribirá en la consola “Tlink Archivo.obj” siendo Archivo el nombre del programa.

La función del enlazador es verificar si existen llamadas a procedimientos de una librería de enlace, en caso de ser así copia cualquier procedimiento requerido y lo combina con el código objeto, generando de esta manera un archivo ejecutable con extensión exe.

Ejemplo.

Ejemplo de un programa en ensamblador que previamente ya paso por el proceso de ensamblado y ligado.

 P1	8/29/2018 10:16 PM	ASM File	1 KB
 P1	10/14/2018 3:38 PM	Application	1 KB
 P1.MAP	10/14/2018 3:38 PM	Linker Address Map	1 KB
 P1.OBJ	10/14/2018 3:38 PM	Object File	1 KB

1.8 Desplegado de mensajes en el monitor.

Para poder desplegar un mensaje en el monitor es necesario que el texto a desplegar se escriba en la RAM de visualización de video, para después ser enviado al monitor mediante el controlador de video. El controlador de video es en sí un microprocesador de propósito especial, que libera a la CPU principal del trabajo de controlar el hardware de video.



Figura 1.8.1. Placa madre.

Para hacer esto se pueden utilizar varios de los servicios con los que cuenta el lenguaje ensamblador, los cuales se describen con detalle en el apéndice A.

Para detener la pantalla y permitir al usuario ver lo que se ha desplegado en la misma se utiliza el servicio 00 de la interrupción 16h.

Ejemplo:

mov dx,65 -> Asignar el Valor 65 ASCII al registro DX

mov ah,02h -> Asignar el valor 02h al registro AH, que corresponde al servicio de impresión de un caracter.

int 21h -> Llamada a la interrupción 21h, de la que se ejecutará el servicio 02h.