2016

Tutorial de ensamblador

Thania Elizabeth Jurado García

Este documento presenta conceptos básicos de ensamblador, así como códigos con su correspondiente explicación y anexos, esto con el fin de que el lector aprenda Ensamblador básico.

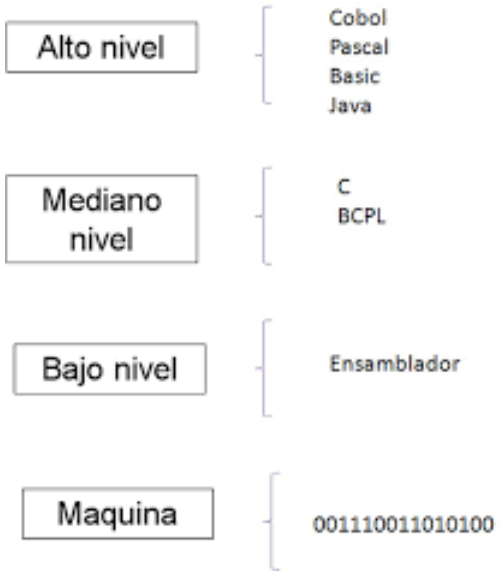
Contenido

Capítulo 0. ¿Qué es ensamblador?

Es el lenguaje de programación para escribir programas informáticos de bajo nivel, y constituye la representación más directa del código máquina específico.

Lenguaje de bajo nivel

Existen cuatro niveles de programación: Alto, medio, bajo y máquina.



Los lenguajes de alto nivel son aquellos cuya sintaxis, es decir, las líneas de código empleadas, son más parecidas al lenguaje humano. Tomando el ejemplo de Java:

```
System.out.println(“Hola”);
```

Interpretándolo del inglés, se leería como:

```
Systema.salida.imprimirLinea(“Hola”);
```

Que en lenguaje humano sería:

```
Muestra el mensaje “Hola”
```

Cabe aclarar que la computadora solamente entiende 1’s y 0’s, por lo que se requiere de un traductor entre el lenguaje humano y el lenguaje máquina. En los lenguajes de bajo nivel, las instrucciones son las más parecidas a como la computadora lo entiende y lo menos parecidas a como el ser humano suele comunicarse. Por ejemplo, el ensamblador usa el siguiente código para mostrar un mensaje en pantalla:

```
MOV DX, OFFSET SALUDO
MOV AH, 09
INT 21H
```

Lo cual, en lenguaje humano significaría:

Ubícate en la variable SALUDO, muévelo al registro de datos. Toma la instrucción 09 para mostrar en pantalla y muévela al registro acumulador en la parte alta. Interrumpe al sistema operativo para que deje de hacer lo que está haciendo (interrupción 21h) para que muestre lo que está guardado en la variable SALUDO.

Historia

La primera computadora en utilizar Ensamblador fue la Univac, desarrollada en los años 50. Antes de ello, las computadoras debían recibir instrucciones en código binario, es decir, 1s y 0s, lo cual resultaba complicado, tardado y hacía más probable cometer errores.

Los sistemas operativos fueron escritos casi exclusivamente en Ensamblador hasta la amplia aceptación de C en los años 70's y principios de los 80's.

Aplicaciones

A pesar de que existen lenguajes de programación con instrucciones más fáciles de entender, Ensamblador es útil para programar en sistemas embebidos (como los temporizadores de los microondas, lavadoras, secadoras). Como se mencionó anteriormente, se programa en bajo nivel, lo cual significa que el "traductor" tardará menos tiempo en pasar de este lenguaje a 1's y 0's al ser más parecido al lenguaje que utiliza la computadora, razón por la cual se emplea para programar en sistemas que cuentan con pocos recursos de memoria. Sin embargo, no es la única aplicación. También se emplea en la domótica (casas automatizadas), programación de drivers (para que las computadoras puedan reconocer un dispositivo o reproducir ciertos formatos de video), aplicaciones de tiempo real (como los que usan los sistemas de navegación de los aviones), entre otros.

Capítulo 1. Teoría básica

Aclaraciones

Ensamblador trabaja con registros del procesador para controlar las instrucciones a realizar y manejar los espacios de memoria. Ensamblador puede trabajar en distintos microprocesadores de distintas arquitecturas (16, 32 o 64 bits). Sin embargo, para este tutorial se empleará el procesador 8086, el cual trabaja sobre 16 bits.

Tipos de Ensambladores

Para poder ejecutar un programa escrito en lenguaje ensamblador es necesario contar con ensambladores y linkeadores. Todos sirven para lo mismo, clasificándose por su modo de trabajar en:

- **Ensambladores cruzados.-** Se usan en computadoras de prueba y luego el programa termina en otro equipo. Para esto se usan traductores.
- **Ensambladores residentes.-** Reside en la RAM, para pequeñas pruebas.
- **Macroensambladores.-** Trabajan con un repertorio fijo de instrucciones.
- **Ensambladores de una sola fase.-** Leen y traducen sobre la marcha. Son sencillos.
- **Ensambladores de dos fases.-** Realizan la traducción en dos etapas.

Cómo Ensamblador maneja los datos

Las instrucciones y variables se manejan por medio de segmentos. Esto es debido a que la arquitectura (el modo en que está hecho) de los procesadores x86 el mayor número de palabra que se podía manejar era de 16 bits de memoria, lo cual no alcanzaba en aquél entonces para un programa normal. Cada segmento proporciona un espacio de memoria para direccionamiento. En pocas palabras, en cada segmento se maneja una parte del programa, y ensamblador manipula el contenido de dichos segmentos mediante los registros de segmento, los cuales sirven como puerta de entrada. Existen 4 segmentos con su correspondiente registro:

Segmentos

Segmento y registro de datos

Ya que este lenguaje es muy restrictivo en cuanto a asignar valores directos se refiere, siempre se utilizan variables. Por ejemplo, retomando el ejemplo de la salida de datos, no es posible hacer lo que en los lenguajes de alto

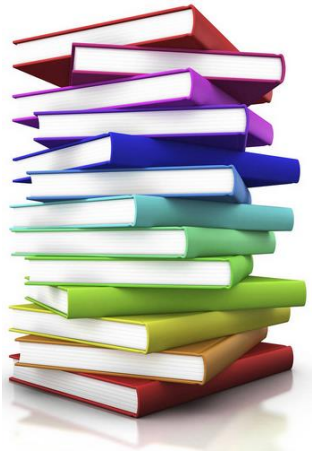
nivel mandando a llamar la función de mostrar y asignándole directamente el mensaje a mostrar. Debe de manejarse por una variable.

En el segmento de datos se declaran e inicializan las variables a utilizar. El registro de datos DS almacena el comienzo del segmento de datos. Si se quiere acceder a una variable en especial, deberá hacer referencia este segmento (el cual dice “aquí empieza”) y también cuánto deberá moverse para llegar a dicha variable. Eso es similar a cuando una persona busca una palabra en el diccionario: Primero se posiciona al principio de la página, luego recorre la hoja hasta llegar a la palabra que está buscando.

Segmento y registro de código

Por supuesto, es necesario tener un área donde se van a escribir las líneas de instrucción. Para la ejecución de código, se emplea el registro de código CS y, más el apuntador de instrucción IP, se mueve por todo el registro, ejecutando las instrucciones que correspondan.

Segmento y registro de pila



La pila es una estructura que consiste en datos apilados. Es un modo de almacenar información de manera temporal. Por ejemplo, si se quiere clasificar libros, estos se apilan según un criterio. El primero en colocarse queda debajo de la pila, y el último en colocarse se encuentra arriba. Si se quisiera retirar el libro del fondo de la pila, habría que remover primero todos los que se encuentran arriba de éste. Lo mismo sucede con la pila de datos en ensamblador. Un programa puede no necesitar segmento de pila, pero es útil para ciertas operaciones, en especial aquellas que manejan cadenas de caracteres. El registro SS permite la colocación en memoria de una pila. La dirección de inicio del segmento de pila se almacena en SS. Esta, más un valor de desplazamiento del apuntador de la pila SP indican el dato actual que se encuentra disponible en la pila.

Segmento y registro extra

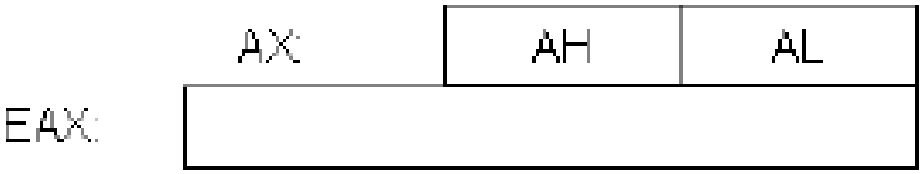
Por cuestiones de memoria, habrá ocasiones en las que será necesario utilizar un segmento extra. Su registro correspondiente es el ES. Por otra parte, en los procesadores 80386 existen registros extra como los FS y GS.

Registros de propósito general

Para el manejo de datos se emplean otros segmentos de memoria. Es similar a cuando una persona está ordenando su armario: La ropa son sus datos y,

el lugar donde pone la ropa temporalmente para luego moverla a su lugar definitivo son los registros.

De manera general, los registros tienen una longitud de 16 bits, y se dividen en dos partes: Alta y baja.



Un registro es similar a una caja con dos compartimientos. Pueden enviarse datos que llenen las dos partes de una vez, o emplearlas por separado. Cada parte tiene un tamaño de 8 bits. Para hacer referencia al registro completo, se escribe, tomando el ejemplo del registro acumulador, AX, siendo A el segmento y X el indicador de que se están usando ambos "compartimientos". Para referirse a la parte baja, se usa AL (L de bajo en inglés) y para usar la parte alta se indica como AH (H de alto en inglés).

Existen procesadores que manejan el EAX, que es un registro extendido.

Los registros de propósito general son:

Registro AX. El registro AX, el acumulador principal, es utilizado para operaciones que implican entrada/salida y la mayor parte de la aritmética. Por ejemplo, las instrucciones para multiplicar, dividir y traducir suponen el uso del AX. También, algunas operaciones generan código más eficiente si se refieren al AX en lugar de a los otros registros.

Registro BX. El BX es conocido como el registro base ya que es el único registro de propósito general que puede ser índice para direccionamiento indexado. También es común emplear el BX para cálculos.

Registro CX. El CX es conocido como el registro contador. Se emplea normalmente como un contador para controlar ciclos (como el i del ciclo for).

Registro DX. El DX es conocido como el registro de datos. Algunas operaciones de entrada/salida requieren uso, y las operaciones de multiplicación y división con cifras grandes suponen al DX y al AX trabajando juntos.

Registros apuntadores

Registro SP. El apuntador de la pila de 16 bits está asociado con el registro SS y proporciona un valor de desplazamiento que se refiere a la palabra actual que está siendo procesada en la pila. Los procesadores 80386 y posteriores tienen un apuntador de pila de 32 bits, el registro ESP. El sistema maneja de forma automática estos registros.

Registro BP. El BP de 16 bits facilita la referencia de parámetros, los cuales son datos y direcciones transmitidos vía pila. Los procesadores 80386 y posteriores tienen un BP ampliado de 32 bits llamado el registro EBP.

Registros Índice.

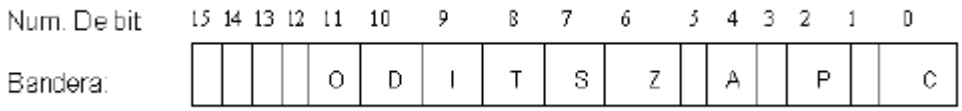
Los registros SI y DI están disponibles para direccionamiento indexado y para sumas y restas.

Registro SI. El registro índice fuente de 16 bits es requerido por algunas operaciones con cadenas (de caracteres). En este contexto, el SI está asociado con el registro DS. Los procesadores 80386 y posteriores permiten el uso de un registro ampliado de 32 bits, el ESI.

Registro DI. El registro índice destino también es requerido por algunas operaciones con cadenas de caracteres. En este contexto, el DI está asociado con el registro ES. Los procesadores 80386 y posteriores permiten el uso de un registro ampliado de 32 bits, el EDI.

Registros de banderas

Estos sirven para indicar el estado actual de la máquina y el resultado del procesamiento. El orden de importancia siempre se maneja de izquierda a derecha.



Las banderas que se utilizan en Ensamblador son:

OF (Overflow, desbordamiento). Indica desbordamiento de un bit de orden alto (más a la izquierda) después de una operación aritmética.

DF (dirección). Designa la dirección hacia la izquierda o hacia la derecha para mover o comparar cadenas de caracteres.

IF (interrupción). Indica que una interrupción externa, como la entrada desde el teclado, sea procesada o ignorada.

TF (trampa). Permite la operación del procesador en modo de un paso. Los programas depuradores, como el DEBUG, activan esta bandera de manera que usted pueda avanzar en la ejecución de una sola instrucción a un tiempo, para examinar el efecto de esa instrucción sobre los registros de memoria.

SF (signo). Contiene el signo resultante de una operación aritmética (0 = positivo y 1 = negativo).

ZF (cero). Indica el resultado de una operación aritmética o de comparación (0 = resultado diferente de cero y 1 = resultado igual a cero).

AF (acarreo auxiliar). Contiene un acarreo externo del bit 3 en un dato de 8 bits para aritmética especializada.

PF (paridad). Indica paridad par o impar de una operación en datos de 8 bits de bajo orden (más a la derecha).

CF (acarreo). Contiene el acarreo de orden más alto (más a la izquierda) después de una operación aritmética; también lleva el contenido del último bit en una operación de corrimiento o de rotación.

Tipos de direccionamiento

El campo de operación de una instrucción especifica la operación que se va a ejecutar. Esta operación debe realizarse sobre algunos datos almacenados en registros de computadora o en palabras de memoria. La manera en que eligen los operandos durante la ejecución del programa depende del modo de direccionamiento de la instrucción. El modo de direccionamiento especifica una regla para interpretar o modificar el campo de dirección de la instrucción antes de que se haga la referencia real al operando. Las computadoras utilizan técnicas de

modo de direccionamiento para acomodar una o las dos siguientes consideraciones:

1. Proporcionar al usuario versatilidad de programación al ofrecer facilidades como apuntadores a memoria, contadores para control de ciclo, indexación de datos y reubicación de datos.

2. Reducir la cantidad de bits en el campo de direccionamiento de la instrucción.

La disponibilidad de los modos de direccionamiento proporciona al programador con experiencia en lenguaje ensamblador la flexibilidad para escribir programas más eficientes en relación con la cantidad de instrucciones y el tiempo de ejecución.

Para comprender los diferentes modos de direccionamiento que se presentaran en esta sección, es imperativo entender el ciclo de operación básico de la computadora. La unidad de control de una computadora está diseñada para recorrer un ciclo de instrucciones que se divide en tres fases principales:

1. Búsqueda de la instrucción de la memoria.

2. Decodificar la instrucción.

3. Ejecutar la instrucción.

Hay un registro en la computadora llamado contador de programa o PC, que lleva un registro de las instrucciones del programa almacenado en la memoria. Pc contiene la dirección de la siguiente instrucción que se va a ejecutar y se incrementa cada vez que se recupera una instrucción de la memoria. La decodificación realizada en el paso 2 determina la operación que se va a ejecutar, el modo de direccionamiento de la instrucción y la posición de los operandos.

Después la computadora ejecuta la instrucción y regresa al paso 1 para hacer la búsqueda de la siguiente instrucción en secuencia.

En algunas computadoras el modo de direccionamiento de la instrucción se especifica con un código binario distinto, como se hace con el código de operación. Otras computadoras utilizan un código binario único que representa la operación y el modo de la instrucción. Pueden definirse instrucciones con diversos modos de direccionamiento y, en ocasiones, se combinan dos o más modos de direccionamiento en una instrucción.

Aunque la mayoría de los modos de direccionamiento modifican el campo de dirección de la instrucción, hay dos modos que no necesitan el campo de dirección. Son los modos implícito e inmediato.

Modo implícito

En este modo se especifican los operandos en forma implícita en la definición de la instrucción. Por ejemplo, la instrucción "complementar acumulador" es una instrucción de modo implícito porque el operando en el registro de acumulador está implícito en la definición de la instrucción. De hecho todas las instrucciones de referencia a registro que utilizan un acumulador son instrucciones de modo implícito.

Las instrucciones de dirección cero en una computadora organizada con pila son instrucciones de modo implícito porque está implícito que los operandos están en la parte superior de la pila.

Modo inmediato

En este modo se especifica el operando en la instrucción misma. En otras palabras, una instrucción de modo inmediato tiene un campo operando, en lugar de un campo de dirección. Un campo de operando contiene el operando real que se va a usar junto con la operación especificada en la instrucción. Las instrucciones de modo inmediato son útiles para inicializar registros en un valor constante.

Se mencionó antes que el campo de dirección de una instrucción puede especificar una palabra de memoria o un registro de procesador. Cuando el campo de dirección especifica un registro de procesador se dice que la instrucción esta en modo de registro.

Modo de registro

En este modo, los operandos están en registros que residen dentro de la CPU. Se selecciona el registro particular de un campo de registro en la instrucción. Un campo k bits puede especificar cualquiera de 2 a la k registros.

Modo indirecto por registro

En este modo la instrucción especifica un registro en la CPU cuyo contenido proporciona la dirección del operando en la memoria. En otras palabras, el registro seleccionado contiene la dirección del operando en lugar del operando mismo. Antes de utilizar una instrucción de modo indirecto por registro, el programador debe asegurarse de que la dirección de memoria del operando está colocada en el registro del procesador con una instrucción previa. Entonces una referencia al registro es equivalente a especificar una dirección de memoria. La ventaja de una instrucción de modo de registro indirecto es que el campo de dirección de la instrucción utiliza menos bits para seleccionar un registro de los que necesitaría para especificar una dirección de memoria en forma directa.

Modo de direccionamiento directo

En este modo la dirección efectiva es igual a la parte de dirección de la instrucción. El operando reside en memoria y su dirección la proporciona en forma directa el campo de dirección de la instrucción. En una instrucción de tipo brinco el campo de dirección especifica la dirección de transferencia de control del programa real.

Modo de direccionamiento indirecto

En este modo, el campo de dirección de la instrucción proporciona la dirección en que se almacena la dirección efectiva en la memoria. El control recupera la instrucción de la memoria y utiliza su parte de dirección para acceder la memoria una vez más con el fin de leer la dirección efectiva.

Unos cuantos modos de direccionamiento requieren que el campo de dirección de la instrucción se sume al contenido de un registro específico en la CPU. En estos modos la dirección efectiva se obtiene del cálculo siguiente:

Dirección efectiva = Parte de la instrucción + El contenido de registro CPU.

El registro de CPU utilizado en el cálculo puede ser el contador de programa, un registro de índice o un registro base. En cualquier caso tenemos un modo de direccionamiento diferente que se utiliza para una aplicación distinta.

Modo de direccionamiento indexado

En este modo el contenido de un registro índice se suma a la parte de dirección de la instrucción para obtener la dirección efectiva. El registro índice es un registro CPU especial que contiene un valor índice. Un campo de dirección de la instrucción define la dirección inicial del arreglo de datos en la memoria. Cada operando del arreglo se almacena en la memoria en relación con la dirección inicial.

La distancia entre la dirección inicial y la dirección del operando es el valor del índice almacenado en el registro de índice. Cualquier operando en el arreglo puede accederse con la misma instrucción siempre y cuando el registro índice contenga el valor de índice correcto. El registro índice puede incrementarse para facilitar el acceso a operandos consecutivos. Nótese que si una instrucción de tipo índice no incluye un campo de dirección en su formato, la instrucción se convierte al modo de operación de indirecto por registro.

Algunas computadoras dedican un registro de CPU para que funcione exclusivamente como un registro índice. De manera implícita este registro participa cuando se utiliza una instrucción de modo índice. En las computadoras con muchos registros de procesador, cualquiera de los registros de la CPU puede

contener el número de índice. En tal caso, el registro debe estar especificado en forma explícita en un campo de registro dentro del formato de instrucción.

Modo de direccionamiento por registro base

En este modo, el contenido de un registro base se suma a la parte de dirección de la instrucción para obtener la dirección efectiva. Esto es similar al modo de direccionamiento indexado, excepto en que el registro se denomina ahora registro base, en lugar de registro índice. La diferencia entre los dos modos está en la manera en que se usan más que en la manera en que se calculan. Se considera que un registro base contiene una dirección base y que el campo de dirección de la instrucción proporciona un desplazamiento en relación con esta dirección base. El modo de direccionamiento de registro base se utiliza en las computadoras para facilitar la localización de los programas en memoria.

Interrupciones

Cuando un niño necesita que le ayuden a abrir un frasco de mermelada, y va hacia donde está su padre o su madre. Quizá ellos estén ocupados, pero el niño los interrumpe para que dejen de hacer lo que están haciendo y lo ayuden. Al abrir el frasco, el niño se retira y sus padres pueden seguir con lo que estaban haciendo. Algo similar ocurre con las interrupciones. Una interrupción es una operación que suspende la ejecución de un programa de modo que el sistema pueda realizar una acción especial. La rutina de interrupción ejecuta y por lo regular regresa el control al procedimiento que fue interrumpido, el cual entonces reasume su ejecución.

Cuando la computadora se enciende, el BIOS y el DOS establecen una tabla de servicios de interrupción en las localidades de memoria 000H-3FFH. La tabla permite el uso de 256 (100H) interrupciones, cada una con un desplazamiento: segmento relativo de cuatro bytes en la forma IP: CS. En pocas palabras, la computadora tiene una tabla de las interrupciones que deberá hacer. Se envía la interrupción, por ejemplo 21H, entonces se busca en la tabla qué es lo que quiere decir 21H y, una vez que lo encuentra, ejecuta la interrupción.

Se dividen en dos tipos:

Interrupciones externas

Una interrupción externa es provocada por un dispositivo externo al procesador. Las dos líneas que pueden señalar interrupciones externas son la línea de interrupción no enmascarable (NMI) y la línea de petición de interrupción (INTR).

La línea NMI reporta la memoria y errores de paridad de E/S. El procesador siempre actúa sobre esta interrupción, aun si emite un CLI para limpiar la bandera de interrupción en un intento por deshabilitar las interrupciones externas. La línea INTR reporta las peticiones desde los dispositivos externos, en realidad, las interrupciones 05H a la 0FH, para cronometro, el teclado, los puertos seriales, el disco duro, las unidades de disco flexibles y los puertos paralelos.

Interrupciones internas

Una interrupción interna ocurre como resultado de la ejecución de una instrucción INT o una operación de división que cause desbordamiento, ejecución en modo de un paso o una petición para una interrupción externa, tal como E/S de disco. Los programas por lo común utilizan interrupciones internas, que no son enmascarables, para acceder los procedimientos del BIOS y del DOS.

Interrupciones del BIOS

El BIOS (Basic Input-Output System / Sistema Básico de Entrada-Salida) es un sistema integrado en la computadora, independiente del sistema operativo. Este se encarga de controlar el modo en que trabaja el hardware (monitor, teclado, mouse, bocinas, etc.), cómo se administra la memoria temporal, el almacenamiento permanente (disco duro, memoria USB) y la comunicación entre el procesador y los demás dispositivos. En pocas palabras, es el responsable del funcionamiento básico de la computadora.

Las interrupciones que maneja el BIOS son:

INT 00H: División entre cero. Llamada por un intento de dividir entre cero. Muestra un mensaje y por lo regular se cae el sistema.

INT 01H: Un solo paso. Usado por DEBUG y otros depuradores para permitir avanzar por paso a través de la ejecución de un programa.

INT 02H: Interrupción no enmascarare. Usada para condiciones graves de hardware, tal como errores de paridad, que siempre están habilitados. Por lo tanto un programa que emite una instrucción CLI (limpiar interrupciones) no afecta estas condiciones.

INT 03H: Punto de ruptura. Usado por depuración de programas para detener la ejecución.

INT 04H: Desbordamiento. Puede ser causado por una operación aritmética, aunque por lo regular no realiza acción alguna.

INT 05H: Imprime pantalla. Hace que el contenido de la pantalla se imprima. Emita la INT 05H para activar la interrupción internamente, y presione las teclas Ctrl +

Prnt Scrn para activarla externamente. La operación permite interrupciones y guarda la posición del cursor.

INT 08H: Sistema del cronometro. Una interrupción de hardware que actualiza la hora del sistema y (si es necesario) la fecha. Un chip temporizador programable genera una interrupción cada 54.9254 milisegundos, casi 18.2 veces por segundo.

INT 09H: Interrupción del teclado. Provocada por presionar o soltar una tecla en el teclado.

INT OBH, INT OCH: Control de dispositivo serial. Controla los puertos COM1 y COM2, respectivamente.

INT 0DH, INT OFH: Control de dispositivo paralelo. Controla los puertos LPT1 y LPT2, respectivamente.

INT 0EH: Control de disco flexible. Señala actividad de disco flexible, como la terminación de una operación de E/S.

INT 10H: Despliegue en vídeo. Acepta el número de funciones en el AH para el modo de pantalla, colocación del cursor, recorrido y despliegue.

INT 11H: Determinación del equipo. Determina los dispositivos opcionales en el sistema y regresa el valor en la localidad 40:10H del BIOS al AX. (A la hora de encender el equipo, el sistema ejecuta esta operación y almacena el AX en la localidad 40:10H).

INT 12H: Determinación del tamaño de la memoria. En el AX, regresa el tamaño de la memoria de la tarjeta del sistema, en términos de kilobytes contiguos.

INT 13H: Entrada/salida de disco. Acepta varias funciones en el AH para el estado del disco, sectores leídos, sectores escritos, verificación, formato y obtener diagnóstico.

Interrupciones del DOS

Los dos módulos del DOS, IO.SYS y MSDOS.SYS, facilitan el uso del BIOS. Ya que proporcionan muchas de las pruebas adicionales necesarias, las operaciones del DOS por lo general son más fáciles de usar que sus contrapartes del BIOS y por lo común son independientes de la máquina.

Las interrupciones desde la 20H hasta la 3FH están reservadas para operaciones del DOS. Algunas de ellas son:

INT 20H: Termina programa. Finaliza la ejecución de un programa .COM, restaura las direcciones para Ctrl + Break y errores críticos, limpia los búfer de registros y regresa

el control al DOS. Esta función por lo regular seria colocada en el procedimiento principal y al salir del, CS contendría la dirección del PSP. La terminación preferida es por medio de la función 4CH de la INT 21H.

INT 21H: Petición de función al DOS. La principal operación del DOS necesita una función en el AH.

INT 22H: Dirección de terminación. Copia la dirección de esta interrupción en el PSP del programa (en el desplazamiento 0AH) cuando el DOS carga un programa para ejecución. A la terminación del programa, el DOS transfiere el control a la dirección de la interrupción. Sus programas no deben de emitir esta interrupción.

INT 23H: Dirección de Ctrl + Break. Diseñada para transferir el control a una rutina del DOS (por medio del PSP desplazamiento 0EH) cuando usted presiona Ctrl + Break o Ctrl + c. La rutina finaliza la ejecución de un programa o de un archivo de procesamiento por lotes. Sus programas no deben de emitir esta interrupción.

INT 24H: Manejador de error crítico. Usada por el dos para transferir el control (por medio del PSP desplazamiento 12H) cuando reconoce un error critico (a veces una operación de disco o de la impresora).Sus programas no deben de emitir esta interrupción.

INT 25H: Lectura absoluta de disco. Lee el contenido de uno o más sectores de disco.

INT 26H: Escritura absoluta de disco. Escribe información desde la memoria a uno o más sectores de disco.

INT 27H: Termina pero permanece residente (reside en memoria). Hace que un programa .COM al salir permanezca residente en memoria.

INT 2FH: Interrupción de multiplex ión. Implica la comunicación entre programas, como la comunicación del estado de un spoiler de la impresora, la presencia de un controlador de dispositivo o un comando del DOS tal como ASSIGN o APPEND.

INT 33H: Manejador del ratón. Proporciona servicios para el manejo del ratón.

Capítulo 2. Comenzando a programar

Estructura básica de un programa en Ensamblador

Comentarios

Los comentarios se hacen escribiendo después de un punto y coma

```
; Esto es un comentario
```

Ensamblador toma todo lo que esté enseguida del punto y coma como comentario, por lo cual el uso de palabras reservadas o su longitud no afectan la ejecución del programa. Incluso, es posible escribir un comentario a la derecha de una instrucción sin siquiera afectarla. Por ejemplo:

```
MOV DX, OFFSET SALUDO; Este es un comentario
```

También pueden hacerse por medio de la directiva COMMENT

```
MOV DX, OFFSET SALUDO COMMENT Este es otro comentario
```

Palabras reservadas

Existen palabras que ensamblador ya entiende como instrucciones, por lo cual no deben de usarse para otro propósito. Por ejemplo, MOV es para mover datos de un segmento a otro, por lo que una variable no se puede llamar MOV.

Identificadores (Variables)

En ensamblador los identificadores son las variables. Es un nombre que se le da a un elemento del programa. Sus características son:

- Pueden usar letras de la A a la Z
- Números desde el 0 al 9, siempre y cuando no se usen como el primer carácter
- Puede usarse Signo de interrogación, guión bajo, signo de pesos y arroba
- También puede usar punto, pero este no debe ser el primer carácter
- No debe coincidir con una palabra reservada
- Máximo 31 caracteres

Instrucciones

Un programa en lenguaje ensamblador consiste en un conjunto de enunciados. Los dos tipos de enunciados son:

1. Instrucciones, tal como MOV y ADD, que el ensamblador traduce a código objeto.

2. Directivas, que indican al ensamblador que realiza una acción específica, como definir un elemento de dato.

A continuación esta el formato general de un enunciado, en donde los corchetes indican una entrada opcional:

[identificador] operación [operando(s)] [;comentarios]

IDENTIFICADOR	OPERACIÓN	OPERANDO	COMENTARIO
COUNT	DB	1	;Nom, Op, Operando
	MOV	AX, 0	;Operación, 2 Operandos

No siempre existe un identificador, así como un comentario o un solo operando. Cada parte debe estar separada por al menos un espacio en blanco o una tabulación.

Directivas para listar: PAGE Y TITLE

La directiva PAGE y TITLE ayudan a controlar el formato de un listado de un programa en ensamblador. Este es su único fin, y no tienen efecto sobre la ejecución subsecuente del programa.

PAGE. Al inicio de un programa, la directiva PAGE designa el número máximo de líneas para listar en una página y el número máximo de caracteres en una línea. Su formato general es:

PAGE [longitud][, ancho]

El ejemplo siguiente proporciona 60 líneas por página y 132 caracteres por línea:

```
PAGE 60, 132
```

El número de líneas por página puede variar desde 10 hasta 255, mientras que el número de caracteres por línea desde 60 hasta 132. La omisión de PAGE causa que el ensamblador tome PAGE 50, 80.

TITLE. Se puede emplear la directiva TITLE para hacer que un titulo para un programa se imprima en la línea 2 de cada página en el listado del programa. Puede codificar TITLE de una vez, al inicio del programa. Su formato general es:

TITLE Texto.

Para el operando texto, una técnica recomendada es utilizar el nombre del programa como se registra en el disco. Por ejemplo:

```
TITLE Prog1 Mi primer programa en ensamblador
```

Declarando segmentos

Para declarar un segmento se emplea la siguiente sintaxis:

Nombre SEGMENT alineación combinar 'clase'

Donde el nombre y el segmento son altamente necesarios, mientras que alineación, combinar y clase son opcionales.

Nombre es el identificador del segmento, SEGMENT indica que se trata de un segmento, lineación indica el límite en el que inicia el segmento. Para el requerimiento típico, PARA, alinea el segmento con el límite de un párrafo, de manera que la dirección inicial es divisible entre 16, o 10H. En ausencia de un operando hace que el ensamblador por omisión tome PARA.

Combinar indica si se combina el segmento con otros segmentos cuando son enlazados después de ensamblar. Los tipos de combinar son STACK, COMMON, PUBLIC y la expresión AT. Puede utilizar PUBLIC y COMMON en donde tenga el propósito de combinar de forma separada programas ensamblados cuando los enlaza. En otros casos, donde un programa no es combinado con otros, puede omitir la opción o codificar NONE.

La entrada clase, encerrada entre apóstrofos, es utilizada para agrupar segmentos cuando se enlazan. Se utiliza la clase 'code' para el segmento de códigos, 'data' por segmento de datos y 'stack' para el segmento de la pila.

Directiva ASSUME

Para que ensamblador sepa cual segmento es cual, se utiliza el ASSUME.

```
ASSUME CS:CODIGO, DS:DATO, SS:PILA, ES:EXTRA
```

De esta manera, Ensamblador asume que el segmento que se llama CODIGO es el segmento de código, que el segmento que se llama DATO es el segmento de datos, y así sucesivamente.

Directivas simplificadas de segmento

Los ensambladores de Microsoft y de Borland proporcionan algunas formas abreviadas para definir segmentos. Para usar estas abreviaturas, inicialice el modelo de memoria antes de definir algún segmento. El formato general (incluyendo el punto inicial) es:

.MODEL modelo de memoria

El modelo de memoria puede ser TINY, SMALL, MEDIUM, COMPACT o LARGE. Los requisitos para cada modelo son:

MODELO	NÚMERO DE SEGMENTOS DE CÓDIGO	NÚMERO DE SEGMENTOS DE DATOS
TINY	*	*
SMALL	1	1
MEDIUM	MÁS DE 1	1
COMPACT	1	MÁS DE 1
LARGE	MÁS DE 1	MÁS DE 1

Los formatos generales (incluyendo el punto inicial) para las directivas que define los segmentos de la pila, de datos y de código son:

.STACK [tamaño] .DATA .CODE [nombre]

Cada una de estas directivas hace que el ensamblador genere el enunciado SEGMENT necesario y su correspondiente ENDS. Los nombres por omisión de los segmentos (que usted no tiene que definir) son STACK, DATA y TEXT (para el segmento de código).

Instrucciones de comparación

En cuanto a la manipulación de datos existen ocasiones en las que es necesario comparar o validar datos para poner condiciones. En vez de usar la sentencia if-else de los lenguajes de alto nivel, se emplea una comparación y luego un salto según el resultado de dicha comparación. Para esto se usa CMP para comparaciones simples, CMPS que aumenta o disminuye 1 en el registro SI y DI para bytes, 2 para datos de tamaño Word y 4 para dobles, REP CMPS para cadenas de cualquier longitud, CMPSB para bytes, CMPSD para datos de tamaño doble y CMPSW para datos de tipo Word o palabra. La sintaxis general es:

CMP dato1, dato2

Saltos

Por otra parte, los saltos (equivalente en Ensamblador del comando else) son muy variados, siendo los más comunes el JMP (salto incondicional), JE (Salta si el resultado de la comparación fue un igual o true) y JNE (Si no lo fue), entre otros. Lo que hacen es ir de la línea de código en la que se encuentran a otra, ejecutando un conjunto de instrucciones específicas.

Ciclos o LOOPs

Dichos saltos se ayuden de registros que sirven como apuntadores. Sin embargo, en vez de usarlos se pueden emplear ciclos LOOP, los cuales se ayudan del registro CX y lo decrementan automáticamente, evitando así ciclos infinitos. Sus variantes son LOOPE (el cual depende del estado de la bandera ZF) y LOOPNE (si la bandera ZF=0).

Incremento y decremento

Habrá momentos en los que, ya sea para manipular un LOOP de una manera especial o hacer un ciclo que funcione de una manera personalizada, se requiera incrementar o decrementar el valor de un contador, es decir, una variable que lleve la cuenta de algo.

Para decrementar un valor se usa:

DEC variable

Y para incrementar se emplea:

INC variable

Ambos son equivalentes de `variable++` y `variable--` de los lenguajes de alto nivel.

Mover datos

Resultaría imposible manipular las variables sin mover datos entre ellas. Para esto se usa:

MOV destino, fuente

Sin embargo, como muchos comandos de Ensamblador, cuenta con variables:

- **MOVS.-** Mueve a DI lo que hay en SI
- **LODS.**- Lleva lo que hay en SI a AL
- **LODSB.**- Para datos de tamaño byte
- **LODSW.**- Para datos de tamaño word
- **LAHF.-** Para mover a AH el estado de las banderas para saber su estado

Posicionarse en un dato

Por otra parte, el mover datos no sólo sirve para asignar datos a variables o realizar copias temporales, sino también para posicionarse en un dato, lo cual puede hacerse de dos formas:

Modo 1:

MOV SI, OFFSET Variable

Modo 2:

LEA SI, Variable

Uso de la pila

Existen ocasiones en las que es necesario almacenar datos de manera temporal para que éstos no se vean afectados por otras operaciones. Para ello se usa la pila de datos. Una pila de datos es como una pila de libros: El primer dato en formar parte de la pila es el último en ser retirado. Para meter y sacar elementos de la pila se utilizan las operaciones PUSH y POP.

POP destino

Saca datos de la pila. POPF extrae hacia la bandera.

PUSH dato

Coloca un valor en la pila. PUSHF decrementa en 2 la bandera SP y mueve el dato hacia la pila.

Operaciones aritméticas

- **ADC destino, fuente** Para sumas con acarreo, donde el acarreo se va a la bandera CF
- **ADD destino, fuente** Para sumas normales
- **DIV fuente** Para dividir entre AX, guardar el resultado en AL y su residuo en DX.
- **IDIV fuente** para división con signo
- **MUL fuente** Para multiplicar por AH (8bits) o por AX (16 bits) y guardar el resultado en AX
- **IMUL fuente** Para multiplicación con signo
- **SBB destino, fuente** Para resta con acarreo, usando la bandera CF
- **SUB destino, fuente** Para restas normales.

Los datos numéricos pueden manejarse en el sistema de numeración que se desee, siempre u cuando se indique por medio de una letra al término del número de qué sistema se trata, siendo el sistema decimal el que se emplea por defecto. Por ejemplo, para indicar un 12 en hexadecimal, se escribe 12H.

Operaciones lógicas

Las operaciones lógicas son aquellas en las que se comparan situaciones y los resultados pueden ser Verdadero o Falso. Los principales son:

- **AND** Verdadero sólo cuando las dos situaciones son Verdaderas.
- **NEG** Complemento de 2
- **NOT** Si es Verdadero lo convierte a Falso, y viceversa.

- **OR** Es verdadero si ambas situaciones son verdaderas o si alguna de ellas o es.
- **XOR** Sólo es verdadera si una situación u otra es verdadera, mas no ambas al mismo tiempo.
- **TEST** Es como el AND, pero sólo afecta a las banderas.

Desplazamiento y rotación

Las instrucciones de corrimiento, que son parte de la capacidad lógica de la computadora, pueden realizar las siguientes acciones:

1. Hacer referencia a un registro o dirección de memoria.
2. Recorre bits a la izquierda o a la derecha.
3. Recorre hasta 8 bits en un byte, 16 bits en una palabra y 32 bits en una palabra doble.
4. Corrimiento lógico (sin signo) o aritmético (con signo).

El segundo operando contiene el valor del corrimiento, que es una constante (un valor inmediato) o una referencia al registro CL. Para los procesadores 8088/8086, la constante inmediata solo puede ser 1; un valor de corrimiento mayor que 1 debe estar contenido en el registro CL. Procesadores posteriores permiten constantes de corrimiento inmediato hasta 31.

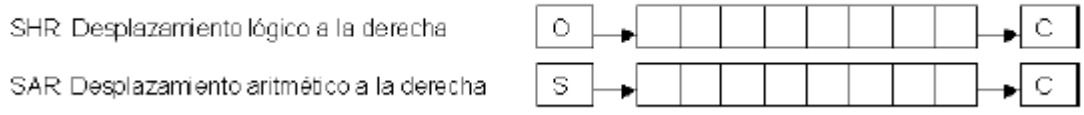
El formato general para el corrimiento es:

Nombre Corrimiento registro/memoria CL/inmediato

Cuando se termina la operación, puede utilizarse JC (Salta si hay acarreo) para ver que bit se ingresó a la bandera de acarreo.

Desplazamiento o corrimiento de bits hacia la derecha

Para esto se emplean los comandos SHR y SAR, los cuales recorren los datos de cada espacio del registro de memoria hacia la derecha.



Ejemplo:

INSTRUCCION	**COMENTARIO**
MOV CL, 03	
MOV AL, 10110111B	; AL=10110111
SHR AL, 01	; AL=01011011 Un corrimiento a la derecha

SHR AL, CL ; AL=00001011 Tres corrimientos adicionales a la derecha

Esto es similar a la fila para algún trámite: Cuando se termina de atender a un cliente las personas que están formadas se mueven un lugar, se recorren. En este caso, cuando se recorren hacia la derecha, el bit que se salió de “la fila de asientos” es enviado a la bandera de acarreo CF, y el bit de más a la izquierda, es decir, el asiento que había, se queda con un cero, ha quedado vacío.

SAR es diferente de SHR porque utiliza el bit de signo para llevar el bit de la izquierda o el asiento que quedó vacío, haciendo que, con el corrimiento, los valores conserven sus signos.

Desplazamiento o corrimiento de bits a la izquierda

Se emplean los comandos SHL y SAL, siendo el SHL para desplazamiento lógico y el SAL para desplazamiento aritmético.

Ejemplo:

INSTRUCCION	**COMENTARIO**
MOV CL, 03	
MOV AL, 10110111B	; AL = 10110111
SHL AL, 01	; AL = 01101110 Un corrimiento a la izquierda
SHL AL, CL	; AL = 01110000 Tres corrimientos mas

Con este recorrido, el bit más a la izquierda se va a la bandera de acarreo y el bit más a la derecha se llena con un cero.

Rotación de bits (Desplazamiento circular)

Se utilizan los comandos ROR (rotación a la derecha) y RCR (rotación a la derecha con acarreo). A diferencia de los anteriores, ningún bit o “cliente” se queda sin silla, simplemente si se sale de la fila de sillas, va y se sienta a la que quedó vacía al otro extremo, haciendo así un desplazamiento circular.

En el caso de que se desee hacer la rotación hacia la izquierda, se utilizan los comandos ROL y RCL.

Obtención de una cadena con la representación hexadecimal

La conversión entre numeración binaria y hexadecimal es sencilla. Lo primero que se hace para una conversión de un número binario a hexadecimal es dividirlo en grupos de 4 bits, empezando de derecha a izquierda. En caso de que el último grupo (el que quede más a la izquierda) sea menor de 4 bits se rellenan los faltantes con ceros.

Tomando como ejemplo el número binario 101011 lo dividimos en grupos de 4 bits y queda:

10; 1011

Rellenando con ceros el último grupo (el de la izquierda):

0010; 1011

Después se toman cada grupo como un número independiente y consideramos su valor en decimal:

0010 = 2; 1011 = 11

Pero como no podemos representar este número hexadecimal como 211 porque sería un error, se deben sustituir todos los valores mayores a 9 por su respectiva representación en hexadecimal, con lo que obtenemos:

2BH (Donde la H representa la base hexadecimal)

Para convertir un número de hexadecimal a binario solo es necesario invertir estos pasos: se toma el primer dígito hexadecimal y se convierte a binario, y luego el segundo, y así sucesivamente hasta completar el número.

Captura y almacenamiento de datos numéricos

Esta representación está basada en la notación científica, esto es, representar un número en dos partes: su mantisa y su exponente.

Poniendo como ejemplo el número 1234000, podemos representarlo como 1.123*10^6, en esta última notación el exponente nos indica el número de espacios que hay que mover el espacio hacia la derecha para obtener el resultado original.

En caso de que el exponente fuera negativo nos estaría indicando el número de espacios que hay que recorrer el punto decimal hacia la izquierda para obtener el original.

Operaciones básicas sobre archivos de disco

Servicios de la interrupción 16h para manejo del teclado.

Función 00h. Lee un carácter. Esta función maneja las teclas del teclado de 83 teclas, pero no acepta entrada de las teclas adicionales en el teclado ampliado de 101 teclas. Retorna en al el carácter, ah el código de rastreo si al=0 es una tecla de función extendida.

Función 01h. Determina si un carácter está presente.

Función 02h. Regresa el estado actual de las teclas Shift.

Función 10h. Lectura de un carácter del teclado.

Función 11h. Determina si está presente un carácter.

MOVS. Mueve un byte, palabra o palabra doble desde una localidad en memoria direccionada por SI a otra localidad direccionada por DI.

LODS. Carga desde una localidad de memoria direccionada por SI un byte en AL, una palabra en AX o una palabra doble en EAX.

STOS. Almacena el contenido de los registros AL, AX, o EAX en la memoria direccionada por SI.

CMPS. Compara localidades de memoria de un byte, palabra o palabra doble direccionadas por SI, DI.

SCAS. Compara el contenido de AL, AX o EAX con el contenido de una localidad de memoria direccionada por SI.

Proceso de creación de un programa

Para la creación de un programa es necesario seguir cinco pasos: **diseño del algoritmo, codificación del mismo, su traducción a lenguaje máquina, la prueba del programa y la depuración.**

En la etapa de diseño se plantea el problema a resolver y se propone la mejor solución, creando diagramas esquemáticos utilizados para el mejor planteamiento de la solución.

La codificación del programa consiste en escribir el programa en algún lenguaje de programación; en este caso específico en ensamblador, tomando como base la solución propuesta en el paso anterior.

La traducción al lenguaje máquina es la creación del programa objeto, esto es, el programa escrito como una secuencia de ceros y unos que pueda ser interpretado por el procesador.

La prueba del programa consiste en verificar que el programa funcione sin errores, o sea que haga lo que tiene que hacer.

Ensamblar, linkear y ejecutar programa

Para ello se deben seguir los siguientes pasos:

1. Escribir el código a utilizar en un archivo de texto.
2. Almacenarlo con extensión .asm en la carpeta masm
3. Ejecutar el archivo masmbat
4. En la línea de comandos que abre, escribir MASM nombreDelArchivo.asm
5. Dar Enter hasta que termine.
6. En esa misma línea de comando escribir LINK nombreDelArhivo.obj
7. Dar Enter hasta que termine.
8. Escribir nombreDelArchivo

En caso de que al realizar este procedimiento surja un “severe error”, deberá corregir el código y almacenar el archivo.asm con otro nombre ya que, de guardarlo en el archivo original, marcará error o simplemente no lo detectará como un programa válido.

El archivo con extensión .asm es un archivo de lenguaje Ensamblador. Al ejecutar MASM nombreDelArchivo se crea un archivo objeto con extensión .obj y, al usarlo en LINK nombreDelArchivo.obj se crea el archivo .exe, que es el ejecutable.

Hola mundo

A continuación se presenta el código de “Hola mundo” en Ensamblador.

```
PILA SEGMENT STACK
     DW 64 DUP(?)
PILA ENDS

DATO SEGMENT
SALUDO DB "Hola mundo",13,10,"$"
DATO ENDS

CODIGO SEGMENT
     ASSUME CS:CODIGO, DS:DATO, SS:PILA

INICIO:
     MOV AX,DATO
     MOV DS,AX
     MOV DX,OFFSET SALUDO

     MOV AH,09h
     INT 21h

     MOV AH, 4CH
     MOV AL, 00H
     INT 21h
CODIGO ENDS
     END INICIO
```

Inicio y final de segmento

```
PILA SEGMENT STACK
     DW 64 DUP(?)
PILA ENDS
```

PILA es el nombre del segmento de pila. Se le puede asignar el nombre que mejor le acomode. SEGMENT es para indicarle a ensamblador que se trata de un segmento y STACK, como se mencionó anteriormente, es la combinación.

PILA ENDS indica el término del segmento de pila.

Asignación de tamaño

Existen diversos tamaños manejados en Ensamblador.

Nombre	Significado	Tamaño
DB	Byte	8 bits
DW	Word	16 bits
DD	Double Word	32 bits

DQ	Quad Word	64 bits

```
DW 64 DUP(?)
```

Esta línea indica que la pila será de tamaño Data Word (16 bits). Indica el tamaño de la pila, a partir de la dirección 64 de la pila.

Declaración de variables

En el siguiente código se muestra la declaración del segmento de datos, dentro de la cual se declara la variable SALUDO.

```
DATO SEGMENT
     SALUDO DB "Hola mundo",13,10,"$"
DATO ENDS
```

SALUDO es el nombre o identificador, DB es el tamaño, “Hola mundo” es el String o cadena de caracteres que se le asigna a SALUDO. 13,10 es el interlineado de la línea de texto (para que salga estética) y el “$” sirve para terminar la cadena. Si se omitiera este símbolo, Ensamblador no sabría que esa cadena ha acabado, lo cual arrojaría un error.

ASSUME

En la siguiente línea se especifican los segmentos que se van a utilizar.

```
ASSUME CS:CODIGO, DS:DATO, SS:PILA
```

Etiquetas

Con ellas le damos nombre a un punto dentro del código, si más tarde dentro del programa deseamos repetir esta parte del código solo tenemos que decir "salta a INICIO" y ya está.

Inicio contiene las instrucciones.

```
INICIO:
     MOV AX,DATO
     MOV DS,AX
     MOV DX,OFFSET SALUDO

     MOV AH,09h
     INT 21h

     MOV AH, 4CH
     MOV AL, 00H
```

```
      INT 21h
CODIGO ENDS
      END INICIO
```

Parametrización, mover datos y puenteo

Antes de comenzar a trabajar, es necesario pasar el segmento de datos al registro acumulador y, del registro acumulador, pasarlo al registro de datos. Para esto se utiliza el comando MOV que sirve para mover datos y presenta la siguiente sintaxis:

MOV DESTINO, FUENTE

En el código tenemos:

```
MOV AX,DATO
MOV DS,AX
```

Cabe destacar que no se pueden mover datos directamente de un registro a otro, por lo que se realiza un “puenteo”. Esto consiste en pasar un dato a un segmento, y de este segmento pasarlo al registro final. De otra manera, esto puede marcar un error.

Mostrar cadena

Existen dos maneras de mostrar una cadena. Esta:

```
MOV DX,OFFSET SALUDO
MOV AH,09h
INT 21h
```

O esta:

```
LEA DX, SALUDO
MOV AH,09h
INT 21h
```

En la primera, OFFSET se posiciona en la parte donde se encuentra la variable saludos, y mueve su dirección de memoria al registro acumulador. En la segunda, LEA DX, SALUDO dice “Carga en el registro DX la dirección de memoria de la variable llamada SALUDO”. Sirven para lo mismo.

MOV AH, 09h mueve la instrucción 09h al registro acumulador en la parte alta, esto con el fin de que se prepare para mostrar algo en pantalla.

INT 21h es la interrupción que hace que el sistema deje de hacer lo que estaba haciendo para mostrar SALUDO.

Devolver control al BIOS

Para que el programa termine correctamente, es necesario devolverle el control al BIOS, de lo contrario, el programa no termina. Esto puede degenerar en un error o en un ciclo infinito.

```
MOV AH, 4CH
MOV AL, 00H
INT 21h
```

Como se mencionó anteriormente, los registros están divididos en dos “compartimientos”. MOV AH, 4CH y MOV AL, 00H están mandando instrucciones a la parte alta y a la parte baja del registro acumulador. Sin embargo, también se puede mandar toda la instrucción a al registro completo. La H puede omitirse. Esta instrucción le devuelve el control al BIOS.

```
MOV AX, 4C00H
INT 21h
```

Y la interrupción 21H es para que se ejecute la acción.

Esto es similar a cuando se le pide prestado un sacapuntas a un compañero. Lo presta, pero tarde que temprano debe devolvérselo ya que éste también necesita usarlo.

Cierre de segmentos y etiquetas

```
CODIGO ENDS
    END INICIO
```

CODIGO ENDS termina el segmento de código y END INICIO cierra la etiqueta de inicio. Si bien no todas las etiquetas requieren un cierre, en este caso es necesario cerrarla para que Ensamblador sepa que ya ha acabado. Cabe destacar que INICIO termina después de que lo hace el segmento de datos. Si lo hiciera antes, el programa terminaría INICIO sin terminar el segmento de datos, por lo cual no sabría que terminó, y esto arrojaría otro error.

Mostrar varios mensajes

```
PILA SEGMENT STACK
     DW 64 DUP(?)
PILA ENDS

DATO SEGMENT

MENSAJE1 DB "Primer mensaje",13,10,"$"
MENSAJE2 DB "Segundo mensaje",13,10,"$"
MENSAJE3 DB "Tercer mensaje",13,10,"$"

DATO ENDS

CODIGO SEGMENT
     ASSUME CS:CODIGO; DS:DATO, SS:PILA

INICIO:
     MOV AX,DATO
     MOV DS,AX
     MOV DX,OFFSET MENSAJE1

     MOV AH,09h
     INT 21h

     MOV DX,OFFSET MENSAJE2
     MOV AH,09h
     INT 21h

     MOV DX,OFFSET MENSAJE3
     MOV AH,09h
     INT 21h

     MOV AH, 4CH
     MOV AL, 00H
     INT 21h

CODIGO ENDS
     END INICIO
```

Cuando se quiere mostrar más de un mensaje, es preferible crear una variable para cada mensaje ya que Ensamblador no admite la asignación directa de texto. Se deberá hacer un posicionamiento en la variable (MOV DX, OFFSET VARIABLE) antes de llamar a la función para mostrar y ejecutar la interrupción.

Capítulo 3. Lectura desde el teclado y manipulación de cadenas

Lectura desde el teclado y conversión a mayúsculas

```
DATO SEGMENT
     CUENTA = 10
     MENSAJE DB "INTRODUCIR UNA CADENA DE 10 CARACTERES",13,10,"$"

     VAR DB CUENTA DUP(0)

     CRLF DB 13,10,"$"
DATO ENDS

CODIGO SEGMENT
     ASSUME DS:DATO,CS:CODIGO

     MAIN:      MOV AX,DATO
          MOV DS, AX

          MOV CX,CUENTA
          LEA SI,VAR

          MOV DX,OFFSET MENSAJE
          MOV AH,09H
          INT 21H


     LEER:      MOV AH,01H
          INT 21H
          CMP AL,"a"
          JB SIGA
          CMP AL,"z"
          JA SIGA
          AND AL,11011111B

     SIGA:      MOV [SI],AL
          INC SI
          DEC CX
          CMP CX,0
          JNE LEER

          MOV DX,OFFSET CRLF
          MOV AH,09H
          INT 21H
```

```
        MOV AH,09H
        MOV DX,OFFSET VAR
        INT 21H

        MOV AX,4C00H
        INT 21H

CODIGO ENDS
    END MAIN
```

Declaración de variables para almacenamiento de datos

En el código se tienen las variables CUENTA, MENSAJE, VAR y CRLF.

```
CUENTA = 10
MENSAJE DB "INTRODUCIR UNA CADENA DE 10 CARACTERES",13,10,"$"
VAR DB CUENTA DUP(0)
CRLF DB 13,10,"$"
```

CUENTA es una constante, a la cual se le asigna el número 10.

VAR es de tipo Data Byte, su longitud va a ser la misma que el valor de CUENTA (es decir, 10). DUP (0) significa que va a evitar la parte de la duplicidad.

CRLF es de tipo Data Byte, contiene el 13,10 para el interlineado y, para que ensamblador reconozca el final de la cadena de caracteres (se le considera así, aunque no tenga letras, por el simple hecho de contener el interlineado) se pone el “$”.

Asignación al registro contador y posicionamiento en un arreglo

```
        MOV CX,CUENTA
        LEA SI,VAR
```

La primera línea asigna un 10 al registro contador, el equivalente al “i” del ciclo for en los lenguajes de alto nivel. En la segunda línea se indica que ahí empiece con el primer espacio del arreglo.

Lectura de cadena: con ECO y sin ECO

Existen dos maneras de leer cadenas. Con y sin eco. Leer con eco significa que el usuario puede ver lo que está escribiendo, y cuando se hace la captura sin eco, el usuario ingresa datos por medio del teclado, más lo que escribe no es visible en pantalla.

Código para capturar con eco:

```
                    mov ah,01H
```

Código para capturar sin eco:

```
mov ah,07H
```

O bien:

```
mov ah,08H
```

Procedimiento para convertir a mayúsculas: Etiquetas, comparaciones, saltos y operador lógico AND

```
LEER:    MOV AH,01H
         INT 21H
         CMP AL,"a"
         JB SIGA
         CMP AL,"z"
         JA SIGA
         AND AL,11011111B

SIGA:    MOV [SI],AL
         INC SI
         DEC CX
         CMP CX,0
         JNE LEER
```

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

MOV AH, 01H borra lo que había almacenado de la tecla anteriormente presionada. Le está pidiendo que vaya y se prepare pues el usuario va a presionar una tecla. INT 21H es para que ejecute la captura.

CMP AL,”a” compara la tecla almacenada en AL con la letra “a” (minúscula).

JB SIGA es un salto. Si el carácter está por abajo de “a” en la tabla ASCII, salta a la etiqueta SIGA.

CMP AL, “z” compara la tecla almacenada en AL con la letra “z” (minúscula).

JA SIGA es un salto. Si el carácter está por encima de “z” en la tabla ASCII, salta a la etiqueta SIGA.

Cabe destacar que los valores de los caracteres son manejados como números según su valor en la tabla ASCII. Si está por debajo de “a” (cuyo valor es 97) significa que si valor es menor, y si está por encima de ”z” (cuyo valor es 122) su valor en la tabla ASCII es mayor. Estas dos comparaciones validan si lo que se ingresó está dentro del rango de la “a” a la “z”. Si no está en dicho rango, salta a SIGA. Si lo está, se transforma en mayúsculas con la línea `AND AL,11011111B` .

En la etiqueta SIGA, se tiene MOV [SI], AL va a tomar lo que hay en el carácter ya transformado. INC SI va a incrementar la posición de SI, es decir, va a avanzar un lugar para almacenar la siguiente letra a teclear en el siguiente espacio de memoria de la variable. DEC CX, decrementa el contador en CX. Esto es similar al “i--” del ciclo for en lenguajes de alto nivel. CMP CX,0 compara si el contador es igual a cero. JNE LEER significa que, si el contador no ha llegado al cero, regresará a ejecutar lo que hay en LEER, es decir, continuará el ciclo. De otra manera, habrá terminado.

Imprimir un salto de línea

```
MOV DX,OFFSET CRLF
MOV AH,09H
INT 21H
```

En CRLF, si bien no se asignó una cadena como tal, tiene almacenado el interlineado. Mostrar esta línea es similar al “\n” de los lenguajes de alto nivel.

Capturar dos cadenas, mostrarlas y convertir a mayúsculas la segunda

```
PILA SEGMENT STACK
     DW 64 DUP(?)
PILA ENDS

DATO SEGMENT

     ingresar db 10,13,'Ingrese primera cadena (Enter para
continuar)', 10,13,'$'
     ingresar2 db 10,13,'Ingrese segunda cadena (Enter para
continuar)', 10,13,'$'
     CUENTA=10
     vtext db CUENTA dup('$')
     vtext2 db CUENTA dup('$')
     vtext3 db CUENTA dup('$')
     mayusculas db 10,13,"$"
     mostrando db 10,13,'Mostrando...', 10,13,'$'

     CRLF DB 13,10,"$"
DATO ENDS

CODIGO SEGMENT

     ASSUME DS:DATO,cs:CODIGO,SS:PILA

MAIN: mov ax,DATO
     MOV DS,AX


        lea dx,ingresar
        mov ah,09
        int 21h

     mov si,00h
leer:
        mov ax,0000
        mov ah,01h
        int 21h

        mov vtext[si],al
        inc si
        cmp al,0dh
        ja leer
        jb leer
```

```
 mov ax,DATO
 MOV DS,AX

    lea dx,ingresar2
    mov ah,09
    int 21h


mov si,00h
leer2:
    mov ax,0000
    mov ah,01h
    int 21h

    mov vtext2[si],al
    inc si
    cmp al,0dh
    ja leer2
    jb leer2

ver:
 mov dx,offset CRLF
    mov ah,09h
    int 21h
 mov dx,offset mostrando

    mov ah,09h
    int 21h
 mov dx,offset CRLF
    mov ah,09h
    int 21h

    mov dx,offset vtext

    mov ah,09h
    int 21h
 mov dx,offset CRLF
    mov ah,09h
    int 21h

       mov dx,offset vtext2

    mov ah,09h
    int 21h
       mov dx,offset CRLF
```

```
         mov ah,09h
         int 21h

            MOV DX, OFFSET VTEXT2

            MOV AH,09H
            INT 21H
            call paraMayus


paraMayus proc
      mov ax,DATO
      mov ds,ax
      mov si,0
      JMP otroMayus


otroMayus:
      mov al,vtext2[si]
      cmp al,'$'
      jz finMay
      cmp al,'z'
      jg sigueMay
      cmp al,'a'
      jl sigueMay
      sub al,20H

sigueMay:mov dl,al
      mov ah,2
      int 21h
      inc si
      jmp otroMayus

finMay:    ;comentario
         ;otro comentario
            mov dx,offset mayusculas
            mov ah,09h
            int 21h

paraMayus endp

            MOV AX,4C00H
            INT 21H
CODIGO ENDS
      END MAIN
```

Invertir cadena

```
datos segment
      mensaje1 db "introduzca una cadena",13,10,"$"
      mensaje2 db "la cadena leida fue: $"
      mensaje3 db "la cadena escrita fue: $"
      nuevalinea db 13,10,"$"
      buf db 255 dup(0)
      fub db 255 dup(0)

datos ends

code segment
      assume cs:code,ds:datos
main:
           mov ax,datos
           mov ds,ax

           lea dx,mensaje1
           mov ah,09h
           int 21h

           xor si,si
           xor di,di
           xor cx,cx
           lea si,buf
           lea di,fub

           leer:
                mov ah,1
                int 21h

                cmp al,13
                je mostrarAntes

                mov [si],al
                inc si
                inc cx
                jmp leer

mostrarAntes:
            mov dx,offset mensaje2
            mov ah,09h
            int 21h

            mov dx,offset nuevalinea
```

```
            mov ah,09h
            int 21h

           mov [si],byte ptr"$"


            mov dx,offset buf
            mov ah,09h
            int 21

           invertir:
                dec si
                mov al,[si]
                mov [di],al
                inc di
                loop invertir

           mov [di],byte ptr"$"

            mov dx,offset nuevalinea
            mov ah,09h
            int 21h

            mov dx,offset mensaje3
            mov ah,09h
            int 21h

            mov dx,offset fub
            mov ah,09h
            int 21h

           mov ax,4c00h
           int 21h

code ends
           end main
```

Preparar las variables para invertir

```
           xor si,si
           xor di,di
           xor cx,cx
           lea si,buf
           lea di,fub
```

Los registros SI, DI y CX (contadores) son limpiados con el operador lógico XOR. LEA SI, BUF es para preparar para la captura de caracteres por el usuario. LEA DI, FUB carga en DI el índice de FUB.

Leer, validando final de cadena con un ENTER

```
leer:
          mov ah,1
          int 21h

          cmp al,13
          je mostrarAntes

          mov [si],al
          inc si
          inc cx
          jmp leer
```

La primera línea permite leer lo que se está ingresando. INT 21H permite ejecutar esta acción.

El segundo par de líneas de código comparan lo ingresado con un Enter. Si lo que se ingresó fue un Enter, saltará a la etiqueta mostrarAntes.

Si no se ingresó un Enter, toma lo que el usuario tecleó y lo almacena en el índice o posición SI. Se incrementa el valor de SI para seguir con la siguiente posición. Se incrementa el valor de CX. Salta incondicionalmente a la etiqueta leer, es decir, captura letras hasta que el usuario teclee un Enter. La etiqueta mostrarAntes sirve para mostrar mensaje2.

```
mov [si],byte ptr"$"
```

Esta línea termina la cadena colocándole un “$”.

Invertir la cadena

```
invertir:
                dec si
                mov al,[si]
                mov [di],al
                inc di
                loop invertir

          mov [di],byte ptr"$"
```

Primero decrementa el contador SI puesto que el ENTER se capturó, pero no tiene nada que ver con la cadena. Hacer esto evita basura o elementos no deseados en la cadena.

En la segunda línea toma lo que estaba en el índice o posición y se manda al acumulador en la parte baja. Luego, con la siguiente línea, se pasa lo que hay en AL al otro arreglo (al arreglo de DI). Nótese que en esta parte del código se ha hecho un puenteo, es decir, se empleó una variable intermedia, en este caso AL, para pasar un carácter del arreglo SI al arreglo DI.

Después se incrementa el valor de DI y se procede a ejecutar el ciclo o LOOP de la etiqueta invertir.

Finalmente, fuera de la etiqueta y, por lo tanto, fuera del ciclo, se agrega un “$” al arreglo de DI para terminar la cadena invertida.

Convertir a minúsculas

```
.MODEL SMALL

.STACK     50H

.DATA
      mensaje db 'Digite la cadena a procesar (24 caracteres): $'
      espacio db 13, 10, '$'
      cade    db 25
      tamano  db ?
      cadena  db 25 dup (' '),'$'

.CODE
MINUSCULAS PROC NEAR
      mov ax, @DATA
      mov ds, ax

      LEA DX, mensaje
      MOV AH, 09H
      INT 21H

      MOV AH, 0AH
      LEA DX, cade
      INT 21H

      LEA DX, cadena
      MOV AH, 09H
      INT 21H

      LEA SI, cadena

SEGUIR:
      MOV AL,[SI]
      MOV BL,65
      MOV BL, AL

      CMP BL,AL
      JG MINUS

      CMP AL, 'Z'

      JA minus

      ADD AL, 20H
      MOV [SI], AL
```

```
MINUS:
      INC SI
      MOV CL, 0DH
      CMP[SI], CL

      JNE seguir

      LEA DX, espacio
      MOV AH, 09H
      INT 21H

      LEA DX, cadena
      MOV AH, 09H
      INT 21H

      LEA DX, cadena
      MOV AH, 09H
      INT 21H

      MOV AH,10H
      INT 16H

      ret

      MINUSCULAS ENDP

      MOV AH, 4CH
      INT 21H
END MINUSCULAS
```

Validar rango de letras Mayúsculas y convertirlas

```
SEGUIR:
      MOV AL,[SI]
      MOV BL,65
      MOV BL, AL

      CMP BL,AL
      JG MINUS

      CMP AL, 'Z'

      JA minus

      ADD AL, 20H
      MOV [SI], AL
```

En la primera línea mueve el arreglo SI al acumulador en la parte alta.

La segunda línea sirve para delimitar el límite inferior, es decir, la letra “A”. JG MINUS es un salto hacia la etiqueta MINUS si el valor en BL es mayor a 65 de la tabla ASCII o, dicho de otra manera, la letra “A”. Luego se compara AL con “Z”. Si es mayor, saltará a MINUS. El código anteriormente descrito evalúa si lo ingresado se encuentra en el rango de la “A” a la “Z”. De no estarlo, no se le podrá aplicar la transformación a minúsculas ya que, o no se trata de una letra, o las letras ya están en minúsculas.

ADD AL, 20H es otro modo de convertir cadenas de mayúsculas a minúsculas. Se puede también empleando un AND como en el ejemplo de código de transformar cadena a mayúsculas. Lo que hace ADD AL, 20H es sumarle 20 hexadecimal (que es 32 en decimal) al valor ASCII de la letra, convirtiéndola en minúscula. Tomando el ejemplo de la letra “A”, su valor en la tabla ASCII es 65. Si se le suma 20H o 32 en decimal, da 97, que es el valor de “a”, convirtiéndola en minúscula.

Por último, la cadena transformada se pasa a la cadena en el índice o posición SI.

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

```
MINUS:
      INC SI
      MOV CL, 0DH
      CMP[SI], CL

      JNE seguir
```

La primera línea incrementa SI, la segunda hace un retorno de acarreo, y luego el retorno de acarreo lo compara con el carácter en el índice o posición SI. Si no son iguales, sigue con el ciclo ejecutando el código de la etiqueta SEGUIR, si sí, sale del ciclo. Se usa SI para determinar si el ciclo se ha acabado o no.

Convertir a mayúsculas e invertir cadena por medio de un menú

```
TITLE Programa que muestra un menu para convertir en mayusculas e
invertir cadena

cr EQU 10
lf EQU 13

VARIABLES SEGMENT PARA PUBLIC

     mensaje0                                                    DB
'################################################################
###############$'
     mensaje1   DB   '################################--BIENVENIDO--
###################################$'
     msj DB
     '################################################################
###################$'
     msj1                                                        DB
     '################################################################
###################$'
     mensaje2 DB     'INGRESE LA CADENA: $'
     msj2                                                        DB
     '################################################################
###################$'
     maximo_caracteres DB 40

     lencad DB 0

     cadena DB 40 DUP(0)

     girat DB 40 DUP(0)

     girat2 DB 40 DUP(0)

     mayusculas DB 40 DUP(0)

     linea_en_blanco DB cr,lf,'$'

     mensaje4 DB 'LA CADENA CAMBIADA ES:$'

     mensaje5 DB 'PRESIONE ENTER PARA SEGUIR...$'

     mensaje3 DB 'DESEA SALIR DEL PROGRAMA? (S/N)$'
```

```
      maximo_caracteres2 DB 2

      lencad2 DB 0

      cadena2 DB 2 DUP(0)

      menu db 'Que desea hacer con la cadena ingresada?',13,10

                db '1)Convertirla a mayusculas',13,10
                db '2)Invertir la cadena',13,10
                db '3)Salir$',13,10

VARIABLES ENDS


CODIGO SEGMENT PARA PUBLIC 'code'

      main PROC FAR

ASSUME CS:CODIGO,DS:VARIABLES,SS:pila,ES:VARIABLES

      mov ax,VARIABLES

      mov ds,ax
      mov es,ax

      LIMPIAR MACRO

                     mov ah,0
                mov al,3h

                int 10h


                mov ax,0600h

                mov bh,0fh

                mov cx,0000h
                mov dx,184Fh
                int 10h

                mov ah,02h

                mov bh,00
                mov dh,00
```

```
            mov dl,00
            int 10h
ENDM

GOTOXY MACRO x,y

      xor bh,bh

      mov dl,x

      mov dh,y

      mov ah,02h

      int 10h

ENDM

IMPRIME MACRO arg1

      push ax

      push dx

      lea dx,arg1

      mov ah,9

      int 21h

      pop dx

      pop ax

ENDM

LEE MACRO arg1

      push ax

      push dx
      lea dx,arg1

      mov ah,10

      int 21h
```

```
            pop dx

            pop ax
      ENDM

INVERTIRCADENA MACRO arg1


pushpila:
            mov al,arg1[bx]

            push ax

            inc bl

            cmp bl,lencad

            jne pushpila

            mov bx,0


poppila:
            pop ax

            mov girat[bx],al

            inc bl

            GOTOXY 24,9

            cmp bl,lencad

            jne poppila

            mov girat[bx],'$'

            GOTOXY 22,14

            imprime girat

            IMPRIME linea_en_blanco

            GOTOXY 0,14
            IMPRIME mensaje4
```

```
           jmp opcionDeSalida


ENDM

INVERTIRCADENA2 MACRO arg1


pushpila4:
           mov al,arg1[bx]
           push ax
           inc bl
           cmp bl,lencad
           jne pushpila4
           mov bx,0

poppila4:
           pop ax
           mov girat2[bx],al
           inc bl
           GOTOXY 24,9
           cmp bl,lencad
           jne poppila4
           mov girat2[bx],'$'
           GOTOXY 22,14
           imprime girat2
           IMPRIME linea_en_blanco
           GOTOXY 0,14
           IMPRIME mensaje4
           jmp opcionDeSalida

ENDM

CAMBIARAMAYUSCULAS MACRO

pushpila2:
      mov al,cadena[bx]
      push ax
      inc bl
      cmp bl,lencad
      jne pushpila2
      mov bx,0

poppila2:
      pop ax
      AND AL,11011111B
```

```
      mov girat[bx],al
      inc bl
      GOTOXY 24,9
      cmp bl,lencad
      jne poppila2
      mov girat[bx],'$'
      GOTOXY 22,14
      INVERTIRCADENA2 girat

ENDM

SALIRDELPROGRAMA MACRO
                mov ax,4c00h
                int 21h
ENDM

INICIO:
           LIMPIAR
           GOTOXY 0,0
           IMPRIME mensaje0
           GOTOXY 1,2
           IMPRIME msj
           GOTOXY 1,3
           IMPRIME msj1
           GOTOXY 1,5
           IMPRIME msj2
           GOTOXY 1,1
           IMPRIME mensaje1
           GOTOXY 1,4
           IMPRIME mensaje2
           GOTOXY 20,4
           LEE maximo_caracteres
           IMPRIME linea_en_blanco
           GOTOXY 1,8
           IMPRIME menu
           IMPRIME linea_en_blanco
           mov ah,08
      int 21h

                cmp al,49
                je llamarmayusculas
                cmp al,50
                je  llamarinvertir
                cmp al,51
                je llamarfin
                jmp opcionDeSalida
```

```
                mov bx,0


llamarinvertir:
     CALL IC
llamarmayusculas:
     CALL CM
llamarfin:
     CALL SDP


IC proc near
     INVERTIRCADENA cadena
ret
IC endp

CM proc near
     CAMBIARAMAYUSCULAS
ret
CM endp

SDP PROC NEAR
     SALIRDELPROGRAMA
RET
SDP ENDP

opcionDeSalida:
          xor lencad,0h
          GOTOXY 25,15
          IMPRIME    mensaje5
          LEE maximo_caracteres2
          IMPRIME linea_en_blanco
          GOTOXY 24,16
          IMPRIME mensaje3
          GOTOXY 39,18
          LEE maximo_caracteres2
          IMPRIME linea_en_blanco

               cmp cadena2[0],'s'
               je salir
               cmp cadena2[0],'S'
               je salir
               jmp inicio
```

```
salir:
           mov ax,4c00h
           int 21h

MAIN ENDP
CODIGO ENDS

pila SEGMENT PARA STACK 'stack'
DB 128 DUP(0)
pila ENDS
END MAIN
```

Conversión cadena a mayúsculas, minúsculas e invertir cadena

```
TITLE Programa que solicita al usuario una cadena,  la convierte a mayúsculas, minúsculas y la invierte

cr EQU 10
lf EQU 13

VARIABLES SEGMENT PARA PUBLIC

      mensaje2 DB     'INGRESE LA CADENA: $'
      maximo_caracteres DB 40

      lencad DB 0

      cadena DB 40 DUP(0)

      girat DB 40 DUP(0)

      girat2 DB 40 DUP(0)

      mayusculas DB 40 DUP(0)

      linea_en_blanco DB cr,lf,'$'

      mensaje5 DB 'PRESIONE ENTER PARA SEGUIR...$'

      mensaje3 DB 'DESEA SALIR DEL PROGRAMA? (S/N)$'

      maximo_caracteres2 DB 2

      lencad2 DB 0

      cadena2 DB 2 DUP(0)

VARIABLES ENDS


CODIGO SEGMENT PARA PUBLIC 'code'

      main PROC FAR

ASSUME CS:CODIGO,DS:VARIABLES,SS:pila,ES:VARIABLES

      mov ax,VARIABLES
```

```
mov ds,ax
mov es,ax


LIMPIAR MACRO

                mov ah,0
           mov al,3h

           int 10h


           mov ax,0600h

           mov bh,0fh

           mov cx,0000h
           mov dx,184Fh
           int 10h

           mov ah,02h

           mov bh,00
           mov dh,00
           mov dl,00
           int 10h
ENDM

IMPRIME MACRO arg1

     push ax

     push dx

     lea dx,arg1

     mov ah,9

     int 21h

     pop dx

     pop ax

ENDM
```

```
      LEE MACRO arg1

            push ax

            push dx
            lea dx,arg1

            mov ah,10

            int 21h
            pop dx

            pop ax
      ENDM

INVERTIRCADENA MACRO arg1


pushpila:
            mov al,arg1[bx]

            push ax

            inc bl

            cmp bl,lencad

            jne pushpila

            mov bx,0


poppila:
            pop ax

            mov girat[bx],al

            inc bl

            cmp bl,lencad

            jne poppila

            mov girat[bx],'$'
```

```
            imprime girat

            IMPRIME linea_en_blanco

ENDM

INVERTIRCADENA2 MACRO arg1


pushpila4:
            mov al,arg1[bx]
            push ax
            inc bl
            cmp bl,lencad
            jne pushpila4
            mov bx,0

poppila4:
            pop ax
            mov girat2[bx],al
            inc bl
            cmp bl,lencad
            jne poppila4
            mov girat2[bx],'$'
            imprime girat2
            IMPRIME linea_en_blanco
            ;IMPRIME mensaje4
            ;jmp opcionDeSalida

ENDM

INVERTIRCADENA3 MACRO arg1


pushpila6:
            mov al,arg1[bx]
            push ax
            inc bl
            cmp bl,lencad
            jne pushpila6
            mov bx,0

poppila6:
            pop ax
            mov girat2[bx],al
            inc bl
```

```
            cmp bl,lencad
            jne poppila6
            mov girat2[bx],'$'
            imprime girat2
            IMPRIME linea_en_blanco
            ;IMPRIME mensaje4
            ;jmp opcionDeSalida

ENDM

CAMBIARAMAYUSCULAS MACRO

pushpila2:
      mov al,cadena[bx]
      push ax
      inc bl
      cmp bl,lencad
      jne pushpila2
      mov bx,0

poppila2:
      pop ax
      AND AL,11011111B
      mov girat[bx],al
      inc bl
      cmp bl,lencad
      jne poppila2
      mov girat[bx],'$'
      INVERTIRCADENA2 girat
ENDM

CAMBIARAMINUSCULAS MACRO

pushpila3:
      mov al,girat2[bx]
      push ax
      inc bl
      cmp bl,lencad
      jne pushpila3
      mov bx,0

poppila3:
      pop ax
      ADD AL,20H
      mov girat[bx],al
      inc bl
```

```
      cmp bl,lencad
      jne poppila3
      mov girat[bx],'$'
      INVERTIRCADENA3 girat
ENDM


  SALIRDELPROGRAMA MACRO
                mov ax,4c00h
                int 21h
ENDM

INICIO:
           LIMPIAR
           IMPRIME mensaje2
           LEE maximo_caracteres
           IMPRIME linea_en_blanco
           IMPRIME linea_en_blanco


                CAMBIARAMAYUSCULAS
                mov bx,0
                CAMBIARAMINUSCULAS
                mov bx,0
                INVERTIRCADENA cadena
                mov bx,0
                jmp opcionDeSalida
                SALIRDELPROGRAMA

opcionDeSalida:
           xor lencad,0h
           IMPRIME    mensaje5
           LEE maximo_caracteres2
           IMPRIME linea_en_blanco
           IMPRIME mensaje3
           LEE maximo_caracteres2
           IMPRIME linea_en_blanco

                cmp cadena2[0],'s'
                je salir
                cmp cadena2[0],'S'
                je salir
                jmp inicio

salir:
           mov ax,4c00h
```

```
          int 21h

MAIN ENDP
CODIGO ENDS

pila SEGMENT PARA STACK 'stack'
DB 128 DUP(0)
pila ENDS
END MAIN
```

Mostrar dígitos hexadecimales de una cadena (Mostrar carácter por caracter)

```
.MODEL SMALL
.CODE
BEGIN:
      MOV AX,@DATA
      MOV DS,AX

      MOV DX,OFFSET Titulo
      MOV AH,09
      INT 21H

      MOV CX,16
      MOV BX, OFFSET Cadena

CICLO:
      MOV AL,CL
      XLAT

      MOV DL,AL

      MOV AH,02
      INT 21H

      MOV DL,10
      INT 21H

      MOV DL,13
      INT 21H

      LOOP CICLO

      MOV AH,4CH
      INT 21H

.DATA
Titulo     DB 13,10,'Lista de numeros hexadecimales del 0 al 15'
                DB 13,10,'$'
      Cadena     DB ' FEDCBA9876543210'

.STACK

END BEGIN
```

Recorrer cadena

```
MOV CX,16
MOV BX, OFFSET Cadena
```

Se asigna 16 al contador. La segunda línea permite el acceso a la cadena donde se encuentran los valores a mostrar.

```
CICLO:
MOV AL,CL
XLAT
```

MOV AL,CL Coloca en AL el número a traducir.

XLAT es una instrucción que convierte el contenido de AL en un número almacenado en la tabla de memoria. Una instrucción XLAT primero añade el contenido de AL a BX para formar una dirección de memoria junto con un segmento de datos. Entonces copia el contenido de esta dirección a AL. Esta es la única instrucción que añade un número de 8 bits a un número de 16 bits.

MOV DL,AL Coloca en DL lo que se va a mostrar.

MOV AH, 02 es una función para mostrar un carácter.

MOV DL, 10 Salta una línea desplegando el carácter 10, lo cual en el código ASCII significa una nueva línea o un salto de línea.

```
MOV DL,13
INT 21H
```

Imprime un ENTER o retorno de carro, cuyo código ASCII es el 13.

LOOP CICLO repite el ciclo el número de veces que indique el valor almacenado en CX.

Imprimir cadena en diagonal

```
.MODEL SMALL
.CODE
INICIO:
      mov ax,@data
      mov ds,ax

      mov dh,posy
      mov dl,posx
      mov ah,0
      mov al,3
      int 10h

      mov dx,offset mensaje
      mov ah,9
      int 21h

      mov dx,offset cadena
      mov ah,0ah
      int 21h

      mov di,offset cadena
      mov dl,byte ptr[di+1]
      mov cl,dl
      mov dh,00
      mov ch,00
      add di,cx
      mov byte ptr[di+2],'$'
      inc dx
      mov temp,dx
      mov di,offset cadena+2

      mov dh,posy
      mov dl,posx
      mov ah,0
      mov al,3
      int 10h

imprime: mov bh,0
      mov dh,posy
      mov dl,posx
      inc posy
      inc posx
      mov ah,02
      int 10h
```

```
      mov ah,0ah
      mov al,byte ptr[di]
      mov cx,1
      inc di
      int 10h
      dec temp
      mov cx,temp
      loop imprime

      mov ax,4c00h
      int 21h

.DATA
mensaje db 'Escriba una cadena de caracteres: ', 10, 13, '$'
cadena db 255,0,255 DUP(?)
posx db 0
posy db 0
temp dw 0
.STACK
END INICIO
```

Iniciar el video en modo 3

```
      mov dh,posy
      mov dl,posx
      mov ah,0
      mov al,3
      int 10h
```

Posiciona en X y Y según los valores de posy y posx. Para esto utiliza el registro de datos en la parte alta y baja. Manda un 03 al registro acumulador para inicializar el video en modo 3. INT 10H es para ejecutar el cambio.

Leer cadena

```
      mov dx,offset cadena
      mov ah,0ah
      int 21h
```

0ah es la instrucción para leer desde el teclado. Lo que se escriba se almacenará en cadena.

Obtener la longitud de una cadena utilizando sumas

```
      mov di,offset cadena
      mov dl,byte ptr[di+1]
      mov cl,dl
```

```
      mov dh,00
      mov ch,00
      add di,cx
      mov byte ptr[di+2],'$'
      inc dx
      mov temp,dx
      mov di,offset cadena+2
```

En la primera línea se posiciona en la variable cadena.

En la segunda manda lo que hay en la posición DI+1 a dl.

En la tercera mueve lo que hay en DL a CL

En la cuarta y quinta se limpian los registros DH y CH, asignándoles ceros.

En la sexta se suman los valores en DI y CX. La suma se almacena en DI.

En la séptima se añade un “$” a la cadena.

En la octava se incrementa DX

En la novena se mueve lo que hay en DX a la variable temp, la cual contendrá la longitud de la cadena.

Luego pone en DI la dirección de la primera letra de la cadena.

Imprimir cadena en diagonal

Posicionar cursor

```
imprime: mov bh,0
      mov dh,posy
      mov dl,posx
      inc posy
      inc posx
      mov ah,02
      int 10h
```

En la primera línea se limpia el registro BH.

En la segunda y tercera línea se mandan las coordenadas de X y Y a las partes del registro de datos.

En la cuarta y quinta línea se incrementan los valores de las posiciones, esto para dar el efecto en diagonal.

MOV AH, 02, como ya se mencionó anteriormente, sirve para mostrar el carácter. Para esto debe ir acompañado de INT 10H.

Imprimir el carácter

```
mov ah,0ah
mov al,byte ptr[di]
mov cx,1
inc di
int 10h
dec temp
mov cx,temp
loop imprime
```

En la primera línea se solicita que se ingrese un enter. Al ingresar el enter, en la siguiente línea, se mueve el carácter que se encuentra en el índice DI de la cadena al registro acumulador en la parte baja, se incrementa CX en 1, se decrementa la longitud de cadena, CX toma el valor de la longitud de cadena y procede a ejecutar el ciclo de imprime. Este código toma cada letra de la cadena y la imprime en forma de diagonal.

Capítulo 4. Manipulación de pantalla

Cambiar color de fondo

```
PILA SEGMENT STACK 'STACK'
     DB    100H DUP (?)
PILA ENDS

DATOS SEGMENT
     MENSAJE DB 'Se ha cambiado el color de la pantalla$'
DATOS ENDS

CODIGO SEGMENT
     ASSUME CS:CODIGO, SS:PILA, DS:DATOS

     INICIO:          MOV AH,06H
                 MOV BH,2FH
                 MOV CX,0000H
                 MOV DX,184FH
                 INT 10H

                 MOV AH,02H
                 MOV BH,00H
                 MOV DX,0000H
                 INT 10H

                 MOV AX,DATOS
                 MOV DS,AX
                 LEA DX,MENSAJE
                 MOV AH,9H
                 INT 21H

                 MOV AX,4C00H
                 INT 21H

CODIGO ENDS
END INICIO
```

Cambiar el modo de video

```
           MOV AH,06H
           MOV BH,2FH
           MOV CX,0000H
           MOV DX,184FH
           INT 10H
```

MOV AH, 06H maneja el scroll up de la ventana.

MOV BH, 2FH usa los primeros 4 bits para el fondo y los otros 4 para la letra. En este caso, el 2 es para el fondo color verde y la letra f el color blanco de las letras.

En MOV CX, 0000h los primeros 8 bits son para el renglón en la esquinza superior izquierda del rectángulo .

En MOV BH,2FH settea el renglón y columna de la esquina inferior derecha en los que se aplican los colores .

INT 10H es una interrupción de video. Sirve para ejecutar el cambio que se hará al modo de video. El código anteriormente descrito cambia color de fondo y de la letra.

```
MOV AH,02H
MOV BH,00H
MOV DX,0000H
INT 10H
```

MOV AH, 02H posiciona el cursor.

MOV BH,00H pone el modo de video en modo texto de 16 colores y 8 páginas.

En MOV DX,0000H indica el renglón y columna de la esquina superior izquierda.

Programa con opciones para seleccionar color de fondo

```
pila segment stack 'stack'
     db    100h dup (?)
pila ends

datos segment
     titulo db 13,10,'                CAMBIAR FONDO DE PANTALLA
',13,10,'$'
     mensaje db 'Presione ENTER si quiere la pantalla azul. Si
quiere morado, presione 2',13,10,'Para salir, presione cualquier
tecla',13,10,'$'

datos ends

codigo segment             ;segmento de codigo
     assume cs:codigo, ss:pila, ds:datos

     inicio:

                mov ah,0
                mov al,3h
                int 10h

                mov ax,0600h
                mov bh,0fh
                mov cx,0000h
                mov dx,184Fh
                int 10h

                mov ah,02h
                mov bh,00
                mov dh,00
                mov dl,00
                int 10h


                mov ax,datos
                mov ds,ax

                lea dx,titulo
                mov ah,9h
                int 21h

                lea dx,mensaje
                mov ah,9h
```

```
                int 21h

                mov ah,08
                int 21h

                cmp al,13
                je  llamarAzul


                cmp al,50
                je  llamarMorado

                jmp fin

     fin:

                mov ax,4c00h
                int 21h
llamarAzul:
     CALL AZULPROC

llamarMorado:
     CALL MORADOPROC

AZULPROC PROC NEAR


mov ah,0
                mov al,3h
                int 10h

                mov ax,0600h
                mov bh,0fh
                mov cx,0000h
                mov dx,184Fh
                int 10h

                mov ah,02h
                mov bh,00
                mov dh,00
                mov dl,00
                int 10h

                mov ah,06h
                mov bh,1fh
                mov cx,0000h
```

```
                mov dx,184fh
                int 10h

                mov ax,4c00h
                int 21h
RET

AZULPROC ENDP


MORADOPROC PROC NEAR

                mov ah,06h
                mov bh,5fh
                mov cx,0000h
                mov dx,184fh
                int 10h


                mov ax,4c00h
                int 21h
RET

MORADOPROC ENDP

codigo ends
end inicio
```

Posicionar carácter en ciertas coordenadas de la pantalla

```
gotoxy macro fila,col
      mov ah,02h
      mov dh,fila
      mov dl,col
      mov bh,0h
      int 10h
endm

pantalla macro que
      mov ah,02h
      mov dl,que
      int 21h
endm

.model small
.data
.code
startup:
      mov ax,@data
      mov ds,ax
      mov ax,0003h
      int 10h
      gotoxy 10,10
      pantalla 41h
      mov ah,01h
      int 21h
      mov ax,4c00h
      int 21h
end startup
```

Posicionar en pantalla

```
gotoxy macro fila,col
      mov ah,02h
      mov dh,fila
      mov dl,col
      mov bh,0h
      int 10h
endm
```

MOV AH, 02H posiciona el cursor. Las siguientes dos líneas de código se asignan al registro de datos. MOV BH, 0H es para enviarle un cero a BH, INT 10 es la interrupción de video.

Jalar el carácter para posicionarlo

```
pantalla macro que
      mov ah,02h
      mov dl,que
      int 21h
endm
```

MOV DL, QUE manda el argument a imprimir a la columna correspondiente.

Posicionar cadena indicada por el usuario en coordenadas indicadas por el usuario

```
TITLE Cadena que solicita una cadena y una posición para mostrarla

gotoxy macro fila,col
      mov ah,02h
      mov dh,fila
      mov dl,col
      mov bh,0h
      int 10h
endm

pantalla macro que
      mov ah,02h
      mov dl,que
      int 21h
endm

imprime macro eztryng
      mov dx,offset eztryng
      mov ah,9
      int 21h
endm

DATOS SEGMENT PARA PUBLIC
      mensaje         DB "INGRESE UN CARACTER: ",13,10,"$"
      mensaje2        DB "INGRESE X del 0 al 9: ",13,10,"$"
      mensaje3        DB "INGRESE Y del 0 al 9: ",13,10,"$"
      caracter   DB 40
      varx       DB ?
      vary       DB ?

vtext db 100 dup('$')

DATOS ENDS
CODIGO SEGMENT PARA PUBLIC 'code'
ASSUME CS:CODIGO,DS:DATOS
startup:
      mov ax,DATOS
      mov ds,ax

      imprime mensaje

       mov si,00h
    leer:
```

```
        mov ax,0000
        mov ah,01h
        int 21h

        mov caracter[si],al
        inc si
        cmp al,0dh
        ja leer
        jb leer

       mov dx,offset caracter
       mov ah,0ah
       int 21h

      imprime caracter

      imprime mensaje2

      mov ah,01h
    int 21h

    sub al,30h
      mov bl,al

      mov varx,al

      imprime mensaje3

      mov ah,01h
    int 21h

    sub al,30h
      mov bl,al
      mov vary,al

      mov ax,0003h
      int 10h
      gotoxy vary,varx
      pantalla caracter[0]

      mov ah,01h
      int 21h
      mov ax,4c00h
      int 21h
CODIGO ENDS
end startup
```

Manipular caracteres numéricos para usarlos como números

```
mov ah,01h
int 21h

sub al,30h
mov bl,al

mov varx,al
```

Dado que Ensamblador maneja la entrada de teclado como caracteres, es necesario aplicar cierta conversión a los números para que estos se puedan manejar como tal.

En las primeras dos líneas se captura el valor numérico. En la tercera resta 30 hexadecimal al valor de la tabla ASCII del número. El resultado se almacena en AL. Luego el valor en este registro se pasa a BL y, finalmente, AL se pasa a la variable varx, la cual se enviará como un parámetro al macro para ubicar el carácter en pantalla.

Capítulo 5. Conversiones

Convertir letras a valor binario

```
DATOS SEGMENT
        ASCII DB 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'
        BINARY DB 64 DUP (?)
        NL DB 13,10,'$'
DATOS ENDS

CODIGO SEGMENT
        ASSUME DS:DATOS, CS:CODIGO
MAIN:
        MOV AX,DATOS
        MOV DS,AX
        MOV AX,0003H
        INT 10H

        LEA SI,ASCII
        LEA DI,BINARY
        MOV DX,OFFSET ASCII
        MOV AH,09H
        INT 21H
        CALL ASCII2BIN
        MOV [DI],BYTE PTR'$'
        MOV DX,OFFSET NL
        MOV AH,09H
        INT 21H
        MOV DX,OFFSET BINARY
        MOV AH,09H
        INT 21H
        MOV AH,01H
        INT 21H
        MOV AX,4C00H
        INT 21H

ASCII2BIN PROC NEAR
        XOR AX,AX
        MOV CX,8

ASCII1:
        MOV AL,[SI]
        PUSH CX
        MOV CX,8

LOOP_SHIFT:
        SHL AL,1
```

```
        JC BIN_UNO
        MOV [DI], BYTE PTR '0'
        JMP CICLO_SHIFT

BIN_UNO:
        MOV [DI], BYTE PTR '1'

CICLO_SHIFT:
        INC DI
        LOOP LOOP_SHIFT
        POP CX
        INC SI
        LOOP ASCII1
        RET


ASCII2BIN ENDP
CODIGO ENDS
END MAIN
```

Preparar modo de video

```
MOV AX,0003H
INT 10H
```

MOV AX,0003H es otra manera de poner el modo de video.

Llamada a de Procedimientos almacenados (Stored procedures) para la conversión de carácter a binario

LEA SI, ASCII va a la variable ASCII y toma la letra “A”.

LEA DI, BINARY hace que, el dato donde se guardará la conversión, apunte hacia el primer índice, la primera posición.

MOV DX, OFFSET ASCII carga lo que está en la variable ASCII. En el primer ciclo toma la “A”

MOV AH, 09H Se prepara para mostrar la letra

INT 21H interrupción para mostrar.

CALL ASCII2BIN llama al procedimiento almacenado ASCII2BIN.

MOV [DI], BYTE PTR ‘$’ Agrega un “$” a la cadena para que se puedan mostrar los datos binarios.

MOV DX, OFFSET NL hace un salto de línea. Muestra la variable que tiene almacenado un salto de línea.

MOV AH, 01H espera a que se ingrese un carácter, el que sea. En este programa, esta línea se utiliza para que el usuario, al presionar cualquier tecla, termine el programa. Una vez capturada la tecla, procede a devolver el control al BIOS y termina su ejecución.

Procedimientos almacenados, limpieza de registros, ciclo LOOP y uso de la pila

```
ASCII2BIN PROC NEAR
      XOR AX,AX
      MOV CX,8
```

Todos los procedimientos almacenados deben seguir la siguiente sintaxis:

Nombre PROC [NEAR|FAR]

Código

RET

Nombre ENDP

Donde NEAR se utiliza para procedimientos en el mismo programa, FAR cuando vienen de otro lado, RET es un return o regreso. Si falta este marcará un error. ENDP es para terminar el procedimiento.

XOR AX, AX es un operador lógico. Su tabla de verdad es:

X	**Y**	**Resultado**
Falso	Falso	Falso
Falso	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Verdadero	Verdadero	Falso

Lo que hace es que, si hay algo almacenado en AX, lo va a eliminar, limpiando así el registro. Otra manera de hacer esto es enviarle ceros de la siguiente manera

```
MOV AX, 0H
```

Esto es de gran utilidad cuando se requiere limpiar un registro para evitar que los datos manejados se empalmen y se genere “basura” (que es cuando aparecen caracteres que no deberían aparecer al mostrar un resultado).

MOC CX,8 es para iniciar en contador en 8, esto para que la función LOOP sepa cuántas veces debe de repetir el ciclo.

```
ASCII1:
```

```
MOV AL,[SI]
PUSH CX
MOV CX,8
```

Este ciclo irá de letra en letra.

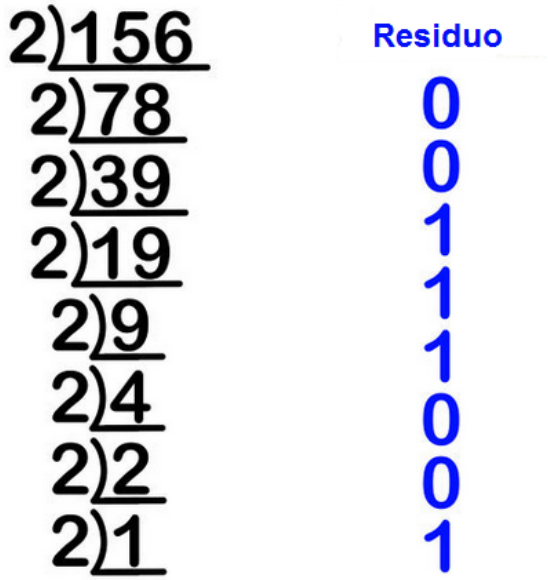
El MOV AL, [SI] mueve el siguiente carácter ASCII al AL. Esto se maneja como un arreglo. Se le dice al programa “Ve esa posición (índice) y dime lo que hay guardado”.

PUSH CX va a meter el valor del contador temporalmente a la pila. Esto es similar a cuando una persona está cargando con muchas cosas, toma su celular y lo deja sobre una mesa mientras se administra cómo cargar todo. Luego, cuando ya requiera tomar de nuevo su celular, lo quitará de la mesa.

MOV CX, 8 se le asigna 8 al contador (de ahí la importancia de respaldar el viejo valor). Este valor se utilizará para otro ciclo dentro del ciclo principal. El ciclo principal va a ir de letra en letra para convertirla a binario. El segundo ciclo va a servir para crear los 1’s y 0’s para la conversión.

```
LOOP_SHIFT:
     SHL AL,1
     JC BIN_UNO
     MOV [DI], BYTE PTR '0'
     JMP CICLO_SHIFT
```

Este ciclo producirá el número binario para la letra correspondiente.



SHL AL,1 en esta línea se va a hacer un Shift a la izquierda. Esto va a mover el siguiente número a la bandera de acarreo CF (Carry flag). Esto se hace porque el sistema binario crece hacia la izquierda. Para convertir un número de decimal a binario, se toma el número a convertir y se divide entre dos. Si la división da un número decimal, se quita la parte decimal para la siguiente división y se coloca un 1 en el residuo. Por ejemplo, 19 entre 2 da 9.5, entonces se toma en cuenta solamente el 9, y se coloca un 1 en el acarreo. En caso contrario, como es el caso de la división de 4 entre 2, se pone un 0. De esta manera, 156 decimal es igual a 00111001 en binario.

En JC BIN_UNO se está usando un Jump On Carry, es decir, un salto si existe un acarreo. Si el número almacenado en el acarreo (residuo) es 1, el siguiente número debe ser 1, entonces salta a la etiqueta BIN_UNO.

MOV [DI], BYTE PTR ‘0’ Si no, el número será cero.

JMP CICLO_SHIFT. Este es un salto incondicional, es decir, no importa los resultados, se va a ir a la etiqueta CICLO_SHIFT.

```
BIN_UNO:
      MOV [DI], BYTE PTR '1'
```

Esta etiqueta pone un 1 a la cadena convertida.

```
CICLO_SHIFT:
      INC DI
      LOOP LOOP_SHIFT
      POP CX
      INC SI
      LOOP ASCII1
      RET

ASCII2BIN ENDP
```

INC DI incrementa DI, el cual permite moverse en la variable binaria. Esto con el fin de que las letras no se almacenen todas en el mismo espacio de memoria, sino que se vaya recorriendo para no empalmar los datos.

LOOP LOOP_SHIFT ejecutará el ciclo realizando lo que hay en la etiqueta LOOP_SHIFT el número de veces que diga CX. El CX que se está manejando aquí es inicialmente de 8. Se usó 8 porque es el número de dígitos que se desea tenga el número binario generado.

POP CX saca el CX de pila. Esto hace una vez que se ha convertido la letra en binario. Este nuevo valor de CX es para trabajar con el siguiente carácter.

INC SI incrementa el valor de SI. Es para que se pueda mover al siguiente carácter y no trabajar siempre con la “A”. Ahora irá con la “B”.

LOOP ASCII ejecuta el ciclo para el siguiente valor ASCII.

RET es para terminar el procedimiento almacenado y regresar al código de donde se le llamó.

ASCII2BIN ENDP indica que ahí acabó el procedimiento.

Capítulo 6. Validación por entrada de teclado

Mostrar mensaje en modo de video, validar ENTER y cualquier tecla

```
.MODEL SMALL
.STACK 100H
.DATA
     TITULO DB 13,10,'    MODO VIDEO    ',13,10
            DB 13,10,'LENGUAJE DE INTERFAZ',13,10,13,10,13,10,'$'

     SALIDA DB 13,10,13,10,'PULSE UNA TECLA PARA SALIR', 13,10,'$'

     TEXTO DB 'PRUEBA DE MENSAJE','$'

.CODE

INICIO:    MOV AH,0FH
     INT 10H

     MOV AH,0
     INT 10H

     MOV AX,@DATA
     MOV DS,AX

     MOV DX,OFFSET TITULO
     MOV AH,09H
     INT 21H

     MOV AH,0AH
     INT 21H

FINAL:
     MOV DX,OFFSET SALIDA
     MOV AH,09H
     INT 21H
     MOV AH,0

     INT 16H

     MOV AH,4CH
     INT 21H

END INICIO
```

Programa con directivas simplificadas de segmento

```
.MODEL SMALL
.STACK 100H
.DATA
```

Es otra manera de declarar los segmentos. En esta, como se mencionó anteriormente, no es necesario indicar el final de los segmentos. Ensamblador lo hace automáticamente en cuanto encuentra otro punto. Para terminar el segmento de código, se termina cerrando una etiqueta.

Cambiar modo de video

```
MOV AH,0FH
INT 10H
```

La primera línea cambia el modo de video. Nótese que se está asignando a la parte alta del registro acumulador.

```
MOV AH,0H
INT 10H
```

Limpia la pantalla. Borra el texto que había y lo deja vacío.

Validación de entrada de teclado

```
MOV AH,0AH
INT 21H
```

Este código pide la entrada de un Enter.

```
MOV AH, 0
Int 16h
```

Esta línea espera a que el usuario presione cualquier tecla. La Interrupción 16H hace que, mientras no se presione alguna tecla, el programa no continúe.

Posicionar menú en pantalla

```
TITLE El siguiente programa invierte una cadena ingresada por el
usuario

cr EQU 10
lf EQU 13

VARIABLES SEGMENT PARA PUBLIC
     mensaje0                                                    DB
'################################################################
###############$'
     mensaje1   DB   '################################--BIENVENIDO--
###################################$'
     msj DB
     '################################################################
###################$'
     msj1                                                        DB
     '################################################################
###################$'
     mensaje2 DB     'INGRESE LA CADENA: $'
     msj2                                                        DB
     '################################################################
###################$'

     maximo_caracteres DB 40

     lencad DB 0

     cadena DB 40 DUP(0)

     girat DB 40 DUP(0)

     linea_en_blanco DB cr,lf,'$'


     mensaje4   DB 'LA CADENA CAMBIADA ES:$'

     mensaje5   DB 'PRESIONE ENTER PARA SEGUIR...$'

     mensaje3   DB 'DESEA SALIR DEL PROGRAMA? (S/N)$'

     maximo_caracteres2    DB 2

     lencad2 DB 0
```

```
      cadena2 DB 2 DUP(0)

VARIABLES ENDS

CODIGO SEGMENT PARA PUBLIC 'code'

      main PROC FAR

            ASSUME CS:CODIGO,DS:VARIABLES,SS:PILA,ES:VARIABLES

            mov ax,VARIABLES
            mov ds,ax
            mov es,ax

            LIMPIAR MACRO
                  mov ax,0600h
                  mov bh,127
                  mov cx,0000h
                  mov dx,184fh
                  int 10h
            ENDM

            GOTOXY MACRO x,y
                  xor bh,bh
                  mov dl,x
                  mov dh,y
                  mov ah,02h
                  int 10h
            ENDM

            IMPRIME MACRO arg1
                  push ax
                  push dx
                  lea dx,arg1
                  mov ah,9
                  int 21h
                  pop dx
                  pop ax
            ENDM

            LEE MACRO arg1
                  push ax
                  push dx
                  lea dx,arg1
                  mov ah,10
                  int 21h
```

```
                pop dx
                pop ax
            ENDM

INICIO:
            LIMPIAR
            GOTOXY 0,0
            IMPRIME mensaje0
            GOTOXY 1,2
            IMPRIME msj
            GOTOXY 1,3
            IMPRIME msj1
            GOTOXY 1,5
            IMPRIME msj2
            GOTOXY 1,1
            IMPRIME mensaje1
            GOTOXY 1,4
            IMPRIME mensaje2
            GOTOXY 20,4
            LEE maximo_caracteres
            IMPRIME linea_en_blanco
            mov bx,0

pushpila:
      mov al,cadena[bx]
      push ax
      inc bl
      cmp bl,lencad
      jne pushpila
      mov bx,0

poppila:
      pop ax
      mov girat[bx],al
      inc bl
      GOTOXY 24,9
      cmp bl,lencad
      jne poppila
      mov girat[bx],'$'
      imprime girat
      IMPRIME linea_en_blanco

      GOTOXY 1,9
      IMPRIME mensaje4
      GOTOXY 25,14
      IMPRIME mensaje5
```

```
      LEE maximo_caracteres2
      IMPRIME linea_en_blanco

      GOTOXY 24,15
      IMPRIME mensaje3
      GOTOXY 39,17
      LEE maximo_caracteres2

      cmp cadena2[0],'s'
      je salir
      cmp cadena2[0],'S'
      je salir
      jmp inicio

salir:
      mov ax,4c00h
      int 21h

main ENDP
CODIGO ENDS

pila SEGMENT PARA STACK 'stack'
      DB  128 DUP(0)
pila ENDS

END main
```

Título

```
TITLE El siguiente programa invierte una cadena ingresada por el usuario
```

Puede omitirse.

Variables globales

```
cr EQU 10
lf EQU 13
```

Las variables globales se declaran e identifican fuera de los segmentos. Esto hace que sean visibles para todos los segmentos.

Segmento extra

Como se mencionó anteriormente, existen ocasiones en las que es necesario o se desea usar un segmento extra. En este código se utilizó dicho segmento en vez del segmento de datos, lo cual se indica mediante la sentencia asume.

```
ASSUME CS:CODIGO,DS:VARIABLES,SS:PILA,ES:VARIABLES
```

Se asume que el segmento VARIABLES será empleado como segmento de datos y segmento extra. Por esto, se hace una triple parametrización.

```
mov ax,VARIABLES
mov ds,ax
mov es,ax
```

MACROS

Una MACRO es un nombre que define un conjunto de instrucciones que serán sustituidas por la macro cuando el nombre de esta aparezca en un programa en el momento de ensamblarlo. Las instrucciones se pueden almacenar en el programa mismo o en otro archivo. Su sintaxis es:

Nombre MACRO argumento1, argumento2….

Código

ENDM

Macro para limpiar pantalla

```
LIMPIAR MACRO
                mov ax,0600h
                mov bh,127
                mov cx,0000h
                mov dx,184fh
                int 10h
ENDM
```

En la primera línea, 06 es un recorrido, 00 es pantalla completa.

En la segunda, calcula las direcciones de los pixeles.

En la tercera, posiciona la esquina superior izquierda.

En la cuarta, posiciona la esquina inferior derecha.

INT 10 es la interrupción para ejecutar los cambios en el modo de video.

Macro para posicionar por medio de coordenadas

```
GOTOXY MACRO x,y
                xor bh,bh
                mov dl,x
                mov dh,y
                mov ah,02h
                int 10h
ENDM
```

Nótese que este MACRO recibe dos parámetros, argumentos, valores o cosas.

En la primera línea limpia el registro BH.

En la segunda envía el valor de X al registro de datos en la parte baja

En la tercera envía el valor de Y al registro de datos en la parte alta

Macro para imprimir

```
IMPRIME MACRO arg1
                push ax
                push dx
                lea dx,arg1
                mov ah,9
                int 21h
                pop dx
                pop ax
ENDM
```

Este MACRO recibe un argumento. Mete AX y DX en la pila para almacenarlos temporalmente. Se posiciona en el parámetro arg1 (el valor que se le envió) y lo manda a DX, manda la instrucción para mostrar, con su correspondiente instrucción. Luego de que lo hubo mostrado, se sacan AX y DX de la pila. Esto es para que, al mostrar los datos, no afecte los valores de AX y DX.

Macro para leer

```
LEE MACRO arg1
                push ax
                push dx
                lea dx,arg1
                mov ah,10
                int 21h
                pop dx
                pop ax
ENDM
```

Almacena temporalmente AX y DX en la pila, se posiciona en arg1, da la instrucción para leer desde el teclado (MOV AH, 10), usa la interrupción 21H para ejecutar la captura, luego se sacan AX y DX de la pila.

Llamar macros

```
INICIO:
          LIMPIAR
          GOTOXY 0,0
          IMPRIME mensaje0
          GOTOXY 1,2
```

```
IMPRIME msj
GOTOXY 1,3
IMPRIME msj1
GOTOXY 1,5
IMPRIME msj2
GOTOXY 1,1
IMPRIME mensaje1
GOTOXY 1,4
IMPRIME mensaje2
GOTOXY 20,4
LEE maximo_caracteres
IMPRIME linea_en_blanco
mov bx,0
```

Nótese que los macros se llaman solamente mencionando su nombre. En caso de que se quieran enviar parámetros, se indican a un lado del nombre del MACRO.

Invertir cadena

```
pushpila:
      mov al,cadena[bx]
      push ax
      inc bl
      cmp bl,lencad
      jne pushpila
      mov bx,0

poppila:
      pop ax
      mov girat[bx],al
      inc bl
      GOTOXY 24,9
      cmp bl,lencad
      jne poppila
      mov girat[bx],'$'
      imprime girat
      IMPRIME linea_en_blanco
```

En pushpila: Se mueve la cadena ingresada al acumulador en la parte baja. Se mueve todo el registro AX a la pila para almacenamiento temporal. Se incrementa el contador BL, se compara BL con la longitud de cadena, si no es igual, es decir, si el ciclo no se ha terminado, se repite pushpila, si sí, se limpia el registro BX.

En poppila: Se saca AX de la pila. De ahí se extrae la cadena del registro y se manda a girat. Incrementa BL. Llama al macro GOTOXY para posicionarse en pantalla. Compara BL con la longitud de cadena, si no son iguales, repite poppila, si sí, manda un “$” a girat. Muestra la cadena invertida. Luego imprime una línea en blanco.

ANEXOS

Tabla de saltos

Instrucción			Condición
JZ	Jump if Zero	salta si cero	ZF=1
JNZ	Jump if Not Zero	salta si no cero	ZF=0
JC	Jump if Carry	salta si acarreo	CF=1
JNC	Jump if Not Carry	salta si no acarreo	CF=0
JO	Jump if Overflow	salta si overflow	OF=1
JNO	Jump if Not Overflow	salta si no overflow	OF=0
JS	Jump if Sign	salta si signo	SF=1
JNO	Jump if Not Sign	salta si no signo	SF=0
JP/JPE	Jump if Parity (Parity Even)	salta si paridad (Paridad Par)	PF=1
JNP/JPO	Jump if Not Parity (Parity Odd)	salta si no paridad (Paridad Par)	PF=0

Instrucción			Condición
JA	Jump if Above	salta si por encima	A>B (sin signo)
JAE	Jump if Above or Equal	salta si por encima o igual	A>=B (sin signo)
JB	Jump if Below	salta si por debajo	A<B (sin signo)
JBE	Jump if Below or Equal	salta si por debajo o igual	A<=B (sin signo)
JE	Jump if Equal	salta si igual	A=B
JG	Jump if Greater	salta si mayor	A>B (con signo)
JGE	Jump if Greater or Equal	salta si mayor o igual	A>=B (con signo)
JL	Jump if Less	salta si menor	A<B (con signo)
JLE	Jump if Less or Equal	salta si menor o igual	A<=B (con signo)

Errores más comunes a la hora de programar en ensamblador

Al ensamblar o linkear un programa, es probable que surjan errores severos o *severe errors*. La manera más eficiente de detectar el error es ubicar:

- Línea de código donde se encuentra el error
- Código de error
- Descripción del error

Conociendo alguno de los 3 datos es posible detectar el error. Sin embargo, en caso de no obtener una respuesta clara, a continuación se presentan algunas de las causas más comunes por las que se producen errores.

Nombre de archivo demasiado largo

Este error suele reconocerse por el mensaje **Unable to open input file: nombreDelPrograma.asm**.

Identificadores (variables) duplicados y referencias a éstos

Tanto en lenguajes de alto nivel como en Ensamblador, no debe de haber dos o más variables con el mismo nombre. Por lo general, se le puede identificar por **Syntax error**.

Mala asignación de datos a segmentos

Esto es causado por no realizar un puenteo adecuado al mover datos, o bien, asignar una cadena directamente a un registro al tratar de mostrar una cadena.

Referencia a identificador, etiqueta, procedimiento almacenado o MACRO inexistente

Eso sucede cuando, o se hace referencia a un identificador no existente, o se escribió mal.

Instrucciones PUSH y POP que no coinciden

Esto sucede cuando se quiere sacar un dato temporalmente almacenado en la pila (con una instrucción POP) cuando no se ha puesto nada en ella por medio de la instrucción PUSH. Esto sucede porque no hay nada que sacar.

Olvidar escribir un RET (return/retorno) en un procedimiento almacenado

Hacer esto provocará que la ejecución del programa continúe en el código después del procedimiento.

Bibliografía

Rincón solidario. (n.d.). *Rincón solidario*. Retrieved Marzo 22, 2016, from Rincón solidario: http://www.rinconsolidario.org/eps/asm8086/CAP6.html

Technical University of Denmark. (2015, Diciembre 23). *agner.* Retrieved Marzo 24, 2016, from agner: http://www.agner.org/optimize/optimizing_assembly.pdf