# ESTRUCTURA DE UN PROGRAMA EN ENSAMBLADOR

Sistemas Digitales con Microprocesadores.

M.C. Juan Carlos Olguín R.

# Sobre el lenguaje Ensamblador

Primeros Pasos.

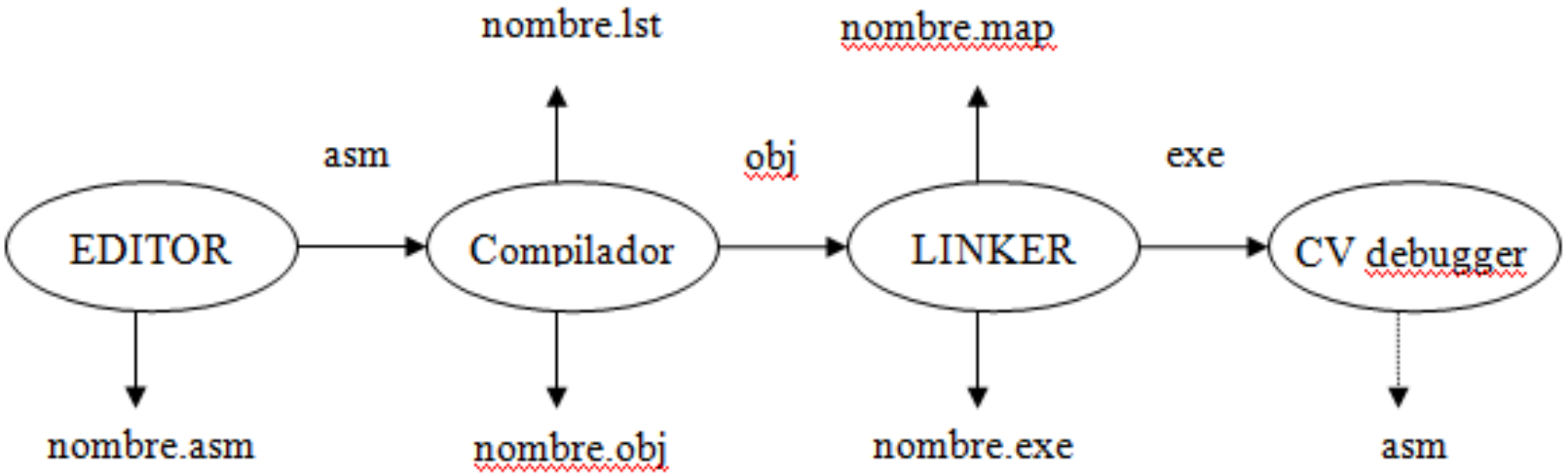
# Software necesario

- Primero un editor para crear el programa fuente.
- Segundo un compilador que no es mas que un programa que "traduce" el programa fuente a un programa objeto.
- Y tercero un enlazador o linker, que genere el programa ejecutable a partir del programa objeto.

- El editor puede ser cualquier editor de textos que se tenga a la mano
- como compilador utilizaremos el MASM (macro ensamblador de Microsoft) ya que es el mas común
- y como enlazador utilizaremos el programa link.

- La extensión usada para que MASM reconozca los programas fuente en ensamblador es .ASM; una vez traducido el programa fuente, el MASM crea un archivo con la extensión .OBJ, este archivo contiene un "formato intermedio" del programa, llamado así porque aun no es ejecutable pero tampoco es ya un programa en lenguaje fuente. El enlazador genera, a partir de un archivo .OBJ o la combinación de varios de estos archivos, un programa ejecutable, cuya extensión es usualmente .EXE

# Diagrama del proceso de ensamblado de un programa.



# Formato interno de un programa

- Para poder comunicarnos en cualquier lenguaje, incluyendo los lenguajes de Programación es necesario seguir un conjunto de reglas, de lo contrario no podrá expresar lo que deseamos.

# Funcionamiento interno (ejecución de un programa)

- Para que un microprocesador ejecute un programa es necesario que éste haya sido ensamblado, enlazado y cargado en memoria.
- Una vez que el programa se encuentra en la memoria, el microprocesador ejecuta los siguientes pasos:
- 1.- Extrae de la memoria la instrucción que va a ejecutar y la coloca en el registro interno de instrucciones.
- 2.- Cambia el registro apuntador de instrucciones (IP) de modo que señale a la siguiente instrucción del programa.

- 3.- Determina el tipo de instrucción que acaba de extraer.
- 4.- Verifica si la instrucción requiere datos de la memoria y, si es así, determina donde están situados.
- 5.- Extrae los datos, si los hay, y los carga en los registros internos del CPU.
- 6.- Ejecuta la instrucción.
- 7.- Almacena los resultados en el lugar apropiado.
- 8.- Regresa al paso 1 para ejecutar la instrucción siguiente.
- Este procedimiento lo lleva a cabo el microprocesador millones de veces por segundo.

# Programas En Ensamblador

- El **lenguaje ensamblador** es un lenguaje que utiliza símbolos en lugar de instrucciones codificadas en lenguaje máquina. También permite hacer referencia a las localidades de memoria por nombres simbólicos en lugar de usar sus direcciones directamente.

- Un programa escrito en lenguaje ensamblador, llamado código fuente, es traducido al código máquina de un microprocesador mediante un programa llamado **ensamblador.**

# Estructura De Un Programa En Ensamblador

- Un programa en el lenguaje ensamblador consiste de una secuencia de proposiciones, **una en cada línea del código fuente.**
- La sintaxis de una proposición es la siguiente:

- [*etiqueta] [instrucción|directiva [operandos]] [;comentario]*

- Donde: *etiqueta, instrucción|directiva, operandos, comentario son los campos de la proposición*.
- Los campos se separan por caracteres blancos: caracteres de espacio y/o de tabulación. Todos los campos son opcionales, pero el campo *operandos sólo pueden estar presente si existe el campo instrucción|directiva*.
-
- ***etiqueta*** *es un nombre simbólico empleado para referirse a números, cadenas de caracteres o* localidades de memoria dentro de un programa. Las etiquetas permiten darle nombre a las variables, constantes y localidades de una instrucción particular.

- Las etiquetas pueden contener los siguientes caracteres: A-Z a-z _ @ $ ? 0-9. Los dígitos 0-9 no pueden usarse como el primer carácter de una etiqueta. Un sólo carácter $ o ? no puede usarse como etiqueta ya que tienen un significado especial.
- Las etiquetas deben ser únicas. Con excepción de las definidas con la directiva =, y las etiquetas locales de macros y subrutinas. Las etiquetas pueden usarse como operandos tantas veces como se desee.
- Una etiqueta puede aparecer por sí sola en una línea. En este caso el valor asociado a la etiqueta es la dirección de la operación en la siguiente línea en el programa.
- Las palabras reservadas no pueden usarse como etiquetas.
- Las etiquetas que aparecen solas en una línea y las que van seguidas de una instrucción deben terminar en dos puntos (:). Las demás etiquetas por lo general no terminan en dos puntos.

- ***instrucción*** *es un mnemónico de una instrucción del procesador. Cada instrucción se traduce* directamente a una instrucción del lenguaje máquina del microprocesador.

- ***directiva*** *es una instrucción para el ensamblador. No produce código ejecutable, sino que controla* varios de los aspectos de cómo opera el ensamblador: el tipo de código generado (8086, 80286,80386, etc.), los segmentos a usar, etc.

- ***operandos*** *consiste de cero, uno o más operandos. Un operando le especifica al ensamblador qué* valor, registro, localidad de memoria, etc., asociar con cada instrucción.

- Hay varias clases de **operandos**: registros, constantes, etiquetas, variables y cadenas.
- El tipo y número de operandos depende de la instrucción o directiva.
- Si hay dos o más, van separados por comas (,)

- ***comentario*** *es cualquier secuencia de caracteres precedidos por un punto y coma (;) y que termina con* el fin de la línea.
- Los comentarios son ignorados por el ensamblador
- Las líneas en blanco se tratan como comentarios.

# Por ejemplo

- considere el programa de la siguiente página. Los números a la izquierda del recuadro no forman parte del programa y sólo sirven para numerar las líneas:

- Las líneas 1 a 7, 9, 16, 20, 28 y 45 son líneas con sólo comentarios.
- Las líneas 8, 10, 15, 17, 19, 21, 27, 29, 34, 39, 44 y 46 son líneas en blanco y son tratadas como comentarios. Se usan para separar las partes de un programa y darle mayor claridad.
- Las líneas 31 y 40 son líneas con sólo una etiqueta
- Las líneas 32, 33, 35 a 38 y 41 a 43 contienen instrucciones.
- Las líneas 11 a 14, 18, 22 a 26, 30 y 47 contienen directivas.

```
;*****************************************************************
; PRIMER.ASM.
;
; Este programa ilustra la estructura de un programa en
; ensamblador. También muestra las principales directivas
; para El ensamblador.
;*****************************************************************

;****** CÓDIGO DE INICIO ************************************

        ideal
        dosseg
        model small
        stack 256

;****** DECLARACIÓN DE CONSTANTES SIMBÓLICAS ***************

cte     equ     10

;****** VARIABLES DEL PROGRAMA *****************************

        dataseg
codsal  db      0
dato1   db      ?
dato2   db      ?
resul   db      ?

;****** CÓDIGO DEL PROGRAMA ********************************

codeseg
inicio:
        mov     ax, @data    ; Inicializa el segmento de
        mov     ds, ax       ; datos

        mov     al, [dato1]  ; resul = dato1 + dato2 + cte
        add     al, [dato2]
        add     al, cte
        mov     [resul], al

salir:
        mov     ah, 04Ch
        mov     al, [codsal]
        int     21h

;****** CÓDIGO DE TERMINACIÓN *******************************

        end    inicio
```

# Programa en Ensamblador

- Normalmente podemos considerar que un programa en ensamblador está formado de las siguientes secciones:

✓Código de inicio
✓Declaración de constantes simbólicas
✓Variables del programa
✓Código del programa
✓Código de terminación

# 1.- Código De Inicio

- El código de inicio de un programa en ensamblador para el 8086 tiene como propósito declarar los nombres de los segmentos en los que se almacenará el código del programa, los datos y la pila.

- También establece el modelo de memoria bajo el que se ensamblará el programa y el tamaño de la pila.

- El código de inicio consta de un conjunto de directivas. En el programa que vimos de ejemplo tenemos:

# Directivas

- **Ideal**
- El modo **ideal** es una notación para simplificar la sintaxis del código inicial de un programa. La directiva **ideal** le dice al Turbo Assembler (Turbo Assembler de Borland International) que utilice el modo ideal. Si se omite se utiliza la notación empleada por la mayoría de los ensambladores, por ejemplo el de Microsoft.

# dosseg

- Esta directiva que forma parte del modo ideal, le instruye al ensamblador que nombre y ordene los segmentos del programa de la misma forma en que lo hacen los compiladores de alto nivel. Esto es adecuado para la mayoría de los programas en ensamblador y sobre todo en las rutinas de ensamblador que se van a ligar a un programa en alto nivel.

# model

- Esta directiva que debe ir después de la directiva **dosseg,** selecciona uno de los 6 modos diferentes que tiene Turbo Assenbler de generar el código de un programa dependiendo del tamaño que tiene el código y los datos.
- Estos modos llamados **modelos de memoria,** difieren en el tipo de apuntadores que el compilador genera por omisión y por lo tanto en la rapidez de ejecución del código.
- Los seis modelos de memoria son: Pequeñito (tiny), pequeño (small), medio (medium), compacto (compact), grande (large) y extenso (huge).

# Modelo pequeñito (tiny)

- En el modelo pequeñito, el código, los datos y la pila deben de estar en un solo segmento, esto es, el programa completo no debe exceder a 64 KB. Todos los registros de segmento son inicializados al mismo valor y permanecen constantes durante la ejecución del programa.
- Bajo este modelo el compilador genera apuntadores cercanos. El código ejecutable producido es el más pequeño y rápido.

# Modelo pequeño (small)

- En el modelo pequeño, el código tiene su propio segmento y los datos y la pila están en otro segmento, esto es, el código no debe exceder a 64 KB ni tampoco los datos y pila. Los registros de segmento permanecen constantes durante la ejecución del programa.
- El código producido bajo este modelo utiliza también apuntadores cercanos y es igual de rápido que en el modelo pequeñito pero el tamaño del programa puede ser del doble.
- Normalmente debemos usar este modelo de memoria a menos que nuestro código o datos exceda los 64 KB.

# Modelo medio (medium)

- En el modelo medio, el código del programa puede ocupar varios segmentos pero los datos y la pila ocupan un solo segmento, esto es, el código puede exceder a 64 KB pero los datos y pila no.
- Por lo tanto, bajo este modelo el código utiliza apuntadores lejanos y los datos apuntadores cercanos.
- Debido a los apuntadores lejanos la velocidad de ejecución del programa se ve degradada.
- Los registros de los segmentos de datos y la pila se inicializan al mismo valor y permanecen constantes durante la ejecución del programa.
- Este modelo se debe usar cuando tengamos un programa de código extenso y pocos datos.

# Modelo compacto (compact)

- En el modelo compacto, el código del programa está limitado a un segmento, los datos pueden ocupar varios segmentos aunque el total de las variables estáticas no pueden exceder los 64 KB.
- Por lo tanto, bajo este modelo el código utiliza apuntadores cercanos y los datos apuntadores lejanos.
- Debido a los apuntadores lejanos también hay degradación en la velocidad de ejecución del programa.
- El registro del segmento de código permanece constante durante la ejecución del programa.
- Este modelo se debe usar cuando tengamos un programa de código pequeño y muchos datos.

# Modelo grande (large)

- En el modelo grande, tanto el código del programa como los datos puede ocupar varios segmentos, aunque el total de las variables estáticas no pueden exceder los 64 KB.
- Los apuntadores en este modelo son lejanos tanto para el código como para los datos.
- Esto hace que la velocidad de ejecución de un programa compilado bajo este modelo sea más lenta que cualquiera de las versiones anteriores.
- Este modelo se debe usar cuando tengamos un programa de código extenso y muchos datos.

# Modelo extenso (huge).

- En el modelo extenso, es igual que el modelo grande, sólo que el total de las variables estáticas si pueden exceder los 64 KB.
- Sin embargo cada variable estática en forma individual no puede exceder a 64 KB.
- Los apuntadores en este modelo son lejanos tanto para el código como para los datos.
- La velocidad de ejecución de un programa compilado bajo este modelo es más lenta que cualquiera de los otros modelos.

# stack

- Esta directiva reserva espacio de memoria para la pila del programa, un área de memoria que almacena valores temporales empleados por las subrutinas y las direcciones de regreso de las subrutinas.

# 2.- Declaración de constantes simbólicas

- Las constantes simbólicas son identificadores que le asociamos a constantes numéricas o a constantes cadenas. El uso de constantes simbólicas en un programa hace que sea más fácil de leer y modificar.

- Hay dos formas de declarar constantes simbólicas:
- mediante la directiva **equ** y mediante la directiva **=**

# Ejemplo:

```
filas   equ     10
cols    equ     5
tam     =       filas * cols
msj     equ     "Hola"
tam     =       0
```

Las directivas **equ** e **=** tienen las siguientes reglas:

- Las constantes simbólicas no son variables. Ni el nombre ni su valor asociado se almacena en el segmento de datos del programa.
- La directiva **equ** permite darle nombre a constantes numéricas, expresiones y cadenas.
- **La directiva =** sólo permite darle nombre a constantes numéricas y expresiones.

- No se le puede cambiar el valor asociado a un símbolo declarado con la directiva **equ.** En cambio a un símbolo declarado con la directiva = se le puede cambiar su valor tantas veces como se desee.
- Las directivas **equ** y = pueden aparecer en cualquier parte del programa. Pero es preferible colocarlas al principio ya que son más visibles.
- Las expresiones declaradas con = se evalúan inmediatamente, mientras que las expresiones declaradas con **equ** se evalúan hasta que el símbolo se usa.

# 3.-Variables del programa

- Las variables de un programa se declaran en el segmento de datos, por lo que la sección de declaración de variables empieza con la directiva **dataseg**. Esta directiva le indica al ensamblador que almacene las variables en el segmento de datos del programa. Las variables pueden estar inicializadas o no. Los valores de las variables inicializadas se almacenan en el código del programa y se cargan en las variables al ejecutar el programa.

- Para declarar variables podemos utilizar las directivas **db, dw, dd o dq. La directiva db** define una variable de un byte, la directiva **dw** define una variable de tipo palabra (dos bytes), la directiva **dd** define una variable de tipo palabra doble (cuatro bytes) y la directiva **dq** define una variable de tipo palabra cuádruple (ocho bytes).

# La sintaxis de estas directivas es:

```
nomVar    db    exp
nomVar    dw    exp
nomVar    dd    exp
nomVar    dq    exp
```

donde *nomVar es el nombre de la variable y exp es una expresión constante cuyo valor se utiliza para* inicializar la variable.

Si la variable no se inicializa *exp se substituye por un signo de interrogación,* ***?*****.**

# Ejemplo:

```
bdato   db      10h                 ; Variable de un byte
                                    ; inicializada a 10h.

cresp   db      'A'                 ; Variable de un byte
                                    ; inicializada a 65d.

bdatx   db      ?                   ; Variable de un byte
                                    ; no inicializada.

wdato   dw      0123h               ; Variable de tipo
                                    ; palabra inicializada
                                    ; a 0123h.

wdatx   dw      ?                   ; Variable de tipo
                                    ; palabra no
                                    ; inicializada.

ddato   dd      01234567h           ; Variable de tipo
                                    ; palabra doble
                                    ; inicializada a
                                    ; 01234567h.

ddatx   dd      ?                   ; Variable de tipo
                                    ; palabra doble
                                    ; no inicializada

qdato   dq      0123456789ABCDEFh   ; Variable de tipo
                                    ; palabra cuadruple
                                    ; inicializada a
```

# 4.- Código del programa

- En esta sección van las instrucciones que el microprocesador ejecutará al correr el programa. Todas las instrucciones van en el segmento de código, por lo que la sección de código del programa empieza con la directiva **codeseg.**
- Esta directiva le dice al ensamblador que almacene las instrucciones en el segmento de código del programa.

- El código del programa en la mayoría de los casos empieza con las siguientes líneas:

```
        codeseg
inicio:
        mov     ax, @data       ; Inicializa el segmento
        mov     ds, ax          ; de datos
```

La primera línea contiene la directiva **codeseg** mencionada anteriormente**.**

La segunda línea establece que ***inicio*** *es una etiqueta asociada a la dirección de la primera instrucción* del programa, que en este ejemplo aparece en la siguiente línea.

Esta etiqueta es usada en la directiva **end explicada en la siguiente sección.**

En la tercera y cuarta línea se carga en el registro DS la dirección de inicio del segmento de datos del programa. El símbolo predefinido **@data** tiene el valor de esa dirección.
La instrucción **mov** mueve datos entre registros o entre registros y localidades de memoria. En este ejemplo la primera instrucción **mov** almacena la dirección del inicio del segmento de datos en el registro **AX** y la segunda instrucción mov mueve esa dirección del registro **AX** al registro de segmento **DS.**

# Nota:

**No**
se puede cargar una dirección directamente a uno de los registros de segmento, por lo que la instrucción:

```
mov     ds, @data          ; No válido
```

**no es válida.**

La dirección debe cargarse primero a un registro de propósito general y luego de ahí moverse al registro de segmento

También la mayoría de los programas que se van a ejecutar bajo el sistema operativo MSDOS deben terminar con las siguientes líneas de código:

```
salir:
        mov     ah, 04Ch
        mov     al, [codsal]
        int     21h
```

El propósito de este código es llamar a una de las rutinas del sistema operativo que termina la ejecución del programa y nos regresa al sistema operativo. Esta rutina que se conoce como una rutina de servicio a interrupción se verá más adelante en el tema de: TIPOS INTERRUPCIONES.

La llamada a esa rutina se hace mediante la instrucción: **int 21**

Es decir, Previamente se cargó en el registro **AH** el valor **04Ch** que especifica que se desea terminar el programa y regresar al sistema operativo y en el registro **AL** el valor de la variable ***codsal,*** *que es el código de* salida del programa.

El programa le regresa al sistema operativo el valor del código de salida en la variable de ambiente **errorlevel.** Por convención, un código de salida de cero denota que el programa terminó con éxito y un código diferente de cero indica que hubo un error durante la ejecución del programa.

El valor del código identifica al error ocurrido.

En los programas en C también se acostumbra regresarle un código de salida al sistema operativo. Esto se hace mediante la proposición return en la función *main():*

```
int main(void)
{
    ...

    return 0;
}
```

# 5.- Código de terminación

```
end  [etiqueta]
```

Todo programa en ensamblador debe contener una directiva **end,** la cual marca el final del código fuente.

El ensamblador ignora cualquier línea después de la directiva **end.**

***etiqueta*** *es el nombre simbólico de la dirección de la instrucción por la que empezará a ejecutarse el* programa. Si se omite, el programa empezará su ejecución en la primera instrucción del código fuente.

En un programa de un sólo módulo, es decir con un archivo con código fuente, la directiva **end debe** siempre especificar el inicio del programa. En un programa que consiste de más de un módulo, sólo la directiva **end** en el módulo que contiene la primera instrucción a ejecutar debe llevar una etiqueta. En los otros módulos la directiva **end** debe aparecer por sí sola.