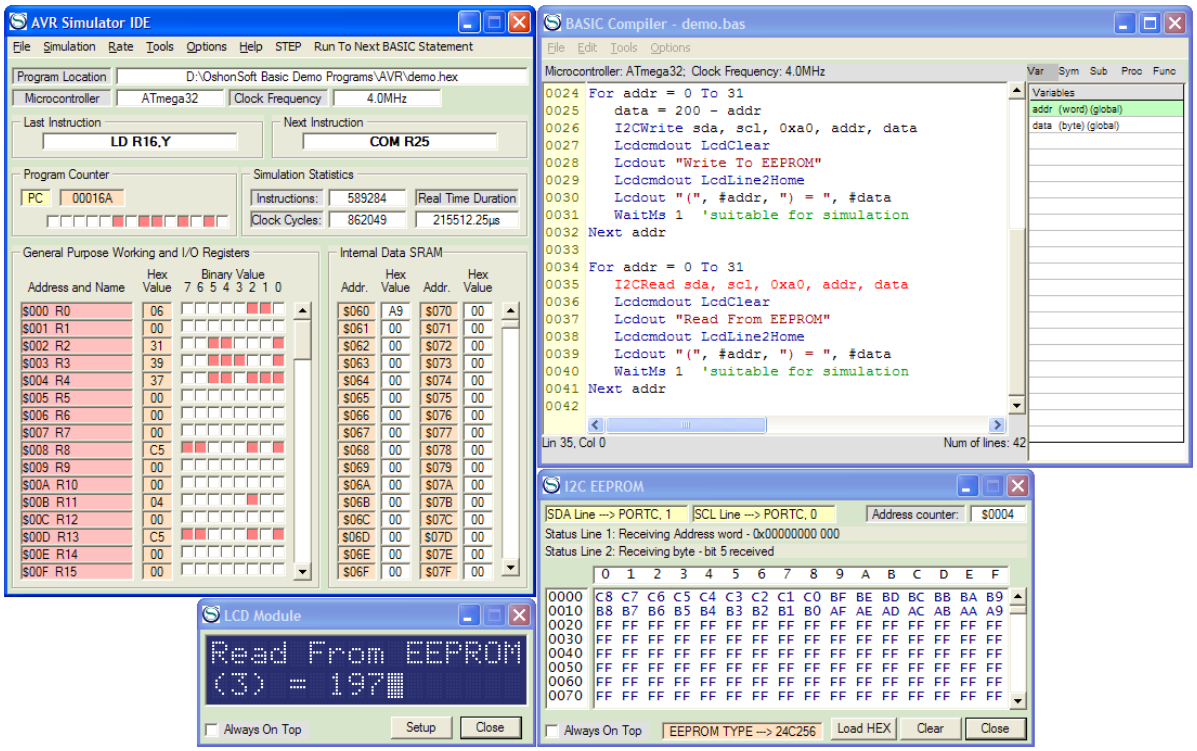
Pontificia Universidad Católica del Ecuador

Nombre: Heidi Noemí Valle García Materia: Arquitectura de Computadoras

ASSEMBLER IDE 2.0



ENLACE PARA DESCARGAR: https://assembler-ide.programas-gratis.net/

Lenguaje de "Bajo Nivel": el usuario se acerca un poco más al lenguaje de máquina. Permiten un acceso más amplio al control físico de la maquina (hardware).

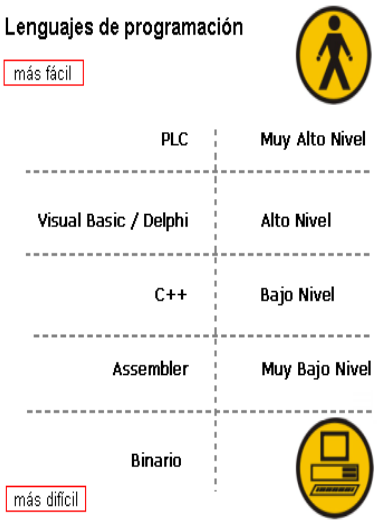
ASSEMBLER

Lenguaje ensamblador de IDE

Inicio como hacer correr un programa en assembler.

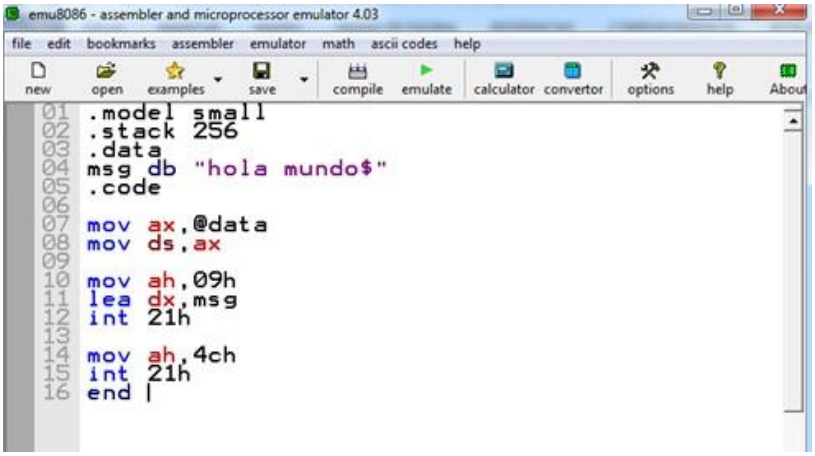
Lo primero que tienen que hacer para correr un programa en assembler es primero crear un programa para este caso haremos el clásico hola mundo, para ello lo haremos un editor de texto para este ejemplo trabajaremos con el notepad++.

Que pueden descargar de forma gratuita desde su página oficial una vez descargado e instalado procederemos a hacer el programa, para ello en la parte de lenguaje del notepad++ seleccionen la parte que dice assembler



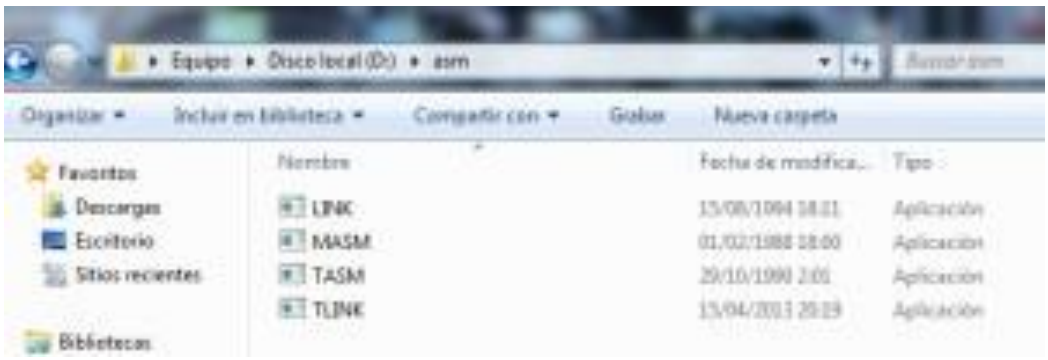
Y CREAMOS EL PROGRAMA QUE IMPRIMA EL HOLA MUNDO.

```
.model small
.stack 256
.data
msg db “hola mundo$” ;comentarios
.code
mov ax,@data
mov ds,ax

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

mov ah,4ch
int 21h
end
```

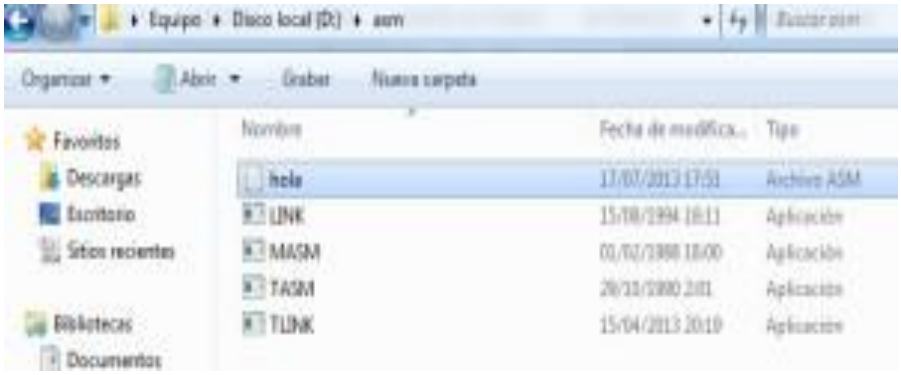


NOTA. - antes de continuar explicaremos un poco el código. Primero definimos el modelo de memoria con .model existen muchos tipos de memoria los más conocidos son el tyni que se usa para programas pequeños, el small para programas medianos y el large para programas largos. Después el tamaño de la pila que su función es un poco más avanzada y conocerán su función cuando hagan programas más complejos pero por ahora solo vean lo como el tamaño del programa, luego definimos las variables de datos con un .data que es donde están la variables y declaramos una variable que se llame msg y lo podemos como tipo de dato que sea DB (define byte) existen más datos como el DW el DD que verán el funcionamiento después, luego ponemos él .code que es donde va el código, para más información acerca del lenguaje assembler se le recomienda leer el libro introducción al lenguaje ensamblador de peter Abel o los artículos de internet de abre los ojos al ensamblador:).

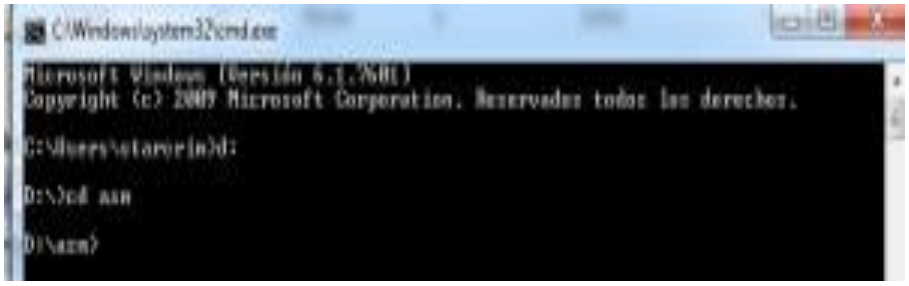
Luego guardamos el programa con la extensión .asm (verificar si tiene esa extensión), guardarlo en una carpeta donde tenga el masm, tasm, tasm y tlink. (luego en el final pronde un link para que descarguen los programas mencionados).



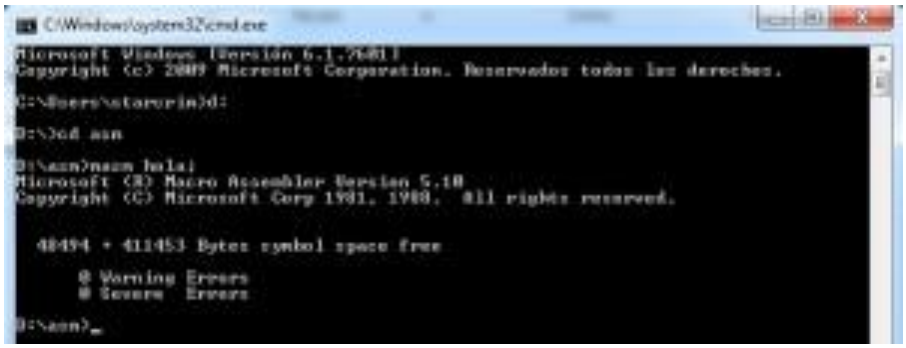
Guarde el programa con el nombre de hola.asm.



LUEGO ENTRAR AL DOS.
Y ENTRAR A LA RUTA DONDE ESTÁ GUARDADO EL PROGRAMA QUE ES D:\asm.



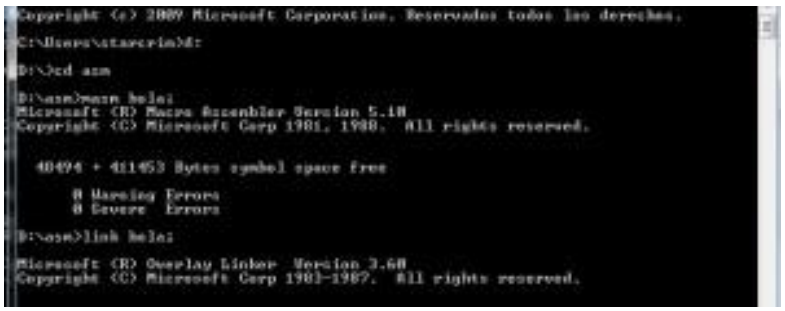
1. primera compilación usando el masm(Microsoft macro assembler).
   Colocamos masm hola; y verificamos que no tenga errores.



En este caso vemos que dice 0 severe Errors(0 errores severos) y 0 warning errors(0 errores peligrosos).

Algunos programas pueden tener 0 errores severos y 1 peligro, pero cuando hay 0 o n peligros el programa igual corre por q solo es una advertencia pero si tiene solo 1 error severo el

programa no puede correr (eso se darán cuenta con la practica), luego colocamos el link hola;



Por último colocamos hola y esto es lo que debería aparecer.

Si verifican la carpeta donde está el programa vemos que se crearon dos nuevos archivos como se muestra a continuación:

2. COMPILACIÓN CON EL tasm(TURBO ASSEMBLER)

Se sigue el mismo procedimiento de la anterior compilación con el masm.
Solo que en vez de colocar masm se coloca tasm y en vez de link tlink.

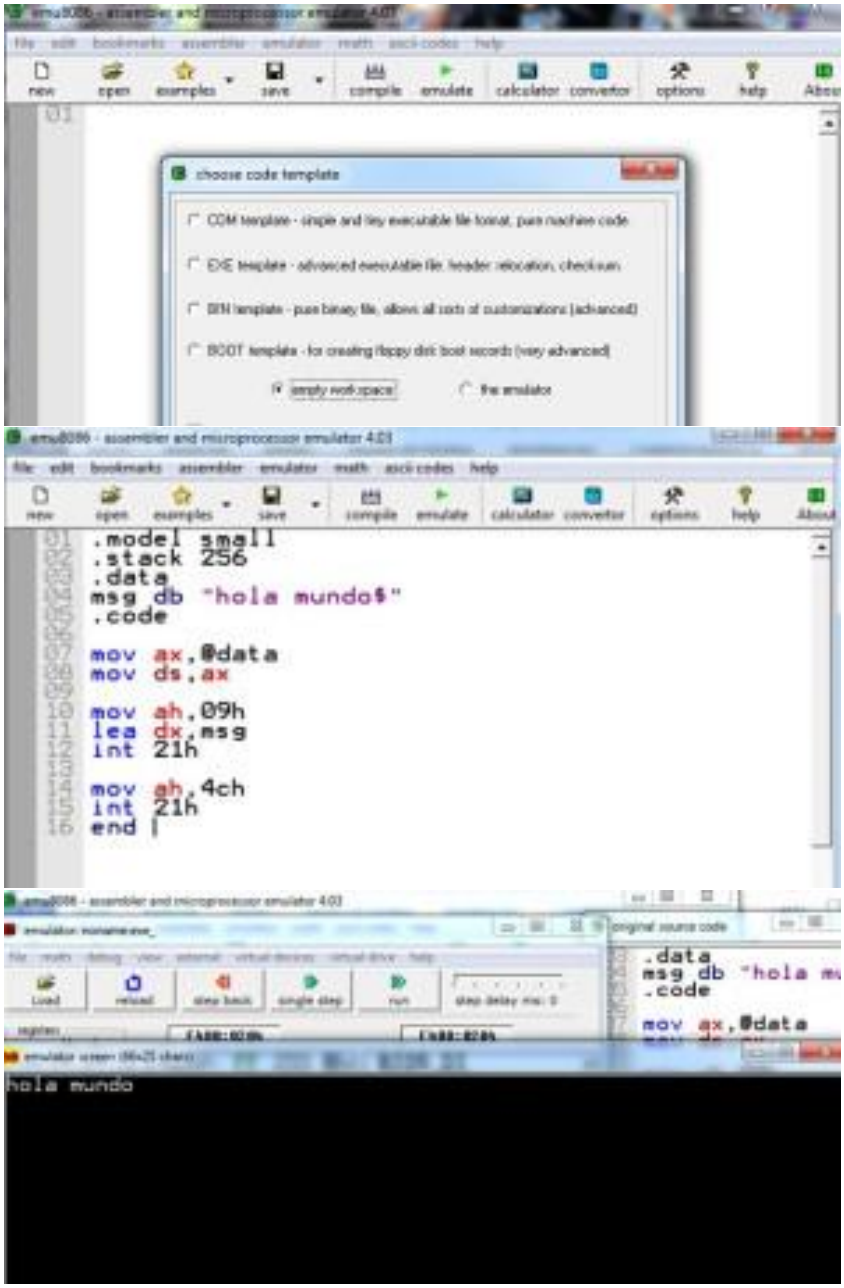
Los warning y errores son los mismos que el masm

Las diferencias se los dejo de tarea, pero se darán cuenta que por un pequeñísimo margen el tasm es mejor que el masm por que tiene funcione propias del compilador y ademas nos permite hacer más saltos que para el masm son errores de compilacion.

3. Ejecución usando el emulator 8086.

Como siempre se deja lo más fácil al final :P, el emu8086 es un ide tipo eclipse, codeblock etc para assembler donde primero creamos un programa en blanco seleccionando la parte de NEW.

Luego seleccionamos la parte que dice documento en blanco y copiamos el código para hacer corre el programa nos situamos en la flecha verde que dice emulate y le damos click para hacer correr y luego le damos Run.



Lo malo del emu8086 es que no tiene todas a las interrupciones del assembler como ser rastreo de teclado modo gráfico y modo de video archivos y algunas más pero para empezar a aprender el lenguaje ensamblador en muy buena herramienta.

REGISTROS DE LA UCP

La UCP tiene 14 registros internos, cada uno de 16 bits. Los primeros cuatro, AX, BX, CX, y DX son registros de uso general y también pueden ser utilizados como registros de 8 bits, para utilizarlos como tales es necesario referirse a ellos como, por ejemplo: AH y AL, que son los bytes alto (high) y bajo (low) del registro AX. Esta nomenclatura es aplicable también a los registros BX, CX y DX.

Los registros son conocidos por sus nombres específicos:

- AX Acumulador
- BX Registro base
- CX Registro contador
- DX Registro de datos
- DS Registro del segmento de datos
- ES Registro del segmento extra
- SS Registro del segmento de pila
- CS Registro del segmento de código
- BP Registro de apuntadores base
- SI Registro índice fuente
- DI Registro índice destino
- SP Registro del apuntador de la pila
- IP Registro de apuntador de siguiente instrucción
- F Registro de banderas

Es posible visualizar los valores de los registros internos de la UCP utilizando el programa Debug. Para empezar a trabajar con Debug digite en el prompt de la computadora:

C:\> **Debug** [Enter]

En la siguiente linea aparecera un guión, éste es el indicador del Debug, en este momento se pueden introducir las instrucciones del Debug. Utilizando el comando:

– **r** [Enter]

Se desplegaran todos los contenidos de los registros internos de la UCP; una forma alternativa de mostrarlos es usar el comando “r” utilizando como parametro el nombre del registro cuyo valor se quiera visualizar. Por ejemplo:

– **rbx**

Esta instrucción desplegará unicamente el contenido del registro BX y cambia el indicador del Debug de **” – ”** a **” : ”**

Estando así el prompt es posible cambiar el valor del registro que se visualizó tecleando el nuevo valor y a continuación [Enter], o se puede dejar el valor anterior presionando [Enter] sin telclear ningún valor.

Es posible cambiar el valor del registro de banderas, así como utilizarlo como estructura de control en nuestros programas como se verá más adelante. Cada bit del registro tiene un nombre y significado especial, la lista dada a continuación describe el valor de cada bit, tanto apagado como prendido y su relación con las operaciones del procesador:

- OVERFLOW
  - NV = no hay desbordamiento;
  - OV = sí lo hay
- DIRECTION
  - UP = hacia adelante;
  - DN = hacia atras;
- INTERRUPTS
  - DI = desactivadas;
  - EI = activadas
- SIGN
  - PL = positivo;
  - NG = negativo
- ZERO
  - NZ = no es cero;
  - ZR = sí lo es
- AUXILIARY CARRY
  - NA = no hay acarreo auxiliar;
  - AC = hay acarreo auxiliar
- PARITY
  - PO = paridad non;
  - PE = paridad par;
- CARRY
  - NC = no hay acarreo;
  - CY = Sí lo hay

LA ESTRUCTURA DEL ENSAMBLADOR

En el lenguaje ensamblador las lineas de código constan de dos partes, la primera es el nombre de la instrucción que se va a ejecutar y la segunda son los parámetros del comando u operandos. Por ejemplo:

add ah bh

Aquí “add” es el comando para ejecutar (en este caso una adición) y tanto “ah” como “bh” son los parámetros. El nombre de las instrucciones en este lenguaje esta formado por dos, tres o cuatro letras. a estas instrucciones tambien se les llama nombres mnemónicos o códigos de operación, ya que representan alguna función que habrá de realizar el procesador.

Existen algunos comandos que no requieren parametros para su operación, as’ como otros que requieren solo un parámetro.

Algunas veces se utilizarán las instrucciones como sigue:

add al,[170]

Los corchetes en el segundo parámetro nos indican que vamos a trabajar con el contenido de la casilla de memoria número 170 y no con el valor 170, a esto se le conoce como direccionamiento directo

NUESTRO PRIMER PROGRAMA

Vamos a crear un programa que sirva para ilustrar lo que hemos estado viendo, lo que haremos será una suma de dos valores que introduciremos directamente en el programa:

El primer paso es iniciar el Debug, este paso consiste unicamente en teclear **debug [Enter]** en el prompt del sistema operativo.

Para ensamblar un programa en el Debug se utiliza el comando "**a**" (assemble); cuando se utiliza este comando se le puede dar como parametro la dirección donde se desea que se inicie el ensamblado, si se omite el parametro el ensamblado se iniciará en la localidad especificada por CS:IP, usualmente 0100H, que es la localidad donde deben iniciar los programas con extensión .COM, y sera la localidad que utilizaremos debido a que debug solo puede crear este tipo específico de programas.

Aunque en este momento no es necesario darle un parametro al comando "a" es recomendable hacerlo para evitar problemas una vez que se haga uso de los registros CS:IP, por lo tanto tecleamos:

– **a0100** [Enter]

Al hacer ésto aparecerá en la pantalla algo como: 0C1B:0100 y el cursor se posiciona a la derecha de estos números, nótese que los primeros cuatro dígitos (en sistema hexagesimal) pueden ser diferentes, pero los últimos cuatro deben ser

0100, ya que es la dirección que indicamos como inicio. Ahora podemos introducir las instrucciones:

- **0C1B:0100 mov ax,0002** ;coloca el valor 0002 en el registro ax
- **0C1B:0103 mov bx,0004** ;coloca el valor 0004 en el registro bx
- **0C1B:0106 add ax,bx** ;le adiciona al contenido de ax el contenido de bx
- **0C1B:0108 int 20** ; provoca la terminación del programa.
- **0C1B:010A**

No es necesario escribir los comentarios que van despues del ";". Una vez digitado el último comando, **int 20**, se le da [Enter] sin escribir nada mas, para volver al prompt del debuger.

La última linea escrita no es propiamente una instrucción de ensamblador, es una llamada a una interrupción del sistema operativo, estas interrupciones serán tratadas mas a fondo en un capítulo posterior, por el momento solo es necesario saber que nos ahorran un gran número de lineas y son muy útiles para accesar a funciones del sistema operativo.

Para ejecutar el programa que escribimos se utliza el comando "**g**", al utilizarlo veremos que aparece un mensaje que dice: "Program terminated normally". Naturalmente con un mensaje como éste no podemos estar seguros que el programa haya hecho la suma, pero existe una forma sencilla de verificarlo, utilizando el comando "r" del Debug podemos ver los contenidos de todos los registros del procesador, simplemente teclee:

– **r** [Enter]

Aparecera en pantalla cada registro con su respectivo valor actual:

AX=0006BX=0004CX=0000DX=0000SP=FFEEBP=0000SI=0000DI=0000

DS=0C1BES=0C1BSS=0C1BCS=0C1BIP=010A NV UP EI PL NZ NA PO NC

0C1B:010A 0F DB oF

Existe la posibilidad de que los registros contengan valores diferentes, pero AX y BX deben ser los mismos, ya que son los que acabamos de modificar.

Otra forma de ver los valores, mientras se ejecuta el programa es utilizando como parámetro para "g" la dirección donde queremos que termine la ejecución y muestre los valores de los registros, en este caso sería: **g108**, esta instrucción ejecuta el programa, se detiene en la dirección 108 y muestra los contenidos de los registros.

También se puede llevar un seguimiento de lo que pasa en los registros utilizando el comando "**t**" (trace), la función de este comando es ejecutar linea por linea lo que se ensambló mostrando cada vez los contenidos de los regitros.

Para salir del Debug se utiliza el comando "q" (quit).

GUARDAR Y CARGAR LOS PROGRAMAS

No sería práctico tener que digitar todo un programa cada vez que se necesite, para evitar eso es posible guardar un programa en el disco, con la enorme ventaja de que ya ensamblado no será necesario correr de nuevo debug para ejecutarlo.

Los pasos a seguir para guardar un programa ya almacenado en la memoria son:

Obtener la longitud del programa restando la dirección final de la dirección inicial, naturalmente en sistema hexadecimal.

Darle un nombre al programa y extensión

Poner la longitud del programa en el registro CX

Ordenar a Debug que escriba el programa en el disco.

Utilizando como ejemplo el programa del capítulo anterior tendremos una idea mas clara de como llevar estos pasos:

Al terminar de ensamblar el programa se vería así:

1. **0C1B:0100 mov ax,0002**
2. **0C1B:0103 mov bx,0004**
3. **0C1B:0106 add ax,bx**
4. **0C1B:0108 int 20**
5. **0C1B:010A**
6. **h 10a 100**
7. **020a 000a**
8. **n prueba.com**
9. **rcx**
10. **CX 0000**
11. **:000a**
12. **w**
13. **Writing 000A bytes**

Para obtener la longitud de un programa se utiliza el comando "h", el cual nos muestra la suma y resta de dos números en hexadecimal. Para obtener la longitud del nuestro le proporcionamos como parámetros el valor de la dirección final de nuestro programa (10A) y el valor de la dirección inicial (100). El primer resultado que nos muestra el comando es la suma de los parámetros y el segundo es la resta.

El comando "n" nos permite poner un nombre al programa.

El comando “rcx” nos permite cambiar el contenido del registro CX al valor que obtuvimos del tamaño del archivo con “h”, en este caso 000a, ya que nos interesa el resultado de la resta de la dirección inicial a la dirección final.

Por último el comando w escribe nuestro programa en el disco, indicandonos cuantos bytes escribió.

Para cargar un archivo ya guardado son necesarios dos pasos:

Proporcionar el nombre del archivo que se cargará.

Cargarlo utilizando el comando “l” (load).

Para obtener el resultado correcto de los siguientes pasos es necesario que previamente se haya creado el programa anterior.

Dentro del Debug escribimos lo siguiente:

1. **n prueba.com**
2. **l**
3. **u 100 109**
4. **0C3D:0100 B80200 MOV AX,0002**
5. **0C3D:0103 BB0400 MOV BX,0004**
6. **0C3D:0106 01D8 ADD AX,BX**
7. **0C3D:0108 CD20 INT 20**

El último comando, “u”, se utiliza para verificar que el programa se cargó en memoria, lo que hace es desensamblar el código y mostrarlo ya desensamblado. Los parámetros le indican a Debug desde donde y hasta donde desensamblar.

Debug siempre carga los programas en memoria en la dirección 100H, a menos que se le indique alguna otra.

CONDICIONES, CICLOS Y BIFURCACIONES

Estas estructuras, o formas de control le dan a la máquina un cierto grado de desición basado en la información que recibe.

La forma mas sencilla de comprender este tema es por medio de ejemplos.

Vamos a crear tres programas que hagan lo mismo: desplegar un número determinado de veces una cadena de caracteres en la pantalla.

1. **a100**
2. **0C1B:0100 jmp 125 ; brinca a la dirección 125H**
3. **0C1B:0102 [Enter]**
4. **e 102 ‘Cadena a visualizar 15 veces’ 0d 0a ‘$’**
5. **a125**
6. **0C1B:0125 MOV CX,000F ; veces que se desplegara la cadena**
7. **0C1B:0128 MOV DX,0102 ; copia cadena al registro DX**
8. **0C1B:012B MOV AH,09 ; copia valor 09 al registro AH**
9. **0C1B:012D INT 21 ; despliega cadena**
10. **0C1B:012F LOOP 012D ; si CX>0 brinca a 012D**
11. **0C1B:0131 INT 20 ; termina el programa.**

Por medio del comando “e” es posible introducir una cadena de caracteres en una determinada localidad de memoria, dada como parámetro, la cadena se introduce entre comillas, le sigue un

espacio, luego el valor hexadecimal del retorno de carro, un espacio, el valor de línea nueva y por último el símbolo '$' que el ensamblador interpreta como final de la cadena. La interrupción 21 utiliza el valor almacenado en el registro AH para ejecutar una determinada función, en este caso mostrar la cadena en pantalla, la cadena que muestra es la que está almacenada en el registro DX. La instrucción LOOP decrementa automáticamente el registro CX en uno y si no ha llegado el valor de este registro a cero brinca a la casilla indicada como parámetro, lo cual crea un ciclo que se repite el número de veces especificado por el valor de CX. La interrupción 20 termina la ejecución del programa.

Otra forma de realizar la misma función pero sin utilizar el comando LOOP es la siguiente:

1. **a100**
2. **0C1B:0100 jmp 125 ; brinca a la dirección 125H**
3. **0C1B:0102 [Enter]**
4. **e 102 'Cadena a visualizar 15 veces' 0d 0a '$'**
5. **a125**
6. **0C1B:0125 MOV BX,000F ; veces que se desplegara la cadena**
7. **0C1B:0128 MOV DX,0102 ; copia cadena al registro DX**
8. **0C1B:012B MOV AH,09 ; copia valor 09 al registro AH**
9. **0C1B:012D INT 21 ; despliega cadena**
10. **0C1B:012F DEC BX ; decrementa en 1 a BX**
11. **0C1B:0130 JNZ 012D ; si BX es diferente a 0 brinca a 012D**
12. **0C1B:0132 INT 20 ; termina el programa.**

En este caso se utiliza el registro BX como contador para el programa, y por medio de la instrucción "DEC" se disminuye su valor en 1. La instrucción "JNZ" verifica si el valor de B es diferente a 0, esto con base en la bandera NZ, en caso afirmativo brinca hacia la dirección 012D. En caso contrario continúa la ejecución normal del programa y por lo tanto se termina.

Una última variante del programa es utilizando de nuevo a CX como contador, pero en lugar de utilizar LOOP utilizaremos decrementos a CX y comparación de CX a 0.

1. **a100**
2. **0C1B:0100 jmp 125 ; brinca a la dirección 125H**
3. **0C1B:0102 [Enter]**
4. **e 102 'Cadena a visualizar 15 veces' 0d 0a '$'**
5. **a125**
6. **0C1B:0125 MOV DX,0102 ; copia cadena al registro DX**
7. **0C1B:0128 MOV CX,000F ; veces que se desplegara la cadena**
8. **0C1B:012B MOV AH,09 ; copia valor 09 al registro AH**
9. **0C1B:012D INT 21 ; despliega cadena**
10. **0C1B:012F DEC CX ; decrementa en 1 a CX**
11. **0C1B:0130 JCXZ 0134 ; si CX es igual a 0 brinca a 0134**
12. **0C1B:0132 JMP 012D ; brinca a la direcci&oauten 012D**
13. **0C1B:0134 INT 20 ; termina el programa**

En este ejemplo se usó la instrucción JCXZ para controlar la condición de salto, el significado de tal función es: brinca si CX=0

El tipo de control a utilizar dependerá de las necesidades de programación en determinado momento.

PD: cabe recalcar que los programas echos en estos ejemplos son .com y no .exe como en el inicio.

DIFERENCIAS ENTRE PROGRAMAS .COM .EXE.

bueno para dar una definición clara de las diferencias entre los programas .com y .exe en assembler es que los programas .exe llevan una pila(stack) definida en el codigo que significa la longitud del programa esta pila la define el programador.

EJEMPLO DE PROGRAMA .EXE

```
.model small ;tipo de memoria a utilizar en el programa puede ser tyni(pequeño),
small(mediano) o large(grande)
.stack 256 ;tamaño de la pila es decir tamaño de las instrucción del programa (para los q no
entiendad tamaño del código :P)
.data ;definición de variable
.code ;codigo
```

lo bueno del programa .exe es que podemos definir todo el programa memoria tamaño etc. los programas .com se pueden crear directamente en el DEBUG o con los compiladores ya mencionados la estructura de los programas .com son.

EJEMPLO DE PROGRAMA .COM

```
STACK SEGMENT STACK  ;segmento de pila
DW 64 DUP (?)
STACK ENDS

DATA SEGMENT ;datos o variables igual que él .data
DATA ENDS
CODE SEGMENT ;segmeto de código mismo que .code
ASSUME CS:CODE, DS:DATA, SS:STACK ;ahora decada segmento como ser el
CS=segmento de codigo el compilador que assuma que el code segment se guarde en el CS
```

cuando ejecuten un programa .com en los compiladores mencionados MASM, TASM saldra un mensage de

WARNING NO STACK

como solo es un mensaje de alerta y no de error el programa correrá igual lo único que dice el compilador es que no se definió un tamaño de la pila.

¿CUÁNDO USAR .exe Y CUANDO .com?

bueno la respuesta es sencilla generalmente se usa .com cuando se quiera hacer un programa pequeño y .exe cuando sea un programa grande