SISTEMAS OPERATIVOS

Lenguaje ensamblador en arquitecturas 8086/8088 parte 1

Elaborado por: Ukranio Coronilla

DOSBox

Para la presente práctica vamos a utilizar el lenguaje ensamblador que originalmente se utilizó para programar las primeras computadoras personales, y de las cuales se derivó el sistema operativo más famoso de los 80s y 90s, el MS-DOS. Con este propósito vamos a instalar en LINUX el emulador DOSBox:

https://en.wikipedia.org/wiki/DOSBox

Ejecutamos en la línea de comandos:

```
sudo apt-get install dosbox
```

Posteriormente creamos una carpeta llamada DOSBOX en `/home/usuario` la cual contendrá todos nuestros programas en ensamblador y las utilerias necesarias. Dichas utilerias(EDIT, TASM, TD y TLINK) debe descargarlas de la liga correspondiente en la plataforma MOODLE y guardarlas en su carpeta DOSBOX ya descomprimidas.
A continuación deberá ejecutar el programa DOSBox lo cual le presentará una pantalla como en la figura 1.

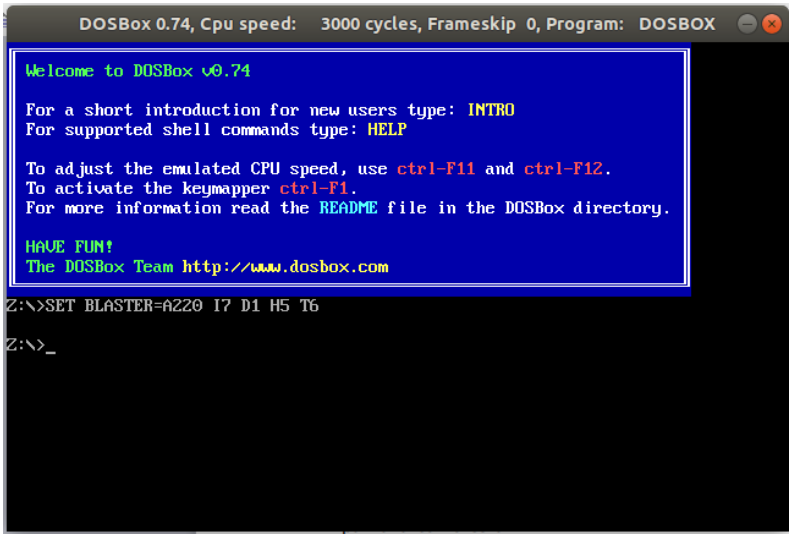

Figura 1

Observe que con la combinación de teclas Ctrl-F11 y Ctrl-F12 puede aumentar o disminuir la velocidad en ciclos del CPU virtual. Esto es de mucha utilidad porque muchos programas viejos (videojuegos vintage) se ejecutarían demasiado rápido si no se disminuye la velocidad del CPU.
Otras combinaciones de teclas como Ctrl-F10 para activar y desactivar el apuntador del mouse se muestran en la siguiente liga:
https://www.dosbox.com/wiki/Special_Keys

Lo que sigue es montar en DOSBox nuestra carpeta DOSBOX, para ello tecleamos en DOSBox:

```
mount c /home/usuario/DOSBOX
```

Y ahora si podemos entrar a la carpeta `/home/usuario/DOSBOX` al teclear:
`C:`

Podremos ver dentro de la carpeta todos los archivos que guardamos con anterioridad. Algunos comandos del sistema operativo MS-DOS se muestran en:
https://es.wikipedia.org/wiki/MS-DOS

Para salir de DOSBox solo tecleamos EXIT.

DESARROLLO EXPERIMENTAL

Cada instrucción de un lenguaje de alto nivel como C, en general está compuesta por varias instrucciones en lenguaje máquina. En contraparte un lenguaje de bajo nivel como es el caso del lenguaje ensamblador, provoca que cada instrucción se codifique como una instrucción en lenguaje máquina.
Aunque el uso de los lenguajes de alto nivel permite realizar una gran cantidad de operaciones con pocas instrucciones, el lenguaje de bajo nivel permite tener un control más directo del hardware, generar códigos ejecutables más pequeños y como consecuencia de ejecución más rápida. Es por estas últimas razones que el lenguaje ensamblador se sigue utilizando dentro de módulos críticos de los sistemas operativos modernos, y que es necesaria su revisión.

Veremos en la presente práctica como se crea un programa ejecutable a partir de código en lenguaje ensamblador, así como algunas características del lenguaje ensamblador.

Dentro del editor sublime copie y pegue el siguiente código (sin los números al inicio que solo sirven de referencia) al cual llamaremos PROG1_1.ASM (todos los programas en ensamblador tienen la extensión .ASM).

```
1 PILASG SEGMENT PARA STACK 'Pila'
2 DW 32 DUP(16) ;Datos repetidos
3 PILASG ENDS
4 DATOSG SEGMENT PARA PUBLIC 'Datos'
5 FLDA DW 250
6 FLDB DW 1FH
7 FLDC DW ?
8 DATOSG ENDS
9 CODIGO SEGMENT PARA PUBLIC 'Codigo'
10 MAIN PROC FAR
```

```
11 ASSUME SS:PILASG, DS:DATOSG, CS:CODIGO
12 MOV AX, DATOSG
13 MOV DS, AX ;Se inicializa DS
14 MOV AX, FLDA
15 ADD AX, FLDB
16 MOV FLDC, AX
17 MOV AX, 4C00H
18 INT 21H
19 MAIN ENDP
20 CODIGO ENDS
21 END MAIN
```

Para que el archivo sea reconocido dentro de DOSBox debe primero editar el archivo, cerrarlo y después arrancar DOSBox. Las modificaciones menores del código puede hacerlas con la utilería EDIT.EXE incluida.

Este programa se debe ensamblar para crear un programa objeto cuya extensión será .OBJ ; posteriormente se debe enlazar para crear el programa ejecutable (.EXE). El ensamblado y el enlazado se llevan a cabo con ayuda de los programas TASM y TLINK fabricados por la empresa de software Borland y que ya deberían estar en la carpeta DOSBOX.
Para crear el programa objeto a partir del código fuente, en la línea de comandos teclee:

```
TASM PROG1_1.ASM
```

Observe que si no ha habido un error se crea el programa objeto `PROG1_1.OBJ`, posteriormente teclee

```
TLINK PROG1_1
```

Lo cual crea el programa ejecutable `PROG1_1.EXE` . Ahora si ejecuta este programa en la línea de comandos, observará que no se imprime nada en pantalla, sin embargo sí se estarán ejecutando las instrucciones del código.

Las personas que idearon como ejecutar software almacenado en RAM con ayuda de un CPU, se les ocurrió dividir los programas en tres importantes áreas de memoria fijas llamados segmentos:

- El segmento de pila (Stack Segment SS) que comprende las líneas de código 1, 2 y 3. Este segmento contiene los datos y direcciones que se requieren guardar temporalmente, para su uso en subrutinas (funciones).
- El segmento de datos (Data Segment DS) que comprende las líneas de código 4 a 8, el cual contiene las variables y constantes que requiere el programa.
- El segmento de código (Code Segment CS) que contiene todo el código del programa.

Al momento de ejecutar el programa, cada segmento se coloca en su correspondiente segmento de la memoria RAM. Observe que cada segmento en el código se inicia con la siguiente sintaxis:

Nombre_segmento SEGMENT alineación combinar 'clase'

Y finaliza con:

Nombre_segmento ENDS

La palabra `PARA`, alinea el segmento en memoria para que inicie en un límite de párrafo (localidades divisibles entre 16 o 0x10). La palabra combinar indica si se combina el segmento con otros segmentos cuando se enlaza. Para la pila se usa `STACK` siempre y `PUBLIC` cuando se requiera combinar programas objetos independientes.
Observe que la clase es el nombre relacionado con cada segmento que se recomienda para los compiladores de ensamblador y que pusimos en español.

De manera análoga a los lenguajes de alto nivel, se pueden incluir funciones o procedimientos en ensamblador. Estos procedimientos se deben ubicar dentro del segmento de código, se definen mediante la palabra reservada `PROC` y tiene la siguiente sintaxis:

Nombre_procedimiento PROC FAR
Código
Nombre_procedimiento ENDP

Así como en lenguaje C, también en ensamblador se tiene una función o procedimiento principal (main), el cual se ejecuta siempre al inicio. La directiva `FAR` se escribe siempre en el procedimiento principal (véase la línea 10). Si se tienen otros procedimientos, `FAR` indica que el procedimiento se ubica en cualquier localidad de memoria. Si se escribe en su lugar `NEAR`, indica que el procedimiento se encuentra en el mismo segmento que el programa.

En la línea 21 se tiene la directiva `END`, la cual finaliza todo el programa y como operando se le pasa el nombre del primer o único `PROC` del código. La línea 11 le indica al ensamblador el nombre que el usuario le adjudica a cada segmento.
Observe que tanto en el segmento de datos como en el de pila, se definen e inicializan variables. El formato para definir variables o constantes es:

[Nombre] Dx expresión

Donde Nombre es el nombre de la variable y puede ser opcional como en el caso de la línea 2. Dx indica el número de bytes que va a ocupar ese dato y se puede definir como:

DB(1 byte), DW(2 bytes), DD(4 bytes), DF(6 bytes), DQ(8 bytes) y DT(10 bytes).

En *expresión* se pone ? para indicar que la variable no está inicializada, o un numero para inicializarla.
Se puede también tener un arreglo, separando los valores mediante comas como sigue:

```
ARREGLO DB 11, 12, 13, 14 ...
```

Estas constantes se encuentran en bytes contiguos y podemos hacer referencia al elemento 3 del arreglo (es decir a 14) mediante ARREGLO+3.
Se pueden duplicar datos con valores idénticos mediante DUP, por ejemplo

```
ALMACEN DB 5 DUP(14)
```

Define la variable almacén, con 5 datos contiguos en memoria repetidos e inicializados a 14.

Las constantes simbólicas se establecen con la directiva `EQU` (similar a las macros en C), por ejemplo

```
TIEMPO EQU 10
DIA EQU HOY
```

Substituye 10 donde aparezca la palabra `TIEMPO` y `HOY` donde aparezca la palabra `DIA` dentro de las líneas de código.
Una cadena de caracteres se define como tipo de dato byte, mediante comillas o apostrofe:

```
MENSAJE DB "Instituto Politecnico Nacional"
```

Todos los comentarios ocupan solo una línea y comienzan con punto y coma, como se muestra en la línea 2.

Vamos ahora a analizar nuestro programa con el depurador que se incluye en la versión completa de desarrollo de Borland C++ denominado Turbo Debugger.

Ejecute TD.EXE y abra el archivo ejecutable PROG1_1.EXE, observe que se tienen 5 ventanas con los contenidos mostrados en la figura 2.

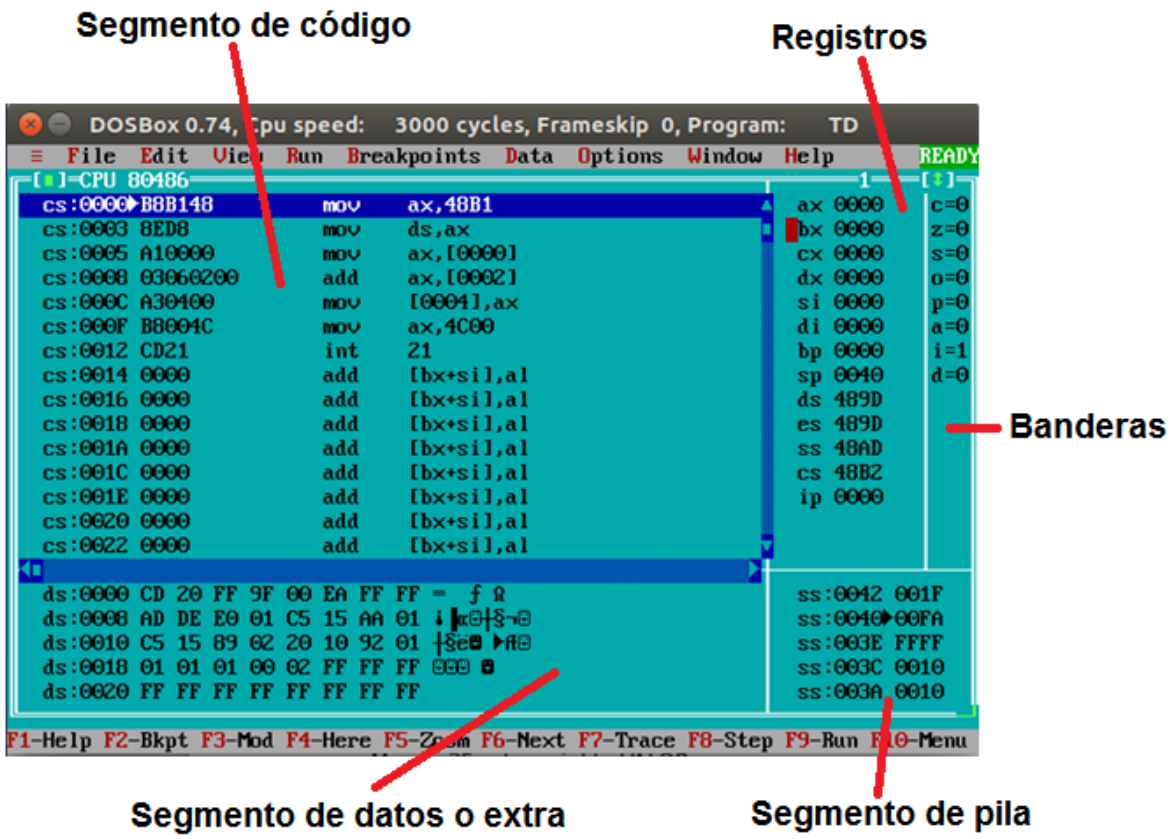

Figura 2

Observe que existe una ventana para cada segmento y esto permite visualizar simultáneamente las áreas de memoria donde se encuentran almacenados el código, los datos y la pila de nuestro programa PROG1_1.EXE.
En las arquitecturas basadas en procesadores 8086 se utiliza un direccionamiento del tipo *segmento:desplazamiento*, y para mayor información es necesario revisar:

https://es.wikipedia.org/wiki/Segmentaci%C3%B3n_de_memoria_del_x86

La ventana del segmento de código indica con el símbolo >, la línea de ensamblador que se ejecutará primero y es similar a la siguiente:

```
CS:0000 > B8B148                    mov ax, 48B1
```

Obsérvese que esta instrucción corresponde a la línea 12 de código y que si queremos saber el valor numérico de CS, lo podemos encontrar dentro de la ventana Registros. En el caso de la figura 2, `CS:0000 = 48B2:0000` indica la dirección de memoria donde inicia la primera instrucción del código. En esta misma instrucción B8 es la representación hexadecimal de la instrucción MOV AX, los últimos 4 dígitos hexadecimales contienen la dirección del segmento de datos en su computadora (vea la 4 y línea 12 de código). Es decir el segmento de memoria que contiene las líneas 4, 5, 6, 7 y 8 del código. Como podemos observar, el segmento de datos se almacena en memoria como 0xB148, pero el valor real del segmento es 0x48B1.
Véase https://es.wikipedia.org/wiki/Endianness

Al cargar el programa del disco a la memoria, automáticamente se establecen las direcciones de los registros CS y SS, pero no se establece DS, por lo que ese registro contiene basura y se debe inicializar. Esta inicialización se lleva acabo con las líneas 12 y 13 como sigue:

```
MOV AX, DATOSG
MOV DS, AX ;Se inicializa DS
```

Véase https://es.wikipedia.org/wiki/Registro_(hardware)

La instrucción `MOV` transfiere los datos referenciados por la dirección del segundo operando a la dirección del primer operando. Es importante que ambos operandos sean del mismo tamaño. Por ejemplo

```
MOV AX, 4C00h
```

Mueve el valor de 16 bits 4C00h al registro de 16 bits (2 bytes) AX del CPU.

Los registros AX, BX, CX, DX son de propósito general con tamaño DW, y se componen de una parte alta y de una parte baja, designadas como AH y AL respectivamente para el caso del registro AX como se muestra en la figura 3.

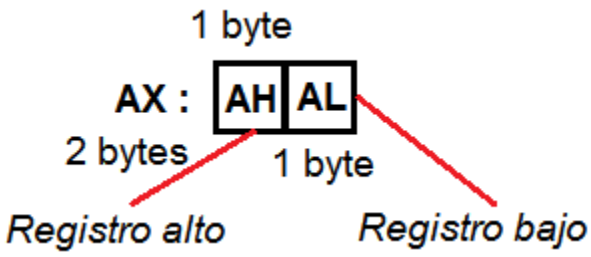

Figura 3

El uso típico de los registros es como sigue:

AX -> Para operaciones aritméticas o datos.
BX -> Para cálculos.
CX -> Se usa como contador o para cálculos.
DX -> Para contener datos que no caben en AX.

También se pueden cambiar los valores de las variables mediante MOV:

```
MOV VALOR, 25
```

Mueve 25 a la variable VALOR

```
MOV VALOR, DS:[38B08]
```

En este caso los corchetes hacen referencia a una dirección de memoria y se mueve el valor contenido en dicha dirección de memoria a la variable VALOR.

Ejercicio 1
En la ventana de registros apunte en una hoja el valor almacenado en DS, el cual contiene basura, ahora observe en la ventana segmento de datos y apunte los primeros 6 bytes almacenados en el segmento DS. De acuerdo a los datos almacenados en las variables dentro del segmento de datos(ver las líneas 5, 6 y 7) ¿Qué debería estar almacenado en esos primeros 6 bytes que acaba de anotar?

Veamos si tiene razón, para lo cual vamos a ejecutar las dos primeras líneas de código al oprimir dos veces la tecla F7 (con CTRL-F2 se reinicia la ejecución), observe detalladamente todos los cambios que se presentan en todas las ventanas, cada que presione F7. Desafortunadamente la ventana segmento de datos o extra va a exhibir el segmento extra. Para ver el segmento de datos vaya a la ventana segmento de datos o extra (para desplazarse entre ventanas utilice la tecla TAB). Posteriormente mantenga apretada la tecla CTRL y después la tecla G, con lo cual puede dar una dirección a la cual se quiere desplazar.
Basta entonces con poner la dirección DS:0000 para ir a dicha dirección...¿Tenía razón?