**Practicando  Assembly**

**1.- Introducción**  
  
Es importante antes de adentrarnos en el mundo de los **Depuradores**, **Desensambladores** y demás *herramientas* que conozcamos y tengamos una buena noción de lo que es el *Assembly*. Empezaremos suave, haciendo breves ejemplos y poco a poco.  
  
**2.- Instrucciones Básicas: Parte I**  
  
Comenzaremos por las instrucciones más básicas, con breves explicaciones y ejemplos:  
  
**MOV, CMP, TEST, ADD, SUB, INC, DEC.**  
  
**MOV**: Se utiliza para mover un dato de un registro a otro.  
  
Ej: MOV **A**, **B**. *El valor de A pasaría a ser igual que el valor de B.*  
  
**CMP**: Compara el valor de dos registros.  
  
Ej: CMP **A**, **B**. *Se efectúa una comparación entre A y B.*  
  
**TEST**: Compara el valor de dos registros.  
  
Ej: TEST **A**, **B**. *Se efectúa una comparación entre A y B.*  
  
Nota: En capitulos posteriores escribiremos la diferencia entre **CMP** y **TEST**.  
  
**ADD**: Suma el valor de dos registros almacenandolo en el primero.  
  
Ej: ADD **A**, **B**. *Se suma a A el valor de B y el resultado se guardaría en  (A).*  
  
**SUB**: Resta el valor de dos registros almacenandolo en el primero.  
  
Ej: SUB **A**, **B**. *Se resta a A el valor de B y el resultado se guardaría en  (A).*  
  
**INC**: Incrementa el valor en 1 de un registro.  
  
Ej: INC **A**. *Se incrementaría el valor de A en 1.*  
  
**DEC**: Decrementa el valor en 1 de un registro.  
  
Ej: DEC **A**. *Se decrementaría el valor de A en 1.*  
  
**2.- Los Registros: Parte I**  
  
Antes de adentrarnos en los **Registros** quiero hacer un breve comentario sobre las **API**.  
  
Bien, en cualquier programa compilado bajo windows se utilizan unas denominadas API, éstas son básicamente las que forman todo lo que percibes, así como ventanas, botones, movimientos del ratón, etc... y también todas las acciones que haces.  
  
La manera de llamarlas es básicamente una estructura. No la explicaré de momento porque es pronto para ello pero os haré un breve ejemplo:

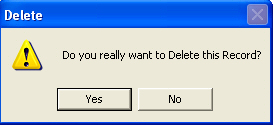
Citar

PUSH EAX  
PUSH EDX  
CALL lstrcatA

Imaginemos esas instrucciones, en éste caso digamos que mete EAX y EDX en la pila (que lo que explicaré más adelante) para así tratar los valores antes de llamar a la **API** *lstrcat* que se encarga de concatenar (juntar) 2 cadenas.  
  
En éste caso si EAX apunta a "hola" y EDX a " mundo" una vez ejecutada la **API** en EAX quedaría la *cadena/string* "hola mundo". Así que como explicaré ahora EAX ha sido utiliza de "**Acumulador**" porque ha acumulado el resultado de la **API**.  
  
Hablaremos de los *Registros Comunes* de los cuales destaco:  
  
**EAX, ECX, EDX, EBX.**  
  
1.- **EAX**: *Acumulador*  
  
- Como ya expliqué, una vez ejecutada una API, se suelen devolver principalmente el o los resultados a **EAX**.  
  
2.-**EBX**: *Base*  
  
- Éste se suele utilizar como Base en los algoritmos. Es decir, si **EBX** vale 10, esa será la BASE de la cantidad de veces que se hace un proceso específico.  
  
3.-**ECX**: *Contador*  
  
- Éste sería en el ejemplo anterior el que hace de contador, con una instrucción tipo ***INC ECX*** cada vez se iría sumando de 1 en 1 hasta llegar al número del registro BASE (EBX) 10, así que utilizando estos dos últimos ejemplos se haría durante 10 veces una funcion (por ejemplo, multiplicar por 3 diez veces). Ej:

Citar

Notas: EAX = 1 ; EBX = 10  
Inicio:  
MUL EAX, 3 ; MUL: Multiplica 2 registros almacenando en el primero.  
INC ECX  
CMP ECX, EBX  
JNE Inicio ; JNE: es un salto condicional (JNE: JUMP IF NOT EQUAL / Salta si no es igual) que va hacia la dirección que le indiques siempre y cuando el resultado de la comparación anterior sea **desigüal**.

4.-**EDX**: *Datos*  
  
- Éste es utilizado también antes de invocar a una **API** para darle los datos.  
  
  
**3.- LA PILA: Parte I**   
  
¿Qué es la **PILA**?  
  
- Técnicamente es una **estructura de datos**, del tipo **LIFO** (del inglés Last In First Out, último en entrar, primero en salir). Texto extraído de la wikipedia..  
  
y vosotros os preguntaréis, ¿que esto de una estructura?, Bien. Imaginad que formamos una ventana, o por ejemplo, el típico mensaje de texto. En éste caso es la **API** *MessageBoxA* .  
  
  
  
En éste caso vemos 3 partes claramente identificables:  
  
- **Título** : "*Delete*".  
- **Cuerpo** : "*Do you really want to Delete this Record?*".  
- **Botones** : " *YesNo*".

Y claro, aquí es donde entra el papel de la **pila**, antes de invocar a la **API** *MessageBoxA*1 tenemos que indicarle todos estos datos para que nos la forme tal y como nosotros le hemos indicado. En Ensamblador, (a nivel de código máquina), ésto sería visualizado así:  
  
**1** MessageBoxA es la API encargada de lanzar ese mensaje tipo "caja de texto" que puse en la imagen.

Citar

PUSH **Título**  
PUSH **Cuerpo**  
PUSH **Botones**  
CALL *MessageBoxA*

Entonces ahí vemos que cada elemento es primero *insertado* en la **pila** mediante la instrucción **PUSH**. La pila tiene la cualidad de lo último que metes, es lo primero que sacas. Es decir, en nuestro caso ahora la pila tendría ésta forma.  
  
Las direcciones son imaginarías, normalmente la pila se visualiza con las correspondientes direcciones hacia donde apuntan (pueden apuntar a una cadena de texto que contenga el título  
  
00401005 **Botones**  
00401000 **Cuerpo**  
00401020 **Título**  
  
de manera que si ahora nos encontráramos con una instrucción tipo **POP EAX** extraeríamos **Botones** por ser la última que se introdujo hacia el **REGISTRO EAX** el cual hemos indicado mediante la instrucción.  
  
Ésta estructura es un ejemplo, de que existe una zona donde se guardan valores MUY importantes en el flujo de un programa, así como por ejemplo, si se ejecutara la instrucción **CALL** *MessageBoxA* y ésta se encontrara por ejemplo en la dirección 402500 (vamos a hacer un ejemplo de donde la dirección en la que se sitúan las instrucciones):  
  
Atendiendo al ejemplo de antes añado las mismas instrucciones pero con sus respectivas direcciones imaginarias.

Citar

**402500**| PUSH **Título**  
**402504**| PUSH **Cuerpo**  
**402508**| PUSH **Botones**  
**40250A**| CALL *MessageBoxA*  
**40250F**| MOV EAX, EBX

En éste caso una vez se ejecute la instrucción **CALL** *MessageBoxA*, se almacenará en la pila también la dirección de **RETORNO**, es decir, la dirección en la que regresará una vez haya sido llamado. Atendiendo al ejemplo de antes la pila en éste caso quedaría así.  
  
00401008 **RETURN to *40250F***  
00401005 **Botones**  
00401000 **Cuerpo**  
00401020 **Título**  
  
Así como vemos, también se a almacenado la dirección hacia donde tendrá que volver despues del *MessageBoxA.  
  
Por el momento vamos teniendo nociones de cada parte pero sin profundizar en nada, si no lo entendéis completamente no os preocupéis, es normal, a medida que lo vayamos viendo todo tomará forma y os saldrá mecánicamente.  
  
4.-* ***LOS REGISTROS: PARTE II [PRÁCTICAS]  
  
Ahora imaginemos varias situaciones, voy a escribir unas instrucciones y quiero que me digáis el valor de los registros finales. (si queréis responder a un privado me alegrará ver que seguis el curso, aunque seamos lentos :P).***

Cita de: Ejercicio 1

MOV EAX, 1530h  
ADD EAX, 1000h  
SUB EAX, 500h  
MOV EBX, 100h  
SUB EAX, EBX

Normalmente cuando crackeamos utilizamos registros de **32 Bit**, entre los cuales podemos encontrarnos *EAX, EBX, ECX, EDX, etc.* Pero éstos se componen de varias partes según tamaño:  
  
EAX (**32 bits**)  
  
se descompone en:  
  
AX (**16BIT**)  
  
que éste mismo se descompone en:  
  
AH y AL (**8 Bit cada uno**)  
  
Para que veáis un ejemplo sobre el valor:  
  
**12345678** => EAX  
1234**5678** => AX  
1234**56**78 => AH  
123456**78** => AL  
  
Siempre debemos estar al tanto de sobre QUE valores trabajamos, es decir, si nos encontramos una instrucción tipo:

Citar

MOV AL, 5

Sabemos que sólamente hay una posibilidad, pero tambien debemos saber a que parte de **EAX** corresponde, porque si fuese *AX* por ejemplo ya cambiaría la posición del movimiento.  
  
Mas adelante veremos como se trabaja con "punteros", y ahí deberemos tener bien claro éstas correspondencias:  
  
**DWORD -> EAX  
WORD   -> AX  
BYTE     -> AH y AL**

Citar

Ejercicio 2:  
  
MOV EAX, 00004932h  
ADD AX, 45h  
SUB AL, 0Ah

5.- **SALTOS CONDICIONALES Y BANDERAS: PARTE I [PRÁCTICAS]**  
  
¿Que son los saltos condicionales? ¿Que son las banderas?  
  
Los saltos condicionales son instrucciones que al ejecutarse hacen que el flujo se traslade a la dirección que apunte.  
  
Es decir, cuando nos encontramos con una instruccion del tipo "**salto condicional**" nos llevará (si se cumple la condicion) hacia donde apunte. Veamos un ejemplo:

Citar

CMP EAX, EBX  
JE 401000

la primera instruccion compara el contenido de **EBX** con el contenido de **EAX**, el resultado de la comparacion pondra a 1 la **BANDERA "Z"** si es corrercto, y a 0 si no lo es.  
y la siguiente instruccion saltará si la comparación ha sido correcta (JUMP IF EQUAL => JE) ha la direccion "**401000**".  
  
En éste ejemplo comprobamos 2 cosas, que hay unas banderas que se activan o bien con 0 o con 1 en función de las operaciones aritméticas que vayan generandose a lo largo del programa, y que hay instrucciones que actuan en función de las mismas, veamos que tipos de satos hay:  
  
JE  
JNE o JNZ  
JB  
JBE  
JP  
JPE  
JL  
JLE  
  
  
[CONTINUARÁ]

|  |  |
| --- | --- |
|  | |
| « *Última modificación: 23 Enero 2011, 20:16 por Shaddy* » |  |