Sistemas Operativos

Primer Actividad Práctica

Repaso de Arquitectura

Trabajo Práctico: Repaso de Interrupciones, direccionamiento de memoria y registros del procesador.

Modalidad: Obligatorio.

Objetivos:

- Abordar la materia Sistemas Operativos desde el concepto “El trabajo basado en interrupciones es una técnica de implementación de sistemas multitarea monoprocesamiento, en la cual las tareas se ejecutan en forma concurrente”.

- Reforzar el concepto de interrupciones desde una visión práctica.

- Identificar una tarea como una estructura de código y datos almacenados en memoria.

- Comprender la ejecución de un proceso, analizando los valores de los registros del procesador.

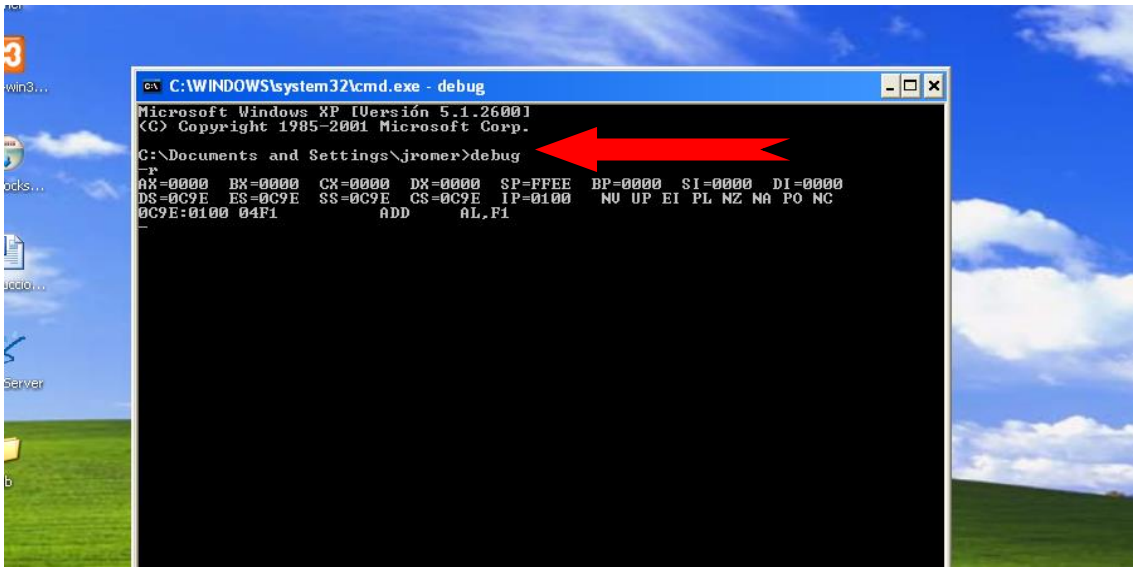
Duración del trabajo práctico: Se propone el mismo en la primer clase de la materia y se realizarán consultas del mismo la segunda clase de la materia.

Mail de consultas juancarlosjromer@gmail.com

Herramientas:

- Sistema Operativo Linux o Windows XP.
- En Windows XP utilizar desde la línea de comandos, el comando debug.
- En Linux Debian instalar “dosemu” que es un emulador de DOS, donde se puede ejecutar el comando debug.

Ejecución del debug en un entorno Windows XP

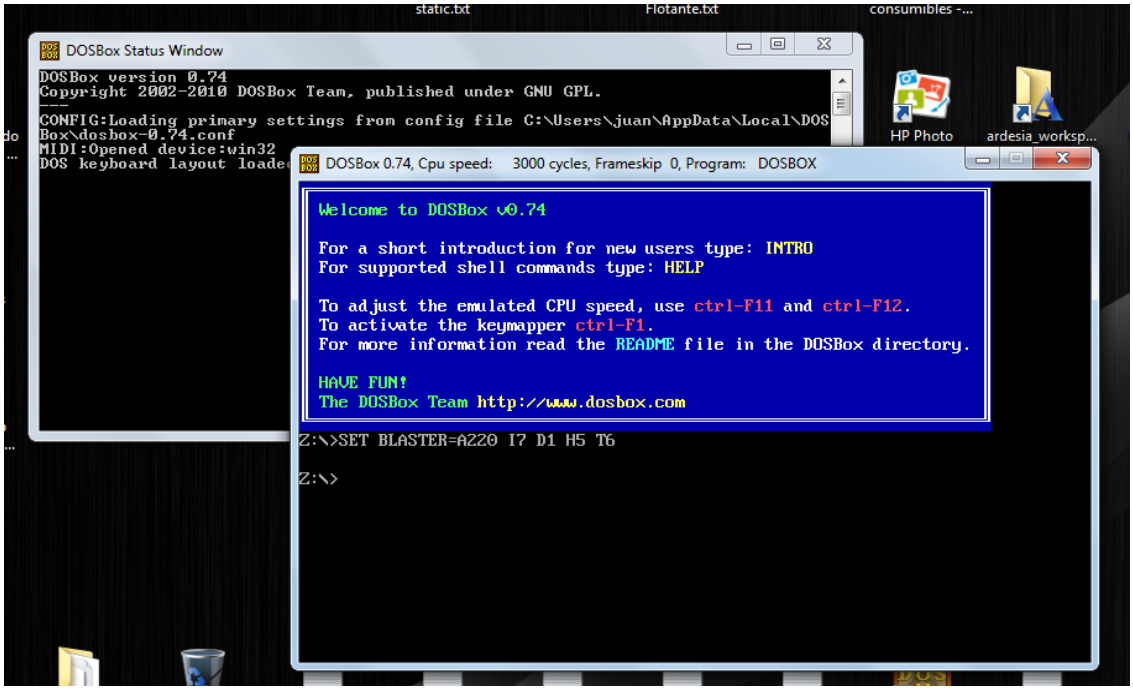


Ejecución del debug en Sistemas Operativos Windows superior a Windows XP

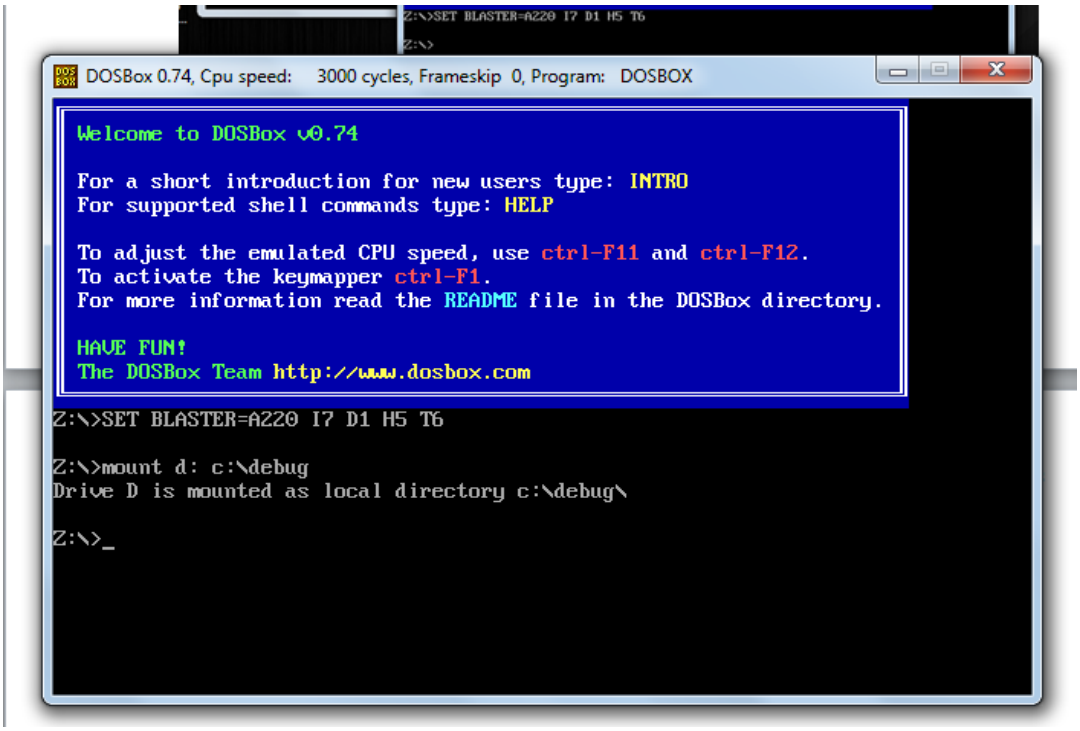
Primer paso: Instalar el emulador del debug DosBox.

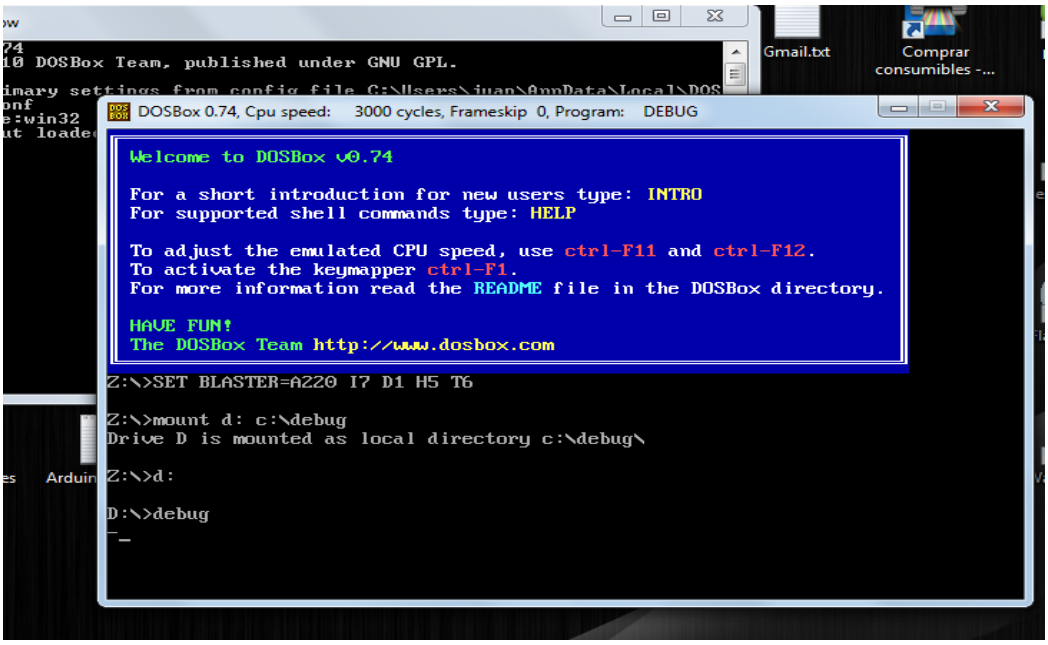
Segundo paso: Copiar el ejecutable debug.exe en un directorio determinado, por ejemplo C:\debug\debug.exe

Tercer paso: Ejecutar DosBox y aparecerá lo siguiente



Cuarto paso: Tipear en el prompt Z:\> mount d: c:\debug presionar ENTER luego tipear d: y presionar ENTER, luego tipear debug y presionar ENTER y ya estarás dentro del debug.



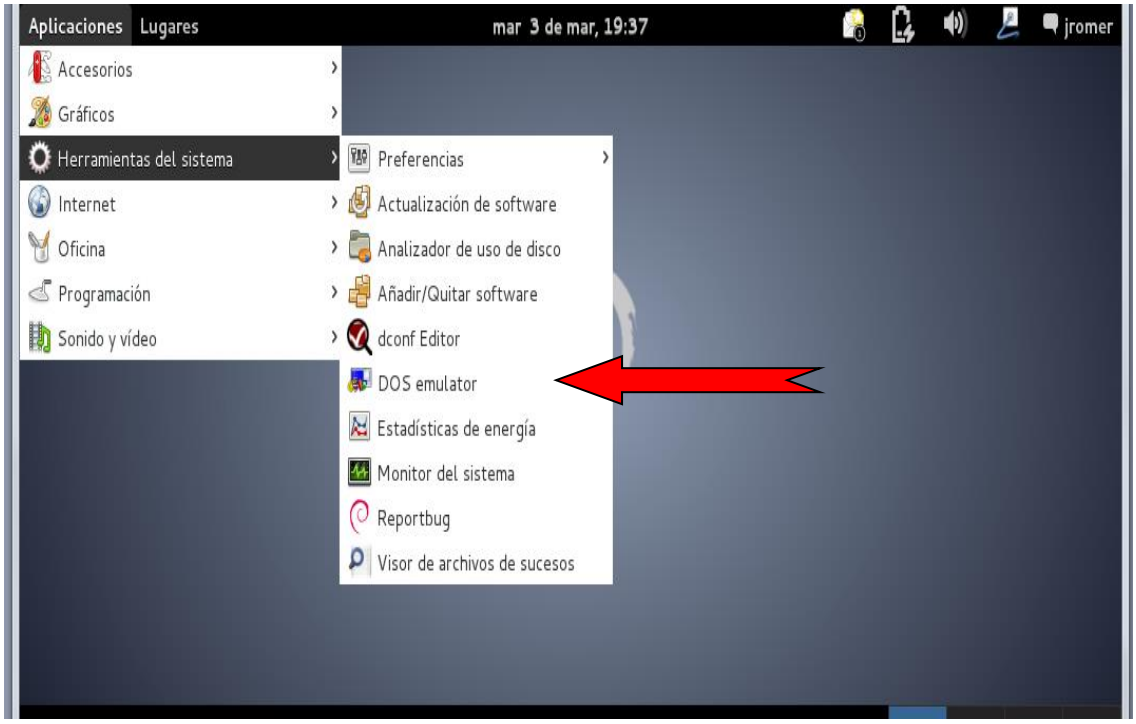


Ejecución del debug en un entorno Linux Debian

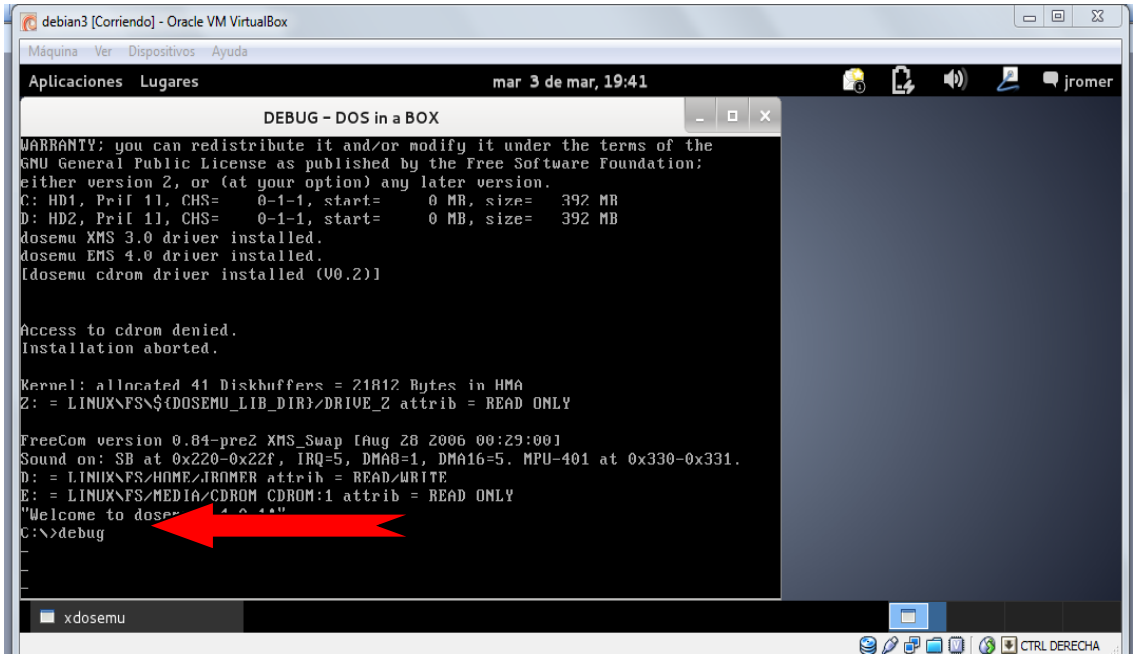
Primer paso: Instalar el emulador del DOS, como lo muestra la siguiente imagen.


sitivos Ayuda
gares
mar 3 de mar, 19:35
jromer@debian32: ~
Archivo Editar Ver Buscar Terminal Ayuda
jromer@debian32:~$ su
Contraseña:
root@debian32:/home/jromer# apt-get install dosemu
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes extras:
libsdl1.2debian
Se instalarán los siguientes paquetes NUEVOS:
dosemu libsdl1.2debian
0 actualizados, 2 se instalarán, 0 para eliminar y 68 no actualizados.
Necesito descargar 2.719 kB de archivos.
Se utilizarán 5.365 kB de espacio de disco adicional después de esta operación.
¿Desea continuar [S/n]? S
Des:1 http://ftp.debian.org/debian/ wheezy/main libsdl1.2debian i386 1.2.15-5 [2
31 kB]
Des:2 http://ftp.debian.org/debian/ wheezy/contrib dosemu i386 1.4.0+svn.2080-1
[2.489 kB]
Descargados 2.719 kB en 14seg. (183 kB/s)
Seleccionando el paquete libsdl1.2debian:i386 previamente no seleccio
Pulse para cambi
an32: ~

Segundo paso: Encontrar el acceso al emulador del DOS, como lo muestra la siguiente imagen.



Tercer paso: Ejecutar el comando debug en el emulador de DOS como lo muestra la siguiente imagen.



Cuarto paso: Leer la siguiente guía rápida de comandos del debug.

Debug

El Debug es una utilidad de MS-DOS que permite visualizar memoria, introducir programas en ella y rastrear su ejecución. Una característica del debug es que despliega todo el código del programa en formato hexadecimal.
Es importante saber que muchos usos de este tipo de utilidades de bajo nivel, requieren un funcionamiento fuera de un Sistema Operativo multiusuario, ya que éstos encapsulan y ocultan muchos aspectos del hardware.
El debug permite ensamblar líneas de código, desensamblar código, ejecutar programas, mostrar áreas de memoria, verificar los registros del CPU.

Comandos de Debug

El ingreso a Debug se realiza de forma sencilla a partir de la línea de comandos de un sistema operativo, tipeando debug. Luego aparece el prompt del debug que es un guión.
Para acceder a la ayuda escribimos el comando "?"

D: DUMP

D [intervalo]
Muestra el contenido de una zona de memoria en hexadecimal y en ASCII. Sin parámetros muestra los primeros 128 bytes a partir de la posición a la que se llegó con el último comando "d". Si se le da un rango, mostrará el contenido de ese rango de memoria.

E DIRECCION: EDIT

E dirección [lista]
Permite editar y modificar, byte por byte, una zona de memoria. Muestra en hexadecimal el byte de esa posición y permite escribir otro valor para cambiarlo. Pulsando la barra espaciadora pasa al byte siguiente, dejando como estaba el anterior si no se ha cambiado, o guardando los cambios si sí se ha modificado. Para terminar la edición se presiona ENTER.

R: REGISTERS

R [registro]
Sin parámetros, muestra el contenido de los registros de la CPU, así como la próxima instrucción a ejecutar."R [REGISTRO]" muestra el contenido del registro especificado y cambia el prompt de "-" a ":" invitando a que se cambie su valor. Presionando ENTER lo deja como estaba.

A: ASSEMBLE

A [dirección]
Sin parámetros ensambla las instrucciones que se introduzcan, guardándolas en la dirección siguiente a la que se llegó con el último comando "a". Cuando se utiliza este comando se le puede dar como parámetro la dirección donde se desea que se inicie el ensamblado, si se omite el parámetro el ensamblado se iniciará en la localización especificada por el contenido de los registros CS:IP, usualmente 0100H, que es la localización donde deben iniciar los programas con extensión .COM, y será la localización que utilizaremos debido a que debug solo puede crear este tipo específico de programas.

F: FILL

F [lista de intervalos]
Llena una zona de memoria con un valor determinado. Como al terminar un programa la zona de memoria en que residía no se borra (poniéndola a cero,

por ejemplo), a menudo es útil para distinguir entre lo que son datos del programa actual y lo que es basura del programa anterior.

Q: QUIT

Q

Salir de debug y volver al DOS.

T: TRACE

T [dirección] [valor]

El comando Trace permite ejecutar un programa instrucción por instrucción, como así también las instrucciones del código de las rutinas de interrupciones.

P: STEP

P [=dirección] [número]

Trace puede ser incómodo si no se quiere depurar el código de las rutinas de interrupción o si ya se sabe el código que hay en las subrutinas y tan sólo interesa seguir avanzando sin entrar en ellas. En estos casos se usa p.

G: GO

El comando “G” permite ejecutar un programa, siempre que el programa contenga la instrucción de interrupción que indica la finalización del programa.

Videos recomendados para el uso del debug extraídos del canal de youtube

www.youtube.com\user\juancarlosjromer\videos

ver vídeos SCI debug 01, SCI debug 02, SCI debug 03

Ejemplo de ayuda para comenzar a conocer el debug

Propósito del ejemplo

- Ingresar un programa a memoria usando el comando del debug A: Assembler.
- Ingresar datos a memoria usando el comando del debug E: Enter.
- Ejecutar el programa usando el comando del debug T: Trace

Problema del Ejemplo

El programa debe mostrar por pantalla los caracteres ASCII que se han almacenado en un área determinada de memoria.

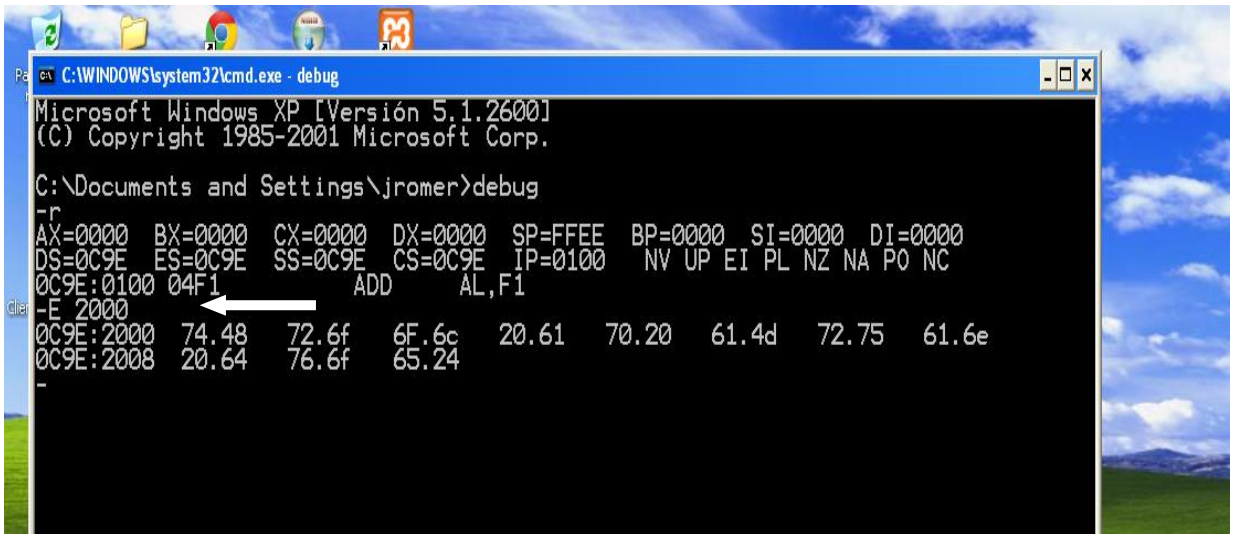
Detalle del problema

Los códigos ASCII en hexadecimal que corresponderán a la cadena de caracteres “Hola Mundo$” termina con el carácter “$” para que el programa pueda identificar fin de cadena.

ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo
0	0	NUL	16	10	DLE	32	20	(space)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

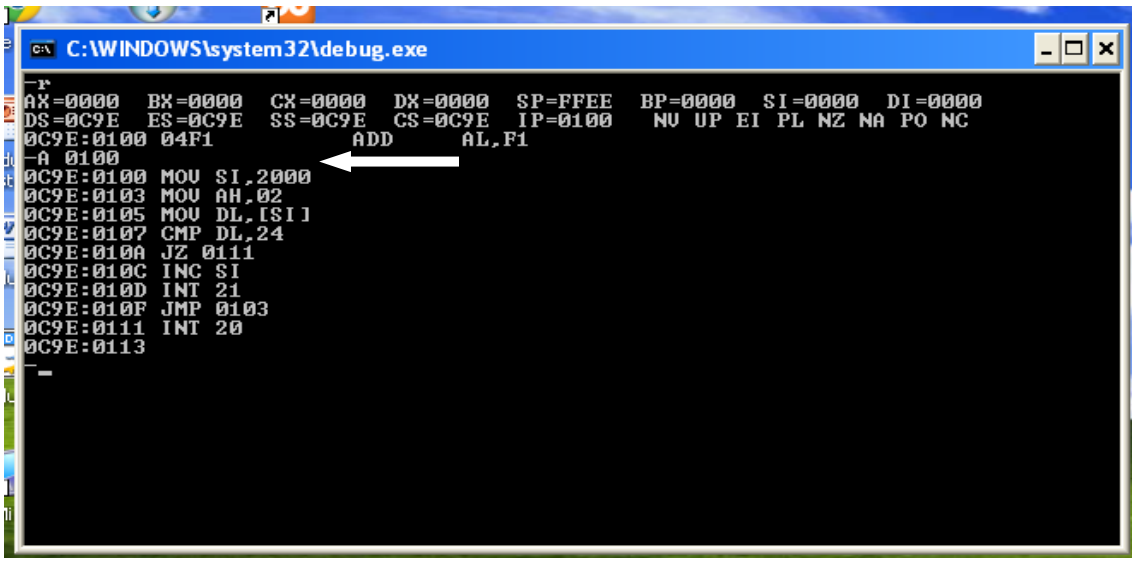
ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

La cadena de caracteres se ingresara a partir de la dirección de memoria 2000Hexa. Correspondiente al segmento de datos DS:0C9E.
Los valores Hexadecimales para “Hola Mundo$” son “48 6F 6C 61 20 4D 75 6E 6F 24 se ingresan a la memoria con el comando E del debug, como muestra la siguiente imagen.



El programa se almacenará a partir de la dirección de memoria 0100Hexa. Correspondiente al segmento de código CS:0C9E, como pueden ver el segmento de datos y el segmento de código es el mismo, junto con el segmento de pila SS:0C9E.

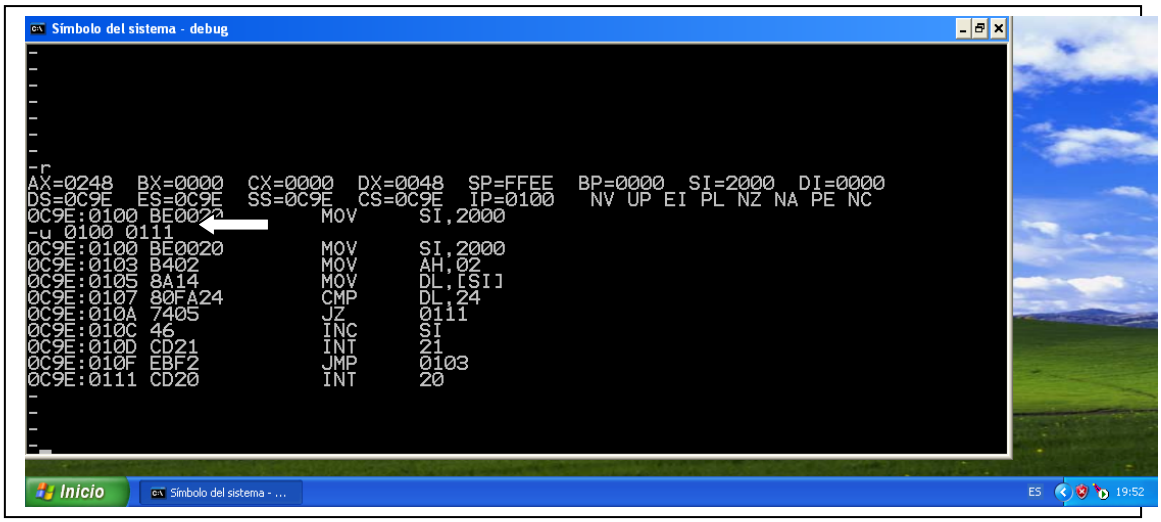
El código se ensambla en memoria con el comando del debug
-A 0100
Como lo muestra la siguiente: después de cada instrucción de assembler ingresada se presiona ENTER para poder ingresar la siguiente instrucción, para finalizar el ingreso del programa se presiona ENTER y volvemos al prompt del debug.



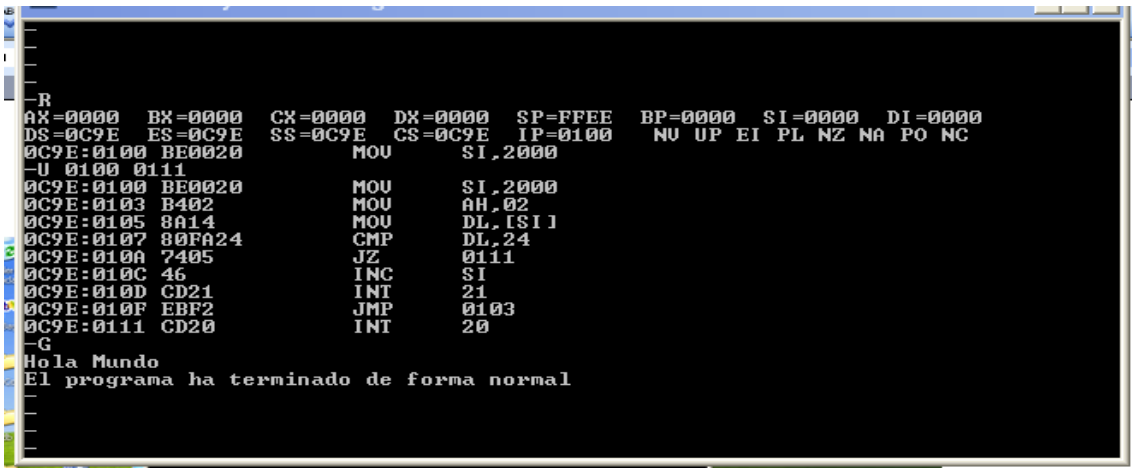
La siguiente tabla tiene un comentario de lo que hace cada instrucción.

Dirección de memoria de la instrucción assembler	Instrucción assembler	Comentario
0C9E:0100	MOV SI,2000	Transferimos el número de la dirección de memoria a partir de la cual ingresaremos los valores ASCII de “Hola Mundo$”
0C9E:0103	MOV AH,02	Colocamos en la parte alta del registro AX, el código 02 de la función de la interrupción 21 que permite mostrar un carácter ASCII en la pantalla
0C9E:0105	MOV DL,[SI]	Colocamos el contenido de la dirección de memoria registrada en el registro SI, en la parte baja del registro DX
0C9E:0107	CMP DL,24	Comparamos el contenido de la parte baja del registro DX, con el valor hexadecimal 24 código ASCII del carácter “$”
0C9E:010A	JZ 0111	Si la comparación anterior determina que son iguales entonces la ejecución del programa se bifurca a la dirección de memoria 0C9E:0111
0C9E:010C	INC SI	Incrementa el contenido del registro SI en 1, de esta manera el registro SI contiene la dirección de memoria siguiente
0C9E:010D	INT 21	Entra en la rutina que permite mostrar un carácter en pantalla. Ejecuta la interrupción 21
0C9E:010F	JMP 0103	La ejecución del programa se bifurca incondicionalmente a la dirección de memoria 0C9E:0103
0C9E:0111	INT 20	Ejecuta la interrupción 20 finalización del programa DOS

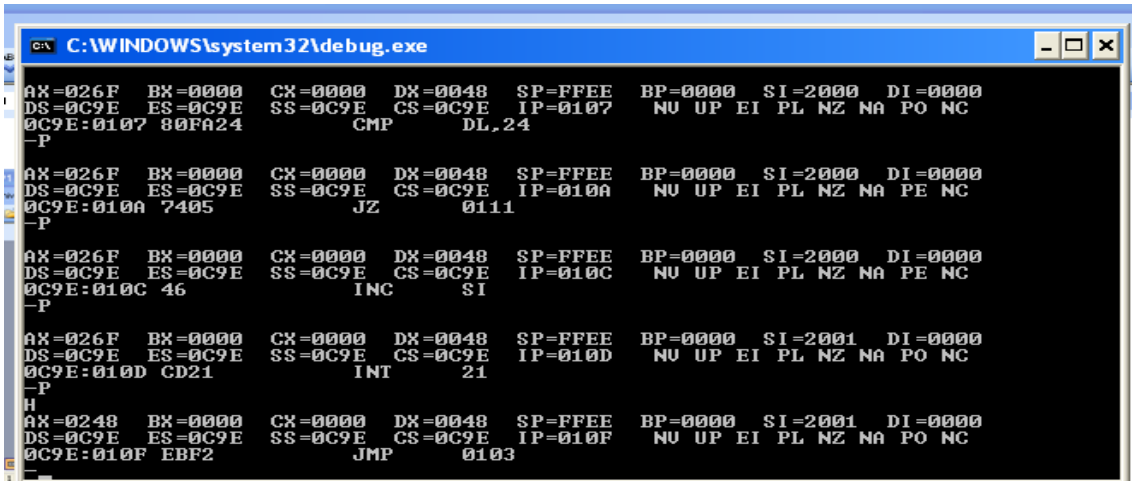
La imagen siguiente nos muestra el programa ya ingresado a memoria, lo visualizamos con el comando U (Unassembler) del debug.



Ahora que ya tenemos el programa almacenado en memoria a partir del desplazamiento 0100 del segmento de memoria del DOS 0C9E, podemos ejecutar el programa que debería mostrar “Hola Mundo” en la consola de DOS.
En nuestra primera ejecución usaremos el comando G del debug, como muestra la siguiente imagen, tienen que recordar que antes de ejecutar el programa el valor del registro IP debe ser 0100Hexa.
Como muestra la siguiente imagen.



Ahora ejecutaremos el programa con el comando P del debug, el comando P ejecuta instrucción por instrucción sin mostrar la ejecución de las instrucciones de la interrupción 21 función 02 en AH, hasta ejecutar la interrupción 20.
Cada vez que queremos ejecutar una instrucción del programa debemos utilizar el comando P.
Como lo muestra la siguiente imagen.



Por último ejecutaremos el programa con el comando T del debug, el comando T ejecuta instrucción por instrucción igual que el comando P, con la diferencia que también muestra la ejecución de cada una de las instrucciones que pertenecen a la subrutina de la interrupción 21 función 02.

Igual que el comando P, cada vez que queremos ejecutar una instrucción debemos utilizar el comando T.
Como muestra la siguiente imagen.

Luego de esta introducción le propondremos al alumno que resuelva dos problemas utilizando el debug y documentando cada paso como lo muestra el ejemplo.

Primer Problema:
Realizar un programa assembler que permita ingresar por teclado una cadena de caracteres que termine con el signo $.
Usar la función de la interrupción 21 que toma el ingreso de datos de a un byte, los bytes ingresados deben almacenarse desde el desplazamiento 2000Hexa del segmento de datos guardado en el registro DS.

Segundo Problema:
Realizar un programa assembler que permita ingresar por teclado una expresión algebraica del tipo número + número =
Debe aparecer el resultado de la suma después del signo =
Para simplificar el problema los números a sumar serán de un digito cada uno, por ejemplo 3 + 4 = 7
Y para simplificarlo aún más la suma de los dos números siempre debe dar un número de un dígito.
No es necesario hacer ningún tipo de validación en el ingreso de datos.

Algunas Observaciones:
Los valores contenidos en los registros SP, BP, DS, ES, SS, CS, que Ud. visualice en su máquina no necesariamente serán iguales a los que Ud. esta viendo en este ejemplo.

Recuerde que cuando Ud. este en la consola DOS del debug, el sistema de numeración utilizado es el Hexadecimal.

Gestión de Memoria y Vector de Interrupciones del DOS

1) Ver video

https://www.youtube.com/watch?v=ZOM7DgxugbI&t=4s

2) Ejercicio

Crear una rutina que muestre por pantalla los caracteres de tabla ASCII de los códigos 30 a 39.
Se utilizará INT 61, y la rutina tiene como última instrucción IRET.
Los 4 bytes que forman La dirección de memoria donde se encontrará el código de máquina de la rutina que resuelve el problema se almacenan en el vector de interrupciones en el desplazamiento (61x4).
Antes de ejecutar el programa y la rutina de interrupción, se debe escribir en memoria los contenidos del vector. Para ello, se debe escribir el comando E 0000:0184 y a partir de esta dirección se debe escribir a la manera de Intel primero el valor de IP (donde está la primera instrucción elegida para la rutina de interrupción) y luego el de CS, que es el que muestra el Debug a la izquierda de los dos puntos que aparecen en cada dirección.
De no realizarse este paso, cuando se ejecuta la instrucción INT 61, la UC irá a la dirección 0000:0184 donde no encontrará la dirección correcta de inicio de la rutina, por lo que seguramente la CPU se “colgará” sin poder saltar la ejecución al lugar adecuado de memoria.
Observar en el DEBUG, que sucede con el FLAG I, después de ejecutar INT 61 y luego de ejecutar IRET.

Buena Suerte y no dejen de consultar cualquier duda que se les presente a juancarlosjromer@gmail.com