UNIVERSIDAD DE EL SALVADOR
FACULTAD MULTIDISCIPLINARIA ORIENTAL
DEPARTAMENTO DE INGENIERIA Y ARQUITECTURA
INGENIERIA DE SISTEMAS INFORMATICOS
GUIA DE DISCUCIÓN No.1:
INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR

x86-64 es una ampliación de la arquitectura x86. La arquitectura x86 fue lanzada por Intel con el procesador Intel 8086 en el año 1978 como una arquitectura de 16 bits. Esta arquitectura de Intel evolucionó a una arquitectura de 32 bits cuando apareció el procesador Intel 80386 en el año 1985, denominada inicialmente i386 o x86-32 y finalmente IA-32. Desde 1999 hasta el 2003, AMD amplió esta arquitectura de 32 bits de Intel a una de 64 bits y la llamó x86-64 en los primeros documentos y posteriormente AMD64. Intel pronto adoptó las extensiones de la arquitectura de AMD bajo el nombre de IA-32e o EM64T, y finalmente la denominó Intel 64.

Desarrollo de la práctica 1.

El ciclo de desarrollo de un programa La obtención de la versión ejecutable de un programa utilizando herramientas de desarrollo para lenguaje ensamblador requiere:

Crear el código fuente en lenguaje ensamblador. Para su creación se utiliza un editor de textos, que para este bloque de prácticas será el editor integrado en el Visual Studio.

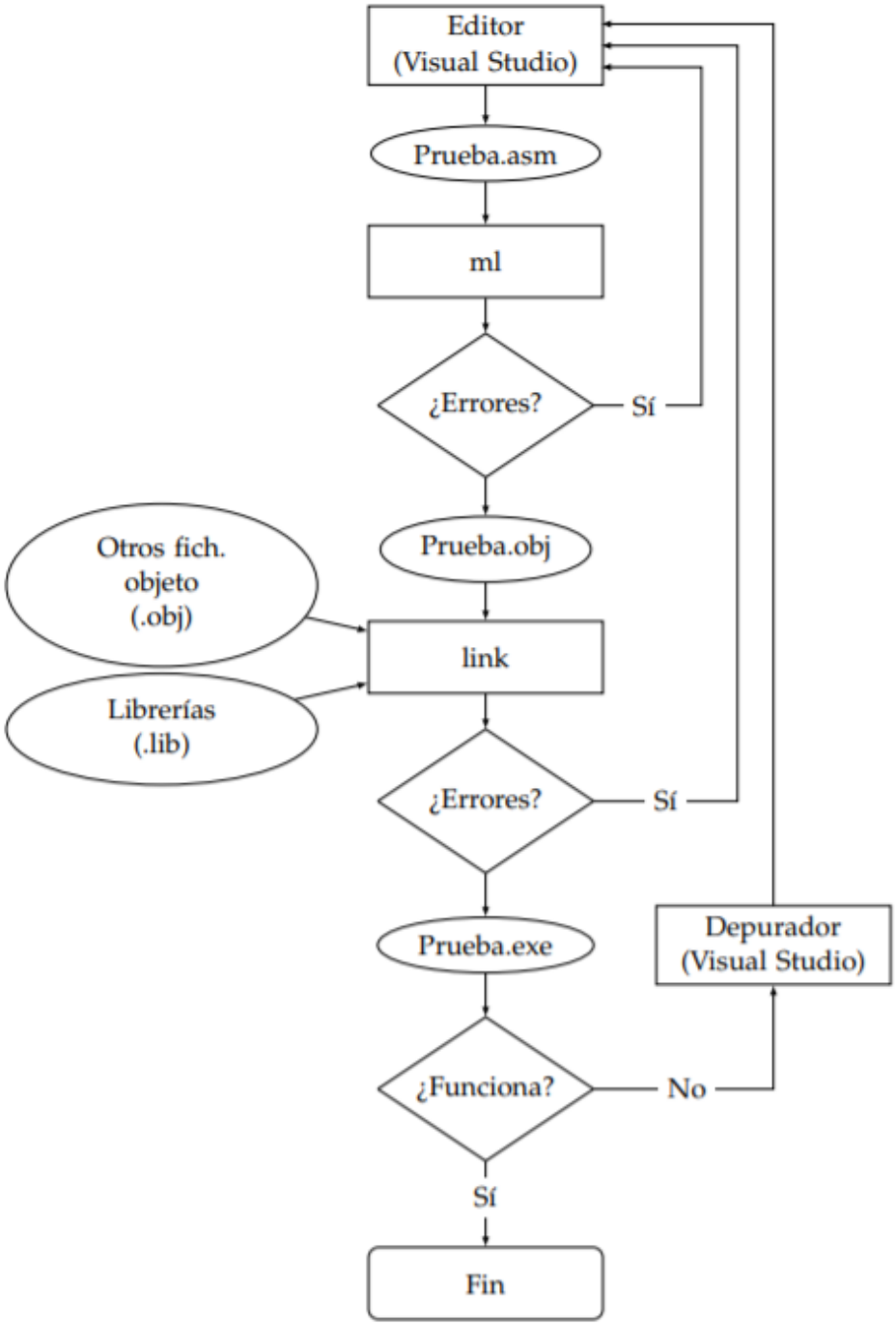
Obtener el código objeto. Esta tarea es realizada por el compilador de ensamblador (ml), que lo genera a partir del fichero fuente. Si se detectan errores en esta fase, es necesario volver al punto anterior para corregirlos.

Obtener el archivo ejecutable, enlazando el código objeto del programa con las librerías necesarias. Esto lo hace el montador de enlaces o linker (link). Si el linker fuese incapaz de hacerlo (porque quedan referencias sin resolver) sería necesario volver al primer punto para corregirlo.

Depurar el ejecutable si no funcionase correctamente. Mediante el depurador se ejecuta paso a paso el programa para comprobar por qué falla. Una vez detectado el origen del error, será necesario volver al primer punto para corregirlo.

Obtener el archivo ejecutable, enlazando el código objeto del programa con las librerías necesarias. Esto lo hace el montador de enlaces o linker (link). Si el linker fuese incapaz de hacerlo (porque quedan referencias sin resolver) sería necesario volver al primer punto para corregirlo.

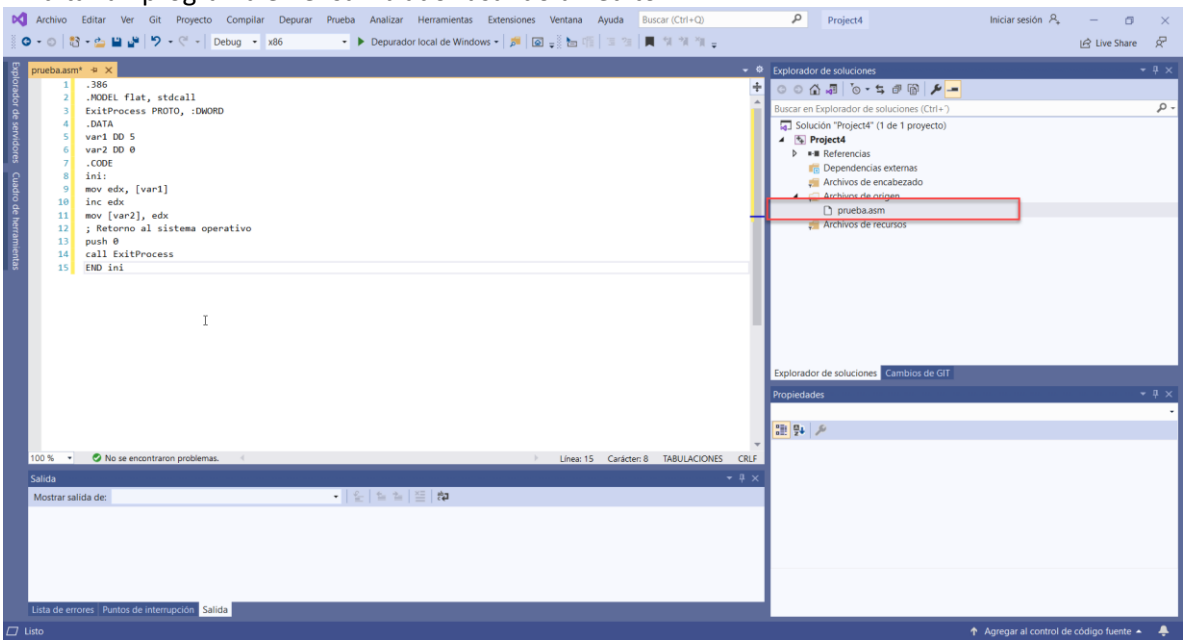
Depurar el ejecutable si no funcionase correctamente. Mediante el depurador se ejecuta paso a paso el programa para comprobar por qué falla. Una vez detectado el origen del error, será necesario volver al primer punto para corregirlo.



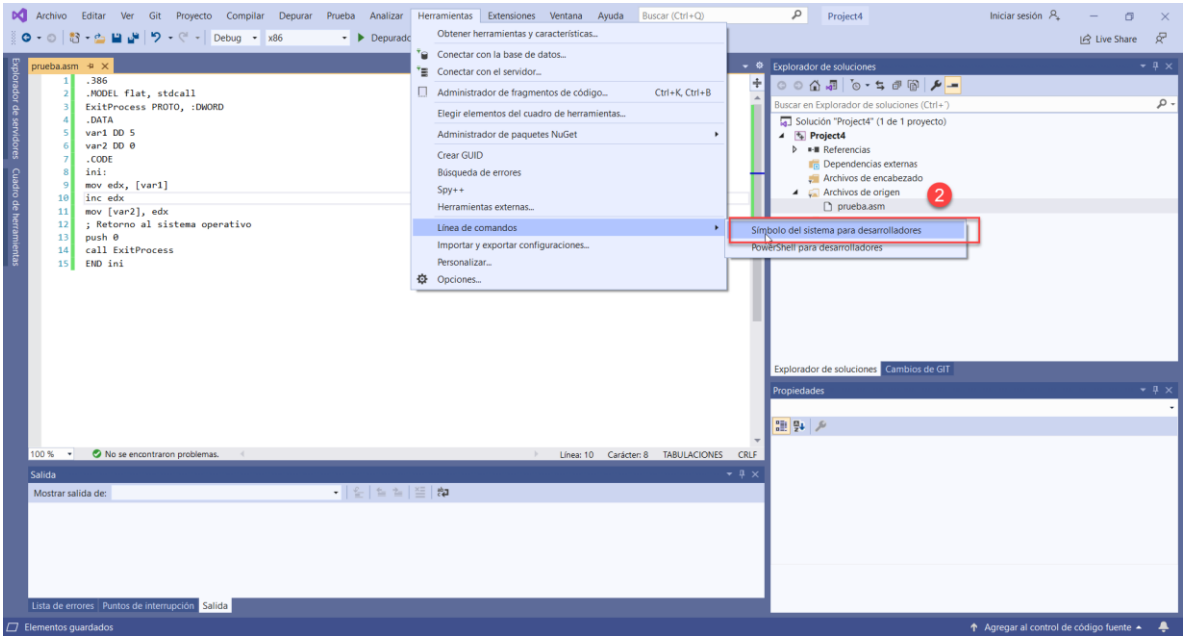
ING. LIGIA ASTRID HERNÁNDEZ BONILLA

CREAR UN EJECUTABLE

1-Editar un programa en ensamblador usando un editor.



2-Hay que arrancar una interfaz de comandos de Windows. Las herramientas del Visual Studio permiten poner en marcha una interfaz de comandos con una configuración especial. Dicha interfaz permitirá al usuario acceder al compilador (ml) y al linker (link) del Visual Studio.



3-Para ensamblar el programa utiliza el programa ml , debemos ubicarnos en la ruta en la que se encuentra almacenado el archivo .asm.
El comando dir sirve para mostrar la lista de archivos y directorios
El comando cd para ingresar a un directorio
cd.. es usado para salir de un directorio.
Ahora digitamos el comando siguiente
ml /c /Cx /coff prueba.asm

Para ejecutarlo presionamos enter.

/c significa. Solo ensambla. No realiza ninguna vinculación.
/ Cx Conserva las mayúsculas y minúsculas en los símbolos Public y extern.
/coff genera un archivo de tipo objeto

```
C:\Users\50375\source\repos\Project4\Project4>ml /c /Cx /coff prueba.asm
Microsoft (R) Macro Assembler Version 14.28.29337.0
Copyright (C) Microsoft Corporation.  All rights reserved.

 Assembling: prueba.asm
```

Los ficheros .obj no contienen una versión totalmente terminada del código máquina de un programa. Esto es debido a que, normalmente, cuando escribimos un programa, no escribimos nosotros todo el código necesario para la correcta ejecución del programa. Así, en muchas ocasiones, dentro de nuestros programas llamamos a funciones que no están escritas por nosotros. Un ejemplo de esto lo tenemos en cualquier programa ensamblador desarrollado para el sistema operativo Windows. En estos programas siempre se llama al procedimiento ExitProcess, cuyo objetivo es devolver el control al sistema operativo. Sin embargo, en nuestros programas este procedimiento no está escrito, solamente lo llamamos. El código máquina de ExitProcess se encuentra en realidad en otro fichero denominado kernel32.lib, el cual pertenece a una categoría de ficheros conocidos como librerías

4-Para enlazar el código objeto se debe enlazar el programa objeto.

```
C:\Users\50375\source\repos\Project4\Project4>link /SUBSYSTEM:CONSOLE prueba.obj kernel32.lib
Microsoft (R) Incremental Linker Version 14.28.29337.0
Copyright (C) Microsoft Corporation.  All rights reserved.


C:\Users\50375\source\repos\Project4\Project4>
```

5-Para depurar el programa deberemos ejecutar el siguiente comando.
devenv /debugexe prueba.exe

PRIMER PROGRAMA EN ENSAMBLADOR
EJERCICIO 1

```
.386
.model flat,stdcall
.stack 4096
ExitProcess proto,dwExitCode:dword
.data
sum dword 0
.code
main proc
	mov eax,5
	add  eax,6
	mov sum,eax
	invoke ExitProcess,0
main endp
end main
```

Explique las modificaciones que debe realizar para que el programa coloque:

- decimal
- binary
- hexadecimal

En el registro AX.

EJERCICIO 2

```
.386
.MODEL flat, stdcall
ExitProcess PROTO, w:DWORD
.DATA
sum dword 0
.CODE
ini:
   mov eax, 10000h          ;
   add eax, 40000h          ;
   sub eax, 20000h          ;
   mov sum,eax
       invoke ExitProcess,0
; Retorno al sistema operativo
push 0
call ExitProcess
END ini
```

¿Cuál es el valor que se le asigna a sum?

EJERCICIO 3

```
.386
.MODEL flat, stdcall
ExitProcess PROTO, w:DWORD
.DATA
array1    WORD    30 DUP(?),0,0
array2    WORD    5 DUP(3 DUP(?))
array3    DWORD   1,2,3,4
digitStr  BYTE   "12345678",0
.CODE
ini:
 mov ecx, LENGTHOF array1;
 mov ecx, LENGTHOF array2;
 mov ecx, LENGTHOF array3;
 mov ecx, LENGTHOF digitStr;
```

```
        invoke ExitProcess,0
; Retorno al sistema operativo
push 0
call ExitProcess
END ini
```

¿Cual es la longitud de los arreglos array1,array2,array3 y digitStr?

EJERCICIO 4

```
.386
.MODEL flat, stdcall
ExitProcess PROTO, w:DWORD
.DATA
dval  DWORD 12345678h
array BYTE  00h,10h,20h,30h
.CODE
ini:
   mov al,  dval
   mov eax, array
        invoke ExitProcess,0
; Retorno al sistema operativo
push 0
call ExitProcess
END ini
```

EJERCICIO 5

```
.386
.model flat,stdcall,c
      includelib     msvcrtd
      includelib     oldnames
      includelib     legacy_stdio_definitions.lib

.data
mensaje   db     "Hola mundo!",0dh,0ah,0
```

```
.code
      extrn   printf:near

public  main

main    proc
      mov eax, offset mensaje
      push eax
      call    printf
      pop eax ;

      ret
main    endp

end
```

Modifique el código para mostrar un menú.
1-Agregar
2-Eliminar
3-Salir

EJERCICIO 6

```
INCLUDE mip115.inc

.data
etiqueta db "Bienvenidos a MiP115", 0

holamensaje BYTE "Esta es una mensaje de bienvenida.", 0dh,0ah
            BYTE "Clic para continuar...", 0

.code
main PROC

          mov    ebx,0
          mov    edx,OFFSET holamensaje          ; este mensaje no tiene etiqueta
          call     MsgBox

          mov    ebx,OFFSET etiqueta
          mov    edx,OFFSET holamensaje
          call     MsgBox
```

```
        exit
main ENDP

END main
```

EJERCICIO 7

```
INCLUDE mip115.inc

.data
etiqueta BYTE "Encuesta",0
pregunta BYTE "Gracias por completar la encuesta"
        BYTE 0dh,0ah
        BYTE "Legustaria recibir resultados?",0
resultados BYTE "Se enviaran a su correo",0dh,0ah,0
.code
main PROC

        mov ebx,OFFSET etiqueta
        mov edx,OFFSET pregunta
        call MsgBoxAsk


        exit
main ENDP

END main
```