I. TEMA: PROGRAMACION EN ENSAMBLADOR

II. OBJETIVO DE LA PRACTICA

Al finalizar la presente práctica, el estudiante:

1. Describe el flujo de trabajo para escribir programas en el lenguaje Ensamblador de los microprocesadores x86 de Intel.
2. Escribe programas en lenguaje Ensamblador para Linux utilizando el compilador NASM

III. TRABAJO PREPARATORIO.

1. Conceptos básicos de organización y arquitectura de computadores.
2. Conocimientos básicos del conjunto de instrucciones ensamblador de los microprocesadores de arquitectura X86

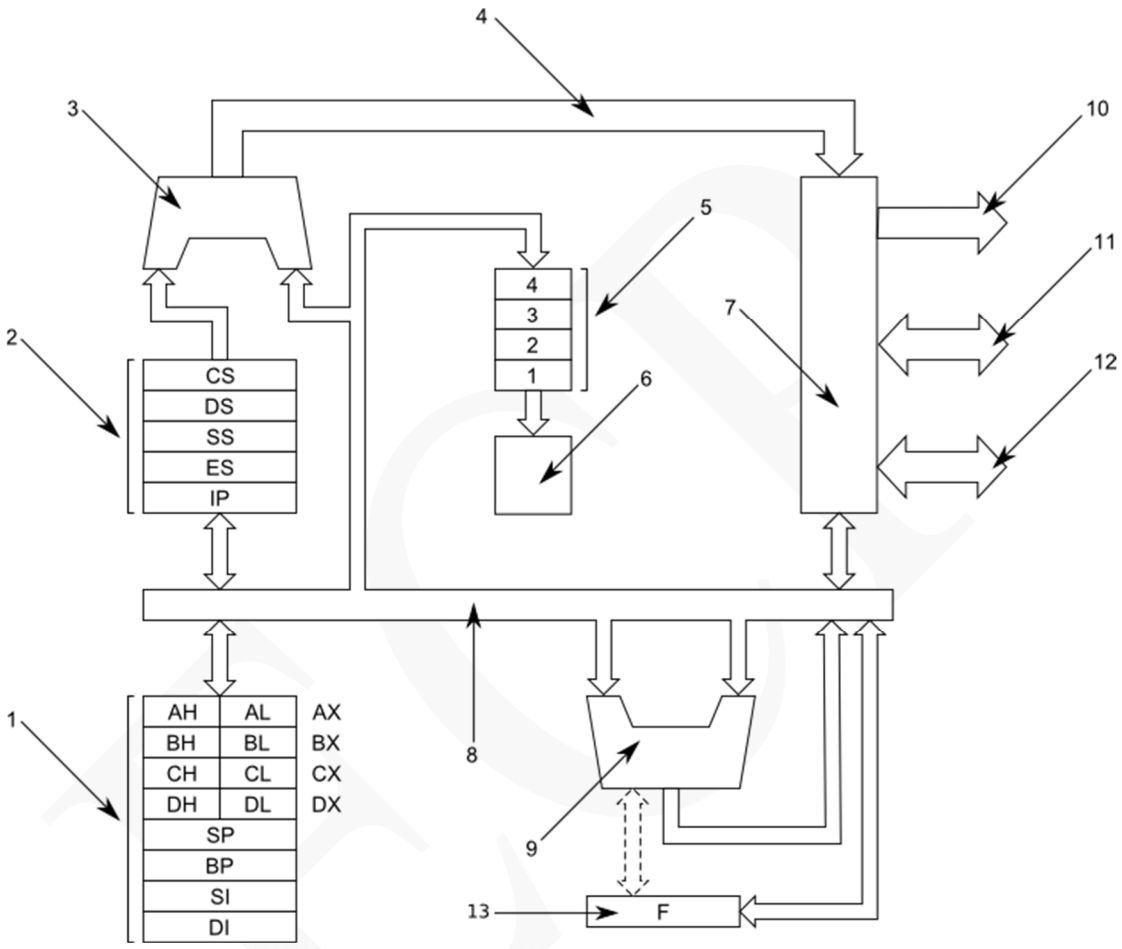
IV. MATERIALES.

1. Sistema operativo Linux
2. Compilador NASM
3. Lista de instrucciones de los procesadores x86
4. Editor de texto Geany, Kate u otro equivalente
5. Librería io.mac e io.o (Para leer datos desde el teclado y mostrar los resultados en el monitor)

V. MARCO TEORICO

ARQUITECTURA DE LOS MICROPROCESADORES X86

Los elementos más importantes de los procesadores de la familia Intel 8086/8088 se muestran en el gráfico y se describe brevemente cada uno de ellos.



1. Registros de propósito general – Registros accesibles al programador y utilizados para el procesamiento de datos.
2. Registros de segmento – Registros para el manejo de segmentos de memoria
3. Sumador de direcciones – Sumador que se utiliza para el cálculo de direcciones para acceso a memoria.
4. Bus interno de direcciones – Líneas de comunicación para direccionar memoria y dispositivos periféricos
5. Cola de instrucciones – Almacena instrucciones pre extraídas de memoria
6. Unidad de ejecución – Ejecuta las instrucciones.
7. Lógica de control del bus – Administra la operación del bus del sistema
8. Bus de datos – Líneas de comunicación para la transferencia de datos entre componentes del sistema.
9. ALU – Unidad Aritmético – Lógica.
10. Bus de direcciones – Líneas para el envío de direcciones desde el CPU a los dispositivos periféricos y memoria.

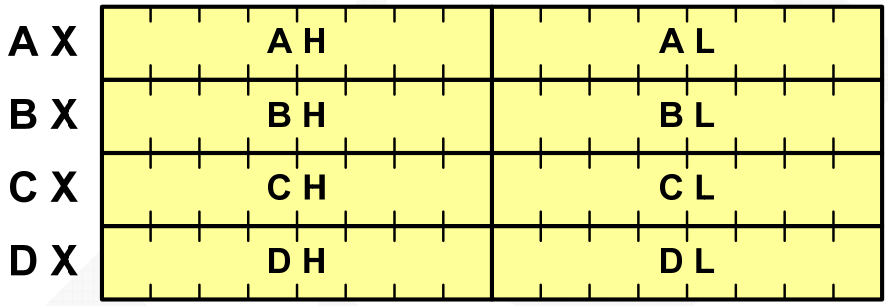
11. Bus de datos – Líneas para el intercambio de datos entre los componentes del computador.
12. Bus de control – Líneas para la transmisión de señales para controlar el comportamiento de los componentes del computador.
13. Registro de banderas – Registras las condiciones resultantes de las operaciones en el CPU

Desde el punto de vista del programador, sin embargo, no todos estos elementos son accesibles, así, por ejemplo, la cola de instrucciones es una estructura de propósito especial, que almacena instrucciones que la unidad de pre extracción trae de la memoria principal en los ciclos libres del bus del sistema con el propósito de reducir el tiempo de procesamiento de instrucciones.

La arquitectura de nivel ISA del microprocesador 8086/8088 se estudia a continuación:

REGISTROS DE PROPOSITO GENERAL

Los registros de propósito general (RPG), se utilizan para el procesamiento de datos. Estos registros son de 16 bits de extensión, pero pueden ser accedidos por bytes.



Los nombres y las funciones de los registros son:

AX (Acumulador) - Se utiliza para almacenar el resultado de operaciones aritméticas y lógicas. Los ocho bits menos significativos se pueden acceder utilizando el nombre de registro AL y los más significativos, a través de AH.

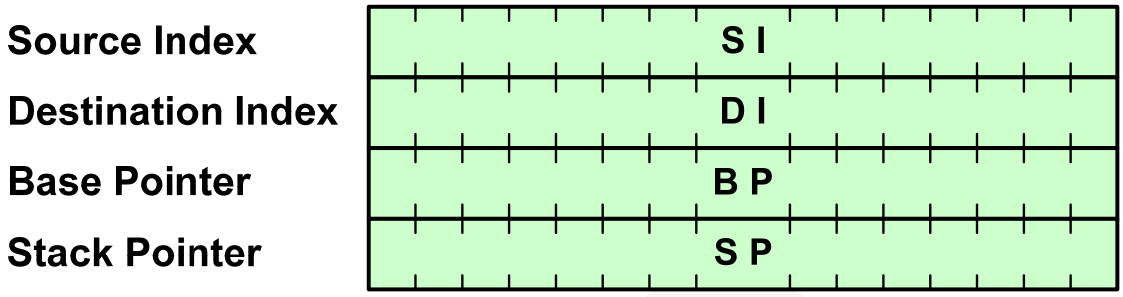
BX (Base) - Se suele utilizar para almacenar la dirección base de un elemento en memoria. También puede ser referenciado como BH y BL

CX (Contador) - Se suele utilizar como contador en algunas operaciones, pero puede ser utilizado para cualquier operación aritmética o lógica. Puede ser referenciado como CH y CL

DX (Datos) - Registro de uso general, contiene la parte más significativa de un producto, después de una multiplicación de 16 o 32 bits o la parte más significativa del dividendo antes de la división y el número de puerto de e/s para operaciones de e/s. puede ser referenciado como DH y DL.

REGISTROS INDICE Y APUNTADORES

Registros de propósito general que se usan frecuentemente para hacer referencia a datos en memoria.



SI (Indice de Fuente) - Se utiliza para almacenar referencias indirectas a datos fuente.

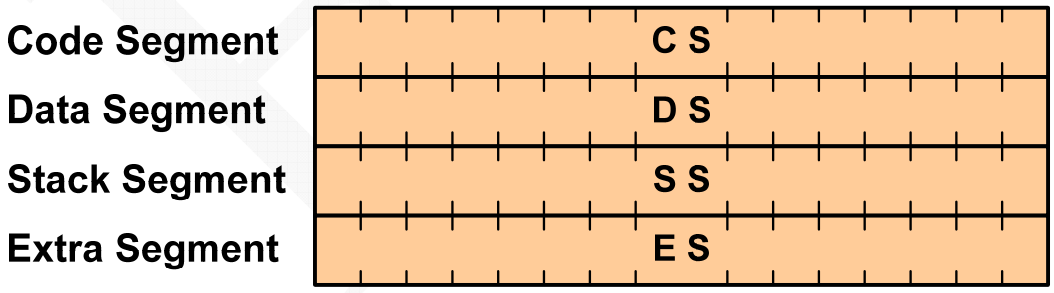
DI (Indice de Destino) - Se utiliza para almacenar referencias indirectas a datos destino.

BP (Apuntador de Base) - Apuntador de uso general, se suele utilizar para hacer referencia a una matriz de datos en una pila de memoria.

SP (Apuntador de Pila) - Se utiliza para hacer referencia a datos en una pila

REGISTRO DE SEGMENTO

Estos registros se utilizan para definir los segmentos que contendrán código o datos de los programas en el computador



CS (Segmento de Código) - Apunta a la dirección inicial, en memoria, donde comienza el código de un programa.

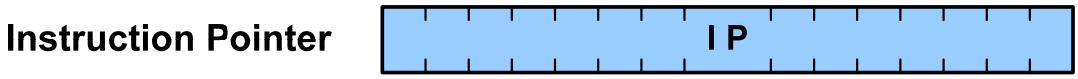
DS (Segmento de Datos) - Apunta a la dirección inicial, en memoria, donde se almacenan los datos que utiliza un programa.

SS (Segmento de Pila) - Apunta a la dirección inicial, en memoria, donde comienza el segmento que pila de un programa.

ES (Segmento Extra) - Se utiliza para definir segmentos adicionales, de ser necesario.

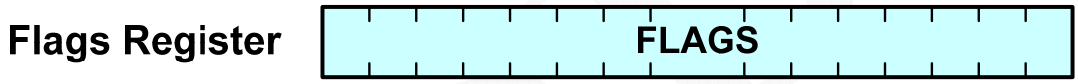
APUNTADOR DE INSTRUCCIONES

IP (Apuntador de Instrucciones) - Este registro se utiliza para hacer referencia a la dirección de la siguiente instrucción que se ejecutará



REGISTRO DE BANDERAS

Este registro almacena información sobre el estado de las operaciones que se ejecutan en el procesador, también se le conoce como PSW o Processor State Word y se utiliza para tomar decisiones de salto condicional.

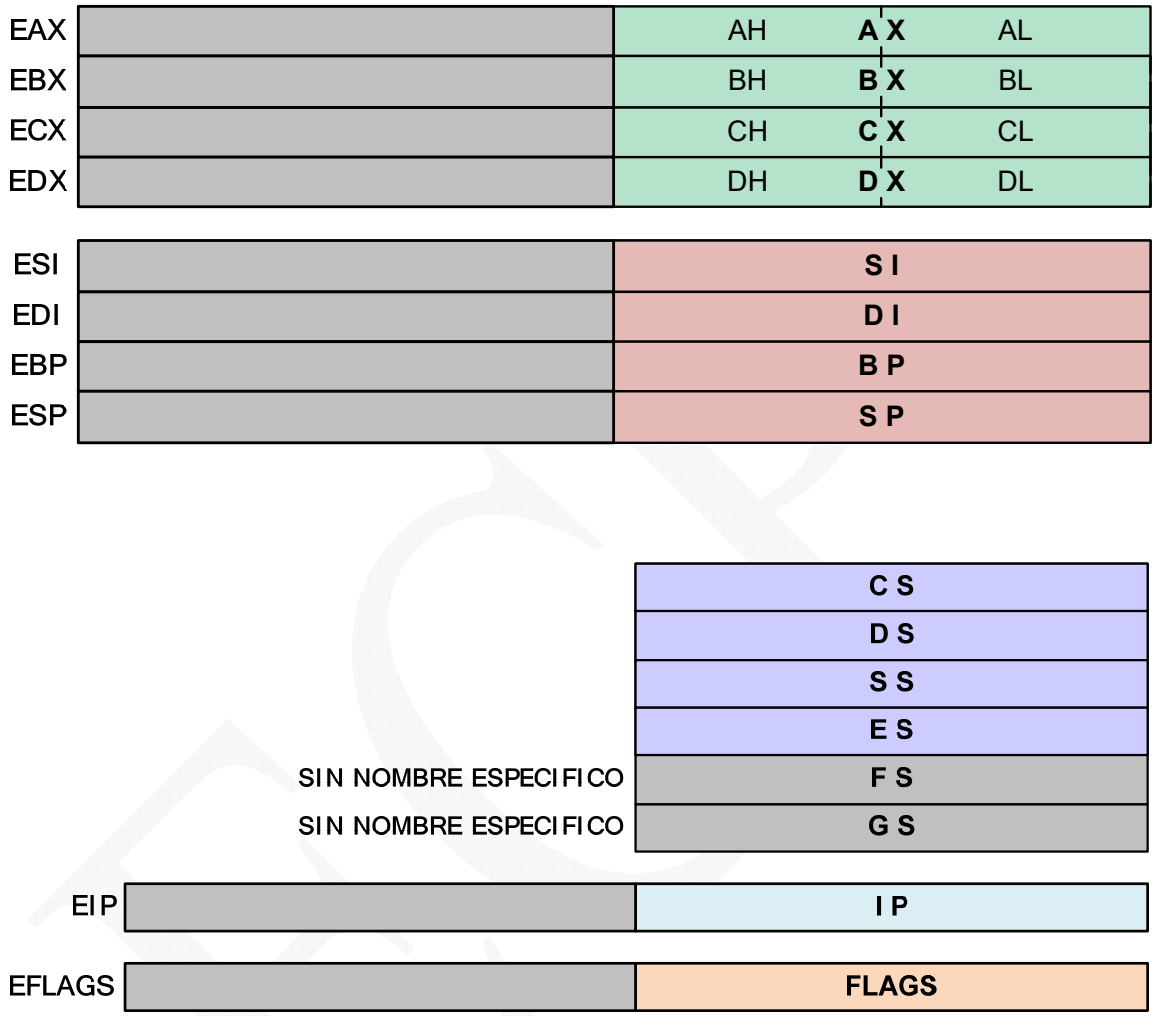


Las condiciones pueden ser:

O (Overflow)	-	Este bit se activa cuando una operación, con signo, genera un resultado cuyo valor excede la capacidad de almacenamiento de los registros.
D (Dirección)	-	Se utiliza para determinar la dirección de incremento (positivo o negativo) automático para operaciones con cadenas utilizando los registros DI o SI.
I (Interrupción)	-	Se activa cuando un dispositivo periférico requiere atención del procesador.
T (Trampa)	-	Se utiliza para tareas de depuración de programas.
S (Signo)	-	Se activa o desactiva dependiendo del signo del resultado de la última operación.
Z (Cero)	-	Se activa si el resultado de la última operación es igual a cero.
A (Acarreo auxiliar)	-	Se activa cuando ocurre un acarreo entre los bits 3 y 4 (BCD).
P (Paridad)	-	Se activa si el número de unos (1) en un número es par.
C (Acarreo)	-	Se activa cuando hay acarreo desde el bit más significativo en una operación de suma o un préstamo en una operación de resta.

MICROPROCESADOR 80386

Por su parte, un procesador 80386, que es la base de los procesadores de Intel hasta la actualidad, presenta la siguiente organización de registros (modelo de programación)



Las funciones de los registros son similares a las indicadas para los procesadores 8086/88 con la diferencia que los registros se amplían a 32 bits y dependiendo de la granularidad de acceso, los registros de datos pueden ser, por ejemplo, AL, AH, AX o EAX si se quiere hacer referencia al byte menos significativo del registro AX, al byte más significativo del registro AX, al registro AX que es la mitad menos significativa del registro EAX o a los 32 bits del registro respectivamente. No se puede hacer referencia a los 16 bits más significativos de estos registros.

INSTRUCCIONES ENSAMBLADOR

Las instrucciones de manipulación de datos, tienen como propósito permitir procesar los datos en el computador. A diferencia de las instrucciones de transferencia de datos, estas instrucciones transforman los datos de acuerdo a la operación aplicada sobre los mismos.

Las instrucciones de manipulación de datos pueden agruparse en:

1. ***Instrucciones aritméticas***. Dentro de este grupo están las instrucciones que permiten realizar operaciones aritméticas sobre los datos.
2. ***Instrucciones lógicas y de manipulación de bits***. Dentro de este grupo están las instrucciones que permiten realizar operaciones lógicas sobre los datos. Se caracterizan porque operan a nivel de bits, es decir, tratan cada bit individualmente.
3. ***Instrucciones de corrimiento***. Pueden a su vez ser:
   i) ***Corrimiento lógico***. Instrucciones de desplazamiento que tratan los datos como secuencias de bits.
   ii) ***Corrimiento aritmético***. Instrucciones de desplazamiento que tratan los datos como números con signo.
   iii) ***Rotación***. Instrucciones de desplazamiento que tratan los datos como secuencias de bits y que difieren de las operaciones de corrimiento, en que los datos que salen por un extremo, vuelven por el extremo opuesto, de modo que después de N rotaciones (para palabras de N bits), el dato original se restablece.

La descripción, sintaxis, ejemplos, así como los indicadores del registro de banderas que cada una de ellas afecta, se detallan a continuación:

INSTRUCCIONES ARITMÉTICAS

ADD	Suma		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
ADD Dest, Orig	reg, reg	ADD CX, SI	
	mem, reg	ADD DATA, AL	
	reg, mem	ADD CL, [EAX]	
	reg, inm	ADD CX, 12	
	mem, inm	ADD DATA, 44	
	acc, inm	ADD AX, 3	

SUB	Sustracción		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
SUB Dest, Orig	reg, reg	SUB CX, SI	
	mem, reg	SUB DATA, AL	
	reg, mem	SUB CL, [EAX]	
	reg, inm	SUB CX, 12	
	mem, inm	SUB DATA, 44	
	acc, inm	SUB AX, 3	

ADC	Suma con acarreo		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
ADC Dest, Orig	reg, reg	ADC AX, BX	
	mem, reg	ADC DATA, AH	
	reg, mem	ADC BL, DATA	
	reg, inm	ADC CX, 3	
	mem, inm	ADC DATA, 32	
	acc, inm	ADC AH, 34	

MOVSX	Transferir con signo extendido		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
MOVSX Dest, Orig	reg, reg	MOVSX BX, AL	
	reg, mem	MOVSX AX, DATA	

MOVZX	Transferir con cero extendido		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
MOVSX Dest, Orig	reg, reg	MOVZX BX, AL	
	reg, mem	MOVZX AX, DATA	

PUSH	Transferir datos a la pila		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
PUSH Orig	reg	PUSH CX	
	mem	PUSH DATA	
	seg	PUSH DS	
	inm	PUSH 10H	

POP	Extraer datos de la pila		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
POP Dest	reg	POP CX	
	mem	POP DATA	
	seg	POP DS	

XCHG	Intercambiar		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
XCHG Dest, Orig	reg, reg	XCHG AX, DX	
	mem, reg	XCHG DATA, CL	
	reg, mem	XCHG CL, DATA	

XLAT	Traducir		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
XLAT		XLAT	

IN	Leer datos de puerto de entrada		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
IN Dest, Orig	acc, puerto	IN AL, 12H	
	acc, DX	IN AL, DX	

OUT	Enviar datos a puerto de salida		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
OUT Dest, Orig	puerto, acc	OUT 12H, AL	
	DX, acc	OUT DX, AX	

LEA	Cargar dirección efectiva		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
LEA Dest, Orig	reg, mem	LEA DI, DATA	

LDS	Cargar apuntador lejano		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
LDS Dest, Orig	reg, mem	LDS DI, DATA	

LES	Cargar apuntador lejano		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
LES Dest, Orig	reg, mem	LES DI, DATA	

LFS	Cargar apuntador lejano		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
LFS Dest, Orig	reg, mem	LFS DI, DATA	

LGS	Cargar apuntador lejano		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
LGS Dest, Orig	reg, mem	LGS DI, DATA	

LSS	Cargar apuntador lejano		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
LSS Dest, Orig	reg, mem	LSS DI, DATA	

LAHF	Carga el registro AH con el registro de banderas		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
LAHF		LAHF	

SAHF	Carga el registro de banderas con el registro AH		
Sintaxis	***Operandos***	***Ejemplo***	***Banderas***
SAHF		SAHF	S Z A P C

Además de estas instrucciones, también se incluyen instrucciones para la transferencia de cadenas, tales como MOVS, LODS, STOS, INS y OUTS

PROGRAMACION EN LENGUAJE ENSAMBLADOR

Para escribir programas en lenguaje ensamblador trabajaremos con el sistema operativo Linux y el compilador de lenguaje ensamblador NASM (Netwide Assembler).

Las ventajas de utilizar estas herramientas, es que ambas, tanto Linux como NASM son herramientas de software libre y tienen, en la actualidad, mejor documentación y soporte que sus versiones en Windows.

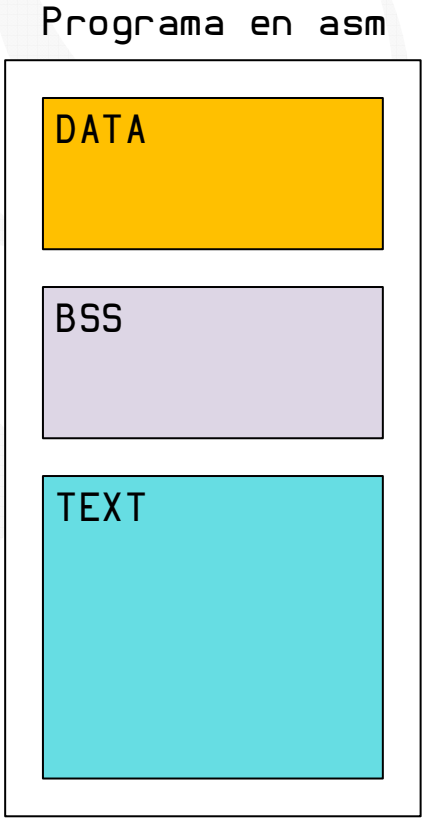
El NASM puede descargarse desde el sitio web del proyecto (http://nasm.sourceforge.net/), donde también pueden conseguirse manuales y versiones de NASM para otras plataformas.

Para escribir los archivos fuente podemos utilizar cualquier editor de texto, sin embargo, se recomienda utilizar el editor Geany, pues éste incorpora una consola desde donde podremos compilar y ejecutar el programa sin salir a otra aplicación.

Para el enlazado (link) utilizaremos el enlazador *ld* de GNU.

ESTRUCTURA DE UN PROGRAMA EN ENSAMBLADOR

Un programa en Ensamblador puede tener las siguientes secciones:



DATA – Contiene las variables inicializadas que utilizará el programa
BSS – Contiene las variables no inicializadas que utilizará el programa
TEXT – Contiene las instrucciones que compone el programa

Por ejemplo:

hola.asm

```
;Nombre          :   hola.asm
;Proposito       :   muestra una cadena en Linux
;Autor           :   Edwin Carrasco
;FCreacion       :   05/03/2008
;FModificacion   :   29/10/2021
;Compilar        :   nasm -f elf hola.asm
;Enlazar         :   ld -m elf_i386 -s hola.o -o hola
;Ejecutar        :   ./hola

section .data
    ;Seccion de variables inicializadas
    strHola:    db  'Hola mundo ...', 10
    longHola:   equ $-strHola

section .text
    ;Seccion de codigo (instrucciones de programa)
    global  _start  ;Punto de entrada al programa (main)

_start:
    mov edx,longHola    ;Longitud de cadena
    mov ecx,strHola     ;Cadena que se mostrara
    mov ebx,1           ;Descriptor de archivos (stdout)
    mov eax,4           ;Llamada de sistema (sys_write)
    int 80h             ;llamada al S.O.

    ;Salir del programa y devolver el control al S.O.
    mov ebx,0   ;Estado de finalizacion del proceso
                ;0 - sin errores
    mov eax,1   ;Llamade de sistema (sys_exit)
    int 80h     ;Llamada al S.O.
```

Terminal

```
ecp@ecp-OAC:~/codigo_ASM$ nasm -f elf hola.asm
ecp@ecp-OAC:~/codigo_ASM$ ld -m elf_i386 -s hola.o -o hola
ecp@ecp-OAC:~/codigo_ASM$ ./hola
Hola mundo ...
ecp@ecp-OAC:~/codigo_ASM$
```

En algunos compiladores, como es el caso de NASM, se admite, también, declarar estas secciones como:

```
.data
.bss
.text
```

Dependiendo de la naturaleza del programa, las secciones **DATA** y **BSS** pueden, como no, estar presentes.

Como una buena práctica de programación, todos nuestros programas deben estar bien documentados, por lo que deben incluir un encabezado como el que se muestra en el programa de ejemplo y utilizar comentarios, donde el programa lo amerite, teniendo en consideración que los comentarios se utilizan para hacer clara una parte del programa y que los comentarios triviales, implícitos o superfluos no ayudan en ese aspecto.

La instrucción:

```
int 80h
```

tiene como propósito realizar una llamada al sistema operativo para que realice una acción definida por los parámetros indicados a través de los registros de propósito general (RPG).

Por ejemplo, las instrucciones

```
mov ebx, 0
mov eax, 1
int 80h
```

se utilizan siempre al final de los programas para indicar el estado de terminación de un proceso a través del valor cargado en el registro ebx (0 – sin errores; 1 – con errores) y la operación que se requiere que ejecute el sistema operativo a través del valor cargado en el registro eax (1 – sys_exit).

A continuación de muestra un ejemplo simple de un programa en Ensamblador: este programa calcula el doble de un número ingresado por el usuario y devuelve el resultado a través del registro AX.

El programa incluye una referencia a la librería io.mac, la cual permite manejar las operaciones de lectura desde el teclado y de escritura en pantalla, que, de otro modo, requerirían ser implementadas y gestionadas por el programador.

doble.asm

```
;Nombre     :   doble
;Proposito  :   muestra el doble de un entero ingresado por el usuario
;Autor      :   Edwin Carrasco
;FCreacion  :   29/10/2021
;FModif.    :   --
;Compilar   :   nasm -f elf doble.asm
;Enlazar    :   ld -m elf_i386 io.o doble.o -o doble
;Ejecutar   :   ./doble

%include "io.mac"

section .data
    strProposito:   db  "ESTE PROGRAMA CALCULA EL DOBLE DE UN ENTERO NATURAL",10,0
    strLeerEntero:  db  9, "Ingrese un entero natural menor a 16384: ",0
    strMsgSalida1:  db  9, "El doble de ", 0
    strMsgSalida2:  db  " es: ", 0

section .text
    global _start:

_start:
    PutStr strProposito     ;Proposito del programa
    PutStr strLeerEntero    ;Pedir datos de entrada
    GetInt ax               ;Leer datos

    ;Procesar
    mov bx, ax  ;bx <-- ax
    add ax, bx  ;ax <-- ax + bx

    ;Mostrar resultado
    PutStr strMsgSalida1
    PutInt bx
    PutStr strMsgSalida2
    PutInt ax
    nwln        ;Imprimir linea

    mov ax, 1   ;Salir del programa
    int 80h
```

```
ecp@ecp-OAC:~/codigo_ASM$ nasm -f elf doble.asm
ecp@ecp-OAC:~/codigo_ASM$ ld -m elf_i386 io.o doble.o -o doble
ecp@ecp-OAC:~/codigo_ASM$ ./doble
ESTE PROGRAMA CALCULA EL DOBLE DE UN ENTERO NATURAL
        Ingrese un entero natural menor a 16384: 654
        El doble de 654 es: 1308
```

Borrador

Terminal

LIBRERÍA IO.MAC

En el lenguaje de programación ensamblador no existen instrucciones para realizar las operaciones de entrada y salida tan fácilmente como en los lenguajes de alto nivel como C o Java. Estas operaciones deben realizarse explícitamente, pero implican un nivel de complejidad que, por el momento, supera nuestros conocimientos, por lo que dejaremos este reto para más adelante.

Para que nuestros programas puedan interactuar con el usuario, utilizaremos la librería ***io.mac***, la cual contiene macros para leer datos desde el teclado, mostrar resultados a través del monitor y otras operaciones de entrada/salida.

Por ejemplo:

La instrucción *PutStr* permite mostrar una cadena a través del monitor.
La instrucción *PutInt* permite mostrar un entero a través del monitor.
La instrucción *PutLint*, permite mostrar un entero largo a través del monitor.
La instrucción *GetInt*, permite leer un entero desde el teclado.

La librería en mención esta compilada, por lo que no podemos acceder a su código y usarlo en nuestros programas. Sin embargo, podemos utilizar sus métodos y enlazar nuestros programas durante el proceso de creación de los mismos.

Para este fin, debemos declarar en el archivo fuente de nuestro programa la sentencia

```
%include "io.mac"
```

Luego, en el proceso de enlace combinamos nuestro archivo con el archivo objeto de la librería mediante la orden:

```
ld -s -o [Nombre del archivo ejecutable] [Nombre el archivo objeto] io.o
```

Por ejemplo:

La orden

```
ld -s -o sumaPares sumaPares.o io.o
```

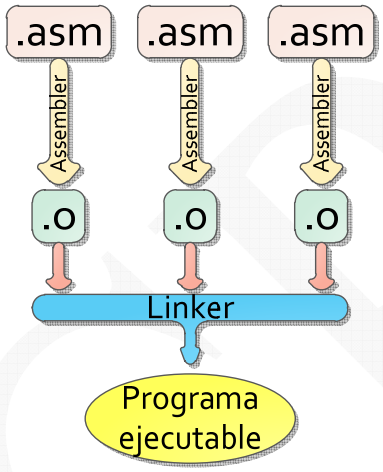
Combina los archivos *io.o* y *sumaPares.o* en el archivo ejecutable *sumaPares*.

De no utilizarse esta librería, nuestros programas solo podrán devolver un resultado a la vez, a través del registro EBX, que se utiliza para devolver el código de estado de finalización del programa en ensamblador al volver al programa que invocó al mismo

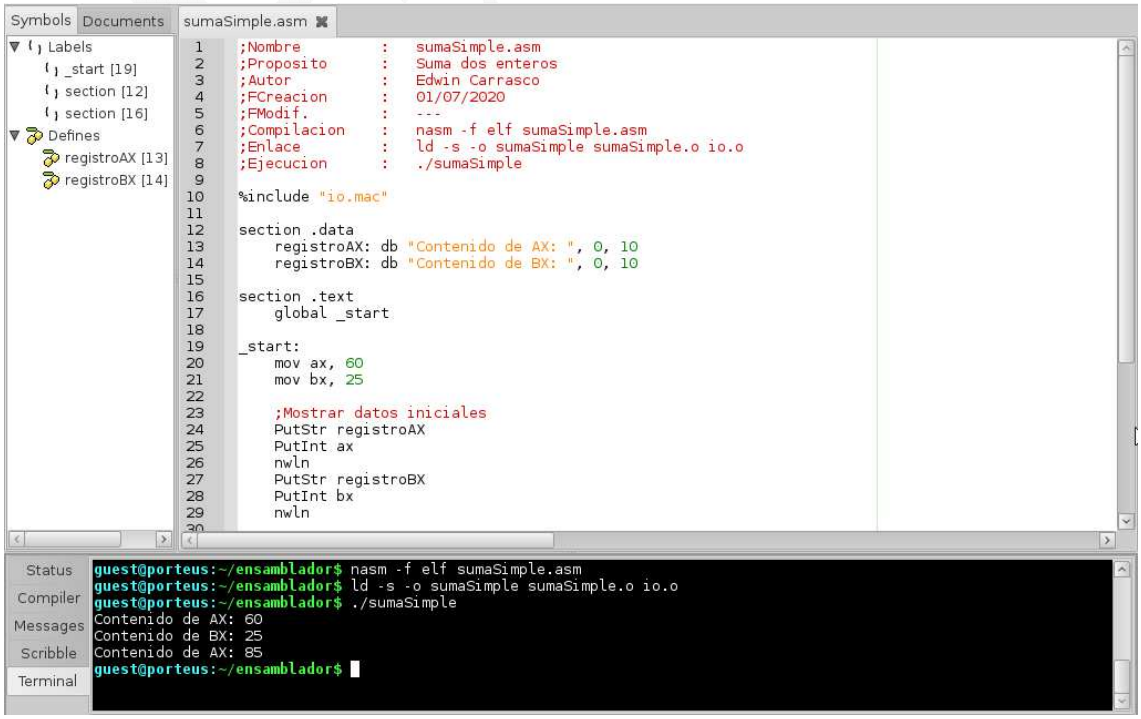
Para hacer más fácil el proceso de compilación y enlace, los archivos ***io.mac*** e ***io.o*** deben estar en la misma carpeta que nuestro archivo fuente.

COMPILACION, ENLAZADO Y EJECUCION DE PROGRAMAS EN ENSAMBLADOR

Las fases que se siguen para crear un programa ejecutable a partir de archivos de código fuente en lenguaje ensamblador se muestran en el gráfico que sigue, en el cual puede verse que uno o varios archivos con código fuente ensamblador (y con extensión *asm*) se compilan para obtener un archivo objeto correspondiente (con extensión *o*). Luego se utiliza un enlazador (linker) para enlazar los programas con extensión *o* y a partir de estos obtener un archivo ejecutable.



En nuestro caso, el programa que se muestra como ejemplo, debe ser editado en un editor de texto cualquiera. Para el ejemplo, se utilizará el editor de texto Geany. La ventaja de este editor, es que soporta coloración de sintaxis NASM y, además, como se mencionó anteriormente, permite acceder, desde la misma interfaz, a la consola, por lo que, desde este mismo entorno, es posible también, compilar, enlazar y ejecutar el programa.



Para compilar el programa utilizamos la sentencia:

```
nasm -f elf sumaSimple.asm
```

Para enlazar el programa objeto resultante `sumaSimple.o` con la librería `io.mac` (`io.o` es el archivo objeto de `io.mac`) y hacerlo ejecutable, debemos utilizar la sentencia:

```
ld -s -o sumaSimple sumaSimple.o io.o
```

Finalmente, para ejecutar el programa resultante, utilizamos la sentencia:

```
./sumaSimple
```

Si los procesos de compilación y enlazado no presentaron problemas, el resultado debería ser:

```
guest@porteus:~/ensamblador$ nasm -f elf sumaSimple.asm
guest@porteus:~/ensamblador$ ld -s -o sumaSimple sumaSimple.o io.o
guest@porteus:~/ensamblador$ ./sumaSimple
Contenido de AX: 60
Contenido de BX: 25
Contenido de AX: 85
guest@porteus:~/ensamblador$
```

En otro caso, debemos proceder a depurar y corregir el programa.

Si compila este programa en un computador de 64 bits y encuentra problemas, cambie en el código fuente, las referencias a los registros de 16 bits (ax, bx, cx, dx, etc.) por referencias a registros de 32 bits (eax, ebx, ecx, edx, etc.) y compile y enlace el programa con las siguientes órdenes:

```
nasm -f elf sumaSimple.asm

ld -m elf_i386 -s -o sumaSimple sumaSimple.o io.o
```

VI. TRABAJO DE LABORATORIO.

1. Escriba un programa que intercambie el contenido de dos registros sin utilizar una variable o registro temporal

Solución

```
;Nombre          : swapValores.asm
;Proposito       : Intercambia el contenido de dos registros
;Autor           : Edwin Carrasco
;FCreacion       : 16/02/2009
;FModificacion   : 18/07/2019
;Compilacion     : nasm -f elf swapValores.asm
;Enlace          : ld -m elf_i386 -s -o swapValores
                   swapValores.o io.o
;Ejecucion       : ./swapValores


%include "io.mac" ; libreria de macros de entrada/salida


section .data
 estadoAX: db "contenido de AX: ",0,10
 estadoDX: db "contenido de DX: ",0,10

section .text
    global  _start

_start:
 mov ax, 12 ; 0000 0000 0000 1110 -> ax=12
 mov dx, 7  ; 0000 0000 0000 0111 -> dx=7

 PutStr estadoAX
 PutInt ax
 nwln
 PutStr estadoDX
 PutInt dx
 nwln

 xor ax, dx ; 0000 0000 0000 1001 -> ax=9
 xor dx, ax ; 0000 0000 0000 1110 -> dx=12
 xor ax, dx ; 0000 0000 0000 0111 -> ax=7

 PutStr estadoAX
 PutInt ax
 nwln
 PutStr estadoDX
 PutInt dx
 nwln

 ;volver al sistema
 mov ax, 1    ; salir del programa
 int 80h
```

Verificamos la funcionalidad de nuestro programa:

```
ecp@ecp-OAC:~/codigo_ASM$ nasm -f elf swapValores.asm
ecp@ecp-OAC:~/codigo_ASM$ ld -m elf_i386 -s -o swapValores swapValores.o io.o
ecp@ecp-OAC:~/codigo_ASM$ ./swapValores
contenido de AX: 12
contenido de DX: 7
contenido de AX: 7
contenido de DX: 12
```

2. Escriba un programa que divida dos números enteros y muestre el cociente y el residuo por pantalla. El usuario debe poder definir los números de entrada

Solución

```
;Nombre        : division.asm
;Proposito     : Divide dos enteros y muestra el cociente y el
                 residuo
;Autor         : Edwin Carrasco
;FCreacion     : 18/07/2019
;FModific.     : --
;Compilacion   : nasm -f elf division.asm
;Enlace        : ld -m elf_i386 -s -o division division.o io.o
;Ejecucion     : ./division

%include "io.mac" ; librerias de e/s

SECTION .DATA
 mensaje:   db   "ESTE PROGRAMA CALCULA EL COCIENTE Y EL
                 RESIDUO",10,0
 dividendo: db   "Ingrese el dividendo: ",0
 divisor:   db   "Ingrese el divisor: ",0
 cociente:  db   "El cociente es: ",0
 residuo:   db   "El residuo es: ",0


SECTION .TEXT
    global  _start

_start:

 ;Indicar que hace el programa
 PutStr mensaje

 ;Leer dividendo
 PutStr dividendo
 GetLInt eax

 ;Leer divisor
 PutStr divisor
 GetLInt ebx

 ;Procesar: Dividir
 div ebx  ;Cociente en eax y residuo en edx

 ;Mostrar cociente
 PutStr cociente
 PutLInt eax
 nwln

 ;Mostrar residuo
 PutStr residuo
 PutLInt edx
 nwln
```

```
;terminar
mov eax, 1
int 80h
```

Verificamos la funcionalidad de nuestro programa:

```
ecp@ecp-OAC:~/codigo_ASM$ nasm -f elf division.asm
ecp@ecp-OAC:~/codigo_ASM$ ld -m elf_i386 io.o division.o -o division
ecp@ecp-OAC:~/codigo_ASM$ ./division
ESTE PROGRAMA CALCULA EL COCIENTE Y EL RESIDUO
Ingrese el dividendo: 139
Ingrese el divisor: 13
El cociente es: 10
El residuo es: 9
```

VII. PRACTICAS DE LABORATORIO

1. Escriba un programa que calcule X = [(A * B) / (C – D)] + E.

2. Escriba un programa que multiplique dos números enteros ingresados por el usuario; le reste 10 al resultado, luego divida este valor entre 15, le reste al cociente 2 y le sume al resto 8; sume los dos valores y aplíquele la operación lógica OR con la máscara 0x00AC. Muestre el resultado por pantalla.

   Un ejemplo de este ejercicio se muestra a continuación:

   Sean:

   A = 10 y B = 12

   Entonces:

   P = A x B = 120

   D = P – 10 = 110

   Q = D/15 = 110/15 = 7

   Q1 = Q – 2 = 7 – 2 = 5

   R = D % 15 = 110 % 15 = 5

   R1 = R + 8 = 5 + 8 = 13

   S = Q1 + R1 = 5 + 13 = 18 = 0x0012

   L = S | 0x00AC = 0x0012 | 0x00AC = **0x00BE = 190**

VIII. EVALUACION

La evaluación de las actividades realizadas en la presente guía de práctica se hará en función de la siguiente tabla:

ACTIVIDAD	SESION 01
	Procedimental
Resolución del ejercicio propuesto 01	10
Resolución del ejercicio propuesto 02	10
TOTAL	**20**

IX. BIBLIOGRAFIA

1. Brey Barry. "*Los Microprocesadores Intel. Arquitectura, Programación e Interfaces*". Prentice Hall 3Ed.
2. Carter P. "PC Assembly Language" 2005
3. Hyde Randall. "*Art of Assembly Language Programming*". Nostarch Press 1Ed.
4. Smith B. E., Johnson M. T. "*Programming the Intel 80386*" Editorial Scott, Foresman And Company, 1987.
5. http://es.wikipedia.org/wiki/Assembly
6. http://nasm.sourceforge.net/
7. http://docs.cs.up.ac.za/programming/asm/derick_tut/
8. http://webster.cs.ucr.edu/Page_Linux/LinuxSysCalls.pdf