I. TEMA: PROGRAMACION EN ENSAMBLADOR – BUCLES

II. OBJETIVOS DE LA PRACTICA

Al finalizar la presente práctica, el estudiante:

1. Entiende la lógica de funcionamiento de las instrucciones de bifurcación de los procesadores con arquitectura X86.
2. Implementa, utilizando el lenguaje ensamblador de los procesadores con arquitectura X86, programas con estructuras de control iterativas.
3. 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 con arquitectura X86.

IV. MATERIALES.

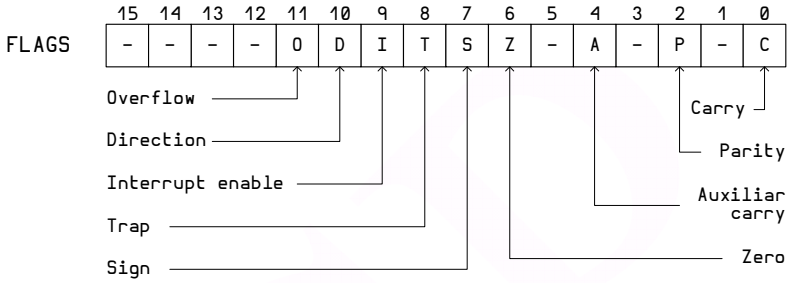
1. Sistema operativo Linux.
2. Compilador NASM.
3. Librería io.mac.
4. Editor de texto o IDE

V. MARCO TEORICO

REGISTRO DE BANDERAS X86

El registro de banderas en los procesadores 8086, almacena información sobre algunas condiciones que pueden presentarse como resultado de las instrucciones que se ejecutan y también sobre condiciones que permiten controlar el modo de operación del procesador.

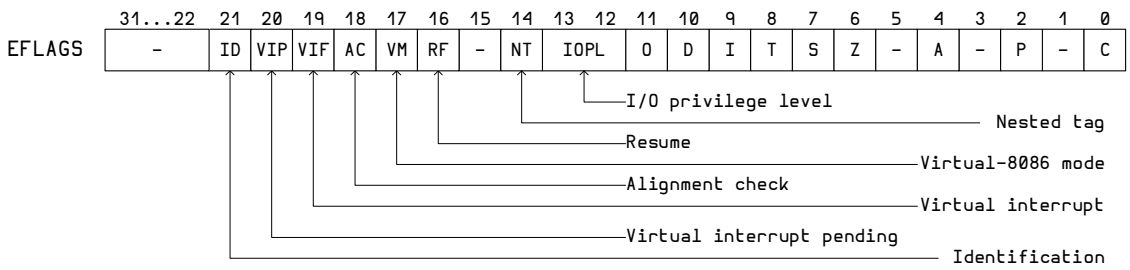
El registro de banderas (Flags Register) del procesador 8086, conocido también como palabra de estado del procesador (PSW – Processor State Word) tiene el siguiente formato:



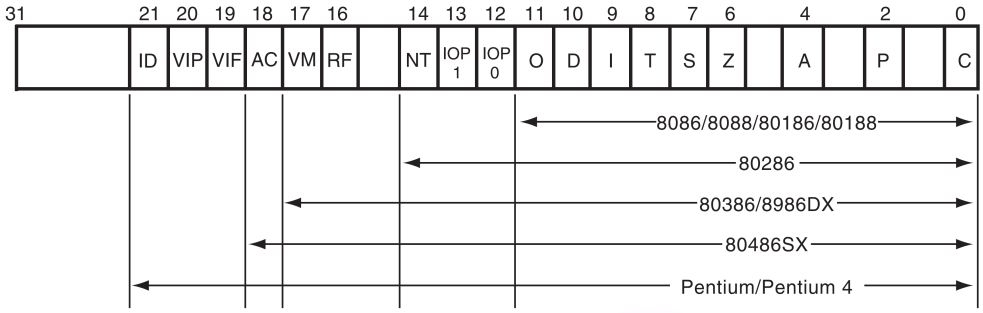
El propósito de cada bandera o indicador es:

O (Desbordamiento)	-	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 requiere atención del CPU.
T (Trampa)	-	Se utiliza para tareas de depuración de programas.
S (Signo)	-	Se modifica de acuerdo al 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.
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.

En los procesadores de Intel actuales el registro de banderas (EFLAG) incluye banderas para dar soporte a capacidades requeridas por aplicaciones modernas:



Estas banderas se han ido agregando en cada uno de los desarrollos de procesadores Intel, como puede apreciarse en la figura tomada del texto de Barry Brey ("*The Intel Microprocessors*" 8va edición)



INSTRUCCIONES DE SALTO

La familia de microprocesadores X86, incluye un gran número de instrucciones de salto. El propósito de estas es permitir que se ejecute una u otra secuencia de instrucciones, dependiendo de condiciones resultantes de la ejecución de instrucciones previas, las cuales son mantenidas en el registro de banderas del procesador.

Una instrucción de salto sigue la siguiente sintaxis

```
Jxx Destino
```

En donde:

Jxx **–** Es la instrucción de salto. En este caso las "x" representan las variaciones de esta instrucción (basadas en la condición que verifican)

Destino– La dirección a donde se saltará en caso que el salto sea efectivo, caso contrario, se ejecutará la instrucción que sigue a `Jxx`.

Los saltos pueden ser incondicionales o condicionales.

Instrucción de salto incondicional.

Esta instrucción se utiliza para saltar, independientemente de cualquier condición. La sintaxis de esta instrucción es la siguiente:

JMP *dest*	Operación: Salta incondicionalmente a *dest*.

Instrucciones de salto condicional

Son instrucciones de salto que verifican alguna condición en el sistema, y dependiendo de la misma, realizan o no el salto.

Estas instrucciones pueden agruparse en:

1. Saltos basados en el valor de una bandera aritmética única

JC *etiqueta*	Saltar si hubo arrastre/préstamo (CF = 1).
JNC *etiqueta*	Saltar si no hubo arrastre/préstamo (CF = 0).
JZ *etiqueta*	Saltar si el resultado es cero (ZF = 1).
JNZ *etiqueta*	Saltar si el resultado no es cero (ZF = 0).
JS *etiqueta*	Saltar si el signo es negativo (SF = 1).
JNS *etiqueta*	Saltar si el signo es positivo (SF = 0).
JP/JPE *etiqueta*	Saltar si la paridad es par (PF = 1).
JNP/JPO *etiqueta*	Saltar si la paridad es impar (PF = 0).

2. Saltos basados en comparaciones sin signo

JB *etiqueta*/**JNAE** *etiqueta*	Saltar a *etiqueta* si es menor.
JBE *etiqueta*/**JNA** *etiqueta*	Saltar a *etiqueta* si es menor o igual.
JE *etiqueta*	Saltar a *etiqueta* si es igual.
JNE *etiqueta*	Saltar a *etiqueta* si es distinto.
JAE *etiqueta*/**JNB** *etiqueta*	Saltar a *etiqueta* si es mayor o igual.
JA *etiqueta*/**JNBE** *etiqueta*	Saltar a *etiqueta* si es mayor.

3. Saltos basados en comparaciones con signo

JL *etiqueta*/**JNGE** *etiqueta*	Saltar a *etiqueta* si es menor.
JLE *etiqueta*/**JNG** *etiqueta*	Saltar a *etiqueta* si es menor o igual.
JE *etiqueta*	Saltar a *etiqueta* si es igual.
JNE *etiqueta*	Saltar a *etiqueta* si es distinto.
JGE *etiqueta*/**JNL** *etiqueta*	Saltar a *etiqueta* si es mayor o igual.
JG *etiqueta*/**JNLE** *etiqueta*	Saltar a *etiqueta* si es mayor.

Instrucción de comparación

Permite actualizar los bits del registro de banderas del microprocesador, de manera que luego estas banderas sean utilizadas por una instrucción de salto posterior

CMP *dest,src*	Operación: *dest - src (sólo afecta flags del registro de banderas).*

IMPLEMENTACIÓN DE ESTRUCTURAS DE CONTROL ITERATIVAS

Las estructuras de control iterativas, son aquellas que permiten la ejecución reiterada de un conjunto de instrucciones, mientas cierta condición no se cumpla.

En ensamblador, estas estructuras se implementan mediante saltos hacia atrás de la posición actual del contador de programas.

Las instrucciones de control de alto nivel tales como las estructuras **`while`** y **`for`** se convierten, después del proceso de compilación en una secuencia de instrucciones ensamblador, todas basadas en instrucciones de bifurcación como las listadas líneas arriba.

A continuación, veremos un ejemplo de esas equivalencias, aclarando que estas no son la única alternativa de correspondencia.

Estructura WHILE

Una instrucción iterativa, que en el lenguaje de programación C se expresa de la siguiente forma:

```
while ( N > 0) {
    acc = acc + N;
    N = N - 1;
}
```

Sería equivalente al siguiente código en ensamblador:

```
dest1:
    add dx,ax    ;   dx es el acumulador
    dec ax       ;   ax es N
    jnz dest1    ;   si N es diferente de cero, saltar
```

Estructura FOR

Un bloque ***`for`*** del lenguaje de programación C tiene la siguiente forma:

```
for (i = 1; i <= 10; i++) {
    acc = acc + i;
}
```

Una implementación equivalente en ensamblador puede ser:

```
  xor ax, ax         ;   ax es el acumulador
  mov cx, 10         ;   cx es el contador

repetir:
  add ax, cx
  loop repetir  ;   cx-- y si cx = 0; termina el bucle
```

VI. TRABAJO DE LABORATORIO.

1. Escriba un programa que muestre la suma de los N primeros números enteros. N debe ser indicado por el usuario

Solución

```
;Nombre     :    nPrimeros.asm
;Proposito  :    calcula la suma de los N primeros enteros
;Autor      :    Edwin Carrasco
;FCreacion  :    13/04/2008
;FModif.    :    11/09/2019
;compilar   :    nasm -f elf nPrimeros.asm
;                ld -m elf_i386 -s -o nPrimeros nPrimeros.o io.o

%include "io.mac"

section .data
   mensaje: db "ESTE PROGRAMA CALCULA LA SUMA DE LOS N RIMEROS
              ENTEROS",10,0
   entrada: db "Ingrese el valor para N: ",0
   salida:  db "La suma es: ",0

section .text
        global   _start

_start:
        PutStr   mensaje  ;indicar que hace el programa
        PutStr   entrada  ;leer datos
        GetInt   ax       ;datos en registro AX
        xor dx, dx        ;Inicializar acumulador

dest1:
        add dx, ax        ;sumar i-esimo entero al acumulador
        dec ax            ;los numeros se suman en orden
                          decreciente
        jnz dest1         ;    todavia hay datos?

        PutStr salida     ;    mostrar resultados
        PutInt dx         ;    DX a pantalla
        nwln

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

```
root@aar:/home/ecp/codigo/AC# nasm -f elf nPrimeros.asm
root@aar:/home/ecp/codigo/AC# ld -m elf_i386 -s -o nPrimeros nPrimeros.o io.o
root@aar:/home/ecp/codigo/AC# ./nPrimeros
ESTE PROGRAMA CALCULA LA SUMA DE LOS N PRIMEROS ENTEROS
Ingrese el valor para N: 100
La suma es: 5050
```

2. Escriba un programa que lea 3 notas de un alumno, calcule su promedio y de acuerdo a este indique si el alumno esta reprobado, desaprobado, aprobado o es sobresaliente. Las notas deben ingresarse en tiempo de ejecución. El mensaje que se debe mostrar según el promedio se india en la tabla adjunta:

NOTA	MENSAJE
0 – 6	Reprobado
7 – 10	Desaprobado
11 – 16	Aprobado
17 – 20	Sobresaliente

Solución

```
;Nombre     :    promedioNotas.asm
;Proposito  :    Calcula el promedio de notas de un alumno y
                 muestra un mensaje acorde:
;                     0-6      : Reprobado
;                     7-10     : Desaprobado
;                     11-16    : Aprobado
;                     17-20    : Sobresaliente
;Autor      :    Edwin Carrasco
;FCreacion  :    28/07/2008
;FModific.  :    19/07/2019
;compilar   :    nasm -f elf promedioNotas.asm
;Enlazar    :    ld -m elf_i386 -s -o promedioNotas
                 promedioNotas.o io.o
;Ejecutar   :    ./promedioNotas


%include "io.mac"

section .data
    msgProposito     db "ESTE PROGRAMA CALCULA EL PROMEDIO DE
                         03 NOTAS DE UN ALUMNO",10,0
    msgPedirDatos    db "Ingrese la ",0
    msgPedirNota     db " nota del estudiante: ",0
    msgReprobado     db " - Reprobado",10,0
    msgDesaprobado   db " - Desaprobado",10,0
    msgBueno         db " - Aprobado", 10,0
    msgSobresaliente db " - Sobresaliente", 10,0
    msgPromedio      db "Promedio: ",0

section .bss
    promedio resw 1

section .text
    global _start

_start:
    xor bx, bx
    mov cx,1
    mov dx, 0
```

```
    PutStr msgProposito

    ;Leer notas
leer_nota:
    PutStr msgPedirDatos
    PutInt cx
    PutStr msgPedirNota
    GetInt ax
    add bx, ax
    inc cx
    cmp cx,4
    jl leer_nota

    ;Calcular promedio
    mov ax, bx
    dec cx
    div cx
    mov [promedio], ax

    ;Mostrar nota
    PutStr msgPromedio
    PutInt [promedio]
    ;nwln

    ;Mostrar comentario
    cmp ax,6
    jg no_reprobo
    PutStr msgReprobado
    jmp final

no_reprobo:
    cmp ax,10
    jg no_desaprobo
    PutStr msgDesaprobado
    jmp final

no_desaprobo:
    cmp ax,16
    jg sobresaliente
    PutStr msgBueno
    jmp final

sobresaliente:
    PutStr msgSobresaliente

final:
    mov ax,1
    int 80h
```

Verificamos la funcionalidad de nuestro programa:

```
root@aar:/home/ecp/codigo/x86# nasm -f elf promedioNotas.asm
root@aar:/home/ecp/codigo/x86# ld -m elf_i386 -s -o promedioNotas promedioNota
s.o io.o
root@aar:/home/ecp/codigo/x86# ./promedioNotas
ESTE PROGRAMA CALCULA EL PROMEDIO DE 03 NOTAS DE UN ALUMNO
Ingrese la 1 nota del estudiante: 13
Ingrese la 2 nota del estudiante: 15
Ingrese la 3 nota del estudiante: 17
Promedio: 15 - Aprobado
root@aar:/home/ecp/codigo/x86#
```

VII. PRACTICAS DE LABORATORIO

1. Escriba un programa iterativo que determine si un número entero positivo N (N < 1000), ingresado por el usuario pertenece a la serie de Fibonacci (1, 1, 2, 3, 5, 8, 13, …). Por ejemplo:

   N = 5 : Pertenece a la serie de Fibonacci.
   N = 7 : No pertenece a la serie de Fibonacci.

2. Escriba un programa iterativo que determine si un número entero positivo N, ingresado por el usuario, es abundante.
   Un número entero positivo se denomina abundante, si este es menor a la suma de sus divisores propios. Por ejemplo:

   12 : Es un número abundante, porque 12 < 1 + 2 + 3 + 4 + 6.
   15 : No es un número abundante, porque 15 > 1 + 3 + 5.

3. Escriba un programa iterativo que determine si un número entero positivo N, ingresado por el usuario, es perfecto.
   Un número entero positivo se denomina perfecto, si este es igual a la suma de sus divisores propios. Por ejemplo:

   06 : Es un número perfecto, porque 6 = 1 + 2 + 3.
   28 : Es un número perfecto, porque 28 = 1 + 2 + 4 + 7 + 14.
   14 : No es un número perfecto, porque 14 ≠ 1 + 2 + 7.

VIII. EVALUACION

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

ACTIVIDAD	Procedimental	
	Sesión 01	Sesión 02
Resolución del ejercicio propuesto 01	--	06
Resolución del ejercicio propuesto 02	--	07
Resolución del ejercicio propuesto 03	--	07
TOTAL	**--**	**20**

IX. BIBLIOGRAFIA

1. Brey B. "*Los Microprocesadores Intel. Arquitectura, Programación e Interfaces*". Prentice Hall 3Ed.
2. Brey B "*The Intel Microprocessors. Architecture, Programming, and Interfacing*" Prentice Hall 8ed.
3. Carter P. "*PC Assembly Language*" 2005
4. Hyde R. "*Art of Assembly Language Programming*". Nostarch Press 1Ed.