Introducción:

Este manual tiene como fin ayudar a comprender los aspectos técnicos del programa para poder facilitar el uso y funcionamiento del mismo, a continuación se explica, el ensamblador utilizado para esta práctica, los requisitos del ensamblador y la forma de poder ejecutar programas utilizando este ensamblador

Descripción del Emulador:

NASM:

El Netwide Assembler o Nasm, es un ensamblador libre para la plataforma Intel x86. Puede ser usado para escribir programas tanto de 16 bits como de 32 bits (IA-32). En el NASM, si se usan las bibliotecas correctas, los programas de 32 bits se pueden escribir de una manera tal para que sean portables entre cualquier sistema operativo x86 de 32 bits. El paquete también incluye un desensamblador, el NDISASM.

Historia:

El NASM fue escrito originalmente por Simon Tatham con ayuda de Julian Hall, y actualmente es desarrollado por un pequeño equipo en SourceForge que le hace mantenimiento. Fue lanzado originalmente bajo su propia licencia, pero más adelante fue cambiada por la licencia GNU Lesser General Public License, seguido de un número de problemas políticos causado por la selección de la licencia. La próxima versión del NASM, la 2.00, actualmente está siendo desarrollada bajo la bifurcación 0.99, e incluirá soporte para el x86-64 (x64/AMD64/Intel 64), junto con la respectiva salida de archivo objeto de 64 bits.

Características:

- El NASM puede generar varios formatos binarios en cualquier máquina, incluyendo COFF (y el ligeramente diferente formato Portable Executable usado porMicrosoft Windows), el a.out, ELF, Mach-O, y el formato binario nativo Minix. El NASM incluso define su propio formato binario, RDOFF, que es usado actualmente solamente por el proyecto del sistema operativo RadiOS).
- La variedad de formatos de la salida permite a uno portar los programas a virtualmente cualquier sistema operativo x86. Además, el NASM puede crear archivos binarios planos, usables para escribir boot loaders (cargadores de arranque), imágenes ROM, y varias facetas del desarrollo sistemas operativos. El NASM incluso puede correr en plataformas diferentes del x86, como SPARC y PowerPC, aunque no puede producir programas usables por esas máquinas.
- El NASM usa la tradicional sintaxis de Intel para el lenguaje ensamblador x86, mientras que otros ensambladores libres, como el ensamblador del GNU (GAS), utilizan la sintaxis de AT&T. También evita características como la generación automática de sobreescritura (override) de segmentos y la relacionada directiva *ASSUME* usada por el MASM y los ensambladores compatibles, pues estas pueden ser a menudo confusas -- los programadores deben seguir por sí mismos el contenido de los registros de segmento y la localización de variables a los que éstos se refieren

Ejemplo Comando de Nasm:

```
nasm -f elf Practica2.asm
ld -m elf_i386 -s -o practica Practica2.o
./practica
```

Ejemplo Comando de Ejecución de NASM para DosBox

```
nasm Practica2.asm -fbin -o practica.com
dosbox ./practica.com -exit
```

DosBox:
Es un emulador que recrea un entorno similar al sistema DOS con el objetivo de poder ejecutar programas y videojuegos originalmente escritos para el sistema operativo MS-DOS de Microsoft en ordenadores más modernos o en diferentes arquitecturas (Como Power PC). También permite que estos juegos funcionen en otros sistemas operativos como GNU/Linux. Fue porque Windows XP ya no se bas en MS-DOS y pasó a basarse a Windows NT.

Descripción del Sistema Operativo:

Sistema Operativo utilizado: Debian 8.5 - Jessie

Debian: Debian o Proyecto Debian (en inglés: Debian Project) es una comunidad conformada por desarrolladores y usuarios, que mantiene un sistema operativo GNU basado en software libre. El sistema se encuentra precompilado, empaquetado y en formato deb para múltiples arquitecturas de computador y para varios núcleos.
El proyecto Debian fue anunciado inicialmente 1993 por Ian Murdock. El nombre Debian proviene de la combinación del nombre de su entonces novia Deborah y el suyo, por lo tanto, Deb (orah) e Ian. Debian 0.01 fue lanzado el 15 de septiembre de 1993, y la primera versión estable fue hecha en 1996.

Requisitos del sistema:

Tipo de instalación	RAM (mínimo)	RAM (recomendado)	Disco duro
Sin escritorio	128 Megabytes	512 Megabytes	2 Gigabytes
Con escritorio	256 Megabytes	1 Gigabyte	10 Gigabyte

Requerimientos de la aplicación:

1. Tener instalado gcc (Compilador de C)
2. Tener instalado dosbox (Segunda forma de poder compilar el programa)
3. Una computadora de 64 bits
4. Un sistema operativo base linux de 64 bits

Pasos para poder compilar y ejecutar un programa:

1. Generar el archivo .asm
2. Compilar el archivo .asm con el siguiente código: **nams -f elf64 NombreArchivo.asm**, este comando traducirá el código en el archivo .asm a un código binario que el procesador pueda entender, como resultado de la ejecución de este código se tendrá un archivo .o donde estará el código binario que el procesador entiende.
3. Generar el ejecutable en base al archivo .o generado en el paso anterior, para esto se introduce lo siguiente en consola: **ld NombreEjecutable NombreArchivo.o**
4. Ejecutar el ejecutable generador el paso anterior se hará de la siguiente forma: **./NombreEjecutable**
5. En consola se podrá ver el resultado del programa escrito

Pasos para poder comilar y ejecutar un programa con dosbox:

1. Generar el archivo .asm
2. Compilar el .asm con el siguiente código: **nasm NombreArchivo.asm -fbin -o NombreCom.com,** este comando traducirá el .asm a código binario entendible por dosbox, como resultado de la ejecución se genera un archivo .com que será el archivo que se compilara en dosbox
3. Ejecutar el siguiente comando en la ruta donde se haya generado el archivo .com: **dosbox ./NombreCom.com -exit**, esto hará que dosbox compile el .com y ejecute el programa, el -exit es para indicarle a dosbox que cuando termine la ejecución se cierre
4.

Código de la solución:

```
;Juan Ramón Veleche Brán
;201314076
;Debian 8.4 -Jessie


org 100h

section .data
	;cadena de caracteres a imprimir en consola
	header: db 0ah,"*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-",0,0ah,0ah,"Universidad de San Carlos de Guatemala",0,0ah,"Faculta de Ingenieria",0,0ah,"Arquitectura de Computadores y Ensambladores 1",0,0ah, "Segundo Semestre 2016",0,0ah,"Seccion A",0,0ah, "Juan Ramon Veleche Bran",0,0ah,"201314076$",0,0ah,0ah
	cad1: db 0ah,0ah,"*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-$",0,0ah,0ah,0ah,24h
	menu: db 0ah,0ah,9h,"1) Modo Calculadora",0ah,0dh,9h,"2) Salir",0ah,0dh,24h
	menu2: db 0ah,0ah,9h,"1) Ingresar operacion",0ah,0dh,9h,"2) Serie Fibonacci",0ah,0dh,9h,"3) Salir",0ah,0dh,24h
	MsgNum: db "Numero: $",0ah,24h
	MsgOp: db "Operador aritmetico: $",0ah,24h
	MsgResultado: db "Resultado: $",0ah,24h
	MsgNegativo: db "es negativo :)$",0ah,24h
	MsgIngresar: db "Ingrese numero entreo 0-19 para calcular el Fibonacci: $",0ah,0dh,24h
	MsgError: db 0ah,0dh,"Numero mayor a 19$",0ah,0dh,24h
	MsgFibonacci: db "Fibonacci de: $",0ah,24h
	MsgIgual: db " = $",0ah,24h
	MsgSignoMas: db  " + $",0ah,24h
	Uno: db "01$",0ah,24h
	Cero: db "00$",0ah,24h
	Signo: db 2Bh,'$'
	Operador: db '0','$'
	Unidad: db '0','$'
	Decena: db '0','$'
	Centena: db '0','$'
	Mil: db '0','$'
	Mas: db '+','$'
	Menos: db '-','$'
	SignoNum1: db 2Bh,'$'
	SignoNum2: db 2Bh,'$'
	SignoResult: db 2Bh,'$'
	uniTot: db 0
```

```
        cenTot: db 0
        decTot: db 0
        milTot: db 0
        SaltoLinea: db 0Ah,'$'
        Numero1: dw 0
        Numero2: dw 0
        Resultado: dw 0
        AuxResultado: dw 0
        Fn1: dw 0
        Fn2: dw 0
        Rfb: dw 0
        aux1: dw 0
        aux2: dw 0
        ValorAux: dw 0


section .text
        global _start

_start:
        mov ah,00h              ; limpiar pantalla
        mov al,03h
        int 10h                 ;llame a la interrupción de video

        mov ah,09h              ;activar la opción de impresión en pantalla
        mov dx,header   ;deposita en dx la cadena a imprimir
        int 21h                 ;interrupción al kernel para la impresión de la cadena

        mov ah,09h
        mov dx,cad1
        int 21h

        mov ah,09h
        mov dx,menu
        int 21h

        call leer_opcion
ret

leer_opcion:

        mov ah,08h
        int 21h

        cmp al,31h
        je imprimir_menu2
```

```
        cmp al,32h
        je Salir
        jmp _start
ret

imprimir_menu2:

        mov ah,00h
        mov al,03h
        int 10h

        mov ah,09h
        mov dx,header
        int 21h

        mov ah,09h
        mov dx,cad1
        int 21h

        mov ah,09h
        mov dx,menu2
        int 21h

        call leer_opcion2

ret

leer_opcion2:

        mov ah,08h
        int 21h

        cmp al,31h
        je ingresar_operacion
        cmp al,32h
        je Serie_Fibonacci
        cmp al,33h
        je _start
        jmp imprimir_menu2
ret

ingresar_operacion:

        mov ah,00h
        mov al,03h
        int 10h
```

```
mov ah,09h
mov dx,header
int 21h

mov ah,09h
mov dx,cad1
int 21h

mov ah,9h
mov dx, MsgNum
int 21h

call recibir_numero

mov ax,[Signo]
mov [SignoNum1],ax

call AsciiToDecimalNum1

loop_recibir:

        mov ah,9h
        mov dx,MsgOp
        int 21h

        call recibir_operador

        mov ah,9h
        mov dx, MsgNum
        int 21h

        mov ax,2bh
        mov [Signo],ax

        call recibir_numero

        mov ax,[Signo]
        mov [SignoNum2],ax

        call AsciiToDecimalNum2

        call HacerOperacion

        mov ax,[Resultado]
        mov [Numero1],ax
```

```
                mov ax,[SignoResult]
                mov [SignoNum1],ax

                jmp loop_recibir

        mov ah,8h
        int 21h
ret

Serie_Fibonacci:

        xor ah,ah
        xor dx,dx

        mov ah,00h              ; limpiar pantalla
        mov al,03h
        int 10h                 ;llame a la interrupción de video

        mov ah,09h              ;activar la opción de impresión en pantalla
        mov dx,header   ;deposita en dx la cadena a imprimir
        int 21h                 ;interrupción al kernel para la impresión de la cadena

        mov ah,09h
        mov dx,cad1
        int 21h

        mov ah,09h
        mov dx,MsgIngresar
        int 21h

        mov ah,8h
        int 21h

        mov [Decena],al

        mov ah,9h
        mov dx,Decena
        int 21h

        mov ah,8h
        int 21h

        mov [Unidad],al

        mov ah,9h
```

```
mov dx,Unidad
int 21h

mov ah,8h
int 21h

call AsciiToDecimalNum1

mov ax,19d
sub ax,[Numero1]
jc Error_Fibonacci
jmp Iniciar


Error_Fibonacci:

        mov ah,9h
        mov dx,MsgError
        int 21h

        mov ah,8h
        int 21h

        jmp imprimir_menu2
ret

Iniciar:
        mov ah,9h
        mov dx,SaltoLinea
        int 21h

        mov ah,9h
        mov dx,MsgFibonacci
        int 21h

        mov ah,9h
        mov dx,Cero
        int 21h

        mov ah,9h
        mov dx,MsgIgual
        int 21h

        mov ah,9h
        mov dx,Cero
        int 21h
```

```
                mov ah,9h
                mov dx,SaltoLinea
                int 21h

                mov ah,9h
                mov dx,MsgFibonacci
                int 21h

                mov ah,9h
                mov dx,Uno
                int 21h

                mov ah,9h
                mov dx,MsgIgual
                int 21h

                mov ah,9h
                mov dx,Uno
                int 21h

                mov ah,9h
                mov dx,SaltoLinea
                int 21h

                call Calcular_Fibonacci

                mov ah,8h
                int 21h
        jmp imprimir_menu2
ret

Calcular_Fibonacci:

        xor ah,ah
        xor dx,dx

        mov ax,0d
        mov [Fn2],ax

        mov ax,1d
        mov  [Fn1],ax

        mov cx,2d

        Inicio_Loop:
```

```
xor ah,ah
xor ax,ax

mov ax,[Fn1]   ;ax = Fibonacci n-1
mov [aux1],ax ;paso el valor a la variable auxiliar para imprimirlo

mov ax,[Fn2]   ;ax = Fibonacci n-2
mov [aux2],ax ;paso el valor a la variable auxiliar para imprimirlo

mov ax,[Fn1]   ;ax = Fbionacci n-1
add ax,[Fn2]   ;ax = ax + Fibonacci n-2
mov [Rfb],ax   ;gurado el valor del Fibonacci en la variable

mov ax,[Fn1]   ; Fibonacci n-2 = Fibonacci n-1
mov [Fn2],ax

mov ax,[Rfb]   ;Fibonacci n-1 = Fibonacci n
mov [Fn1],ax

mov ah,9h
mov dx,MsgFibonacci          ; imprimir en pantalal el mensaje
int 21h

mov ax,cx
mov [Resultado],ax           ;para poder imprimir el resultado

call DecimalToAscii

mov ah,9h
mov dx,MsgIgual              ;imprimir en pantalla el mensaje
int 21h

mov ax,[aux1]
mov [Resultado],ax

call DecimalToAscii

mov ah,9h
mov dx,MsgSignoMas
int 21h

mov ax,[aux2]
mov [Resultado],ax

call DecimalToAscii
```

```
                mov ah,9h
                mov dx,MsgIgual
                int 21h

                mov ax,[Rfb]
                mov [Resultado],ax

                call DecimalToAscii

                mov ah,9h
                mov dx,SaltoLinea
                int 21h

                mov ax,[Numero1]
                sub ax,1d
                sub ax,cx                       ;condicon de salida

                inc cx

                jnc Inicio_Loop
ret

recibir_numero:

        mov ah,08h
        int 21h

        cmp al,2Dh      ;menos
        je esSigno
        cmp al,2Bh      ;mas
        je esSigno
        cmp al,71h
        je Salir_Numero
        jmp esNumero

        esSigno:
                cmp al,2dh
                je Imprimir_menos
                jmp Imprimir_mas

                Imprimir_menos:
                        mov al,2dh
                        mov [Signo],al
                        mov ah,9h
                        mov dx,Menos
                        int 21h
```

```
                jmp Seguir

        Imprimir_mas:
                mov al,2bh
                mov [Signo],al
                mov ah,9h
                mov dx,Mas
                int 21h

        Seguir:
                mov ah,08h
                int 21h

    esNumero:
        mov [Decena],al

        mov ah,9h
        mov dx,Decena
        int 21h

        mov ah,08h
        int 21h

        mov [Unidad],al

        mov ah,9h
        mov dx,Unidad
        int 21h

        mov ah,8h
        int 21h

        mov ah,9h
        mov dx,SaltoLinea
        int 21h
    ret

    Salir_Numero:

        mov ah,8h
        int 21h

        jmp imprimir_menu2
ret

recibir_operador:
```

```
        mov ah,08h
        int 21h

        cmp al,71h
        je Salir_Op
        jmp Continuar

        Salir_Op:
                mov ah,8h
                int 21h

                jmp imprimir_menu2

        Continuar:
                mov [Operador],al

                mov ah,9h
                mov dx,Operador
                int 21h

                mov ah,8h
                int 21h

                mov ah,9h
                mov dx,SaltoLinea
                int 21h
ret

HacerOperacion:

        mov al,[Operador]
        cmp al,2BH     ;suma
        je HacerSuma
        cmp al,2DH     ;resta
        je HacerResta
        cmp al,2AH     ;multiplicacion
        je HacerMultiplicacion
        cmp al,2FH     ;division
        je HacerDivison
ret

HacerSuma:

        mov ah,9h
        mov dx, MsgResultado
```

```
int 21h

mov al,[SignoNum1]
mov bl,[SignoNum2]
cmp al,2bh
je andPositivo_Suma
cmp al,2dh
je andNegativo_Suma

andPositivo_Suma:
        cmp bl,2bh
        je Signos_Iguales_Suma
        jmp Signos_Distintos_Positivo_Suma

andNegativo_Suma:
        cmp bl,2dh
        je Signos_Iguales_Suma
        jmp Signos_Distintos_Negativo_Suma

Signos_Iguales_Suma:
        cmp al,2bh
        je Sumar_Normal
        jmp Sumar_Negativos

Signos_Distintos_Positivo_Suma:

        mov ax,[Numero1]
        sub ax,[Numero2]
        mov [Resultado],ax
        jc esNegativo_Suma
        jmp mostrarResultado_Suma

Signos_Distintos_Negativo_Suma:
        mov ax,[Numero2]
        sub ax,[Numero1]
        mov [Resultado],ax
        jc esNegativo_Suma
        jmp mostrarResultado_Suma

esNegativo_Suma:

        mov ah,9h
        mov dx,Menos
        int 21h

        mov ax,[Resultado]
```

```
                neg ax
                mov [Resultado],ax

                mov dx,2DH
                mov [SignoResult],dx
                jmp mostrarResultado_Suma

        Sumar_Normal:

                mov ax,[Numero1]
                add ax,[Numero2]
                mov [Resultado],ax

                mov dx,2bh
                mov [SignoResult],dx
                jmp mostrarResultado_Suma

        Sumar_Negativos:

                mov ah,9h
                mov dx,Menos
                int 21h

                mov ax,[Numero1]
                add ax,[Numero2]
                mov [Resultado],ax

                mov bx,2dh
                mov [SignoResult],bx

        mostrarResultado_Suma:

                call DecimalToAscii

                mov ah,9h
                mov dx,SaltoLinea
                int 21h
ret

HacerResta:

        mov ah,9h
        mov dx, MsgResultado
        int 21h

        mov al,[SignoNum1]
```

```
mov bl,[SignoNum2]
cmp al,2dh
je andNum
cmp al,2bh
je andPositivo

andNum:
        cmp bl,2dh
        je Signos_Iguales
        jmp Signos_Distintos_Negativo

andPositivo:
        cmp bl,2bh
        je Signos_Iguales
        jmp Signos_Distintos_Positivo

Signos_Distintos_Positivo:
        mov ax,[Numero1]
        mov bx,[Numero2]
        add ax,bx
        mov [Resultado],ax
        mov dx,2bh
        mov [SignoResult],dx
        jmp mostrarResultado

Signos_Distintos_Negativo:
        mov ax,[Numero1]
        mov bx,[Numero2]
        add ax,bx
        mov [Resultado],ax
        mov dx,2dh
        mov [SignoResult],dx
        mov ah,9h
        mov dx,Menos
        int 21h
        jmp mostrarResultado

Signos_Iguales:
        cmp al,2dh
        je Sumar_Op2
        cmp al,2bh
        je Restar_Normal

Sumar_Op2:
        mov ax,[Numero1]
        mov dx,[Numero2]
```

```
        sub ax,dx
        mov [Resultado],ax
        jc esPositivo
        mov ah,9h
        mov dx,Menos
        int 21h
        mov dx,2dh
        mov [SignoResult],dx
        jmp mostrarResultado

Restar_Normal:

        mov ax,[Numero1]
        sub ax,[Numero2]
        mov [Resultado],ax
        jc esNegativo
        mov dx,2bh
        mov [SignoResult],dx
        jmp mostrarResultado

esNegativo:

        mov ah,9h
        mov dx,Menos
        int 21h

        mov ax,[Resultado]
        neg ax
        mov [Resultado],ax

        mov dx,2DH
        mov [SignoResult],dx
        jmp mostrarResultado

esPositivo:

        mov ah,9h
        mov dx,Mas
        int 21h

        mov ax,[Resultado]
        neg ax
        mov [Resultado],ax

        mov dx,2bH
        mov [SignoResult],dx
```

```
        mostrarResultado:

                call DecimalToAscii

                mov ah,9h
                mov dx,SaltoLinea
                int 21h
ret

HacerMultiplicacion:
        mov ah,9h
        mov dx,MsgResultado
        int 21h

        mov al,[SignoNum1]
        mov bl,[SignoNum2]
        cmp al,bl
        je Signos_Iguales_Mult
        jmp Signos_Distintos_Mult

        Signos_Iguales_Mult:

                mov ax,[Numero1]
                mov bx,[Numero2]
                mul bx
                mov [Resultado],eax

                mov dx,2bh
                mov [SignoResult],dx

                jmp mostrarResultado_mult

        Signos_Distintos_Mult:

                mov ah,9h
                mov dx,Menos
                int 21h

                mov ax,[Numero1]
                mov bx,[Numero2]
                mul bx
                mov [Resultado],eax

                mov dx,2dh
                mov [SignoResult],dx
```

```
        mostrarResultado_mult:

                call DecimalToAscii

                mov ah,9h
                mov dx,SaltoLinea
                int 21h
ret

HacerDivison:
        mov ah,9h
        mov dx,MsgResultado
        int 21h

        mov al,[SignoNum1]
        mov bl,[SignoNum2]
        cmp al,bl
        je Signos_Iguales_Div
        jmp Signos_Distintos_Div

        Signos_Iguales_Div:

                xor dx,dx
                xor ax,ax
                xor bx,bx

                mov ax,[Numero1]
                mov bx,[Numero2]
                div bx
                mov [Resultado],ax

                mov dx,2bh
                mov [SignoResult],dx

                jmp mostrarResultado_div

        Signos_Distintos_Div:

                mov ah,9h
                mov dx,Menos
                int 21h

                xor dx,dx
                xor ax,ax
                xor bx,bx
```

```
                mov ax,[Numero1]
                mov bx,[Numero2]
                div bx
                mov [Resultado],ax

                mov dx,2dh
                mov [SignoResult],dx

        mostrarResultado_div:

                call DecimalToAscii

                mov ah,9h
                mov dx,SaltoLinea
                int 21h
ret

DecimalToAscii:
        ;se limpian los registors
        xor ah, ah
   xor al, al
   xor ax, ax
        xor dl,dl
        xor dx,dx

        mov ax,[Resultado]
        mov [AuxResultado],ax

        mov ax,[AuxResultado]
        sub ax,1000d
        jc MenorAMil
        jmp ConvertirMil

        MenorAMil:
                mov ax,[Resultado]
                sub ax,100d
                jc MenorACien
                jmp ConvertirCien

        MenorACien:
                xor dx,dx
                xor ax,ax
                xor bx,bx
                xor ah,ah
           ;decenas
```

```
    mov al,[AuxResultado]      ;se mueve ax el valor del dividendo
    mov bl,10d                 ;se mueve a bl el valor del divisor
    div bl                             ;se efectual la división


        mov [decTot], al       ;al almacena el valor del cociente
        mov [uniTot], ah       ;ah almacena el valor del residuo

        mov ah,02h                     ;macro para impresión
        mov dl, [decTot]       ;se mueve a dl el valor de las decenas
        add dl,30h                     ;se convierte la decena de decimal a ascii
        int 21h                        ;llamada al kernel

        mov ah, 02h
        mov dl, [uniTot]       ;se mueve a dl el valor de la unidad
        add dl,30h                     ;se convierte la unidad de deicmal a ascii
        int 21h                        ;llamda al kernel
ret

ConvertirMil:
        xor bl,bl
        xor ax,ax
        xor al,al
        xor ah,ah

        mov bx,1000d
        mov ax,[AuxResultado]
        div bx

        mov [milTot],al
        mov [AuxResultado],dx

        mov ah,02h
        mov dl,[milTot]
        add dl,30h
        int 21h

ConvertirCien:

        xor bl,bl
        xor ax,ax
        xor al,al
        xor ah,ah

        mov bl,100d
        mov ax,[AuxResultado]
```

```
                div bl

                mov [cenTot],al
                mov [AuxResultado],ah

                mov ah,02h
                mov dl,[cenTot]
                add dl,30h
                int 21h

                jmp MenorACien
ret

AsciiToDecimalNum1:

        mov dx,0d                       ;se mueve a dx el valor de 0 decimal
   mov [Numero1],dx  ;se inicializa la variable donde se guardara la conversión en cero

   ;limpieza de registros
        xor al,al
        xor bl,bl
        xor ax,ax

        mov al, 10                      ;se mueve a al 10 para sumar decenas
        mov bl, [Decena]        ;se obtiene el valor de la decena
        sub bl, 30h             ;se convierte el valor de la decena de ascii a decimal
        mul bl                          ;se multiplica el valor de la decena por 10
        add [Numero1], ax       ;se agrega el resultado a la variable

        mov bl, [Unidad]        ;se mueve a bl el valor de la unidad, por se unidad no es
necesario multiplicarlo por 10
        sub bl, 30h             ;se convierte el valor de la unidad de ascii a decimal
        add [Numero1], bl ;se suma el valo de la unidad al resulado
ret

AsciiToDecimalNum2:

        mov dx,0d                       ;se mueve a dx el valor de 0 decimal
   mov [Numero2],dx  ;se inicializa la variable donde se guardara la conversión en cero

   ;limpieza de registros
        xor al,al
        xor bl,bl
        xor ax,ax

        mov al, 10                      ;se mueve a al 10 para sumar decenas
```

```
        mov bl, [Decena]        ;se obtiene el valor de la decena
        sub bl, 30h             ;se convierte el valor de la decena de ascii a decimal
        mul bl                  ;se multiplica el valor de la decena por 10
        add [Numero2], ax       ;se agrega el resultado a la variable

        mov bl, [Unidad]        ;se mueve a bl el valor de la unidad, por se unidad no es necesario multiplicarlo por 10
        sub bl, 30h             ;se convierte el valor de la unidad de ascii a decimal
        add [Numero2], bl ;se suma el valo de la unidad al resulado
ret

Salir:
        mov ah,04ch             ;termina el programa
        int 21h                 ;llama el kernel para realizar la acción
ret
```