



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Práctica No.3

Introducción a los sistemas operativos Linux y
Windows (3)

Unidad de aprendizaje: Sistemas Operativos

Grupo: 2CM8

Integrantes del equipo:

Domínguez Morán Joaquín
Carrillo Balcazar Eduardo Yair
Ruiz López Luis Carlos

Profesor:

Jorge Cortes Galicia

8 de octubre de 2018

1. Competencias.

El alumno aprende a programar aplicaciones sencillas a nivel ensamblador bajo los sistemas operativos Linux y Windows utilizando la interfaz de interrupciones respectiva de cada sistema, mediante la comprensión de la estructura general e instrucciones para el lenguaje ensamblador del procesador Intel de 32 bits.

2. Desarrollo.

2.1. Linux

1. Inicie sesión en Linux.
2. Cree una carpeta llamada Borrador para que allí mantenga sus programas. Al finalizar su sesión de trabajo respalde sus programas en una memoria USB y elimine esta carpeta con su contenido.
3. Capture el siguiente código:

```
1 segment .data          ;segmento de datos
2 cadena db 'Progama en ensamblador para Linux',0xA ;cadena a imprimir
3
4 segment .text           ;Segmento de código
5 global _start           ;Punto de entrada al programa(usado en el enlazador ld)
6 _start:                 ;Inicio del programa
7 mov edx,38d             ;Longitud de cadena
8 mov ecx,cadena           ;Cadena a escribir
9 mov ebx,1               ;Salida estandar
10 mov eax,4              ;Numero de llamada al sistema "sys_write"
11 int 0x80               ;Interrupción de llamadas al sistema del kernel de Linux
12 mov eax,1              ;Numero de llamada al sistema "sys_write"
13 int 0x80               ;Interrupción de llamadas al sistema del kernel de Linux
```

4. Guarde su archivo con extensión .asm dentro de Borrador y ensamble desde una consola con el siguiente comando :
nasm -f elf -o nombre_archivo.o nombre_archivo.asm
5. Enlace el código objeto intermedio generado en el paso anterior con el siguiente comando:
ld -m elf_i386 -o nombre_ejecutable nombre_archivo.o
6. Al final de estos dos pasos obtendrá un programa ejecutable, pruebe el funcionamiento de su aplicación.

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ nasm -f elf -o 2.o 2.a
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ ld -m elf_i386 -o 2.2
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ ./2
Programa en ensamblador para Linux
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$

```

7. Capture el siguiente código:

```

1 segment .bss                                ;Segmento de datos
2 cadena resd 50                              ;Espacio en memoria para la cadena al almacenar
3
4 segment .text                                ;Segmento de código
5 global _start                               ;punto de entrada al programa(usado con el enlazador ld)
6 _start:                                     ;Inicio del programa
7     mov edx,50d                              ;Longitud del bufer
8     mov ecx,cadena                          ;Cadena a leer
9     mov ebx,0                                ;Entrada estándar
10    mov eax,3                                ;Número de llamada al sistema "sys_read"
11    int 0x80                                ;Interrupción de llamadas al sistema del kernel de Linux
12    mov edx,50d                              ;Longitud de cadena
13    mov ecx,cadena                          ;Cadena a escribir
14    mov ebx,1                                ;Salida estándar
15    mov eax,4                                ;Número de llamada al sistema "sys_write"
16    int 0x80                                ;Interrupción de llamadas al sistema del kernel de Linux
17    mov eax,1                                ;Número de llamadas al sistema "sys_exit"
18    int 0x80                                ;Interrupción de llamadas al sistema al kernel de Linux

```

8. Ensamble, enlace y ejecute esta aplicación.

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ nasm -f elf -o 7.o 7.asm
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ ld -m elf_i386 -o 7.7.o
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ ./7
hola mundo
hola mundo
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$

```

9. Programe una aplicación en ensamblador que genere un contador de 0 a 9, mostrando en pantalla el conteo generado.
10. Programe una aplicación en ensamblador que copie tres cadenas dadas (cadena1, cadena2, cadena3) a una nueva cadena (cadena4). La copia de las cadenas deberá ser intercalando los caracteres de cada cadena. Las cadenas cadena1, cadena2 y cadena3 deben ser ingresadas por teclado. Muestre en pantalla el contenido de la cadena4.

11. Programe una aplicación en ensamblador que muestre en pantalla la longitud de una cadena que haya sido ingresada por teclado. Considere el caso de que la cadena tenga 10 caracteres o más.
12. Programe una aplicación en ensamblador que concatene diez cadenas (cadena 1 hasta cadena 10) ingresadas por teclado, mostrando en pantalla lo siguiente : el contenido de la cadena concatenada, la cadena concatenada en sentido inverso y la longitud de dicha cadena.
13. Programe una aplicación en ensamblador que implemente una calculadora con las cuatro operaciones básicas. A través de un menú dé la posibilidad de seleccionar la operación a realizar. Maneje dígitos enteros positivos en el intervalo [0, 255].

2.2. Windows

1. Inicie sesión en Windows.
2. Cree una carpeta llamada Borrador para que allí mantenga sus programas. Al finalizar su sesión de trabajo respalde sus programas en una memoria USB y elimine esta carpeta con su contenido.
3. Capture el siguiente código:

```

1 segment .data
2     cadImprimir db
3     'Ensamblado en Windows',0xA           ;Segundo argumento para la
4                                           ;llamada al sistema _WriteConsoleA()
5 segment .bss
6     handleConsola resd 1                  ;Primer argumento para la
7                                           ;llamada al sistema _WriteConsoleA()
8     longitudCadena resd 1                ;Tercer argumento para la
9                                           ;llamada al sistema _WriteConsoleA()
10    caractEscritos resd 1                 ;Cuarto argumento para la
11                                           ;llamada al sistema _WriteConsoleA()
12    ultimoArgumento resd 1                ;Quinto argumento para la
13                                           ;llamada al sistema _WriteConsoleA()
14 segment .text
15 global _main
16 extern _GetStdHandle@4                   ;Acceso externo a la
17                                           ;llamada al sistema _GetStdHandle()
18 extern _WriteConsoleA@20                 ;Acceso externo a la
19                                           ;llamada al sistema _WriteConsoleA()
20 extern _ExitProcess@4                    ;Acceso externo a la
21                                           ;llamada al sistema _ExitProcess()
22 _main:
23     push dword-11                         ;Argumento pasado por la pila y
24                                           ;usado en _GetStdHandle() para la salida estandar
25     call _GetStdHandle@4                  ;Invocación de GetStdHandle()
26     mov [handleConsola],eax              ;Devolucion del manejador de
27                                           ;consola para escribir en el registro eax
28

```

```

29     xor  eax, eax                ;Limpieza del registro eax(eax=0)
30     mov  eax, 23d               ;eax=23 caracteres de
31                                     ;longitud de la cadena a imprimir
32     mov  [longitudCadena], eax   ;Se guarda la longitud en memoria
33     xor  eax, eax                ;Limpieza del registro eax(eax=0)
34     mov  eax, 0d                ;eax=0 valor del ultimo
35                                     ;argumento de _WriteConsoleA()
36     mov  [ultimoArgumento], eax  ;Se guarda el valor del
37                                     ;ultimo arguento en memoria
38
39     push dword [ultimoArgumento] ;Quito argumento de
40                                     ;_WriteConsoleA() pasado a la pila
41     push dword caractEscritos    ;Cuarto argumento de
42                                     ;_WriteConsoleA() pasado a la pila
43     push dword [longitudCadena]  ;Tercer argumento de
44                                     ;_WriteConsoleA() pasado a la pila
45     push dword cadImprimir       ;Segundo argumento de
46                                     ;_WriteConsoleA() pasado a la pila
47     push dword [handleConsola]   ;Primer argumento de
48                                     ;_WriteConsoleA() pasado a la pila
49     call _WriteConsoleA@20        ;Invocaci n de _WriteConsoleA()
50
51     xor  eax, eax                ;Limpieza del registro eax(eax=0)
52     mov  eax, 0d                ;eax=0 valor del argumento de _ExitProcess()
53     mov  [ultimoArgumento], eax  ;Se guarda el valor del argumento en memoria
54     push dword [ultimoArgumento] ;Argumento del ExitProcess() pasado por la pila
55     call _ExitProcess@4          ;Invocacion de ExitProcess()

```

4. Guarde su archivo con extensión .asm dentro de Borrador y ensamble desde una consola con el siguiente comando :
`nasm -f win32 -o nombre_archivo.obj nombre_archivo.asm`
5. Enlace el código objeto intermedio generado en el paso anterior con el siguiente comando:
`ld nombre_archivo.obj -m i386pe -e _main -L "ruta_al_archivo_kernel32" -o nombre_ejecutable.exe`
6. Al final de estos dos pasos obtendrá un programa ejecutable, pruebe el funcionamiento de su aplicación.

```

C:\Windows\System32\cmd.exe
-Calculadora-
1.- Suma
2.- Resta
3.- Multiplicacion
4.- Division

Elija una opcion:
3

Ingresa los numeros
2
Numero 2: 2
La multiplicacion es: 4

```

7. Capture el siguiente código:

```

1 segment .bss
2 handleConsola resd 1 ;Primer argumento para la llamada al sistema _ReadConsoleA()
3 cadLeer resd 30 ;Segundo argumento para la llamada al sistema _ReadConsoleA()
4 longitudCadena resd 1 ;Tercer argumento para la llamada al sistema _ReadConsoleA()
5 caractLeidos resd 1 ;Cuarto argumento para la llamada al sistema _ReadConsoleA()
6 ultimoArgumento resd 1 ;Quinto argumento para la llamada al sistema _ReadConsoleA()
7 segment .text
8 global _main
9 extern _GetStdHandle@4 ;Acceso externo a la llamado al sistema _GetStdHandle()
10 extern _WriteConsoleA@20 ;Acceso externo a la llamado al sistema _WriteConsoleA()
11 extern _ReadConsoleA@20 ;Acceso externo a la llamado al sistema _ReadConsoleA()
12 extern _ExitProcess@4 ;Acceso externo a la llamado al sistema _ExitProcess()
13 _main:
14 push dword-10 ;Argumento pasado por la pila y usado en _GetStdHandle()
15 ;para la entrada estandar
16 call _GetStdHandle@4 ;Invocaci n de _GetStdHanle()
17 mov [handleConsola],eax ;Devolucion del manejador de consola
18 ;para la lectura en el registro
19
20 xor eax,eax ;Limpieza del registro eax (eax=0)
21 mov eax,30d ;eax=30 caracteres de longitud de la cadena a leer
22 mov [longitudCadena],eax ;Se guarda la longitud en memoria
23 xor eax,eax ;Limpieza del registro eax (eax=0)
24 mov eax,0d ;eax=0 valor del ultimo argumento de _ReadConsoleA()
25 mov [ultimoArgumento],eax ;Se guarda el valor del ultimo argumento en memoria
26
27 push dword [ultimoArgumento] ;Quinto argumento de _ReadConsoleA() pasado por la pila
28 push dword caractLeidos ;Cuarto argumento de _ReadConsoleA() pasado por la pila
29 push dword [longitudCadena] ;Tercer argumento de _ReadConsoleA() pasado por la pila
30 push dword cadLeer ;Segundo argumento de _ReadConsoleA() pasado por la pila
31 push dword [handleConsola] ;Primer argumento de _ReadConsoleA() pasado por la pila
32 call _ReadConsoleA@20
33
34 xor eax,eax ;Limpieza del registro eax (eax=0)
35 push dword-11 ;Argumento pasado por la pila y usado en _GetStdHandle()
36 ;para la salida estandar
37 call _GetStdHandle@4 ;Invocacion de _GetStdHandle()
38 mov [handleConsola],eax ;Devolucion del manejador de consola
39 ;para escritura en el registro eax
40
41 xor eax,eax ;Limpieza del registro eax(eax=0)
42 mov eax,30d ;eax=30 caracteres del longitud de la cadena a imprimir
43 mov [longitudCadena],eax ;Se guarda la longitud en memoria
44 xor eax,eax ;Limpieza del registro eax(eax=0)
45 mov eax,0d ;eax=0 valor del ultimo argumento de _WriteConsoleA()
46 mov [ultimoArgumento],eax ;Se guarda el valor del ultimo argumento en memoria
47
48 push dword [ultimoArgumento] ;Quinto argumento de WriteConsoleA() pasado por la pila
49 push dword caractLeidos ;Cuarto argumento de WriteConsoleA() pasado por la pila
50 push dword [longitudCadena] ;Tercer argumento de WriteConsoleA() pasado por la pila
51 push dword cadLeer ;Segundo argumento de WriteConsoleA() pasado por la pila
52 push dword [handleConsola] ;Primer argumento de WriteConsoleA() pasado por la pila
53 call _WriteConsoleA@20 ;Invocacion de WriteConsoleA()
54
55 xor eax,eax ;Limpieza del registro eax(eax=0)

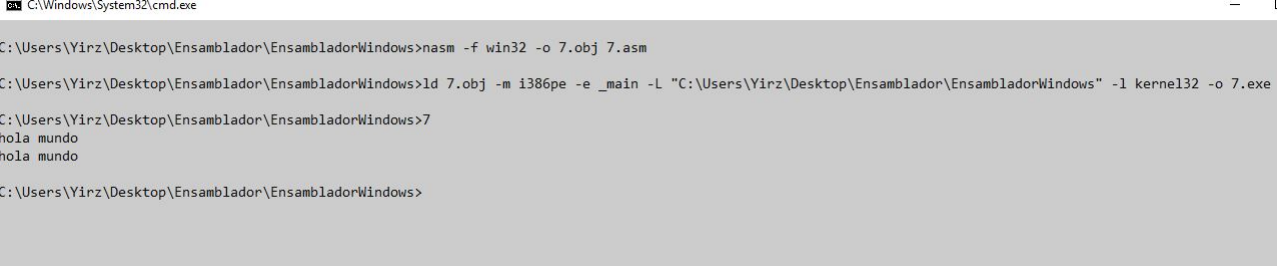
```

```

56 mov eax,0d          ;eax=0 valor del argumento de _ExitProcess()
57 mov [ultimoArgumento],eax ;Se guarda el valor del argumento en memoria
58 push dword [ultimoArgumento] ;Argumento del ExitProcess() pasado por la pila
59 call _ExitProcess@4    ;Invocacion de ExitProcess()

```

8. Ensamble, enlace y ejecute esta aplicación.



```

C:\Windows\System32\cmd.exe

C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>nasm -f win32 -o 7.obj 7.asm

C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>ld 7.obj -m i386pe -e _main -L "C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows" -l kernel32 -o 7.exe

C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>7
hola mundo
hola mundo

C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>

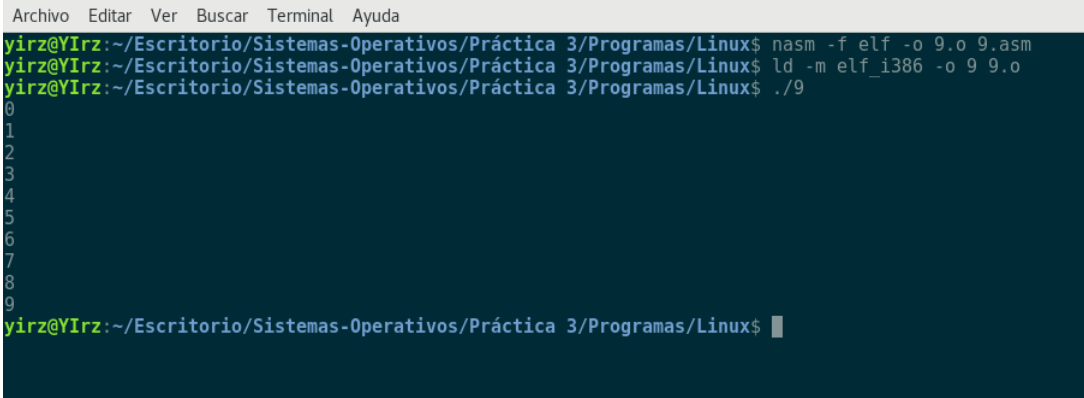
```

9. Programe las aplicaciones de los puntos 9 al 13 de la sección de Linux usando el ensamblador para Windows.

2.3. Programas Desarrollados

2.3.1. Linux

■ Contador



```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
yirz@Yirz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ nasm -f elf -o 9.o 9.asm
yirz@Yirz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ ld -m elf_i386 -o 9 9.o
yirz@Yirz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ ./9
0
1
2
3
4
5
6
7
8
9
yirz@Yirz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$

```

■ Concatenación de 3 cadenas

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ nasm -f elf -o 10.o 10.asm
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ ld -m elf_i386 -o 10 10.o
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ ./10
Ingrese la primer cadena: hola
Ingrese la segunda cadena: mundo
Ingrese la tercer cadena: joaquin
La cadena final es: hmjouolnaadquoin
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$

```

■ Longitud de una cadena

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ nasm -f elf -o 11.o 11.asm
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ ld -m elf_i386 -o 11 11.o
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$ ./11
Ingrese una cadena: yaircarrillobalcazar
La longitud de la cadena que ingresaste es: 20
yirz@YIrz:~/Escritorio/Sistemas-Operativos/Práctica 3/Programas/Linux$

```

■ Concatenación de 10 cadenas, Inversa y Longitud

```

Cadena1:
Hola
Cadena2:
Mundo
Cadena3:
Sistemas
Cadena4:
Operativos
Cadena5:
Practica
Cadena6:
3
Cadena7:
Ensamblador
Cadena8:
Concatenar
Cadena9:
10
Cadena10:
Cadenas
Cadena Concatenada:
HolaMundoSistemasOperativosPractica3EnsambladorConcatenar10Cadenas
Cadena Invertida:
Cadenas10ConcatenarEnsamblador3PracticaOperativosSistemasMundoHola
Longitud de Cadena:
66

```


- Calculadora



```
joaquinipn@joaquinIPN: ~/Documentos/Borrar
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
joaquinipn@joaquinIPN:~/Documentos/Borrar$ nasm -f elf -o main.o main.asm
joaquinipn@joaquinIPN:~/Documentos/Borrar$ ld -m elf_i386 -o main main.o
joaquinipn@joaquinIPN:~/Documentos/Borrar$ ./main

SimpleCalculator

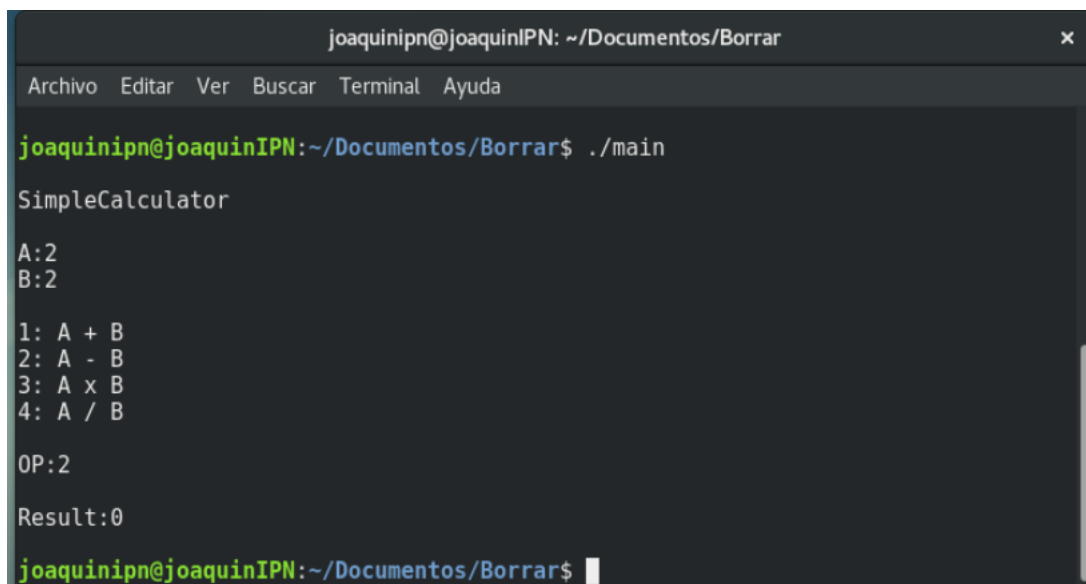
A:1
B:1

1: A + B
2: A - B
3: A x B
4: A / B

OP:1

Result:2

joaquinipn@joaquinIPN:~/Documentos/Borrar$
```



```
joaquinipn@joaquinIPN: ~/Documentos/Borrar
Archivo  Editar  Ver  Buscar  Terminal  Ayuda

joaquinipn@joaquinIPN:~/Documentos/Borrar$ ./main

SimpleCalculator

A:2
B:2

1: A + B
2: A - B
3: A x B
4: A / B

OP:2

Result:0

joaquinipn@joaquinIPN:~/Documentos/Borrar$
```

```
joaquinipn@joaquinIPN: ~/Documentos/Borrar
Archivo  Editar  Ver  Buscar  Terminal  Ayuda

joaquinipn@joaquinIPN:~/Documentos/Borrar$ ./main
SimpleCalculator
A:3
B:2

1: A + B
2: A - B
3: A x B
4: A / B

OP:3
Result:6
joaquinipn@joaquinIPN:~/Documentos/Borrar$
```

```
joaquinipn@joaquinIPN: ~/Documentos/Borrar
Archivo  Editar  Ver  Buscar  Terminal  Ayuda

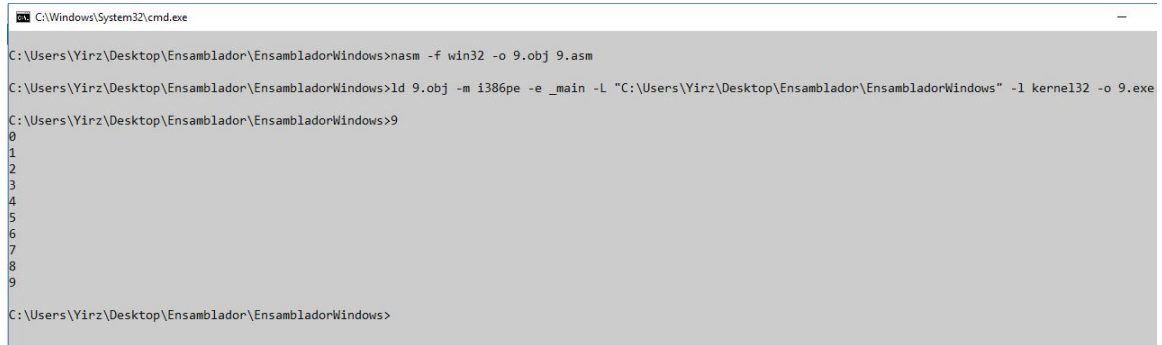
joaquinipn@joaquinIPN:~/Documentos/Borrar$ ./main
SimpleCalculator
A:4
B:2

1: A + B
2: A - B
3: A x B
4: A / B

OP:4
Result:2
joaquinipn@joaquinIPN:~/Documentos/Borrar$
```

2.3.2. Windows

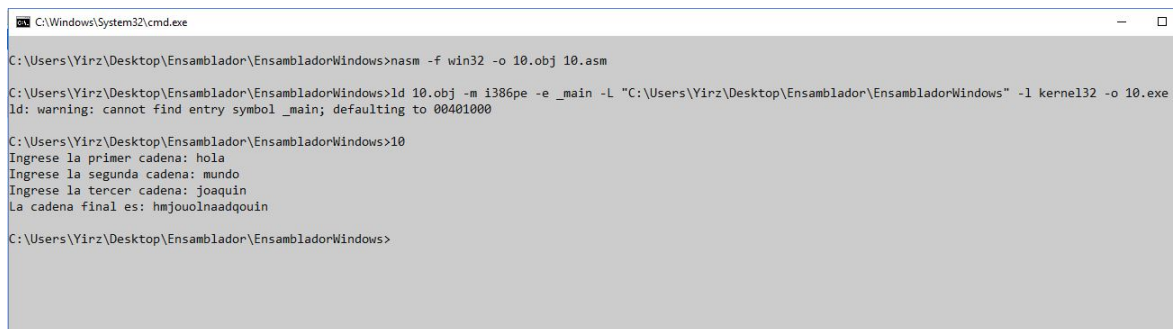
■ Contador



```
C:\Windows\System32\cmd.exe

C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>nasm -f win32 -o 9.obj 9.asm
C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>ld 9.obj -m i386pe -e _main -L "C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows" -l kernel32 -o 9.exe
C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>9
0
1
2
3
4
5
6
7
8
9
C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>
```

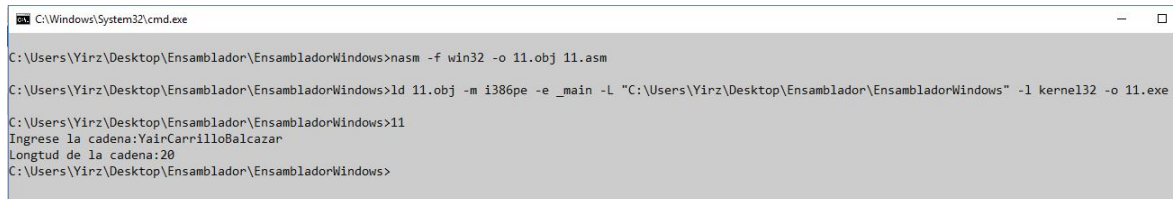
■ Concatenación de 3 cadenas



```
C:\Windows\System32\cmd.exe

C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>nasm -f win32 -o 10.obj 10.asm
C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>ld 10.obj -m i386pe -e _main -L "C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows" -l kernel32 -o 10.exe
ld: warning: cannot find entry symbol _main; defaulting to 00401000
C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>10
Ingrese la primer cadena: hola
Ingrese la segunda cadena: mundo
Ingrese la tercer cadena: joaquin
La cadena final es: hmjouolnaadqouin
C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>
```

■ Longitud de una cadena



```
C:\Windows\System32\cmd.exe

C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>nasm -f win32 -o 11.obj 11.asm
C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>ld 11.obj -m i386pe -e _main -L "C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows" -l kernel32 -o 11.exe
C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>11
Ingrese la cadena: YairCarrilloBalcazar
Longitud de la cadena:20
C:\Users\Yirz\Desktop\Ensamblador\EnsambladorWindows>
```

■ Concatenación de 10 cadenas, Inversa y Longitud

```
C:\Users\Alejandro>C:\Users\Alejandro\Desktop\borrar\win10.exe
Ingresa 10 cadenas no vacias:
s
k
d
f
g
h
d
s
w
f
10
Resultado: sksdfghdswf
Sentido Inverso: fwsdhgfdsk
Longitud: 11
```

■ Calculadora

```
Seleccionar C:\Windows\System32\cmd.exe
C:\Users\Yirz\Documents\Yirz\5to Semestre\Sistemas-Operativos\Ensamblador\EnsambladorWindows>13

-Calculadora-
1.- Suma
2.- Resta
3.- Multiplicacion
4.- Division

Elija una opcion:
1

Ingresa los numeros
2
Numero 2: 2
La suma es: 4
```

```
Seleccionar C:\Windows\System32\cmd.exe
C:\Users\Yirz\Documents\Yirz\5to Semestre\Sistemas-Operativos\Ensamblador\EnsambladorWindows>13

-Calculadora-
1.- Suma
2.- Resta
3.- Multiplicacion
4.- Division

Elija una opcion:
2

Ingresa los numeros
3
Numero 2: 2
La resta es: 1
C:\Users\Yirz\Documents\Yirz\5to Semestre\Sistemas-Operativos\Ensamblador\EnsambladorWindows>13
```

```
Seleccionar C:\Windows\System32\cmd.exe

-Calculadora-
1.- Suma
2.- Resta
3.- Multiplicacion
4.- Division

Elija una opcion:
3

Ingresa los numeros
2
Numero 2: 2
La multiplicacion es: 4
```

```
Seleccionar C:\Windows\System32\cmd.exe
C:\Users\Yirz\Documents\Yirz\5to Semestre\Sistemas-Operativos\Ensamblador\EnsambladorWindows>13

-Calculadora-
1.- Suma
2.- Resta
3.- Multiplicacion
4.- Division

Elija una opcion:
4

Ingresa los numeros
6
Numero 2: 2
La division es: 3
```

2.4. Código Fuente.

2.4.1. Linux

■ Contador

```
1 segment .data
2  nuevaLinea db  "" ,0xA
3 segment .bss
4  i resb 1
5 segment .text
6 global _start
7 _start:
8  mov al, '0'
9  bucle:
10
11  mostrarValor:
12  mov [i], al
13
14  mov eax, 4
15  mov ebx, 1
16  mov ecx, i
17  mov edx, 1
18  int 0x80
19  mostrarLinea:
20  mov eax, 4
21  mov ebx, 1
22  mov ecx, nuevaLinea
23  mov edx, 1
24  int 0x80
25
26  incrementar:
27  mov al, [i]
28  inc al
29
30  comprobarFin:
31  cmp al, 58
32  je  salida
33  jmp bucle
34
35  salida:
36  mov eax, 1
37  mov ebx, 0
38  int 0x80
```

■ Concatenación de 3 cadenas

```

1 SYS_EXIT    equ 1
2 SYS_READ    equ 3
3 SYS_WRITE   equ 4
4 STDIN       equ 0
5 STDOUT      equ 1
6 segment .data
7     mensaje1 db 'Ingrese la primer cadena: '
8     len1 equ $- mensaje1
9     mensaje2 db 'Ingrese la segunda cadena: '
10    len2 equ $- mensaje2
11    mensaje3 db 'Ingrese la tercer cadena: '
12    len3 equ $- mensaje3
13    mensaje4 db 'La cadena final es: '
14    len4 equ $- mensaje4
15 segment .bss
16    cad1 resb 50
17    cad2 resb 50
18    cad3 resb 50
19    cad4 resb 151
20 segment .text
21 global _start
22 _start:
23 obtenerCadena1:
24     pedirCadena1:
25         mov eax, SYS_WRITE
26         mov ebx, STDOUT
27         mov ecx, mensaje1
28         mov edx, len1
29         int 0x80
30     leerCadena1:
31         mov eax, SYS_READ
32         mov ebx, STDIN
33         mov ecx, cad1
34         mov edx, 50d
35         int 0x80
36 obtenerCadena2:
37     pedirCadena2:
38         mov eax, SYS_WRITE
39         mov ebx, STDOUT
40         mov ecx, mensaje2
41         mov edx, len2
42         int 0x80
43     leerCadena2:
44         mov eax, SYS_READ
45         mov ebx, STDIN
46         mov ecx, cad2
47         mov edx, 50d
48         int 0x80
49 obtenerCadena3:
50     pedirCadena3:
51         mov eax, SYS_WRITE
52         mov ebx, STDOUT

```

```
53     mov ecx , mensaje3
54     mov edx , len3
55     int 0x80
56 leerCadena3:
57     mov eax , SYS_READ
58     mov ebx , STDIN
59     mov ecx , cad3
60     mov edx , 50d
61     int 0x80
62 intercalar:
63 moverCadenas:
64     mov eax , cad1
65     mov ebx , cad2
66     mov edi , cad3
67     mov esi , cad4
68 moverCadena1:
69     mov dl , byte[eax]
70     cmp dl , 0xA
71     je moverCadena2
72     mov byte[esi] , dl
73     inc eax
74     inc esi
75 moverCadena2:
76     mov dl , byte[ebx]
77     cmp dl , 0xA
78     je moverCadena3
79     mov byte[esi] , dl
80     inc ebx
81     inc esi
82 moverCadena3:
83     mov dl , byte[edi]
84     cmp dl , 0xA
85     je mostrarCadenaFinal
86     mov byte[esi] , dl
87     inc edi
88     inc esi
89     jmp moverCadena1
90 mostrarCadenaFinal:
91     inc esi
92     mov byte[esi] , 0xA
93     mov eax , SYS_WRITE
94     mov ebx , STDOUT
95     mov ecx , mensaje4
96     mov edx , len4
97     int 0x80
98     mov eax , SYS_WRITE
99     mov ebx , STDOUT
100    mov ecx , cad4
101    mov edx , 151d
102    int 0x80
103 salida:
104    mov eax , SYS_EXIT
105    mov ebx , 0
106    int 0x80
```


■ Longitud de una cadena

```

1 SYS_EXIT equ 1
2 SYS_READ equ 3
3 SYS_WRITE equ 4
4 STDIN equ 0
5 STDOUT equ 1
6 segment .data
7     mensaje1 db 'Ingrese una cadena: '
8     len1 equ $- mensaje1
9     mensaje2 db 'La longitud de la cadena que ingresaste es: '
10    len2 equ $- mensaje2
11    cr db 0xA
12 segment .bss
13    cad resb 1515
14    digito resb 1
15    temp resb 16
16 segment .text
17 global _start
18 _start:
19 pedirCadena:
20     mov eax, SYS_WRITE
21     mov ebx, STDOUT
22     mov ecx, mensaje1
23     mov edx, len1
24     int 0x80
25
26     mov eax, SYS_READ
27     mov ebx, STDIN
28     mov ecx, cad
29     mov edx, 1515
30     int 0x80
31
32     mov esi, cad ;Movemos la cadena leida a este registro
33
34 mostrarAvisoLongitud:
35     mov eax, SYS_WRITE
36     mov ebx, STDOUT
37     mov ecx, mensaje2
38     mov edx, len2
39     int 0x80
40     mov ax, 0 ;Aqui guardaremos la longitud de la cadena
41     mov bx, 10 ;Nos servira para extraer los digitos de la longitud
42     mov cx, 0 ;Contador de digitos de la longitud
43     mov dx, 0 ;Residuo de las divisiones
44
45 calculoLongitud:
46     mov BL, byte[esi] ;Copiamos el byte actual de la cadena
47     cmp BL, 0xA ;Preguntamos si estamos en el fin
48     je enteroACadena ;Si si estamos, ya terminamos de contar
49     inc esi ;Si no, incrementamos el puntero de la cadena
50     inc ax ;y el contador de la longitud
51     jmp calculoLongitud ;Bucle
52

```

```
53 enteroACadena:           ;Obtenemos digito por digito
54     mov bx, 10             ;Divisor
55     div bx                 ;Hacemos la division dx:ax / bx
56     add dx, '0'           ;El residuo se almacena en dx y el cociente en ax,
57                             ;lo convertimos a caracter
58     push dx                ;Lo ponemos en el tope de la pila
59     inc cx                 ;Incrementamos el numero de digitos
60     mov dx, 0
61     cmp ax, 0              ;Preguntamos si ya terminamos de procesar el numero
62     jne enteroACadena     ;No hemos terminado, repetimos
63
64 imprimirDigitos:          ;Imprimimos digito por digito desde la pila
65     pop dx
66     mov [digito], dx
67     mov [temp], cx         ;Guardamos el valor de cx para que no se pierda
68     mov eax, SYS_WRITE
69     mov ebx, STDOUT
70     mov ecx, digito
71     mov edx, 1
72     int 0x80
73     mov cx, [temp]         ;Restauramos el valor de cx
74     dec cx                 ;Ya tenemos un digito menos que procesar
75     cmp cx, 0              ;Si ya no tenemos digitos pendientes,
76     jne imprimirDigitos   ;nos salimos del bucle
77
78 salida:
79     mov eax, SYS_WRITE
80     mov ebx, STDOUT
81     mov ecx, cr
82     mov edx, 1
83     int 0x80
84
85     mov eax, SYS_EXIT
86     mov ebx, 0
87     int 0x80
```

■ Concatenación de 10 cadenas, Inversa y Longitud

```
1 segment .data
2 mensaje1 db 'Cadena1:',0xA
3 lon1 equ $-mensaje1
4 mensaje2 db 'Cadena2:',0xA
5 lon2 equ $-mensaje2
6 mensaje3 db 'Cadena3:',0xA
7 lon3 equ $-mensaje3
8 mensaje4 db 'Cadena4:',0xA
9 lon4 equ $-mensaje4
10 mensaje5 db 'Cadena5:',0xA
11 lon5 equ $-mensaje5
12 mensaje6 db 'Cadena6:',0xA
13 lon6 equ $-mensaje6
14 mensaje7 db 'Cadena7:',0xA
15 lon7 equ $-mensaje7
16 mensaje8 db 'Cadena8:',0xA
17 lon8 equ $-mensaje8
18 mensaje9 db 'Cadena9:',0xA
19 lon9 equ $-mensaje9
20 mensaje10 db 'Cadena10:',0xA
21 lon10 equ $-mensaje10
22 concat db 'Cadena Concatenada:',0xA
23 lonconcat equ $-concat
24 invertida db 'Cadena invertida:',0xA
25 loninv equ $-invertida
26 longitud db 'Longitud de Cadena',0xA
27 loncadena equ $-longitud
28
29 segment .bss
30 cadena1 resb 20
31 cadena2 resb 20
32 cadena3 resb 20
33 cadena4 resb 20
34 cadena5 resb 20
35 cadena6 resb 20
36 cadena7 resb 20
37 cadena8 resb 20
38 cadena9 resb 20
39 cadena10 resb 20
40 cadenaconcat resb 100
41 cadenainv resb 100
42
43
44 lona resb 8
45 lonb resb 8
46 cont1 resb 8
47 cont2 resb 8
48
49 segment .text
50 global _start
51 _start:
52
53 mov byte[cont1],0x30
54 mov byte[cont2],0x30
```

```
55  mov  byte[lon a],0x30
56  mov  byte[lon b],0x30
57
58  mov  eax,4
59  mov  ebx,1
60  mov  ecx,mensaje1
61  mov  edx,lon1
62  int  0x80
63  mov  eax,3
64  mov  ebx,0
65  mov  ecx,cadena1
66  mov  edx,20d
67  int  0x80
68
69  mov  eax,4
70  mov  ebx,1
71  mov  ecx,mensaje2
72  mov  edx,lon2
73  int  0x80
74  mov  eax,3
75  mov  ebx,0
76  mov  ecx,cadena2
77  mov  edx,20d
78  int  0x80
79  mov  eax,4
80  mov  ebx,1
81  mov  ecx,mensaje3
82  mov  edx,lon3
83  int  0x80
84  mov  eax,3
85  mov  ebx,0
86  mov  ecx,cadena3
87  mov  edx,20d
88  int  0x80
89
90
91  mov  eax,4
92  mov  ebx,0
93  mov  ecx,mensaje4
94  mov  edx,lon4
95  int  0x80
96  mov  eax,3
97  mov  ebx,0
98  mov  ecx,cadena4
99  mov  edx,20d
100  int  0x80
101
102  mov  eax,5
103  mov  ebx,0
104  mov  ecx,mensaje5
105  mov  edx,lon5
106  int  0x80
107  mov  eax,3
108  mov  ebx,0
109  mov  ecx,cadena5
```

```
110 mov edx,20d
111 int 0x80
112
113 mov eax,6
114 mov ebx,0
115 mov ecx,mensaje6
116 mov edx,lon6
117 int 0x80
118 mov eax,3
119 mov ebx,0
120 mov ecx,cadena6
121 mov edx,20d
122 int 0x80
123
124 mov eax,7
125 mov ebx,0
126 mov ecx,mensaje7
127 mov edx,lon7
128 int 0x80
129 mov eax,3
130 mov ebx,0
131 mov ecx,cadena7
132 mov edx,20d
133 int 0x80
134
135 mov eax,8
136 mov ebx,0
137 mov ecx,mensaje8
138 mov edx,lon8
139 int 0x80
140 mov eax,3
141 mov ebx,0
142 mov ecx,cadena8
143 mov edx,20d
144 int 0x80
145
146 mov eax,9
147 mov ebx,0
148 mov ecx,mensaje9
149 mov edx,lon9
150 int 0x80
151 mov eax,3
152 mov ebx,0
153 mov ecx,cadena9
154 mov edx,20d
155 int 0x80
156
157 mov eax,10
158 mov ebx,0
159 mov ecx,mensaje10
160 mov edx,lon10
161 int 0x80
162 mov eax,3
163 mov ebx,0
164 mov ecx,cadena10
```

```
165  mov  edx,20d
166  int  0x80
167
168  mov  esi,cadena1
169  mov  edi,cadenaconcat
170
171
172  concatena1:
173  mov  al,[esi]
174  inc  esi
175  mov  [edi],al
176  inc  edi
177  cmp  al,0
178  jne  concatena1
179  mov  esi,cadena3
180  sub  edi,2
181
182  concatena2:
183  mov  al,[esi]
184  inc  esi
185  mov  [esi],al
186  inc  edi
187  cmp  al,0
188  jne  concatena2
189  mov  esi,cadena4
190  sub  edi,2
191
192  concatena3:
193  mov  al,[esi]
194  inc  esi
195  mov  [esi],al
196  inc  edi
197  cmp  al,0
198  jne  concatena3
199  mov  esi,cadena5
200  sub  edi,2
201
202  concatena4:
203  mov  al,[esi]
204  inc  esi
205  mov  [esi],al
206  inc  edi
207  cmp  al,0
208  jne  concatena4
209  mov  esi,cadena6
210  sub  edi,2
211
212  concatena5:
213  mov  al,[esi]
214  inc  esi
215  mov  [esi],al
216  inc  edi
217  cmp  al,0
218  jne  concatena5
219  mov  esi,cadena7
```

```
220  sub  edi,2
221
222  concatena6:
223  mov  al,[esi]
224  inc  esi
225  mov  [esi],al
226  inc  edi
227  cmp  al,0
228  jne  concatena6
229  mov  esi,cadena8
230  sub  edi,2
231
232  concatena7:
233  mov  al,[esi]
234  inc  esi
235  mov  [esi],al
236  inc  edi
237  cmp  al,0
238  jne  concatena7
239  mov  esi,cadena9
240  sub  edi,2
241
242  concatena8:
243  mov  al,[esi]
244  inc  esi
245  mov  [esi],al
246  inc  edi
247  cmp  al,0
248  jne  concatena8
249  mov  esi,cadena10
250  sub  edi,2
251
252  concatena9:
253  mov  al,[esi]
254  inc  esi
255  mov  [esi],al
256  inc  edi
257  cmp  al,0
258  jne  concatena9
259  int  0x80
260
261  mov  eax,4
262  mov  ebx,1
263  mov  ecx,concat
264  int  0x80
265
266  mov  eax,4
267  mov  ebx,1
268  mov  ecx,cadenaconcat
269  mov  edx,100d
270  int  0x80
271
272  mov  esi,cadena10
273  mov  edi,cadenainv
274
```

```
275 contar :
276 mov al,[esi]
277 inc esi
278 mov [edi],al
279 inc edi
280 cmp al,0
281 jne contar
282 mov esi,cadena9
283 sub edi,2
284
285 concatenarla:
286 mov al,[esi]
287 inc esi
288 mov [edi],al
289 inc edi
290 cmp al,0
291 jne concatenarla
292 mov esi,cadena8
293 sub edi,2
294
295 concatenar2a:
296 mov al,[esi]
297 inc esi
298 mov [edi],al
299 inc edi
300 cmp al,0
301 jne concatenar2a
302 mov esi,cadena7
303 sub edi,2
304
305 concatenar3a:
306 mov al,[esi]
307 inc esi
308 mov [edi],al
309 inc edi
310 cmp al,0
311 jne concatenar3a
312 mov esi,cadena6
313 sub edi,2
314
315 concatenar4a:
316 mov al,[esi]
317 inc esi
318 mov [edi],al
319 inc edi
320 cmp al,0
321 jne concatenar4a
322 mov esi,cadena5
323 sub edi,2
324
325 concatenar5a:
326 mov al,[esi]
327 inc esi
328 mov [edi],al
329 inc edi
```



```
330  cmp  al,0
331  jne  concatenar5a
332  mov  esi,cadena4
333  sub  edi,2
334
335  concatenar6a:
336  mov  al,[esi]
337  inc  esi
338  mov  [edi],al
339  inc  edi
340  cmp  al,0
341  jne  concatenar6a
342  mov  esi,cadena3
343  sub  edi,2
344
345  concatenar7a:
346  mov  al,[esi]
347  inc  esi
348  mov  [edi],al
349  inc  edi
350  cmp  al,0
351  jne  concatenar7a
352  mov  esi,cadena3
353  sub  edi,2
354
355  concatenar8a:
356  mov  al,[esi]
357  inc  esi
358  mov  [edi],al
359  inc  edi
360  cmp  al,0
361  jne  concatenar8a
362  mov  esi,cadena2
363  sub  edi,2
364
365  concatenar9a:
366  mov  al,[esi]
367  inc  esi
368  mov  [edi],al
369  inc  edi
370  cmp  al,0
371  jne  concatenar9a
372  mov  esi,cadena1
373  sub  edi,2
374
375  concatenar10a:
376  mov  al,[esi]
377  inc  esi
378  mov  [edi],al
379  inc  edi
380  cmp  al,0
381  jne  concatenar10a
382
383  mov  eax,4
384  mov  ebx,1
```

```
385 mov ecx,invertida
386 mov edx,loninv
387 int 0x80
388
389 mov eax,4
390 mov ebx,1
391 mov ecx,cadenainv
392 mov edx,100d
393 int 0x80
394
395 mov edi,cadenaconcat
396 mov esi,cadenainv
397
398 llenamos:
399 cmp byte[edi],0x00
400 je invierte
401 inc edi
402 inc byte[cont1]
403 inc byte[cont2]
404 jmp llenamos
405 invierte:
406 cmp byte[cont2],0x30
407 dec edi
408 inc esi
409 jmp invierte
410
411 acomoda:
412 mov esi,cadenaconcat
413 mov edi,cadenaconcat
414
415 contarb:
416 mov al,[esi]
417 inc esi
418 mov [edi],al
419 inc edi
420 cmp byte[lona],0x39
421 jne sumara
422 mov byte[lona],0x30
423 cmp byte[lonb],0x39
424 jne sumarb
425 mov byte[lonb],0x30
426 jmp sacar
427
428 sumarb:
429 add byte[lona],1d
430 jmp sacar
431
432 sumara:
433 add byte[lona],1d
434
435 sacar:
436 cmp al,0
437 jne contarb
438 int 0x80
439 mov eax,4
```

```
440  mov ebx,1
441  mov ecx,longitud
442  mov edx,loncadena
443  int 0x80
444  sub byte[lona],2d
445  cmp byte[lona], '.'
446  jne ignorar
447  mov byte[lona], '8'
448  sub byte[lona],1d
449
450  ignorar:
451  cmp byte[lona], '/'
452  jne igual
453  mov byte[lona], '9'
454  sub byte[lonb],1d
455
456  igual:
457  cmp byte[lonb],0x30
458  je imprimir
459  mov eax,4
460  mov ebx,1
461  mov ecx,lonb
462  mov edx,8d
463  int 0x80
464
465  imprimir:
466  mov eax,4
467  mov ebx,1
468  mov ecx,lona
469  mov edx,8d
470  mov byte[ecx+1],0xA
471  int 0x80
472  mov eax,1
473  int 0x80
```

■ Calculadora

```

1 segment.data;
2 Title db 10, 'SimpleCalculator ', 0xA;
3 SizeTitle equ $ - Title;
4 Prompt1 db 10,'A: ',0;
5 SizePrompt1 equ $ - Prompt1;
6 Prompt2 db 'B: ',0;
7 SizePrompt2 equ $ - Prompt2;
8 SumPromt db 10,'1: A + B',10,0;
9 SizeSumPrompt equ $ - SumPrompt;
10 SubPrompt db '2: A - B',10,0;
11 SizeSubPrompt equ $ - SubPrompt;
12 MulPrompt db '3: A x B',10,0;
13 SizeMulPrompt equ $ - MulPrompt;
14 DivPrompt db '4: A / B',10,0;
15 SizeDivPrompt equ $ - DivPrompt;
16 OPPrompt db 10, 'OP: ',0;
17 ResultPrompt db 10,'Result: ',0;
18 SizeResPrompt equ $ - ResultPrompt;
19 NotOpPrompt db 10,'Unknow',10,0
20 SizeNotOpPrompt equ $ - NotOpPrompt;
21 NewLine db 10,10,0;
22 SizeNewLine equ $ - NewLine;
23
24 segment.bss;
25 OptionNumber resb 2;
26 NumberA resb 2;
27 NumberB resb 2;
28 ResultNumber resb 2;
29
30 segment .text;
31 globAL _start;
32 _start::;
33
34 GetNumbers::;
35 ShowPrompt::;
36 mov EAX, 4;
37 mov EBX, 1;
38 mov ECX, Title;
39 mov EDX, SizeTitle;
40 int 0x80;
41
42 ShowAndGetANumber::;
43 mov EAX, 4;
44 mov EBX, 1;
45 mov ECX, Prompt1;
46 mov EDX, SizePrompt1;
47 int 0x80;
48 mov EAX, 3;
49 mov EBX, 0;
50 mov ECX, NumberA;
51 mov EDX, 2;
52 int 0x80;
53
54 ShowAndGetBNumber:

```

```
55  mov EAX, 4;
56  mov EBX, 1;
57  mov ECX, Prompt2;
58  mov EDX, SizePrompt2;
59  int 0x80;
60  mov EAX, 3;
61  mov EBX, 0;
62  mov ECX, NumberB;
63  mov EDX, 2;
64  int 0x80;
65
66  GetOperation:
67  mov EAX, 4;
68  mov EBX, 1;
69  mov ECX, SumPrompt;
70  mov EDX, SizeSumPrompt;
71  int 0x80;
72  mov EAX, 4;
73  mov EBX, 1;
74  mov ECX, SubPrompt;
75  mov EDX, SizeSubPrompt;
76  int 0x80;
77  mov EAX, 4;
78  mov EBX, 1;
79  mov ECX, MulPrompt;
80  mov EDX, SizeMulPrompt;
81  int 0x80;
82  mov EAX, 4;
83  mov EBX, 1;
84  mov ECX, DivPrompt;
85  mov EDX, SizeDivPrompt;
86  int 0x80;
87  mov EAX, 4;
88  mov EBX, 1;
89  mov ECX, OPPrompt;
90  mov EDX, SizeOpPrompt;
91  int 0x80;
92
93  GetTheOperation:
94  mov EAX, 3;
95  mov EBX, 0;
96  mov ECX, OptionNumber;
97  mov EDX, 2;
98  int 0x80;
99
100 SelectOperation:;
101 DirectionOfNumber:;
102  mov AH, [OptionNumber];
103  sub AH, '0';
104  FromASCIIToNumber:;
105  mov AL, [NumberA];
106  mov BL, [NumberB];
107  sub AL, '0';
108  sub BL, '0';
109
```

```
110 ChooseOperation::
111     cmp AH, 1;
112     je Sum;
113     cmp AH, 2;
114     je Sub;
115     cmp AH, 3;
116     je Mul;
117     cmp AH, 4;
118     je Div;
119
120 ShowNoVALidOperation: ;
121     mov EAX, 4;
122     mov EBX, 1;
123     mov ECX, NotOpPrompt;
124     mov EDX, SizeNotOpPrompt;
125     int 0x80;
126     jmp SecureExit;
127
128 DoTheRealOperation::
129 Sum::
130     add AL, BL;
131     add AL, '0';
132     mov [ResultNumber], AL;
133     jmp ShowResult;
134
135 Sub::
136     sub AL, BL;
137     add AL, '0' ;
138     mov [ResultNumber], AL;
139     jmp ShowResult;
140
141 Mul:
142     mul BL;
143     add AX, '0';
144     mov [ResultNumber], AX;
145     jmp ShowResult;
146
147 Div:
148     mov DX, 0;
149     mov AH, 0;
150     div BL;
151     add AX, '0';
152     mov [ResultNumber], AX;
153     jmp ShowResult;
154
155 ShowResult:
156     mov EAX, 4;
157     mov EBX, 1;
158     mov ECX, ResultPrompt;
159     mov EDX, SizeResPrompt;
160     int 0x80;
161     mov EAX, 4;
162     mov EBX, 1;
163     mov ECX, ResultNumber;
164     mov EDX, 2;
```

```
165     int 0x80;
166
167 SecureExit::
168     mov EAX, 4;
169     mov EBX, 1;
170     mov ECX, NewLine;
171     mov EDX, SizeNewLine;
172     int 0x80;
173     mov EAX, 1;
174     mov EBX, 0;
175     int 0x80;
```

2.4.2. Windows

■ Contador

```
1 segment .data
2     nuevaLinea db " ",0xA
3 segment .bss
4     contador resd 1
5     handleConsola resd 1
6     leído resd 1
7
8 segment .text
9 global _main
10 extern _GetStdHandle@4
11 extern _WriteConsoleA@20
12 extern _ExitProcess@4
13 _main:
14     mov al, '0'
15 bucle:
16     ImprimirValor:
17         mov [contador], al
18         push dword -11
19         call _GetStdHandle@4
20         mov [handleConsola], eax
21
22         push dword 0
23         push dword leído
24         push dword 1
25         push dword contador
26         push dword [handleConsola]
27         call _WriteConsoleA@20
28
29         push dword -11
30         call _GetStdHandle@4
31         mov [handleConsola], eax
32         push dword 0
33         push dword leído
34         push dword 1
35         push dword nuevaLinea
36         push dword [handleConsola]
37         call _WriteConsoleA@20
```

```
38
39
40 Incrementar:
41     mov al,[contador]
42     inc al
43 comprobar:
44     cmp al,58
45     je salida
46     jmp bucle
47
48 salida:
49     push dword 0
50     call _ExitProcess@4
```

■ Concatenación de 3 cadenas

```
1 STDIN      equ dword -10
2 STDOUT     equ dword -11
3
4 segment .data
5     mensaje1 db 'Ingresa la primer cadena: '
6     len1 equ $- mensaje1
7     mensaje2 db 'Ingresa la segunda cadena: '
8     len2 equ $- mensaje2
9     mensaje3 db 'Ingresa la tercer cadena: '
10    len3 equ $- mensaje3
11    mensaje4 db 'La cadena final es: '
12    len4 equ $- mensaje4
13
14 segment .bss
15    cad1 resb 50
16    cad2 resb 50
17    cad3 resb 50
18    cad4 resb 151
19    handle resb 1
20    leído resb 2
21
22 segment .text
23     global _start
24     extern _GetStdHandle@4
25     extern _WriteConsoleA@20
26     extern _ReadConsoleA@20
27     extern _ExitProcess@4
28
29 _start:
30
31 obtenerCadena1:
32
33 pedirCadena1:
34     push dword STDOUT
35     call _GetStdHandle@4
36     mov [handle], EAX
37     push dword 0
38     push dword leído
```



```
39     push dword len1
40     push dword mensaje1
41     push dword [handle]
42     call _WriteConsoleA@20
43
44 leerCadena1:
45     push dword STDIN
46     call _GetStdHandle@4
47     mov [handle], EAX
48     push dword 0
49     push dword leído
50     push dword 50
51     push dword cad1
52     push dword [handle]
53     call _ReadConsoleA@20
54
55 obtenerCadena2:
56
57 pedirCadena2:
58     push dword STDOUT
59     call _GetStdHandle@4
60     mov [handle], EAX
61     push dword 0
62     push dword leído
63     push dword len2
64     push dword mensaje2
65     push dword [handle]
66     call _WriteConsoleA@20
67
68 leerCadena2:
69     push dword STDIN
70     call _GetStdHandle@4
71     mov [handle], EAX
72     push dword 0
73     push dword leído
74     push dword 50
75     push dword cad2
76     push dword [handle]
77     call _ReadConsoleA@20
78
79 obtenerCadena3:
80
81 pedirCadena3:
82     push dword STDOUT
83     call _GetStdHandle@4
84     mov [handle], EAX
85     push dword 0
86     push dword leído
87     push dword len3
88     push dword mensaje3
89     push dword [handle]
90     call _WriteConsoleA@20
91
92 leerCadena3:
93     push dword STDIN
```

```
94     call _GetStdHandle@4
95     mov [handle], EAX
96     push dword 0
97     push dword leído
98     push dword 50
99     push dword cad3
100    push dword [handle]
101    call _ReadConsoleA@20
102
103    intercalar:
104
105    moverCadenas:
106        mov EAX, cad1
107        mov EBX, cad2
108        mov EDI, cad3
109        mov ESI, cad4
110
111    moverCadena1:
112        mov DL, byte[EAX]
113        cmp DL, 0xD
114        je moverCadena2
115
116        mov byte[ESI], DL
117        inc EAX
118        inc ESI
119
120    moverCadena2:
121        mov DL, byte[EBX]
122        cmp DL, 0xD
123        je moverCadena3
124
125        mov byte[ESI], DL
126        inc EBX
127        inc ESI
128
129    moverCadena3:
130        mov DL, byte[EDI]
131        cmp DL, 0xD
132        je mostrarCadenaFinal
133
134        mov byte[ESI], DL
135        inc EDI
136        inc ESI
137        jmp moverCadena1
138
139
140    mostrarCadenaFinal:
141        push dword STDOUT
142        call _GetStdHandle@4
143        mov [handle], EAX
144        push dword 0
145        push dword leído
146        push dword len4
147        push dword mensaje4
148        push dword [handle]
```

```

149     call _WriteConsoleA@20
150
151     push dword STDOUT
152     call _GetStdHandle@4
153     mov [handle], EAX
154     push dword 0
155     push dword leído
156     push dword 151
157     push dword cad4
158     push dword [handle]
159     call _WriteConsoleA@20
160
161 salida:
162     push dword 0
163     call _ExitProcess@4

```

■ Longitud de una cadena

```

1 STDIN      equ dword -10
2 STDOUT     equ dword -11
3
4 segment .data
5     mensaje1 db 'Ingrese la cadena:'
6     len1 equ $- mensaje1
7     mensaje2 db 'Longtud de la cadena:'
8     len2 equ $- mensaje2
9     longitud dw 0
10    leído db 0
11    handleConsola resd 0
12 segment .bss
13    cad resd 1515
14    longitudCad resd 2
15
16
17 segment .text
18     global _main
19     extern _GetStdHandle@4
20     extern _WriteConsoleA@20
21     extern _ReadConsoleA@20
22     extern _ExitProcess@4
23 _main:
24     PedirCadena:
25     push dword STDOUT
26     call _GetStdHandle@4
27     mov [handleConsola], eax
28     push dword 0
29     push dword leído
30     push dword len1
31     push dword mensaje1
32     push dword [handleConsola]
33     call _WriteConsoleA@20
34
35     push dword STDIN
36     call _GetStdHandle@4

```

```

37     mov [handleConsola], EAX
38     push dword 0
39     push dword longitud
40     push dword 1515
41     push dword cad
42     push dword [handleConsola]
43     call _ReadConsoleA@20
44
45     push dword STDOUT
46     call _GetStdHandle@4
47     mov [handleConsola], EAX
48     push dword 0
49     push dword leido
50     push dword len2
51     push dword mensaje2
52     push dword [handleConsola]
53     call _WriteConsoleA@20
54
55     dec word[longitud]      ;Borramos el CR
56     dec word[longitud]      ;Borramos el LF
57     mov ax, word[longitud]  ;Aquí guardaremos la longitud de la cadena
58     mov bx, 0               ;Contador de digitos de la longitud
59     mov cx, 10              ;Nos servira para extraer los digitos de la longitud
60     mov dx, 0               ;Residuo de las divisiones
61
62 enteroACadena:             ;Obtenemos digito por digito
63     mov cx, 10              ;Divisor
64     div cx                  ;Hacemos la division DX:AX / BX
65     add dx, '0'             ;El residuo se almacena en DX y el cociente en AX, lo convertemos a char
66     push dx                 ;Lo ponemos en el tope de la pila
67     inc bx                  ;Incrementamos el numero de digitos
68     mov dx, 0               ;Regresamos DX a 0 para que la division se haga correctamente
69     cmp ax, 0               ;Preguntamos si ya terminamos de procesar el numero
70     jne enteroACadena       ;No hemos terminado, repetimos
71
72 imprimirDigitos:           ;Imprimimos digito por digito desde la pila
73     pop dx
74     mov [longitudCad], dx
75
76     push dword STDOUT
77     call _GetStdHandle@4
78     mov [handleConsola], eax
79     push dword 0
80     push dword leido
81     push dword 1
82     push dword longitudCad
83     push dword [handleConsola]
84     call _WriteConsoleA@20
85
86     dec bx                  ;Ya tenemos un digito menos que procesar
87     cmp bx, 0               ;Si ya no tenemos digitos pendientes,
88     jne imprimirDigitos     ;nos salimos del bucle
89
90 salida:
91     push dword 0

```

```
92      call _ExitProcess@4
```

■ Concatenación de 10 cadenas, Inversa y Longitud

```
1 segment .data
2  instruccion db 'Introduzca las 10 cadenas de tamaño 10.',0xA
3  ult db 'Cadena concatenada:',0xA
4  vol db 'Cadena volteada:',0xA
5  tam db 'Tamaño de cadena: '
6  len db '100',0xA
7 segment .bss
8  handleConsola resd 1
9  longitudCadena resd 1
10 caractLeidos resd 1
11 ultimoArgumento resd 1
12 cadena1 resb 12
13 cadena2 resb 12
14 cadena3 resb 12
15 cadena4 resb 12
16 cadena5 resb 12
17 cadena6 resb 12
18 cadena7 resb 12
19 cadena8 resb 12
20 cadena9 resb 12
21 cadena10 resb 12
22 cadenafinal resb 120
23 cadenavolt resb 120
24 cadenaux resb 120
25 auxcad resb 12
26
27 segment .text
28 global _main
29 extern _GetStdHandle@4
30 extern _WriteConsoleA@20
31 extern _ReadConsoleA@20
32 extern _ExitProcess@4
33
34 _main:
35  push dword -11
36  call _GetStdHandle@4
37  mov [handleConsola],eax
38
39  xor eax,eax
40  mov eax,41d
41  mov [longitudCadena],eax
42  xor eax,eax
43  mov eax,0d
44  mov [ultimoArgumento],eax
45
46  push dword [ultimoArgumento]
47  push dword caractLeidos
48  push dword [longitudCadena]
49  push dword instruccion
50  push dword [handleConsola]
51  call _WriteConsoleA@20
```

```
52 ;1
53 push dword -10
54 call _GetStdHandle@4
55 mov [handleConsola], eax
56
57 xor eax, eax
58 mov eax, 12d
59 mov [longitudCadena], eax
60 xor eax, eax
61 mov eax, 0d
62 mov [ultimoArgumento], eax
63
64 push dword [ultimoArgumento]
65 push dword caractLeidos
66 push dword [longitudCadena]
67 push dword cadena1
68 push dword [handleConsola]
69 call _ReadConsoleA@20
70 ;2
71 xor eax, eax
72 mov eax, 12d
73 mov [longitudCadena], eax
74 xor eax, eax
75 mov eax, 0d
76 mov [ultimoArgumento], eax
77
78 push dword [ultimoArgumento]
79 push dword caractLeidos
80 push dword [longitudCadena]
81 push dword cadena2
82 push dword [handleConsola]
83 call _ReadConsoleA@20
84 ;3
85 xor eax, eax
86 mov eax, 12d
87 mov [longitudCadena], eax
88 xor eax, eax
89 mov eax, 0d
90 mov [ultimoArgumento], eax
91
92 push dword [ultimoArgumento]
93 push dword caractLeidos
94 push dword [longitudCadena]
95 push dword cadena3
96 push dword [handleConsola]
97 call _ReadConsoleA@20
98
99 push dword -10
100 call _GetStdHandle@4
101 mov [handleConsola], eax
102
103 xor eax, eax
104 mov eax, 12d
105 mov [longitudCadena], eax
106 xor eax, eax
```

```
107 mov eax,0d
108 mov [ultimoArgumento],eax
109
110 push dword [ultimoArgumento]
111 push dword caractLeidos
112 push dword [longitudCadena]
113 push dword cadena4
114 push dword [handleConsola]
115 call _ReadConsoleA@20
116
117 push dword -10
118 call _GetStdHandle@4
119 mov [handleConsola], eax
120
121 xor eax,eax
122 mov eax,12d
123 mov [longitudCadena],eax
124 xor eax,eax
125 mov eax,0d
126 mov [ultimoArgumento],eax
127
128 push dword [ultimoArgumento]
129 push dword caractLeidos
130 push dword [longitudCadena]
131 push dword cadena5
132 push dword [handleConsola]
133 call _ReadConsoleA@20
134
135 push dword -10
136 call _GetStdHandle@4
137 mov [handleConsola], eax
138
139 xor eax,eax
140 mov eax,12d
141 mov [longitudCadena],eax
142 xor eax,eax
143 mov eax,0d
144 mov [ultimoArgumento],eax
145
146 push dword [ultimoArgumento]
147 push dword caractLeidos
148 push dword [longitudCadena]
149 push dword cadena6
150 push dword [handleConsola]
151 call _ReadConsoleA@20
152
153 push dword -10
154 call _GetStdHandle@4
155 mov [handleConsola], eax
156
157 xor eax,eax
158 mov eax,12d
159 mov [longitudCadena],eax
160 xor eax,eax
161 mov eax,0d
```

```
162 mov [ultimoArgumento],eax
163
164 push dword [ultimoArgumento]
165 push dword caractLeidos
166 push dword [longitudCadena]
167 push dword cadena7
168 push dword [handleConsola]
169 call _ReadConsoleA@20
170
171 push dword -10
172 call _GetStdHandle@4
173 mov [handleConsola], eax
174
175 xor eax,eax
176 mov eax,12d
177 mov [longitudCadena],eax
178 xor eax,eax
179 mov eax,0d
180 mov [ultimoArgumento],eax
181
182 push dword [ultimoArgumento]
183 push dword caractLeidos
184 push dword [longitudCadena]
185 push dword cadena8
186 push dword [handleConsola]
187 call _ReadConsoleA@20
188
189 push dword -10
190 call _GetStdHandle@4
191 mov [handleConsola], eax
192
193 xor eax,eax
194 mov eax,12d
195 mov [longitudCadena],eax
196 xor eax,eax
197 mov eax,0d
198 mov [ultimoArgumento],eax
199
200 push dword [ultimoArgumento]
201 push dword caractLeidos
202 push dword [longitudCadena]
203 push dword cadena9
204 push dword [handleConsola]
205 call _ReadConsoleA@20
206
207 push dword -10
208 call _GetStdHandle@4
209 mov [handleConsola], eax
210
211 xor eax,eax
212 mov eax,12d
213 mov [longitudCadena],eax
214 xor eax,eax
215 mov eax,0d
216 mov [ultimoArgumento],eax
```



```
217
218 push dword [ultimoArgumento]
219 push dword caractLeidos
220 push dword [longitudCadena]
221 push dword cadena10
222 push dword [handleConsola]
223 call _ReadConsoleA@20
224 ;CONCATENAR CADENAS
225 xor edi, edi
226 xor esi, esi
227 xor eax, eax
228 xor ebx, ebx
229 xor ecx, ecx
230
231 mov edi, cadenafinal
232 mov esi, cadena1
233 mov eax, cadena2
234 mov ebx, cadena3
235 mov ecx, cadena4
236
237 pricad:
238 mov dl, byte[esi]
239 cmp dl, 0xA
240 je cond
241 mov byte[edi], dl
242 inc edi
243 inc esi
244 jmp pricad
245
246 cond:
247 dec edi
248 jmp seccad
249
250 seccad:
251 mov dl, byte[eax]
252 cmp dl, 0xA
253 je condsec
254 mov byte[edi], dl
255 inc edi
256 inc eax
257 jmp seccad
258
259 condsec:
260 dec edi
261 jmp tercad
262
263 tercad:
264 mov dl, byte[ebx]
265 cmp dl, 0xA
266 je condter
267 mov byte[edi], dl
268 inc edi
269 inc ebx
270 jmp tercad
271
```

```
272 condter:
273 dec edi
274 jmp cuacad
275
276 cuacad:
277 mov dl, byte[ecx]
278 cmp dl, 0xA
279 je condcua
280 mov byte[edi], dl
281 inc edi
282 inc ecx
283 jmp cuacad
284
285 condcua:
286 dec edi
287 xor esi, esi
288 xor eax, eax
289 xor ebx, ebx
290 xor ecx, ecx
291 mov esi, cadena5
292 mov eax, cadena6
293 mov ebx, cadena7
294 mov ecx, cadena8
295 jmp cincad
296
297 cincad:
298 mov dl, byte[esi]
299 cmp dl, 0xA
300 je condcin
301 mov byte[edi], dl
302 inc edi
303 inc esi
304 jmp cincad
305
306 condcin:
307 dec edi
308 jmp seicad
309
310 seicad:
311 mov dl, byte[eax]
312 cmp dl, 0xA
313 je condsei
314 mov byte[edi], dl
315 inc edi
316 inc eax
317 jmp seicad
318
319 condsei:
320 dec edi
321 jmp siecad
322
323 siecad:
324 mov dl, byte[ebx]
325 cmp dl, 0xA
326 je condsie
```

```
327 mov byte[edi], dl
328 inc edi
329 inc ebx
330 jmp siecad
331
332 condsie:
333 dec edi
334 jmp ochcad
335
336 ochcad:
337 mov dl, byte[ecx]
338 cmp dl, 0xA
339 je condoch
340 mov byte[edi], dl
341 inc edi
342 inc ecx
343 jmp ochcad
344
345 condoch:
346 xor esi, esi
347 xor eax, eax
348 mov esi, cadena9
349 mov eax, cadena10
350 dec edi
351 jmp nuecad
352
353 nuecad:
354 mov dl, byte[esi]
355 cmp dl, 0xA
356 je condnue
357 mov byte[edi], dl
358 inc edi
359 inc esi
360 jmp nuecad
361
362 condnue:
363 dec edi
364 jmp diecad
365
366
367 ;HACER ESTO PARA LA ULTIMA CADENA
368
369 diecad:
370 mov dl, byte[eax]
371 mov byte[edi], dl
372 inc edi
373 inc eax
374 cmp dl, 0xA
375 je ultcad
376 jmp diecad
377
378 ultcad:
379 inc edi
380 mov byte[edi], 0xA
381 dec edi
```

```
382 dec edi
383 xor esi, esi
384 mov esi, cadenaux
385 jmp voltear
386
387 ;PROCESO PARA VOLTEAR CADENA
388 ;cadenafinal es la que ya est concatenada
389
390 voltear:
391 mov dl, byte[edi]
392 cmp dl, 0h
393 je condvol
394 mov byte[esi], dl
395 dec edi
396 inc esi
397 jmp voltear
398
399 condvol:
400 inc esi
401 mov byte[esi], 0xA
402
403 ;IMPRIME TEXTO Y CADENA CONCATENADA
404
405 xor eax, eax
406 push dword -11
407 call _GetStdHandle@4
408 mov [handleConsola], eax
409
410 xor eax, eax
411 mov eax, 20d
412 mov [longitudCadena], eax
413 xor eax, eax
414 mov eax, 0d
415 mov [ultimoArgumento], eax
416
417 push dword [ultimoArgumento]
418 push dword caractLeidos
419 push dword [longitudCadena]
420 push dword ult
421 push dword [handleConsola]
422 call _WriteConsoleA@20
423
424 xor eax, eax
425 push dword -11
426 call _GetStdHandle@4
427 mov [handleConsola], eax
428
429 xor eax, eax
430 mov eax, 102d
431 mov [longitudCadena], eax
432 xor eax, eax
433 mov eax, 0d
434 mov [ultimoArgumento], eax
435
436 push dword [ultimoArgumento]
```

```
437 push dword caractLeidos
438 push dword [longitudCadena]
439 push dword cadenafinal
440 push dword [handleConsola]
441 call _WriteConsoleA@20
442
443 ;IMPRIME TEXTO Y CADENA VOLTEADA
444
445 xor eax, eax
446 push dword -11
447 call _GetStdHandle@4
448 mov [handleConsola], eax
449
450 xor eax, eax
451 mov eax, 16d
452 mov [longitudCadena], eax
453 xor eax, eax
454 mov eax, 0d
455 mov [ultimoArgumento], eax
456
457 push dword [ultimoArgumento]
458 push dword caractLeidos
459 push dword [longitudCadena]
460 push dword vol
461 push dword [handleConsola]
462 call _WriteConsoleA@20
463
464 xor eax, eax
465 push dword -11
466 call _GetStdHandle@4
467 mov [handleConsola], eax
468
469 xor eax, eax
470 mov eax, 104d
471 mov [longitudCadena], eax
472 xor eax, eax
473 mov eax, 0d
474 mov [ultimoArgumento], eax
475
476 push dword [ultimoArgumento]
477 push dword caractLeidos
478 push dword [longitudCadena]
479 push dword cadenaux
480 push dword [handleConsola]
481 call _WriteConsoleA@20
482
483 xor eax, eax
484 push dword -11
485 call _GetStdHandle@4
486 mov [handleConsola], eax
487
488 xor eax, eax
489 mov eax, 18d
490 mov [longitudCadena], eax
491 xor eax, eax
```

```
492 mov eax,0d
493 mov [ultimoArgumento],eax
494
495 push dword [ultimoArgumento]
496 push dword caractLeidos
497 push dword [longitudCadena]
498 push dword tam
499 push dword [handleConsola]
500 call _WriteConsoleA@20
501
502 xor eax, eax
503 push dword -11
504 call _GetStdHandle@4
505 mov [handleConsola],eax
506
507 xor eax,eax
508 mov eax,3d
509 mov [longitudCadena],eax
510 xor eax,eax
511 mov eax,0d
512 mov [ultimoArgumento],eax
513
514 push dword [ultimoArgumento]
515 push dword caractLeidos
516 push dword [longitudCadena]
517 push dword len
518 push dword [handleConsola]
519 call _WriteConsoleA@20
520
521 xor eax, eax
522 mov eax, 0d
523 mov [ultimoArgumento], eax
524 push dword [ultimoArgumento]
525 call _ExitProcess@4
```

■ Calculadora

```

1 global _main
2 extern _GetStdHandle@4
3 extern _ExitProcess@4
4 extern _WriteConsole@20
5 extern _ReadConsole@20
6
7 segment .data
8 cadenaMsj: db 10,10,'-Calculadora-',10,'1.- Suma',10,'2.- Resta',10,
9 '3.- Multiplicacion',10,'4.- Division',10,10,'Elija una opcion: ',0xA
10 cadLen equ $ - cadenaMsj
11 Msj1: db 10,10,'Ingresa los numeros',0xA
12 cadLen1 equ $ - Msj1
13 Msj2: db 'Numero 1: ',0
14 cadLen2 equ $ - Msj2
15 Msj3: db 'Numero 2: ',0
16 cadLen3 equ $ - Msj3
17 rsuma: db 'La suma es: ',0
18 lrsuma equ $ - rsuma
19 rresta: db 'La resta es: ',0
20 lrresta equ $ - rresta
21 rmulti: db 'La multiplicacion es: ',0
22 lrmulti equ $ - rmulti
23 rdiv: db 'La division es:',0
24 lrdiv equ $ - rdiv
25 handle: db 0
26 cadenaEscrita: db 0
27 StringNumLeida: db 0
28
29 segment .bss
30 StringNum1 resb 30
31 StringNum2 resb 30
32 longNum1 resb 30
33 longNum2 resb 30
34 Num1 resb 30
35 Num2 resb 30
36 resu resb 30
37 resuDiv resb 30
38 auxAcu resb 30
39 aux resb 30
40 auxCon resb 30
41 op resb 2
42
43 segment .text
44 _main:
45 call mostrarMenu
46 push dword -10
47 call _GetStdHandle@4
48 mov [handle],eax
49 push dword 0d
50 push StringNum1Leida
51 push 5d
52 push op
53 push dword [handle]
54 call _ReadConsole@20

```

```
55  call  imprimirMsj1
56  push  dword -10
57  call  _GetStdHandle@4
58  mov  [handle],eax
59  push  dword 0d
60  push  StringNum1Leida
61  push  5d
62  push  StringNum1
63  push  dword [handle]
64  call  _ReadConsole@20
65  call  imprimirMsj3
66  push  dword -10
67  call  _GetStdHandle@4
68  mov  [handle],eax
69  push  dword 0d
70  push  StringNum1Leida
71  push  5d
72  push  StringNum2
73  push  dword [handle]
74  call  _ReadConsole@20
75  call  longitudN1
76  call  longitudN2
77  call  convANum1
78  call  convANum2
79  sub  byte[op], '0'
80  cmp  byte[op], 1d
81  je  sumar
82  cmp  byte[op], 2d
83  je  restar
84  cmp  byte[op], 3d
85  je  multiplicar
86  cmp  byte[op], 4d
87  je  dividir
88  push  0
89  call  _ExitProcess@4
90
91  sumar:
92  mov  eax,[Num1]
93  add  eax,[Num2]
94  mov  [resu],eax
95  push  dword -11
96  call  _GetStdHandle@4
97  mov  [handle],eax
98  push  dword 0
99  push  cadenaEscrita
100 push  lrsuma
101 push  rsuma
102 push  dword [handle]
103 call  _WriteConsole@20
104 call  convAText
105 push  0
106 call  _ExitProcess@4
107 ret
108
109 restar:
```



```
110 mov eax,[Num1]
111 sub eax,[Num2]
112 mov [resu],eax
113 push dword -11
114 call _GetStdHandle@4
115 mov [handle],eax
116 push dword 0
117 push cadenaEscrita
118 push lrresta
119 push rresta
120 push dword [handle]
121 call _WriteConsole@20
122 call convAText
123 push 0
124 call _ExitProcess@4
125 ret
126
127 multiplicar:
128 mov eax,[Num1]
129 mov ebx,[Num2]
130 mul ebx
131 mov [resu],eax
132 push dword -11
133 call _GetStdHandle@4
134 mov [handle],eax
135 push dword 0
136 push cadenaEscrita
137 push lrmulti
138 push rmulti
139 push dword [handle]
140 call _WriteConsole@20
141 call convAText
142 push 0
143 call _ExitProcess@4
144 ret
145
146 dividir:
147 xor edx,edx
148 mov eax,[Num1]
149 mov ebx,[Num2]
150 div ebx
151 mov [resu],eax
152 push dword -11
153 call _GetStdHandle@4
154 mov [handle],eax
155 push dword 0
156 push cadenaEscrita
157 push lrdiv
158 push rdiv
159 push dword [handle]
160 call _WriteConsole@20
161 call convAText
162 push 0
163 call _ExitProcess@4
164 ret
```

```
165
166 longitudN1:
167     mov esi,StringNum1
168     mov byte[longNum1],0d
169     lpLN1:
170         mov al,byte[esi]
171         cmp al,0xA
172         jz finLongN1
173         add byte[longNum1],1d
174         inc esi
175         loop lpLN1
176     finLongN1:
177         dec byte[longNum1]
178         ret
179
180 longitudN2:
181     mov esi,StringNum2
182     mov byte[longNum2],0d
183     lpLN2:
184         mov al,byte[esi]
185         cmp al,0xA
186         jz finLongN2
187         add byte[longNum2],1d
188         inc esi
189         loop lpLN2
190     finLongN2:
191         dec byte[longNum2]
192         ret
193
194 convANum1:
195     mov esi,StringNum1
196     mov ebx,[longNum1]
197     dec ebx
198     add esi,ebx
199     mov al,[esi]
200     sub al,0
201     mov [auxAcu],al
202     mov ecx,10d
203     mov ebx,[longNum1]
204     mov [auxCon],ebx
205     dec byte[auxCon]
206     al:
207         cmp byte[auxCon],0d
208         je fin1
209         dec esi
210         mov al,[esi]
211         sub al,0
212         mov [aux],al
213         mov eax,[aux]
214         mul ecx
215         add [auxAcu],eax
216         mov eax,ecx
217         mov ecx,10d
218         mul ecx
219         mov ecx,eax
```

```
220     dec byte [auxCon]
221     loop a1
222     fin1:
223     cmp byte [longNum1],3
224     jb termino1
225     add byte [auxAcu],1d
226     termino1:
227     mov eax,[auxAcu]
228     mov [Num1],eax
229     ret
230
231 convANum2:
232     mov esi,StringNum2
233     mov ebx,[longNum2]
234     dec ebx
235     add esi,ebx
236     mov al,[esi]
237     sub al,'0'
238     mov [auxAcu],al
239     mov ecx,10d
240     mov ebx,[longNum2]
241     mov [auxCon],ebx
242     dec byte [auxCon]
243     a2:
244     cmp byte [auxCon],0d
245     je fin2
246     dec esi
247     mov al,[esi]
248     sub al,'0'
249     mov [aux],al
250     mov eax,[aux]
251     mul ecx
252     add [auxAcu],eax
253     mov eax,ecx
254     mov ecx,10d
255     mul ecx
256     mov ecx,eax
257     dec byte [auxCon]
258     loop a2
259     fin2:
260     cmp byte [longNum2],3
261     jb termino2
262     add byte [auxAcu],1d
263     termino2:
264     mov eax,[auxAcu]
265     mov [Num2],eax
266     ret
267 convAText:
268     mov eax,[resu]
269     mov ecx,10d
270     mov ebx,0d
271     aCAT:
272     xor edx,edx
273     div ecx
274     push edx
```

```
275     inc ebx
276     cmp byte[aux],2d
277     jbe finCAT
278     loop aCAT
279 finCAT:
280     cmp al,1d
281     jb sacar
282     push eax
283     inc ebx
284 sacar:
285     cmp ebx,0d
286     je salir
287     pop ecx
288     mov [aux],ecx
289     add byte[aux], '0'
290     push dword -11
291     call _GetStdHandle@4
292     mov [handle],eax
293     push dword 0
294     push StringNum1Leida
295     push 1
296     push aux
297     push dword [handle]
298     call _WriteConsole@20
299     dec ebx
300     loop sacar
301
302 salir:
303     ret
304
305 mostrarMenu:
306     push dword -11
307     call _GetStdHandle@4
308     mov [handle],eax
309     push dword 0
310     push cadenaEscrita
311     push cadLen
312     push cadenaMsj
313     push dword [handle]
314     call _WriteConsole@20
315     ret
316
317 imprimirMsj1:
318     push dword -11
319     call _GetStdHandle@4
320     mov [handle],eax
321     push dword 0
322     push cadenaEscrita
323     push cadLen1
324     push Msj1
325     push dword [handle]
326     call _WriteConsole@20
327     ret
328
329 imprimirMsj2:
```

```
330 push dword -11
331 call _GetStdHandle@4
332 mov [handle],eax
333 push dword 0
334 push cadenaEscrita
335 push cadLen2
336 push Msj2
337 push dword [handle]
338 call _WriteConsole@20
339 ret
340
341 imprimirMsj3:
342 push dword -11
343 call _GetStdHandle@4
344 mov [handle],eax
345 push dword 0
346 push cadenaEscrita
347 push cadLen3
348 push Msj3
349 push dword [handle]
350 call _WriteConsole@20
351 ret
```

3. Observaciones.

Primeramente, cabe mencionar que el NASM produce principalmente código objeto, que por lo general no son ejecutables por sí mismos. La única excepción a esto son los binarios planos que son inherentemente limitados en el uso moderno. Para traducir los archivos objeto a programas ejecutables, se debe usar un enlazador apropiado.

Además, 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 gestores 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.

4. Análisis Crítico.

Como hemos visto en prácticas anteriores, ambos sistemas operativos tiene diferentes llamadas al sistema y los diferentes códigos que tienen estos. En esta práctica, observamos como utilizar el lenguaje ensamblador NASM tanto en Windows como en Linux con sus diferencias en tanto en compilación como en llamadas al sistema.

Las aplicaciones de un lenguaje ensamblador son extensas, por ejemplo el lenguaje ensamblador es típicamente usado en el ROM de arranque del sistema (BIOS en los sistemas compatible IBM PC). Este código de bajo nivel es usado, entre otras cosas, para inicializar y probar el hardware del sistema antes de cargar el sistema operativo, y está almacenado en el ROM. Una vez que ha tomado lugar un cierto nivel de inicialización del hardware, la ejecución se transfiere a otro código, típicamente escrito en lenguajes de alto nivel; pero el código corriendo inmediatamente después de que es aplicada la energía usualmente está escrito en lenguaje ensamblador.

Muchos compiladores traducen lenguajes de alto nivel a lenguaje ensamblador primero, antes de la compilación completa, permitiendo que el código en ensamblador sea visto para propósitos de depuración y optimización. Lenguajes de relativo bajo nivel, como C, con frecuencia proveen sintaxis especial para empotrar lenguaje ensamblador en cada plataforma de hardware. El código portable del sistema entonces puede usar estos componentes específicos a un procesador a través de una interface uniforme.

El lenguaje ensamblador también es valioso en ingeniería inversa, puesto que muchos programas solamente son distribuidos en una forma de código de máquina. El código de máquina es usualmente fácil de trasladar hacia lenguaje ensamblador para luego ser cuidadosamente examinado en esta forma, pero es muy difícil de trasladar hacia un lenguaje de alto nivel.

5. Conclusiones.

Finalmente, sabemos que las capas del modelo de un sistema operativos requieren un conjunto de interfaces las cuales permiten comunas a las distintas capas que componen al modelo. Estas interfaces contienen funciones de comunicación especializadas entre las capas, anteriormente se verificaron las interfaces de comandos y llamadas al sistema, en la presente práctica se llevo acabo el análisis de las funcionalidades de la capa de interrupciones, se pudo observar que esta capa mantiene las interrupciones a nivel software como también a nivel hardware del equipo de computo. Para obtener la funcionalidad requerida de la interrupción se utiliza un lenguaje de programación a bajo nivel, como fue el lenguaje ensamblador para los sistemas operativos Linux y Windows.

Aunque los lenguajes ensambladores son diferentes de un microprocesador a otro, sin embargo, las funciones que realizan cada uno de ellos son las mismas, es decir, todos los lenguajes ensambladores tienen instrucciones de carga y almacenamiento, aritméticas y lógicas, de salto incondicional, etc. Con estas instrucciones se pueden realizar diferentes programas, como los desarrollados en la presente práctica.