**Sistemas Operativos**

**Práctica 3. Introducción a los sistemas operativos Linux y Windows (3)**

**Grupo: 2CM9**

**Equipo:**

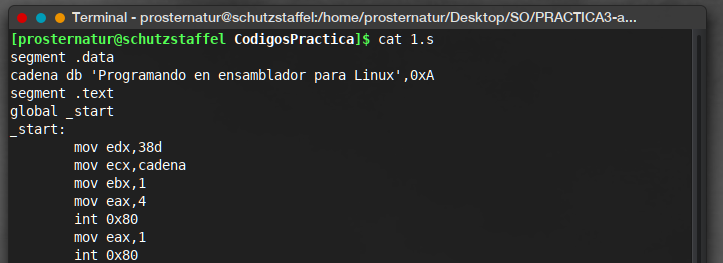
* **Guerra Vargas Irving Cristóbal**
* **Reyes Valenzuela Alejandro**
* **Dávila Méndez Juan Manuel**

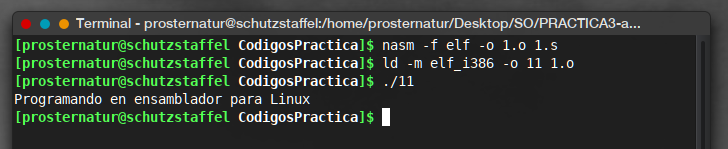
**Competencia.**

* **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 competencia de la estructura general e instrucciones para el lenguaje ensamblador del procesador Intel de 32 bits.**

**Desarrollo (códigos en la práctica)**

Código que imprime “Programando en Ensamblador para Linux”:

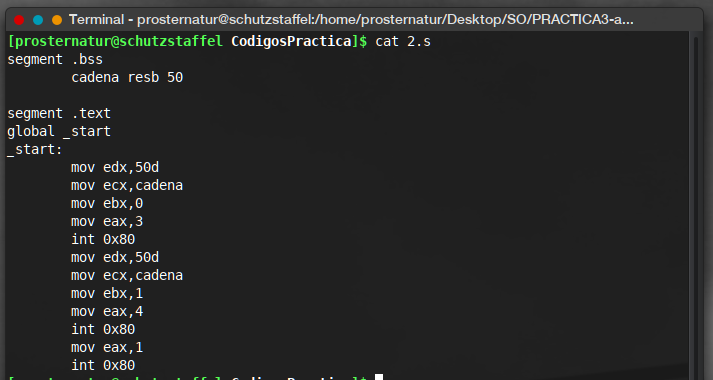


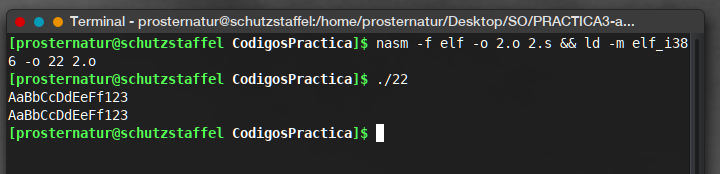
****

Observaciones del código capturado:

Se guarda en los registros (edx,ecx,ebx y eax) ciertos parámetros que se utilizan para distintos propósitos antes de hacer una interrupción. Dichos parámetros pueden decir de entrada (por ejemplo, stdin), de salida (un archivo, stdout, etc…) así como de tamaño (por ejemplo, en edx se guarda la longitud de la cadena).

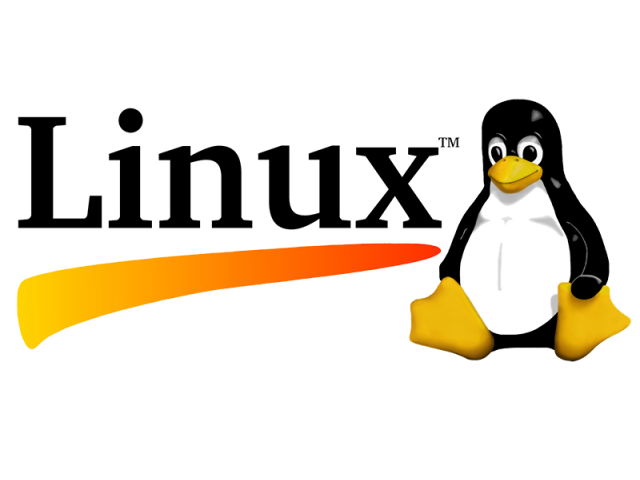
Código que imprime que solicita una cadena y después la imprime:





Observaciones: Similar al programa que sólo imprime una cadena, este programa recibe una cadena y después la muestra, únicamente cambiando los ‘parámetros’ en los registros antes de realizar una interrupción al sistema. Se observa que después de pasar los parámetros es forzosamente necesaria la instrucción para realizar una interrupción al sistema (int 0x80).

**Desarrollo.**

****

**Contador del 0 al 9.**

Programe una aplicación en ensamblador que genere un contador de 0 a 9, mostrando en pantalla el conteo generado.

**Código:**

**segment .data**

**cadena db ' ',**

**longcadena equ $ - cadena**

**segment .bss**

**contador resb 2**

**segment .text**

**global \_start**

**\_start: ;nasm -f elf -o 1 contadorchido.asm && ld -m elf\_i386 -o Cont 1**

**xor edi,edi ; inicializar edi a 0**

**ciclochido:**

**mov ecx, edi ;;;Almacenar contador en ecx**

**add ecx, 0x30 ;;; Valor ASCII**

**mov [contador], ecx ;;;Mover numero al contenido de contador**

**mov ecx, contador ;;;Cadena a escribir**

**mov eax, 4 ;;;Llamada a sys\_write**

**mov ebx, 1 ;;;Salida estandar (stdout)**

**mov edx, 1 ;;;Longitud de la cadena**

**int 0x80 ;;;Llamada al kernel**

**;;; imprimir espacio**

**mov ecx,cadena ;;;Cadena a escribir**

**mov eax,4 ;;;Llamada a sys\_write**

**mov ebx,1 ;;;STDOUT**

**mov edx,longcadena ;;;Logntiud**

**int 0x80**

**inc edi ;incrementar el contador en uno**

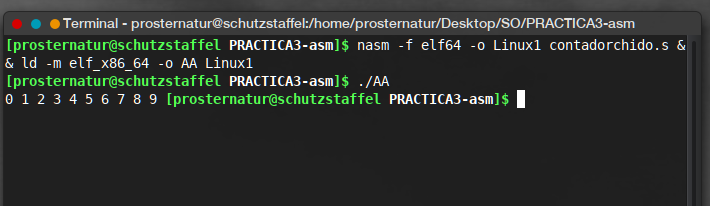
**cmp edi,9 ;si es diferente, seguir con el ciclo**

**jle ciclochido ;jle en lugar de loop para poder usar edi**

**mov eax,1 ;sys\_exit**

**int 0x80 ;llamada al kernel**

**Capturas:**

****

**Copia tres cadenas.**

Programe una aplicación en ensamblador que copie tres cadenas dadas (cadena1, cadena2, cadena3) a una nueva cadena (cadena4). La copia de las cadenas deberá er intercalando los caracteres de cada cadena.

**Código:**

**%macro printer\_chidori 2  
 mov EAX, 4  
 mov EBX, 1  
 mov ECX, %1 ;ArgChar  
 mov EDX, %2 ;ArgTam  
 int 80h  
%endmacro  
%macro receptor\_chidori 2   
 mov eax, 3  
 mov ebx, 0  
 mov ecx, %1 ;ArgChar  
 mov edx, %2 ;ArgTam  
 int 80h  
%endmacro  
%macro MoveREG 4 ;Este macro copia el primer caracter de %2 al argumento1 (%1)  
 mov %1, byte[%2];Se usan los ultimos 8 bytes (registro 8bytes)  
 cmp %1, 0xA ;Se compara si es el terminador  
 je %3 ;En dado caso, se salta al argumento3  
 mov byte[%4], %1;De no ser asi, pasa el primer char a la cadena final(%4)(EDI)  
%endmacro  
segment .data   
 label1 db 'Introduzca la 1er cadena: ', 0xA  
 label2 db 'Introduzca la 2da cadena: ', 0xA  
 label3 db 'Introduzca la 3ra cadena: ', 0xA  
segment .bss   
 str1 resb 50   
 str2 resb 50   
 str3 resb 50   
 Finalstring resb 150   
segment .text   
 global \_start   
 \_start:   
 printer\_chidori label1,26 ;mostrar etiqueta1  
 receptor\_chidori str1, 50 ;ingresa texto1  
 printer\_chidori label2,26   
 receptor\_chidori str2, 50  
 printer\_chidori label3,26  
 receptor\_chidori str3, 50   
 DirectionOfstrings:   
 mov EDI, Finalstring ;cargar en EDI la direcion de Finalstring   
 mov ESI, str1   
 mov EAX, str2   
 mov EBX, str3   
 Movestr1: ;Arg1-8bytes, Arg2-AddrCadenaLeer, Arg3-LabelSalto,Arg4-AddrCadenaDestino  
 MoveREG DL, ESI, Movestr2, EDI  
 inc EDI ;incrementar EDI (cadena final)  
 inc ESI ;incrementar ESI (str1)  
 Movestr2:   
 MoveREG DL, EAX, Movestr3, EDI  
 inc EDI ;incrementar EDI (cadena final)   
 inc EAX ;incrementar EAX (str2)  
 Movestr3:   
 MoveREG DL, EBX, ShowResult, EDI  
 inc EDI ;incrementar EDI (cadena final)  
 inc EBX ;incrementar EBX (str3)  
 jmp Movestr1 ;Regresar a Movestr1  
 ShowResult:  
 mov DL, byte[ESI] ;Verificar nuevamente si str1 no esta vacia  
 cmp DL, 0xA ;comparar con '\0'  
 jne Movestr1 ;si no esta vacia, regresar a Movestr1  
 mov DL, byte[EAX] ;Verificar str2  
 cmp DL, 0xA   
 jne Movestr2 ;regresar a Movestr2 si no esta vacia  
 inc EDI ;agregar uno para meter salto de linea  
 mov byte[EDI],0xA ;incrustar salto de linea  
 printer\_chidori Finalstring,150  
 SecureExit:  
 mov EAX, 1   
 mov EBX, 0   
 int 80h**

**Capturas:**

****

**Longitud de cadena**

Programe una aplicación en ensamblador que muestre en la pantalla la longitud de una cadena que haya sido ingresada por teclado.

**Código:**

**segment .bss**

**cadena resb 1000**

**segment .data**

**msg db 'Caracteres presentes en la cadena: '**

**longmsg equ $ - msg**

**segment .text ;nasm -f elf64 -o ec EntradaCadena.asm && ld -m elf\_x86\_64 -o EX ec**

**global \_start ;se usan espacios de 64bit**

**\_start:**

**mov edx,1000 ;longitud maxima de la cadena a leer**

**mov ecx,cadena ;buffer de entrada**

**mov ebx,0 ;stdin**

**mov eax,3 ;sys\_read**

**int 0x80 ;llamada al sistemma**

**mov edx,longmsg ;;; imprimir mensaje linea 5**

**mov ecx,msg**

**mov ebx,1**

**mov eax,4**

**int 0x80 ;CALCULO DE CARACTERES EN UNA CADENA**

**mov edi,cadena ;cargar la ubicacion de cadena a edi (scasb)**

**mov ecx, 0 ;inicializar ecx a 0**

**mov al, 0 ;inicializar al con el valor nulo**

**not ecx ;valor maximo de ecx (2^32)**

**repne scasb ;repeat-not-equal-al -- usar funcion del procesador para contar bytes**

**not ecx ;extrae un uno de ecx**

**add ecx,-2 ;extrae 2 porque tambien se cuenta el caracter nulo**

**mov edi,ecx ;en edi se carga el numero que contiene ecx, para la subrutina**

**call print\_int ;llamar subrutina para imprimir**

**mov eax,1 ;sys\_exit**

**int 0x80 ;;; SUBRUTINA PARA IMPRIMIR ENTEROS**

**global print\_int ;print\_int recibe en edi el num**

**print\_int:**

**mov eax, edi ; en eax se guarda el num proveniente de edi**

**mov ecx, 10 ; dividir en base 10**

**push rcx ; ecx se agrega al stack con el registro rcx de 64bit**

**mov rsi, rsp**

**sub rsp, 16 ;utilizar 16bytes para la zona de contenidos**

**.toascii\_digit: ;ciclo para convertir a ascii el digito obtenido**

**mov edx,0 ;se limpia edx**

**div ecx ;en edx se almacena el residuo**

**add edx, '0' ;se agrega cero a edx**

**dec rsi**

**mov [rsi], dl ;solo se buscan los ultimos bytes (primer digito)**

**test eax,eax ;mientras eax no sea 0**

**jnz .toascii\_digit ;condicional**

**mov eax, 1 ; Escritura**

**mov edi, 1 ;stdout**

**lea edx, [rsp+16+1] ;Se trunca el apuntador +1 por el caracter nulo**

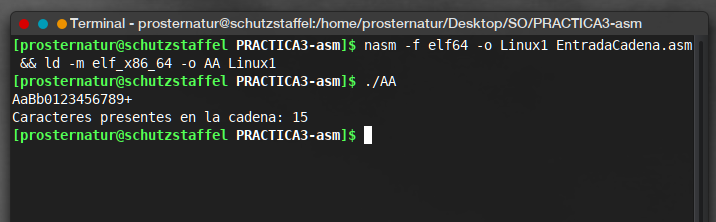
**sub edx, esi ;Pasar la longitud a edx**

**syscall ;llamada al sistema (64bit)**

**add rsp, 24 ;regresar 24bytes que se reservaron anteriormente**

**ret ;retorno**

**Capturas:**

****

**Concatenar 10 cadenas**

Programe una aplicación en ensamblador que concatene diez cadenas.

**Código:**

**segment .data**

**msg1 db 'Cadena 1:',0xA**

**longitud1 equ $-msg1**

**msg2 db 'Cadena 2:',0xA**

**longitud2 equ $-msg2**

**msg3 db 'Cadena 3:',0xA**

**longitud3 equ $-msg3**

**msg4 db 'Cadena 4:',0xA**

**longitud4 equ $-msg4**

**msg5 db 'Cadena 5:',0xA**

**longitud5 equ $-msg5**

**msg6 db 'Cadena 6:',0xA**

**longitud6 equ $-msg6**

**msg7 db 'Cadena 7:',0xA**

**longitud7 equ $-msg7**

**msg8 db 'Cadena 8:',0xA**

**longitud8 equ $-msg8**

**msg9 db 'Cadena 9:',0xA**

**longitud9 equ $-msg9**

**msg10 db 'Cadena 10:',0xA**

**longitud10 equ $-msg10**

**msg11 db 'Concatenada:',0xA**

**longitud11 equ $-msg11**

**msg12 db 'Inversa:',0xA**

**longitud12 equ $-msg12**

**msg13 db 'Longitud:',0xA**

**longitud13 equ $-msg13**

**segment .bss**

**sub1 resb 20**

**sub2 resb 20**

**sub3 resb 20**

**sub4 resb 20**

**sub5 resb 20**

**sub6 resb 20**

**sub7 resb 20**

**sub8 resb 20**

**sub9 resb 20**

**sub10 resb 20**

**cdtta resb 100**

**ttdac resb 100**

**attdc resb 100**

**lona resb 8**

**lonb resb 8**

**cont1 resb 8**

**cont2 resb 8**

**segment .text**

**global \_start**

**\_start:**

**mov byte [cont1],0x30**

**mov byte [cont2],0x30**

**mov byte [lona],0x30**

**mov byte [lonb],0x30**

**mov eax,4**

**mov ebx,1**

**mov ecx,msg1**

**mov edx,longitud1**

**int 0x80**

**mov eax,3**

**mov ebx,0**

**mov ecx,sub1**

**mov edx,20d**

**int 0x80**

**mov eax,4**

**mov ebx,1**

**mov ecx,msg2**

**mov edx,longitud2**

**int 0x80**

**mov eax,3**

**mov ebx,0**

**mov ecx,sub2**

**mov edx,20d**

**int 0x80**

**mov eax,4**

**mov ebx,1**

**mov ecx,msg3**

**mov edx,longitud3**

**int 0x80**

**mov eax,3**

**mov ebx,0**

**mov ecx,sub3**

**mov edx,50d**

**int 0x80**

**mov eax,4**

**mov ebx,1**

**mov ecx,msg4**

**mov edx,longitud4**

**int 0x80**

**mov eax,3**

**mov ebx,0**

**mov ecx,sub4**

**mov edx,50d**

**int 0x80**

**mov eax,4**

**mov ebx,1**

**mov ecx,msg5**

**mov edx,longitud5**

**int 0x80**

**mov eax,3**

**mov ebx,0**

**mov ecx,sub5**

**mov edx,20d**

**int 0x80**

**mov eax,4**

**mov ebx,1**

**mov ecx,msg6**

**mov edx,longitud6**

**int 0x80**

**mov eax,3**

**mov ebx,0**

**mov ecx,sub6**

**mov edx,50d**

**int 0x80**

**mov eax,4**

**mov ebx,1**

**mov ecx,msg7**

**mov edx,longitud7**

**int 0x80**

**mov eax,3**

**mov ebx,0**

**mov ecx,sub7**

**mov edx,50d**

**int 0x80**

**mov eax,4**

**mov ebx,1**

**mov ecx,msg8**

**mov edx,longitud8**

**int 0x80**

**mov eax,3**

**mov ebx,0**

**mov ecx,sub8**

**mov edx,50d**

**int 0x80**

**mov eax,4**

**mov ebx,1**

**mov ecx,msg9**

**mov edx,longitud9**

**int 0x80**

**mov eax,3**

**mov ebx,0**

**mov ecx,sub9**

**mov edx,50d**

**int 0x80**

**mov eax,4**

**mov ebx,1**

**mov ecx,msg10**

**mov edx,longitud10**

**int 0x80**

**mov eax,3**

**mov ebx,0**

**mov ecx,sub10**

**mov edx,50d**

**int 0x80**

**mov esi,sub1**

**mov edi,cdtta**

**count:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne count**

**mov esi,sub2**

**sub edi,2**

**concatenar1:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne concatenar1**

**mov esi,sub3**

**sub edi,2**

**concatenar2:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne concatenar2**

**mov esi,sub4**

**sub edi,2**

**concatenar3:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne concatenar3**

**mov esi,sub5**

**sub edi,2**

**concatenar4:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne concatenar4**

**mov esi,sub6**

**sub edi,2**

**concatenar5:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne concatenar5**

**mov esi,sub7**

**sub edi,2**

**concatenar6:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne concatenar6**

**mov esi,sub8**

**sub edi,2**

**concatenar7:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne concatenar7**

**mov esi,sub9**

**sub edi,2**

**concatenar8:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne concatenar8**

**mov esi,sub10**

**sub edi,2**

**concatenar9:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne concatenar9**

**int 0x80**

**mov eax,4**

**mov ebx,1**

**mov ecx,msg11**

**mov edx,longitud11**

**int 0x80**

**mov eax,4**

**mov ebx,1**

**mov ecx,cdtta**

**mov edx,100d**

**int 0x80**

**mov esi,sub10**

**mov edi,ttdac**

**contar:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne contar**

**mov esi,sub9**

**sub edi,2**

**Str1a:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne Str1a**

**mov esi,sub8**

**sub edi,2**

**Str2a:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne Str2a**

**mov esi,sub7**

**sub edi,2**

**Str3a:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne Str3a**

**mov esi,sub6**

**sub edi,2**

**Str4a:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne Str4a**

**mov esi,sub5**

**sub edi,2**

**Str5a:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne Str5a**

**mov esi,sub4**

**sub edi,2**

**Str6a:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne Str6a**

**mov esi,sub3**

**sub edi,2**

**Str7a:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne Str7a**

**mov esi,sub3**

**sub edi,2**

**Str8a:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne Str8a**

**mov esi,sub2**

**sub edi,2**

**Str9a:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne Str9a**

**mov esi,sub1**

**sub edi,2**

**Str10a:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp al,0**

**jne Str10a**

**mov eax,4**

**mov ebx,1**

**mov ecx,msg12**

**mov edx,longitud12**

**int 0x80**

**mov eax,4**

**mov ebx,1**

**mov ecx,ttdac**

**mov edx,100d**

**int 0x80**

**mov edi,cdtta**

**mov esi,attdc**

**fill:**

**cmp byte [edi],0x00**

**je inverse**

**inc edi**

**inc byte [cont1]**

**inc byte [cont2]**

**jmp fill**

**inverse:**

**cmp byte [cont2],0x30**

**je sort**

**mov al,[edi-2]**

**mov [esi],al**

**dec byte [cont2]**

**dec edi**

**inc esi**

**jmp inverse**

**sort:**

**mov esi,cdtta**

**mov edi,cdtta**

**countB:**

**mov al,[esi]**

**inc esi**

**mov [edi],al**

**inc edi**

**cmp byte [lona],0x39**

**jne sumA**

**mov byte [lona],0x30**

**cmp byte [lonb],0x39**

**jne sumB**

**mov byte [lonb],0x30**

**jmp OutN**

**sumB:**

**add byte [lonb],1d**

**jmp OutN**

**sumA:**

**add byte [lona],1d**

**OutN:**

**cmp al,0**

**jne countB**

**int 0x80**

**mov eax,4**

**mov ebx,1**

**mov ecx,msg13**

**mov edx,longitud13**

**int 0x80**

**sub byte[lona],2d**

**cmp byte[lona],'.'**

**jne Omite**

**mov byte[lona],'8'**

**sub byte[lonb],1d**

**Omite:**

**cmp byte[lona],'/'**

**jne Omitir**

**mov byte[lona],'9'**

**sub byte[lonb],1d**

**Omitir:**

**cmp byte[lonb],0x30**

**je ImprimeCad**

**mov eax,4**

**mov ebx,1**

**mov ecx,lonb**

**mov edx,8d**

**int 0x80**

**ImprimeCad:**

**mov eax,4**

**mov ebx,1**

**mov ecx,lona**

**mov edx,8d**

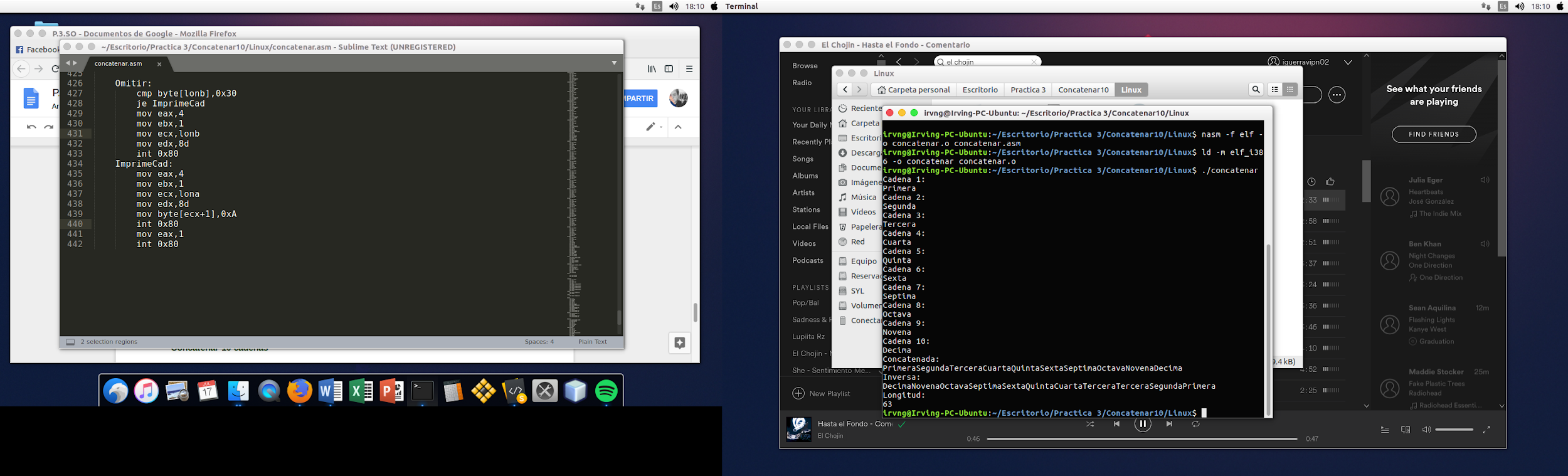
**mov byte[ecx+1],0xA**

**int 0x80**

**mov eax,1**

**int 0x80**

**Capturas:**

****

**Calculadora**

Programe una aplicación en ensamblador que implemente una calculadora con las cuatro operaciones básicas.

**Código:**

**segment .data**

**msg1 db "--> Calculadora en Ensamblador <--",0xA,0xD**

**longitud1 equ $-msg1**

**msgSuma db 0xA,0xD,"1.-Suma" ,0xA,0xD**

**longitudSuma equ $-msgSuma**

**msgResta db "2.-Resta" ,0xA,0xD**

**longitudResta equ $-msgResta**

**msgMultiplicacion db "3.-Mulltiplicacion",0xA,0xD**

**longitudMultiplicacion equ $-msgMultiplicacion**

**msgDivision db "4.-Division" ,0xA,0xD**

**longitudDivision equ $-msgDivision**

**msgOpcion db 0xA,0xD,"Digite la opcion: "**

**longitudOpcion equ $-msgOpcion**

**msgNumero1 db "Digita el numero 1: "**

**longitudNum1 equ $-msgNumero1**

**msgNumero2 db "Digita el numero 2: "**

**longitudNum2 equ $-msgNumero2**

**msgOpcionInvalida db "Error de opcion",0xA**

**longitudInvalida equ $-msgOpcionInvalida**

**msgFinal db 0xD,"Fin de programa",0xA**

**longitudFinal equ $-msgFinal**

**msgResultado db 0xD,"El resultado es: ",0xA**

**longitudResultado equ $-msgResultado**

**SaltoDeLinea db 0xA**

**longitudSalto equ $-SaltoDeLinea**

**segment .bss**

**opc resb 1**

**num1 resw 2**

**num2 resw 2**

**resultado resw 2**

**segment .text**

**global \_start**

**\_start:**

**mov eax,4**

**mov ebx,1**

**mov ecx,msg1**

**mov edx,longitud1**

**int 80h**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgSuma**

**mov edx,longitudSuma**

**int 80h**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgResta**

**mov edx,longitudResta**

**int 80h**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgMultiplicacion**

**mov edx,longitudMultiplicacion**

**int 80h**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgDivision**

**mov edx,longitudDivision**

**int 80h**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgOpcion**

**mov edx,longitudOpcion**

**int 80h**

**mov eax,3**

**mov ebx,0**

**mov ecx,opc**

**mov edx,2**

**int 80h**

**;pasa a ASCII**

**mov ah,[opc]**

**sub ah, '0'**

**cmp ah,1**

**je suma**

**cmp ah,2**

**je resta**

**cmp ah,3**

**je multiplicar**

**cmp ah,4**

**je dividir**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgOpcionInvalida**

**mov edx,longitudInvalida**

**int 80h**

**jmp salida**

**suma:**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgNumero1**

**mov edx,longitudNum1**

**int 80h**

**mov eax,3**

**mov ebx,0**

**mov ecx,num1**

**mov edx,2**

**int 80h**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgNumero2**

**mov edx,longitudNum2**

**int 80h**

**mov eax,3**

**mov ebx,0**

**mov ecx,num2**

**mov edx,2**

**int 80h**

**mov eax, 4**

**mov ebx, 1**

**mov ecx, msgResultado**

**mov edx, longitudResultado**

**int 80h**

**mov eax, [num1]**

**mov ebx, [num2]**

**sub eax, '0'**

**sub ebx, '0'**

**add eax,ebx**

**add eax,'0'**

**mov [resultado],eax**

**mov eax,4**

**mov ebx,1**

**mov ecx,resultado**

**mov edx,1**

**int 80h**

**jmp salida**

**resta:**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgNumero1**

**mov edx,longitudNum1**

**int 80h**

**mov eax,3**

**mov ebx,0**

**mov ecx,num1**

**mov edx,2**

**int 80h**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgNumero2**

**mov edx,longitudNum2**

**int 80h**

**mov eax,3**

**mov ebx,0**

**mov ecx,num2**

**mov edx,2**

**int 80h**

**mov eax, 4**

**mov ebx, 1**

**mov ecx, msgResultado**

**mov edx, longitudResultado**

**int 80h**

**mov eax,[num1]**

**mov ebx,[num2]**

**sub eax,ebx**

**add eax,'0'**

**mov [resultado], eax**

**mov eax,4**

**mov ebx,1**

**mov ecx,resultado**

**mov edx,1**

**int 80h**

**jmp salida**

**multiplicar:**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgNumero1**

**mov edx,longitudNum1**

**int 80h**

**;Leer num1**

**mov eax,3**

**mov ebx,0**

**mov ecx,num1**

**mov edx,2**

**int 80h**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgNumero2**

**mov edx,longitudNum2**

**int 80h**

**mov eax,3**

**mov ebx,0**

**mov ecx,num2**

**mov edx,2**

**int 80h**

**mov eax, 4**

**mov ebx, 1**

**mov ecx, msgResultado**

**mov edx, longitudResultado**

**int 80h**

**mov eax,[num1]**

**mov ebx,[num2]**

**sub eax,'0'**

**sub ebx,'0'**

**mul ebx**

**add eax,'0'**

**mov [resultado],eax**

**;Se imprime el resultado**

**mov eax,4**

**mov ebx,1**

**mov ecx,resultado**

**mov edx,1**

**int 80h**

**jmp salida**

**dividir:**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgNumero1**

**mov edx,longitudNum1**

**int 80h**

**mov eax,3**

**mov ebx,0**

**mov ecx,num1**

**mov edx,2**

**int 80h**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgNumero2**

**mov edx,longitudNum2**

**int 80h**

**mov eax,3**

**mov ebx,0**

**mov ecx,num2**

**mov edx,2**

**int 80h**

**mov eax, 4**

**mov ebx, 1**

**mov ecx, msgResultado**

**mov edx, longitudResultado**

**int 80h**

**mov al,[num1]**

**mov bl,[num2]**

**sub al,'0'**

**sub bl,'0'**

**div bl**

**add ax,'0'**

**mov [resultado],ax**

**mov eax,4**

**mov ebx,1**

**mov ecx,resultado**

**mov edx,1**

**int 80h**

**jmp salida**

**salida:**

**mov eax,4**

**mov ebx,1**

**mov ecx,SaltoDeLinea**

**mov edx,longitudSalto**

**int 80h**

**mov eax,4**

**mov ebx,1**

**mov ecx,msgFinal**

**mov edx,longitudFinal**

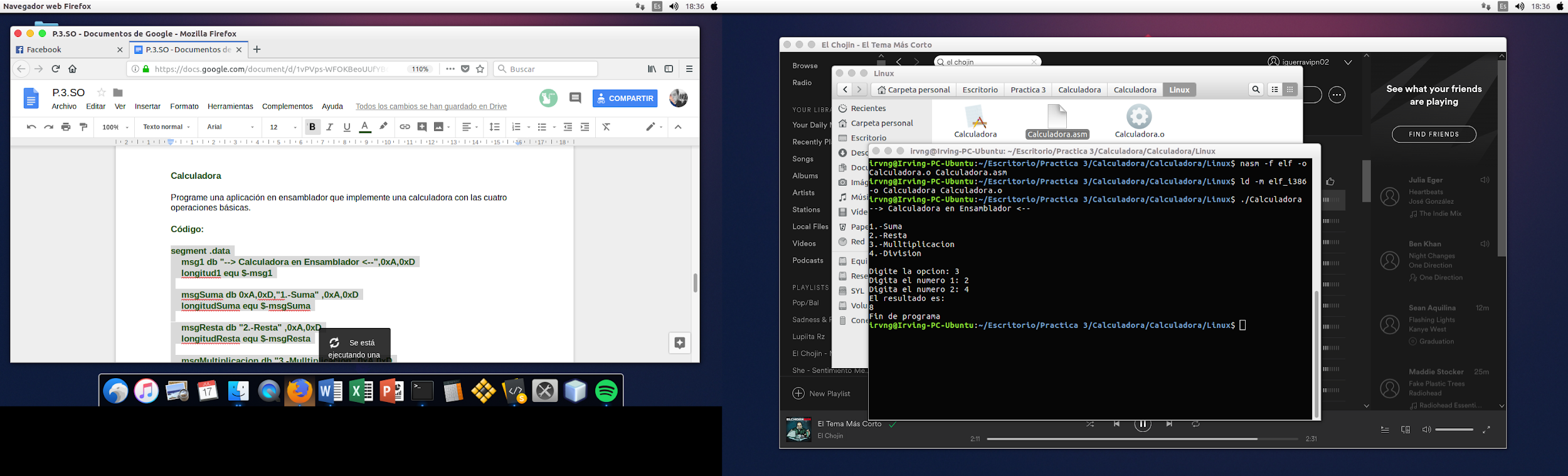
**int 80h**

**mov eax,1**

**mov ebx,0**

**int 80h**

**Capturas:**

****

****

**Contador**

**Código:**

**segment .data**

**handleConsola dd 1 ;1er argumento para la llamada al sistema \_WriteConsoleA()**

**cadenaImprimir db '0',0xA ;El 10=0xA es para que haga un salto de linea**

**longitudCadena dd 2 ;3er argumento para la llamada al sistema \_WriteConsoleA()**

**;Tamaño=2, se cuenta el salto de linea**

**caractEscritos dd 1 ;4to argumento para la llamada al sistema \_WriteConsoleA()**

**ultimoArgumento dd 1 ;5to argumento para la llamada al sistema \_WriteConsoleA()**

**counter dd 0**

**segment .text**

**global \_main**

**extern \_GetStdHandle@4 ;Acceso externo a la llamada al sistema \_GetStdHandle()**

**extern \_WriteConsoleA@20 ;Acceso externo a la llamada al sistema \_WriteConsoleA()**

**extern \_ExitProcess@4 ;Acceso externo a la llamada al sistema \_ExitProcess()**

**\_main:**

**push dword -11 ;Argumento pasado por la pila y usado en \_GetStdHandle() para la salida estándar**

**call \_GetStdHandle@4 ;Invocación de \_GetStdHandle()**

**mov [handleConsola],eax ;Devolución del manejador de consola para escritura en el registro eax**

**xor eax,eax ;Limpieza del registro eax (eax=0)**

**mov eax,2d ;longitud de cadena a imprimir**

**mov [longitudCadena],eax ;Se guarda la longitud en memoria**

**xor eax,eax ;Limpieza del registro eax (eax=0)**

**mov eax,0d**

**mov [ultimoArgumento],eax ;Valor del 5to argumento para \_WriteConsoleA()**

**;Loop**

**loop:**

**push dword [ultimoArgumento] ;5to argumento de \_WriteConsoleA() pasado por pila**

**push dword caractEscritos ;4to argumento de \_WriteConsoleA() pasado por pila**

**push dword [longitudCadena] ;3er argumento de \_WriteConsoleA() pasado por pila**

**push dword cadenaImprimir ;2do argumento de \_WriteConsoleA() pasado por pila**

**push dword [handleConsola] ;1er argumento de \_WriteConsoleA() pasado por pila**

**call \_WriteConsoleA@20 ;Invocación de \_WriteConsoleA()**

**inc byte [cadenaImprimir] ;Agrega 1 byte al primer byte del numero**

**inc dword [counter] ;Agrega 1 a la direccion de memoria**

**mov eax, [counter] ;Se guarda en memoria**

**cmp eax,10 ;Regresa al ciclo mientras sea menor a 10**

**jb loop**

**xor eax,eax ;Limpieza del registro eax (eax=0)**

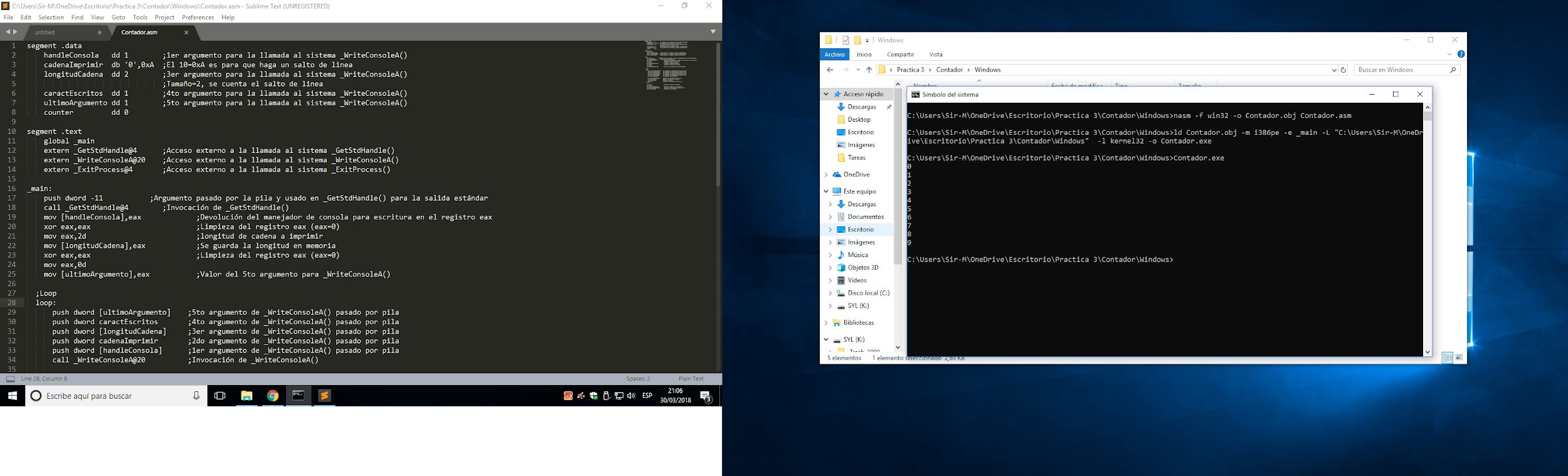
**mov eax,0d**

**mov [ultimoArgumento],eax ;Valor del argumento para \_ExitProcess()**

**push dword [ultimoArgumento] ;Argumento de \_ExitProcess() pasado por pila**

**call \_ExitProcess@4 ;Invocación de \_ExitProcess**

**Capturas:**

****

**Intercalar**

**Código:**

**segment .data**

**cont dd 0**

**contP dd 0**

**msg1 db 'Introduzca la 1ra cadena: '**

**msg2 db 'Introduzca la 2da cadena: '**

**msg3 db 'Introduzca la 3er cadena: '**

**msg4 db 'Cadena final: '**

**%macro entrada\_cadena 2 ;argumento uno cadena**

**push dword -10 ;argumento dos longitud cadena**

**call \_GetStdHandle@4**

**mov [handleConsola],eax**

**xor eax,eax**

**mov eax,%2**

**mov [longitudCadena],eax**

**xor eax,eax**

**push 0**

**push dword caractLeidos**

**push dword [longitudCadena]**

**push dword %1 ;unico argumento**

**push dword [handleConsola]**

**call \_ReadConsoleA@20**

**xor eax,eax**

**%endmacro**

**%macro imp\_texto 2 ;un argumento es la cadena y el otro es la**

**push dword -11 ;longitud de la cadena**

**call \_GetStdHandle@4**

**mov [handleConsola],eax**

**xor eax,eax**

**mov eax, %2**

**mov [longitudCadena],eax**

**xor eax,eax**

**mov eax,0**

**mov [ultArg],eax**

**push dword [ultArg]**

**push dword caracEscritos**

**push dword [longitudCadena]**

**push dword %1**

**push dword [handleConsola]**

**call \_WriteConsoleA@20**

**xor eax,eax**

**%endmacro**

**segment .bss**

**handleConsola resd 1**

**longitudCadena resd 1**

**caractLeidos resd 1**

**caracEscritos resd 1**

**ultArg resd 1**

**strA resb 30**

**strB resb 30**

**strC resb 30**

**strFinal resb 90**

**segment .text**

**global \_main**

**extern \_GetStdHandle@4**

**extern \_ReadConsoleA@20**

**extern \_ExitProcess@4**

**extern \_WriteConsoleA@20**

**\_main:**

**imp\_texto msg1,26**

**entrada\_cadena strA,30**

**imp\_texto msg2,26**

**entrada\_cadena strB,30**

**imp\_texto msg3,26**

**entrada\_cadena strC,30**

**xor eax,eax**

**ciclop:**

**mov eax,strA**

**add eax,[contP]**

**mov esi, eax ;origen strA+contP**

**xor eax,eax**

**mov eax,strFinal**

**add eax,[cont]**

**mov edi, eax ;destino strFinal+cont**

**xor eax,eax**

**movsb**

**inc dword [cont]**

**mov eax,strB**

**add eax,[contP]**

**mov esi, eax ;origen strA+contP**

**xor eax,eax**

**mov eax,strFinal**

**add eax,[cont]**

**mov edi, eax ;destino strFinal+cont**

**xor eax,eax**

**movsb**

**inc dword [cont]**

**mov eax,strC**

**add eax,[contP]**

**mov esi, eax ;origen strA+contP**

**xor eax,eax**

**mov eax,strFinal**

**add eax,[cont]**

**mov edi, eax ;destino strFinal+cont**

**xor eax,eax**

**movsb**

**inc dword [cont]**

**inc dword [contP]**

**xor eax,eax**

**mov eax,[cont]**

**cmp eax,90**

**jb ciclop ;sale del ciclo hasta comparar los 90 espacios**

**xor eax,eax**

**imp\_texto msg4,16**

**imp\_texto strFinal,90**

**push 0**

**call \_ExitProcess@4**

**segment .data**

**cont dd 0**

**contP dd 0**

**msg1 db 'Introduzca la 1ra cadena: '**

**msg2 db 'Introduzca la 2da cadena: '**

**msg3 db 'Introduzca la 3er cadena: '**

**msg4 db 'Cadena final: '**

**%macro entrada\_cadena 2 ;argumento uno cadena**

**push dword -10 ;argumento dos longitud cadena**

**call \_GetStdHandle@4**

**mov [handleConsola],eax**

**xor eax,eax**

**mov eax,%2**

**mov [longitudCadena],eax**

**xor eax,eax**

**push 0**

**push dword caractLeidos**

**push dword [longitudCadena]**

**push dword %1 ;unico argumento**

**push dword [handleConsola]**

**call \_ReadConsoleA@20**

**xor eax,eax**

**%endmacro**

**%macro imp\_texto 2 ;un argumento es la cadena y el otro es la**

**push dword -11 ;longitud de la cadena**

**call \_GetStdHandle@4**

**mov [handleConsola],eax**

**xor eax,eax**

**mov eax, %2**

**mov [longitudCadena],eax**

**xor eax,eax**

**mov eax,0**

**mov [ultArg],eax**

**push dword [ultArg]**

**push dword caracEscritos**

**push dword [longitudCadena]**

**push dword %1**

**push dword [handleConsola]**

**call \_WriteConsoleA@20**

**xor eax,eax**

**%endmacro**

**segment .bss**

**handleConsola resd 1**

**longitudCadena resd 1**

**caractLeidos resd 1**

**caracEscritos resd 1**

**ultArg resd 1**

**strA resb 30**

**strB resb 30**

**strC resb 30**

**strFinal resb 90**

**segment .text**

**global \_main**

**extern \_GetStdHandle@4**

**extern \_ReadConsoleA@20**

**extern \_ExitProcess@4**

**extern \_WriteConsoleA@20**

**\_main:**

**imp\_texto msg1,26**

**entrada\_cadena strA,30**

**imp\_texto msg2,26**

**entrada\_cadena strB,30**

**imp\_texto msg3,26**

**entrada\_cadena strC,30**

**xor eax,eax**

**ciclop:**

**mov eax,strA**

**add eax,[contP]**

**mov esi, eax ;origen strA+contP**

**xor eax,eax**

**mov eax,strFinal**

**add eax,[cont]**

**mov edi, eax ;destino strFinal+cont**

**xor eax,eax**

**movsb**

**inc dword [cont]**

**mov eax,strB**

**add eax,[contP]**

**mov esi, eax ;origen strA+contP**

**xor eax,eax**

**mov eax,strFinal**

**add eax,[cont]**

**mov edi, eax ;destino strFinal+cont**

**xor eax,eax**

**movsb**

**inc dword [cont]**

**mov eax,strC**

**add eax,[contP]**

**mov esi, eax ;origen strA+contP**

**xor eax,eax**

**mov eax,strFinal**

**add eax,[cont]**

**mov edi, eax ;destino strFinal+cont**

**xor eax,eax**

**movsb**

**inc dword [cont]**

**inc dword [contP]**

**xor eax,eax**

**mov eax,[cont]**

**cmp eax,90**

**jb ciclop ;sale del ciclo hasta comparar los 90 espacios**

**xor eax,eax**

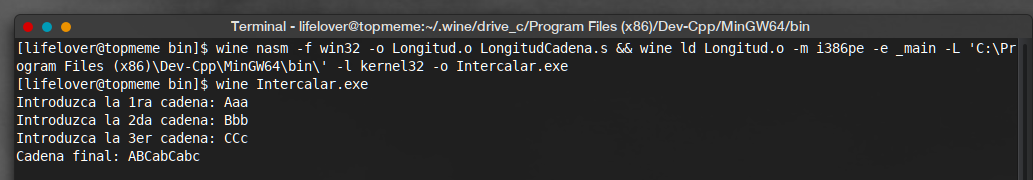
**imp\_texto msg4,16**

**imp\_texto strFinal,90**

**push 0**

**call \_ExitProcess@4**

**Capturas:**

****

**Longitud**

**Código:**

**segment .data**

**msg db "Ingresa una cadena para obtener su longitud:", 10**

**longitud1 equ $-msg**

**msg2 db "La longitud de la cadena ingresada es: "**

**longitud2 equ $-msg2**

**handleSalida dd 0**

**caracEscritos dd 0**

**handleEntrada dd 0**

**caracLeidos dd 0**

**longCad dd 0**

**longCadFinal dd 0**

**segment .bss**

**cadena resb 100**

**segment .text**

**global \_main**

**extern \_GetStdHandle@4**

**extern \_WriteConsoleA@20**

**extern \_ReadConsoleA@20**

**extern \_ExitProcess@4**

**\_main:**

**push -11 ;Argumento pasado por la pila y usado en \_GetStdHandle() para la salida estándar**

**call \_GetStdHandle@4 ;Invocación de \_GetStdHandle()**

**mov [handleSalida], eax ;Devolución del manejador de consola para escritura en el registro eax**

**;Imprimir msg**

**push 0 ;5to argumento de \_WriteConsoleA() pasado por pila**

**push dword caracEscritos ;4to argumento de \_WriteConsoleA() pasado por pila**

**push longitud1 ;3er argumento de \_WriteConsoleA() pasado por pila**

**push dword msg ;2do argumento de \_WriteConsoleA() pasado por pila**

**push dword [handleSalida] ;1er argumento de \_WriteConsoleA() pasado por pila**

**call \_WriteConsoleA@20 ;Invocación de \_WriteConsoleA()**

**;Cambiar a modo lectura**

**xor eax,eax ;Limpieza del registro eax (eax=0)**

**push dword -10 ;Argumento pasado por la pila y usado en \_GetStdHandle() para la entrada estándar**

**call \_GetStdHandle@4 ;Invocación de \_GetStdHandle()**

**mov [handleEntrada],eax ;Devolución del manejador de consola para lectura en el registro eax**

**;leer cadena**

**push 0 ;5to argumento de \_ReadConsoleA() pasado por pila**

**push dword caracLeidos ;4to argumento de \_ReadConsoleA() pasado por pila**

**push 100 ;3er argumento de \_ReadConsoleA() pasado por pila**

**push dword cadena ;2do argumento de \_ReadConsoleA() pasado por pila**

**push dword [handleEntrada] ;1er argumento de \_ReadConsoleA() pasado por pila**

**call dword \_ReadConsoleA@20 ;Invocación de \_ReadConsole**

**xor esi, esi ;Apuntador a origen para operaciones con cadenas de caracteres**

**;Ciclo para conteo de caracteres de la cadena**

**loop1:**

**mov al, [cadena+esi]**

**cmp al, 0**

**je continuar**

**cmp al, 10**

**je continuar**

**cmp al, 13**

**je continuar**

**inc dword [longCad]**

**inc esi**

**jmp loop1**

**continuar:**

**xor eax,eax**

**mov eax, [longCad]**

**loop2:**

**xor edx,edx**

**mov edx,0**

**mov ecx, 10**

**div ecx**

**add edx,48**

**push edx ;Actualiza y se coloca el nuevo valor en el tope de la pila**

**cmp eax,0 ;Se mantiene en el loop si es distinto de 0**

**jne loop2**

**;Imprimir msg2**

**push 0**

**push dword caracEscritos**

**push dword longitud2**

**push dword msg2**

**push dword [handleSalida]**

**call \_WriteConsoleA@20**

**;Imprimir longitud de cadena**

**imprimir:**

**pop edx ;lee el valor del tope de la pila**

**cmp edx, 48 ;Salta si es menor a 48**

**jl terminar**

**cmp edx, 57 ;salta si es mayor a 57**

**jg terminar**

**mov [longCadFinal], edx**

**;imprimir la longitud de cadena**

**push 0 ;5to argumento de \_WriteConsoleA() pasado por pila**

**push dword caracEscritos ;4to argumento de \_WriteConsoleA() pasado por pila**

**push dword 1 ;3er argumento de \_WriteConsoleA() pasado por pila**

**push dword longCadFinal ;2do argumento de \_WriteConsoleA() pasado por pila**

**push dword [handleSalida] ;1er argumento de \_WriteConsoleA() pasado por pila**

**call \_WriteConsoleA@20 ;Invocación de \_WriteConsoleA()**

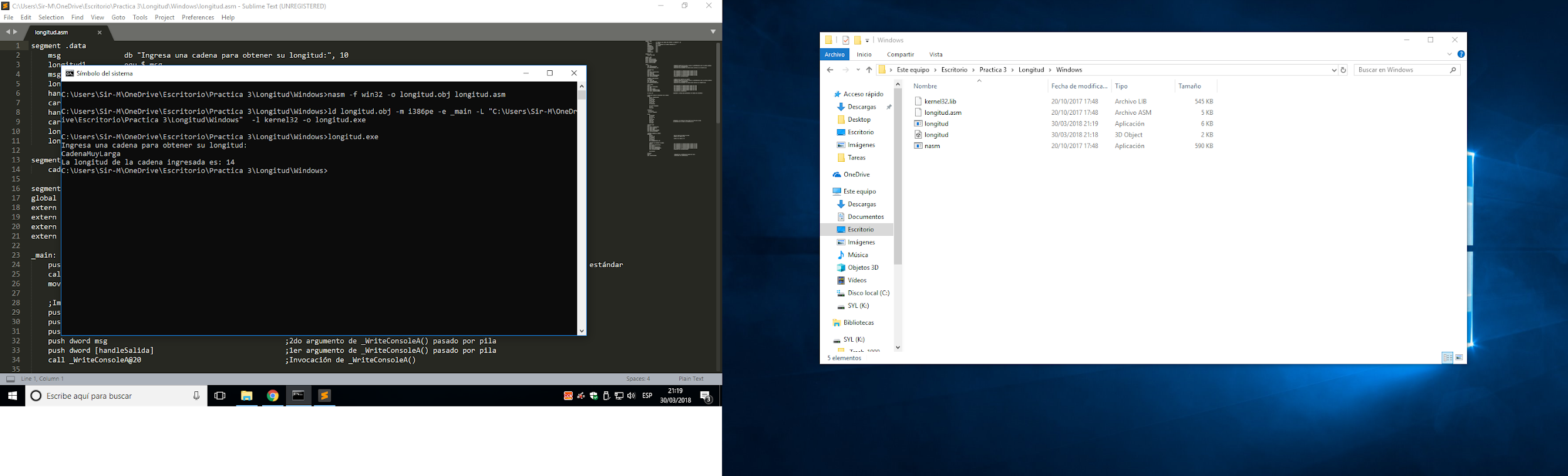
**jmp imprimir**

**terminar:**

**push 0 ;Argumento de \_ExitProcess() pasado por pila**

**call \_ExitProcess@4 ;Invocación de \_ExitProcess()**

**Capturas:**

****

**Calculadora**

**Código:**

**global \_main**

**extern \_GetStdHandle@4**

**extern \_ExitProcess@4**

**extern \_WriteConsole@20**

**extern \_ReadConsole@20**

**segment .data**

**cadImprimir: db 10,10,'CALCULADORA',10,'1) SUMA',10,'2) RESTA',10,'3)´PRODUCTO',10,'4) COCIENTE',10,10,'SELECC: ',0xA ;Impresión del menú**

**cadena equ $ - cadImprimir ;Leemos la opción deseada**

**mensajeNum1: db 'NUM1: ',0**

**pideNum1 equ $ - mensajeNum1**

**mensajeNum2: db 'NUM2: ',0**

**pideNum2 equ $ - mensajeNum2 ;pedimos los operandos**

**rsuma: db 'SUMA: ',0**

**lrsuma equ $ - rsuma ;Impresión de resultados acorde a las opciones deseadas**

**rresta: db 'RESTA: ',0**

**lrresta equ $ - rresta**

**rmulti: db 'PRODUCTO: ',0**

**lrmulti equ $ - rmulti**

**rdiv: db 'COCIENTE: ',0**

**lrdiv equ $ - rdiv**

**handle: db 0**

**USERINPUT: db 0**

**READSTRING: db 0**

**segment .bss ;Declaración de variables contenedoras que utilizaran las operaciones**

**StringNum1 resb 30**

**StringNum2 resb 30**

**longNum1 resb 30**

**longNum2 resb 30**

**Num1 resb 30**

**Num2 resb 30**

**resu resb 30**

**resuDiv resb 30**

**auxAcu resb 30**

**aux resb 30**

**auxCon resb 30**

**op resb 2**

**segment .text**

**\_main:**

**call PRINTMENU ;Llamamos al menu**

**push dword -10**

**call \_GetStdHandle@4**

**mov [handle],eax**

**push dword 0d**

**push READSTRING**

**push 5d**

**push op**

**push dword [handle]**

**call \_ReadConsole@20**

**call ASKFORNUM1**

**push dword -10**

**call \_GetStdHandle@4**

**mov [handle],eax**

**push dword 0d**

**push READSTRING**

**push 5d**

**push StringNum1**

**push dword [handle]**

**call \_ReadConsole@20**

**call ASKFORNUM2**

**push dword -10**

**call \_GetStdHandle@4**

**mov [handle],eax**

**push dword 0d**

**push READSTRING**

**push 5d**

**push StringNum2**

**push dword [handle]**

**call \_ReadConsole@20**

**call longitudN1**

**call longitudN2**

**call convANum1**

**call convANum2**

**sub byte[op],'0' ;REVISAMOS LA OPERACIÓN SELECCIONADA**

**cmp byte[op],1d**

**je OPSUMA**

**cmp byte[op],2d**

**je OPRESTA**

**cmp byte[op],3d**

**je OPPROD**

**cmp byte[op],4d**

**je OPCOCI**

**push 0**

**call \_ExitProcess@4**

**OPSUMA: ;proceso de operación suma**

**mov eax,[Num1]**

**add eax,[Num2]**

**mov [resu],eax**

**push dword -11**

**call \_GetStdHandle@4**

**mov [handle],eax**

**push dword 0**

**push USERINPUT**

**push lrsuma**

**push rsuma**

**push dword [handle]**

**call \_WriteConsole@20**

**call convAText**

**push 0**

**call \_ExitProcess@4**

**ret**

**OPRESTA: ;proceso de operación resta**

**mov eax,[Num1]**

**sub eax,[Num2]**

**mov [resu],eax**

**push dword -11**

**call \_GetStdHandle@4**

**mov [handle],eax**

**push dword 0**

**push USERINPUT**

**push lrresta**

**push rresta**

**push dword [handle]**

**call \_WriteConsole@20**

**call convAText**

**push 0**

**call \_ExitProcess@4**

**ret**

**OPPROD: ;proceso de operación producto**

**mov eax,[Num1]**

**mov ebx,[Num2]**

**mul ebx**

**mov [resu],eax**

**push dword -11**

**call \_GetStdHandle@4**

**mov [handle],eax**

**push dword 0**

**push USERINPUT**

**push lrmulti**

**push rmulti**

**push dword [handle]**

**call \_WriteConsole@20**

**call convAText**

**push 0**

**call \_ExitProcess@4**

**ret**

**OPCOCI: ;proceso de operación cociente**

**xor edx,edx**

**mov eax,[Num1]**

**mov ebx,[Num2]**

**div ebx**

**mov [resu],eax**

**push dword -11**

**call \_GetStdHandle@4**

**mov [handle],eax**

**push dword 0**

**push USERINPUT**

**push lrdiv**

**push rdiv**

**push dword [handle]**

**call \_WriteConsole@20**

**call convAText**

**push 0**

**call \_ExitProcess@4**

**ret**

**longitudN1: ;obtenemos la logitud del primer numero**

**mov esi,StringNum1**

**mov byte[longNum1], 0d**

**lpLN1:**

**mov al,byte[esi]**

**cmp al,0xA**

**jz finLongN1**

**add byte[longNum1],1d**

**inc esi**

**loop lpLN1**

**finLongN1:**

**dec byte[longNum1]**

**ret**

**longitudN2: ;obtenemos la logitud del segundo numero**

**mov esi,StringNum2**

**mov byte[longNum2], 0d**

**lpLN2:**

**mov al,byte[esi]**

**cmp al,0xA**

**jz finLongN2**

**add byte[longNum2],1d**

**inc esi**

**loop lpLN2**

**finLongN2:**

**dec byte[longNum2]**

**ret**

**convANum1: ;Obtenemos el valor 'real' del primer número**

**mov esi,StringNum1**

**mov ebx,[longNum1]**

**dec ebx**

**add esi,ebx**

**mov al,[esi]**

**sub al,'0'**

**mov [auxAcu], al**

**mov ecx,10d**

**mov ebx,[longNum1]**

**mov [auxCon],ebx**

**dec byte[auxCon]**

**a1:**

**cmp byte[auxCon],0d**

**je fin1**

**dec esi**

**mov al,[esi]**

**sub al,'0'**

**mov [aux],al**

**mov eax,[aux]**

**mul ecx**

**add [auxAcu],eax**

**mov eax,ecx**

**mov ecx,10d**

**mul ecx**

**mov ecx,eax**

**dec byte[auxCon]**

**loop a1**

**fin1:**

**cmp byte[longNum1],3**

**jb termino1**

**add byte[auxAcu],1d**

**termino1:**

**mov eax,[auxAcu]**

**mov [Num1],eax**

**ret**

**convANum2: ;Obtenemos el valor 'real' del segundo número**

**mov esi,StringNum2**

**mov ebx,[longNum2]**

**dec ebx**

**add esi,ebx**

**mov al,[esi]**

**sub al,'0'**

**mov [auxAcu], al**

**mov ecx,10d**

**mov ebx,[longNum2]**

**mov [auxCon],ebx**

**dec byte[auxCon]**

**a2:**

**cmp byte[auxCon],0d**

**je fin2**

**dec esi**

**mov al,[esi]**

**sub al,'0'**

**mov [aux],al**

**mov eax,[aux]**

**mul ecx**

**add [auxAcu],eax**

**mov eax,ecx**

**mov ecx,10d**

**mul ecx**

**mov ecx,eax**

**dec byte[auxCon]**

**loop a2**

**fin2:**

**cmp byte[longNum2],3**

**jb termino2**

**add byte[auxAcu],1d**

**termino2:**

**mov eax,[auxAcu]**

**mov [Num2],eax**

**ret**

**convAText:**

**mov eax,[resu]**

**mov ecx,10d**

**mov ebx,0d**

**aCAT:**

**xor edx,edx**

**div ecx**

**push edx**

**inc ebx**

**cmp byte[aux],2d**

**jbe finCAT**

**loop aCAT**

**finCAT:**

**cmp al,1d**

**jb sacar**

**push eax**

**inc ebx**

**sacar:**

**cmp ebx,0d**

**je salir**

**pop ecx**

**mov [aux],ecx**

**add byte[aux],'0'**

**push dword -11**

**call \_GetStdHandle@4**

**mov [handle],eax**

**push dword 0**

**push READSTRING**

**push 1**

**push aux**

**push dword [handle]**

**call \_WriteConsole@20**

**dec ebx**

**loop sacar**

**salir:**

**ret**

**PRINTMENU: ;proceso del menú**

**push dword -11**

**call \_GetStdHandle@4**

**mov [handle],eax**

**push dword 0**

**push USERINPUT**

**push cadena**

**push cadImprimir**

**push dword [handle]**

**call \_WriteConsole@20**

**ret**

**ASKFORNUM1: ;proceso de pedir el primer numero**

**push dword -11**

**call \_GetStdHandle@4**

**mov [handle],eax**

**push dword 0**

**push USERINPUT**

**push pideNum1**

**push mensajeNum1**

**push dword [handle]**

**call \_WriteConsole@20**

**ret**

**ASKFORNUM2: ;proceso de pedir el segundo numero**

**push dword -11**

**call \_GetStdHandle@4**

**mov [handle],eax**

**push dword 0**

**push USERINPUT**

**push pideNum2**

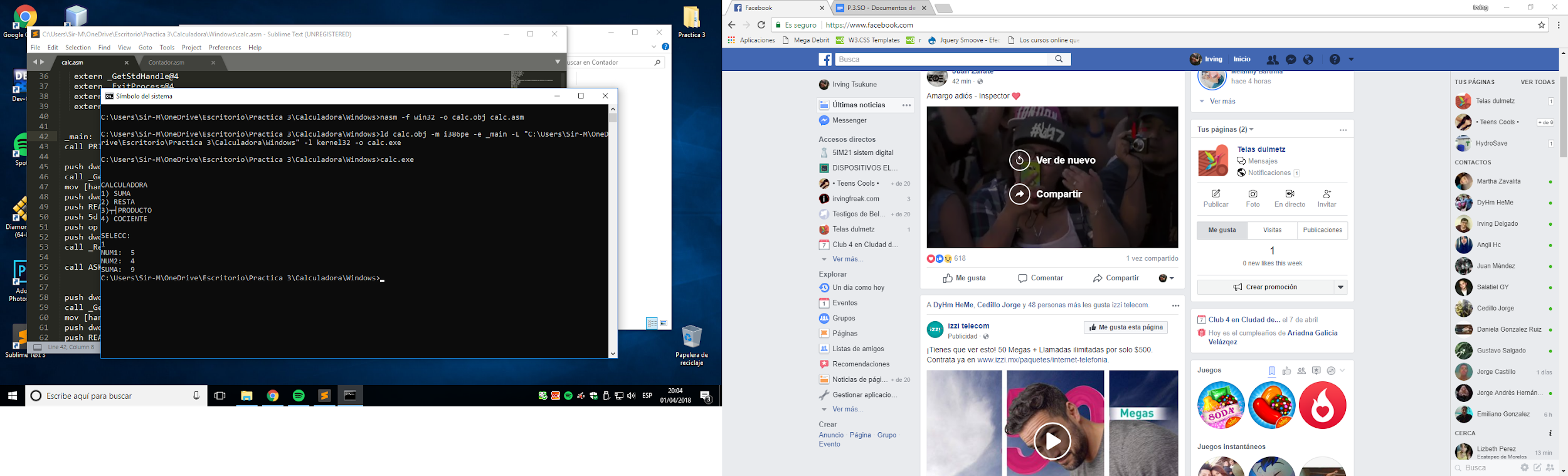
**push mensajeNum2**

**push dword [handle]**

**call \_WriteConsole@20**

**ret**

**Capturas:**

****

**Observaciones:**

* Por lo general, cada llamada al sistema en ambos sistemas operativos requiere que en ciertos registros se guarden datos muy específicos. Esto implica que el desarrollador tendrá que tener especial cuidado con no equivocarse en cosas tan simples como un número.
* El desarrollo en ambos sistemas tiene sus pros y sus contras. Por ejemplo, en linux no es necesario un orden para almacenar datos en los registros; en Windows hay una interrupción (int 21h) que facilita mucho las cosas. Sin embargo, en Windows las llamadas al sistema necesitan bastantes argumentos (que, a pesar de ser varios, tienen el nombre de lo que hacen) y en Linux se utilizan sólo números, que pueden llegar a ser engorrosos.
* Hay características bastante antiguas que los procesadores implementaron desde sus inicios que fueron útiles al realizar los programas. Por ejemplo, para contar caracteres en una cadena.
* Hay 2 convenciones para escribir código ensamblador: la de AT&T y la de Intel, aunque ambas terminan haciendo lo mismo al final.
* Es muy fácil equivocarse al realizar un programa, ya sea porque no se hizo una llamada al sistema, por pasar mal un argumento a un registro o por no utilizar bien la memoria.

**Análisis Crítico.**

En esta práctica, es posible ver, aún más a fondo cómo es que funciona una computadora, con especial detalle en lo que realiza el procesador y qué necesita para ejecutar cada instrucción. Como estudiantes de una ingeniería en sistemas, hasta el momento no se había visto, al menos en lenguajes de alto (o bien, intermedio) nivel cómo es que el procesador ejecuta cada acción que el usuario desea. Sin embargo, al realizar programas en lenguaje ensamblador, no sólo es posible darse cuenta de cómo se realiza cada una de ellas, también se observa qué tan significativa es la traducción que realiza un lenguaje (como C) para simplificar una gran variedad de instrucciones.

Así mismo, fue posible observar que incluso para realizar algo tan simple en cualquier lenguaje de alto se necesita una gran cantidad de código en lenguaje ensamblador para realizar la misma acción.

**Conclusiones.**

**Reyes Valenzuela:** El lenguaje ensamblador si me causó bastantes problemas al momento de trabajar con él, ya que el simple hecho de compilar un programa lleva dos pasos más que al hacerlo con otro lenguaje, como C o Java, sin embargo, es importante conocer este lenguaje de bajo nivel, ya que con él aprendemos un poco acerca de cómo realiza el sistema operativo las operaciones que en un lenguaje de alto nivel, son triviales, tales como la selección de una función, un ciclo o una operación aritmética.

**Guerra Vargas:** Esta práctica fue un pilar muy importante para el entendimiento de cómo programar a nivel ensamblador tanto en windows como en Linux. Como siempre se nos enseñó, Linux es un sistema operativo enfocado para desarrolladores y programadores de software, y el nivel ensamblador no es una excepción, la compatibilidad y lo amigable que es este sistema operativo al usar ensamblador es el claro ejemplo, a comparacion de windows, que es un software que no va enfocado a esto, puesto que incluso se necesita hacer el uso de llamadas especificas si quieres programar en ensamblador en windows, ya que el sistema tuvo que adaptarse para que programadores pudieran hacerlo en windows a nivel ensamblador. Más enfocado a la práctica, ensamblador es un lenguaje complicado pero muy interesante, al manejar direcciones de memoria y llamadas al sistema.

Difícil, sí, pero muy rápido y eficaz.

**Dávila Méndez:** En general, considero significativo el aprendizaje que se obtuvo al realizar la práctica, pues desconocía en gran medida cómo el procesador realiza cada acción que uno como usuario solicita. Observé que prácticamente para todo son necesarias apenas unas cuantas instrucciones, entre las que se encuentran mov, lea, add, jge, jne, etc... cosa que en C es totalmente diferente (el uso de las instrucciones es más claro). A pesar de que sí aprendí cómo usar varias de estas instrucciones, considero que programar en ensamblador es muy poco práctico, pues el usuario puede cometer errores muy fácilmente, sin dejar de lado que el código no es portable entre sistemas operativos (a diferencia de ANSI C). Sin embargo, es interesante conocer a detalle el funcionamiento de una computadora a bajo nivel.