PRÁCTICA 3: INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS LINUX Y WINDOWS (3)

ALUMNO: BASTIDA PRADO JAIME ARMANDO

PROFESOR: CORTÉS GALICIA JORGE

GRUPO: 2CM9

Marzo 2018

Índice

1.	Competencias	3
2.	Desarrollo	3
	2.1. Linux	3
	2.1.1. Punto 6	3
	2.1.2. Punto 8	3
	2.1.3. Punto 9	3
	2.1.4. Código	3
	2.1.5. Punto 10	5
	2.1.6. Código	5
	2.1.7. Punto 11	8
	2.1.8. Código	8
	2.1.9. Punto 12	11
	2.1.10. Código	11
	2.1.11. Punto 13	17
	2.1.12. Código	17
	2.2. Windows	23
	2.2.1. Punto 6	23
	2.2.2. Punto 8	23
	2.2.3. Punto 9	23
	2.2.4. Código	23
	2.2.5. Punto 10	25
	2.2.6. Código	25
	2.2.7. Punto 11	29
	2.2.8. Código	29
	2.2.9. Punto 12	32
	2.2.10. Código	32
	2.2.11. Punto 13	39
	2.2.12. Código	39
3.	Análisis Crítico	46
4.	Observaciones	46
5.	Conclusión	46

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

2. Desarrollo

2.1. Linux

2.1.1. Punto 6

De inicio no me ha parecido algo muy complicado programar en ensamblador, el código es conciso y sencillo.

2.1.2. Punto 8

Otra vez el código aunque un poco laborioso no esta demasiado alejado de como se realiza una llamada al sistema.

2.1.3. Punto 9

Programe una aplicación en ensamblador que genere un contador de 0 a 9, mostrando en pantalla el conteo generado. Consejo: revise las instrucciones de ensamblador CMP, JMP, JE, JNE e INC.

Ensamblamos, enlazamos y ejecutamos el programa y nos aparecerá lo siguiente en consola:

```
james@james-Lenovo-ideapad-320-15ABR:~/Documents/ESCOM_SE
OPERATIVOS/1_Parcial/3_Practice/9_Point$ nasm -f elf -o C
james@james-Lenovo-ideapad-320-15ABR:~/Documents/ESCOM_SE
OPERATIVOS/1_Parcial/3_Practice/9_Point$ ld -m elf_i386 -
.o
james@james-Lenovo-ideapad-320-15ABR:~/Documents/ESCOM_SE
OPERATIVOS/1_Parcial/3_Practice/9_Point$ ./Countera.out
0
1
2
3
4
5
6
7
8
9
```

Figura 1:

2.1.4. Código

```
segment .text
global _start

_start:
    mov ecx, 0

increase:
    mov eax, ecx
    add eax, 48
```

```
push eax
                  mov eax, esp
                  call strprintLF
                  pop eax
                  inc ecx
                  cmp ecx, 10
                  jne increase
                  call exit
; String print with line feed function
strprintLF:
    call
             strprint
                           ; push eax onto the stack to preserve it \mathbf{while} we use the
    push
             eax
        eax register in this function
            eax, OAh ; move OAh into eax - OAh is the ascii character for a
        linefeed
                           ; push the linefeed onto the stack so we can get the
    push
             eax
       address
                           ; move the address of the current stack pointer into eax
    mov
            eax, esp
        for sprint
                           ; call our sprint function
             strprint
                           ; remove our linefeed character from the stack
             eax
    pop
                           ; restore the original value of eax before our function was
    pop
             eax
          called
{\rm re}\, t
; String printing function
strprint:
    push
             edx
    push
             ecx
    push
             _{
m ebx}
    push
             eax
    call
             \operatorname{strlen}
             edx, eax
    mov
    pop
             eax
             ecx, eax
    mov
    mov
             \operatorname{ebx}, 1
             eax, 4
    mov
    int
             0x80
             ebx
    pop
    pop
             ecx
             edx
    \operatorname{pop}
; String length calculation function
strlen:
    push
             ebx
             ebx, eax
    mov
         nextchar:
                       byte [eax], 0
             cmp
             _{
m j}\,{
m z}
                       finished
             inc
                       eax
             _{\mathrm{jmp}}
                       nextchar
         finished:
             \operatorname{sub}
                       eax, ebx
             pop
                      ebx
r\,e\,t
exit:
         mov eax, 1
                                                                         ; sys_exit
```

 $\begin{array}{ccc} \textbf{int} & 0x80 \\ \textbf{ret} \end{array} \hspace{0.2in} ; \textbf{Llamada al sistema}$

2.1.5. Punto 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 cadena 4. Consejo: revise el uso de los registros índice SI y DI.

Ensamblamos, enlazamos y ejecutamos el programa y nos aparecerá un mensaje solicitando ingresar las tres cadenas respectivas, para después mostrar la cuarta cadena:

```
james@james-Lenovo-ideapad-320-15ABR:~/Documents/ESCOM_SEMESTRE
OPERATIVOS/1_Parcial/3_Practice/10_Point$ ./CopyS.out
Enter the first string: hola
Enter the second string: soy
Enter the third string: james
The fourth string is: hsjooalym
james@james-Lenovo-ideapad-320-15ABR:~/Documents/ESCOM_SEMESTRE
OPERATIVOS/1_Parcial/3_Practice/10_Point$
```

Figura 2:

2.1.6. Código

```
segment .bss
                                                               ; Save 1 byte more for the '
string1 resb 21
    0' (null)
string2 resb 21
string3 resb 21
string4 resb 61
segment .data
msg1 db 'Enter_the_first_string:_',0
msg2 db 'Enter_the_second_string:_',0
msg3 db 'Enter_the_third_string:_',0
msg4 db 'The_fourth_string_is:_
segment .text
global _start
_start:
                                                               ; Move the address of msg1
        mov eax, msg1
             into eax
         call strprint
                                                               ; Print the first message
        mov eax, string1
                                                               ; Move the address of
             string1 into eax
        mov ebx, 20d
                                                               ; Size of bytes to read
         call strread
                                                               ; Call strwrite
        mov\ eax\ ,\ msg2
                                                               ; Print the second message
         call strprint
        mov eax, string2
                                                               ; Read the second string 2
        mov ebx, 20d
         call strread
                                                               ; Print the third message
        mov eax, msg3
         call strprint
```

```
call strread
        mov eax, string1
                                                 ; eax contains the address of
           string1
        mov\ ebx\,,\ string2
                                                 ; ebx contains the address of
           string2
                                                 ; ecx contains the address of
        mov ecx, string3
           string3
        mov esi, string4
                                                  ; esi contains the address of
           \operatorname{string} 4
        copiaChar:
                mov edi, [eax]
                                                 ; edi is used just to hold the value
                     of the string
                ; we copy then the value into the
                mov edi, [ebx]
                mov [esi], edi
                inc esi
                inc esi
                                                          ; move the pointer in the
                inc eax
                    string
                cmp byte [eax], 0xA
                                                ; check if it has reached the end of
                     the string
                jz finish
                                                          ; in case of, jump to the
                    finish process
                inc ebx
                cmp\ byte\ [\,ebx\,]\;,\ 0xA
                jz finish
                inc ecx
                cmp byte [ebx], 0xA
                jz finish
                jmp copiaChar
                                                 ; in the case any string has reached
                    the end continue
        finish:
                mov eax, msg4
                call strprint
                mov\ byte\ [\ esi\ ]\ ,\ 0
                                                 ; put the NULL value into the string
                mov eax, string4
                                                 ; print the string4
                call strprint
                mov eax, 0xA
                push eax
                mov eax, esp
                call strprint
        call exit
                                                                  ; exit program
; Lee una cadena
strread:
                                                          ; ebx contiene la cantidad
        \quad \text{mov } \operatorname{edx}\,, \ \operatorname{ebx}
            de bytes a leer
        mov ecx, eax
                                                          ; contiene la direcci n de
           la cadena en la cual guardar
        mov ebx, 0
mov eax, 3
                                                                  ;Entrada est ndar
                                                                  ; sys_read
        int 0x80
                                                                  ;Llamada al sistema
```

; Read the third string 3

```
; Salimos de la
        ret
            funci n
;Imprime una cadena
strprint:
        push edx
                                                                      ; Guardamos el valor
             de los registros a ocupar
        push ecx
        push ebx
        push eax
        call strlen
                                                                      ;Llamamos a strlen
        mov edx, eax
                                                             ; edx contiene ahora la
            longitud de la cadena
        pop eax
                                                                      ; eax contiene la
             direcci n de la cadena a imprimir
                                                             ;Le pasamos la direcci n
        mov ecx, eax
           de la cadena a ecx
        mov ebx, 1
mov eax, 4
                                                                      ; Salida est ndar
                                                                       ; sys_write
        int 0x80
                                                                      ;Llamada al sistema
        pop ebx
                                                                      ; Rescatamos el
           valor de nuestros registros
        pop ecx
        pop edx
                                                                      ; Salimos de la
ret
    funcin
; Calcula la longitud de una cadena
strlen:
        push ebx
                                                                      ; Guardamos el valor
             de ebx
        \operatorname{mov}\ \operatorname{ebx}\ ,\ \operatorname{eax}
                                                             ; Movemos la direcci n de
            eax en ebx
        nextchar:
                 cmp\ byte\ [\,eax\,]\;,\ 0
                                                             ;Es el fin de la candena?
                                                                      ; Caso de S
                 jz finished
                     terminamos
                 inc eax
                                                                      ; Caso de No nos
                      seguimos moviendo en la candena
                 jmp nextchar
        finished:
                 sub eax, ebx
                                                             ; eax contendr la longitud
                      de la cadena
                 pop ebx
                                                                      ; Rescatamos el
                     valor que ten a ebx
                                                                      ; Salimos de la
    funci n
; Cierra el programa
exit:
        mov eax, 1
                                                                      ; sys_exit
        int 0x80
                                                                       ;Llamada al sistema
ret
```

2.1.7. Punto 11

Programe una apicación 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.

Ensamblamos, enlazamos y ejecutamos el programa y nos aparecerá un mensaje solicitando ingresar la cadena, para después mostrar su longitud:

```
james@james-Lenovo-ideapad-320-15ABR:~/Documents/ESCOM_SEM

OPERATIVOS/1_Parcial/3_Practice/11_Point$ ./Strlen.out

Enter the string: hola mi nombre es james

The length is: 23

james@james-Lenovo-ideapad-320-15ABR:~/Documents/ESCOM_SEM

OPERATIVOS/1_Parcial/3_Practice/11_Point$
```

Figura 3:

2.1.8. Código

```
segment .bss
string resb 101
segment .data
msg1 db 'Enter_the_string:_',0
msg2 db 'The_length_is:_',0
segment .txt
global _start
_start:
        mov eax, msg1
                                                           ; print msg
        call strprint
                                                           ; string is the buffer
        mov eax, string
        mov\ ebx\ ,\ 100d
                                                           ;100 bytes to read
        call strread
                                                            ; read the bytes into string
        mov eax, msg2
        call strprint
        mov eax, string
                                                           ; save the address of string
             into eax
        call strlen
                                                                    ; calculate the
            length of string, eax will contain the length
                                                                    ;To not count the '
        dec eax
            n'
        call iprintLF
        call exit
; void iprint(Integer number)
 Integer printing function (itoa)
iprint:
                             ; preserve eax on the stack to be restored after
    push
            eax
        function runs
    push
           ecx
                             ; preserve ecx on the stack to be restored after
        function runs
    push
            edx
                              ; preserve edx on the stack to be restored after
        function runs
                             ; preserve esi on the stack to be restored after
    push
           esi
        function runs
```

```
ecx, 0
                                          ; counter of how many bytes we need to print in the end
      mov
divideLoop:
                                          ; count each byte to print - number of characters
      inc
                  ecx
      mov
                  edx, 0
                                           ; empty edx
                                           ; mov 10 into esi
      mov
                  esi , 10
                                          ; divide eax by esi
      idiv
                  esi
      add
                  edx, 48
                                           ; convert edx to it 's_ascii_representation_-_edx_holds_
            the_remainder_after_a_divide_instruction
___push___edx___;_push_edx_(string_representation_of_an_intger)_onto_
___cmp___eax,_0___;_can_the_integer_be_divided_anymore?
____jnz____divideLoop____;_jump_if_not_zero_to_the_label_divideLoop
printLoop:
___dec___ecx___; _count_down_each_byte_that_we_put_on_the_stack
___mov___eax,_esp___;_mov_the_stack_pointer_into_eax_for_printing
___call___strprint____; _call_our_string_print_function
___pop____eax____; _remove_last_character_from_the_stack_to_move_esp_
___cmp___ecx,_0___;_have_we_printed_all_bytes_we_pushed_onto_the_stack?
___jnz___printLoop____; _jump_is_not_zero_to_the_label_printLoop
___pop___esi___esi___; restore_esi_from_the_value_we_pushed_onto_the_stack_
      at_the_start
  ___pop____edx____; _restore_edx_from_the_value_we_pushed_onto_the_stack_
      at_the_start
___pop___ecx___ecx___; _restore_ecx_from_the_value_we_pushed_onto_the_stack_
      at_the_start
\verb| Lumpop = | lump | 
      at_the_start
ret
; _void_iprintLF (Integer_number)
; _Integer_printing_function_with_linefeed_(itoa)
iprintLF:
___call___iprint____;_call_our_integer_printing_function
___push____eax____;_push_eax_onto_the_stack_to_preserve_it_while_we_use_
      the _eax _ register _in _ this _ function
     _mov____eax_,_0Ah____;_move_0Ah_into_eax_-_0Ah_is_the_ascii_character_for_a_
      linefeed
___push___eax____;_push_the_linefeed_onto_the_stack_so_we_can_get_the_
      address
____mov____eax,_esp____;_move_the_address_of_the_current_stack_pointer_into_
      eax_for_sprint
call___strprint____; call_our_sprint_function
___pop___eax___;_remove_our_linefeed_character_from_the_stack
___pop___eax___; restore_the_original_value_of_eax_before_our_function
      \_was \_called
ret
; Lee_una_cadena_del_tama o_de_n_bytes_especificados_en_edx
strread:
______; ebx_contiene_la_cantidad_
      de_bvtes_a_leer
LLLLLLmovlecx, leax LLLLLLLLLLLLLLLLLLLLLL; contiene laldirecci n del
      la_cadena_en_la_cual_guardar
____mov_ebx,_0____;Entrada_est ndar
____int_0x80____:Llamada_al_sistema
ret ____; Salimos_de_la_
      funci n
;Imprime_una_cadena
strprint:
```

push_edxpush_edx	; Guardamos el valor
_de_los_registros_a_ocupar	
push_ecx	
push_ebx	
push_eax	
call_strlen	Llamamos_a_strlen
$\verb"$; edx.contiene.ahora.la.
longitud_de_la_cadena	
pop_eax	; eax_contiene_la_
direcci n_de_la_cadena_a_imprimir	
	v 1 1.
mov-ecx, -eax	:Le_pasamos_la_direcci_n_
de_la_cadena_a_ecx	0.1:1 1
Lucial moveebx, 1 Lucial Lucia	; Salida est ndar
int 0-20	
int_0x80	::::::;Llamada_al_sistema
pop_ebx	, Pagastamag al
valor_de_nuestros_registros	, Rescatamos er
pop ody	
ret	·Salimos de la
funcin	
	
;;_int_strlen(String_message)	
; ;_int_strlen(String_message) ;_String_length_calculation_function	
; _String _length _calculation _function	
; _String_length_calculation_function strlen:	
;_String_length_calculation_function strlen: pushebx	
;_String_length_calculation_function strlen: pushebx	
; String length calculation function strlen: pushebxnextchar:	
; String length calculation function strlen: pushebxmovebx, Leax	
; String length calculation function strlen: pushebxnextchar:	
; String length calculation function strlen:	
; String length calculation function strlen:	
; String length calculation function strlen:	
; String length calculation function strlen:	
; String length calculation function strlen:	
; String length calculation function strlen:	
; String length calculation function strlen:	
;_String_length_calculation_function strlen:pushebxneaxnextchar:nextchar:nextchar:nextchar:nextcharnextcharnextcharnextcharnextcharnextcharnextcharnextcharnextcharnextcharnextcharnextcharnextcharnextcharnextchar	
;_String_length_calculation_function strlen:pushebxneaxnextchar:nextchar:pishebxinceaxincinceaxincinceaxinceaxinceax	
;_String_length_calculation_function strlen:pushebxmovebx,_eaxnextchar:nextchar:nextchar:nextcharinceaxnextcharinished:subeax,_ebxsubebx ret; Cierra_el_programa exit:	
;_String_length_calculation_function strlen:pushebxmovebx,_eaxnextchar:nextchar:nextchar:nextcharinceaxnextcharinished:subeax,_ebxsubeax,_ebxsubebx ret;;Cierra_el_programa exit:mov_eax,_l	
;_String_length_calculation_function strlen:pushebxmovebx,_eaxnextchar:nextchar:nextchar:nextcharinceaxnextcharinished:subeax,_ebxsubebx ret; Cierra_el_programa exit:	

2.1.9. Punto 12

Programe una apicación en ensamblador que concatene diez cadenas (cadena1 hasta cadena10) 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.

Ensamblamos, enlazamos y ejecutamos el programa y nos aparecerá un mensaje solicitando ingresar las cadenas, para después mostrar la cadena concatenada, la cadena concatenada en sentido inverso y su longitud:

```
james@james-Lenovo-ideapad-320-15ABR:~/Documents/ESCOM_SEME
OPERATIVOS/1_Parcial/3_Practice/12_Point$ ./Strcat.out
Enter the 1 string: h
Enter the 2 string: o
Enter the 3 string: l
Enter the 4 string: a
Enter the 5 string: m
Enter the 6 string: i
Enter the 7 string: n
Enter the 8 string: o
Enter the 9 string: m
Enter the 10 string: b
The final string is: holaminomb
The length is: 10
```

Figura 4:

2.1.10. Código

```
segment .bss
string1 resb 101
string2 resb 101
string3 resb 101
string4 resb 101
string5 resb 101
string6 resb 101
string7 resb 101
string8 resb 101
string9 resb 101
string10 resb 101
stringf resb 1001
                                                                 ; Final string
stringfi resb 1001
                                                                 ; Final string inverted
segment .data
msg1 db 'Enter_the_1_string:_',0
msg2 db 'Enter_the_2_string:_
msg3 db 'Enter_the_3_string:_',0
msg4 db 'Enter_the_4_string:_',0
msg5 db 'Enter_the_5_string:_',0
msg6 db 'Enter_the_6_string:_',0
msg7 db 'Enter_the_7_string:_',0
msg8 db 'Enter_the_8_string:_',0
msg9 db 'Enter_the_9_string:_',0
msg10 db 'Enter_the_10_string:_',0
msg11 db 'The_final_string_is:_',0
msg12 db 'The_final_string_inverted_is:_',0
msg13 db 'The_length_is:_',0
segment .txt
global _start
_start:
                                                                ; First string
        mov eax, msg1
```

call strprint $mov\ eax\,,\ string\,1$ mov ebx, 100d call strread mov eax, msg2 ; Second string call strprint $\begin{array}{lll} mov \ eax \,, & string \, 2 \\ mov \ ebx \,, & 100 \, d \end{array}$ call strread mov eax, msg3 ; Third string call strprint call strread mov eax, msg4 ; Fourth string call strprint mov eax, string4 mov ebx, 100dcall strread ; Fifth string mov eax, msg5 call strprint $mov\ eax\,,\ string\,5$ $mov\ ebx\ ,\ 100d$ call strread ; Sexth string mov eax, msg6 call strprint mov eax, string6 mov ebx, 100d call strread mov eax, msg7 ; Seventh string call strprint call strread mov eax, msg8 ; Eighth string call strprint mov eax, string8 mov ebx, 100d call strread mov eax, msg9 ; Ninth string call strprint $mov\ eax\,,\ string\,9$ mov ebx, 100d call strread mov eax, msg10 ; Tenth string call strprint $mov\ eax\,,\ string 10$ mov ebx, 100d call strread

 $mov\ esi\ ,\ stringf$; ESI now contains the reference to stringf $mov\ edi\,,\ string1$ call cat $mov\ edi\ ,\ string 2$ call cat mov edi, string3 call cat mov edi, string4 call cat mov edi, string5 call cat mov edi, string6 call cat mov edi, string7 call cat mov edi, string8 call cat mov edi, string9 call cat $mov\ edi\ ,\ string 10$ call cat mov eax, msg11 call strprint $\quad \text{mov byte } \left[\, \operatorname{esi} \,\right], \ 0$; Print the stringf $mov\ eax\,,\ stringf$ call strprint mov esi, stringfi ; ESI holds the reference to strinf inverted mov edi, string10 call cat mov edi, string9 call cat mov edi, string8 call cat mov edi, string7 call cat mov edi, string6 call cat $mov\ edi\,,\ string\,5$ call cat mov edi, string4 call cat mov edi, string3 call cat mov edi, string2 call cat

```
mov edi, string1
       call cat
                                                   ; Print a line feed '\n'
       mov eax, 0xA
       push eax
       mov eax, esp
       call strprint
       pop eax
       mov eax, msg12
       call strprint
                                                   ; Print stringf inverted
       mov byte [esi], 0
       mov eax, stringfi
       call strprint
       mov eax, 0xA
                                                   ; Print a line feed '\n'
       push eax
       mov eax, esp
       call strprint
       pop eax
       mov eax, msg13
       call strprint
       mov eax, stringf
       call strlen
                                                          ; After calling
          strlen EAX will contain the length of the stringf
       call iprintLF
       call exit
; void iprint (Integer number)
 Integer printing function (itoa)
iprint:
                         ; preserve eax on the stack to be restored after
   push
       function runs
   push
                         ; preserve ecx on the stack to be restored after
          ecx
       function runs
                         ; preserve edx on the stack to be restored after
   push
          edx
      function runs
   push
          esi
                         ; preserve esi on the stack to be restored after
      function runs
          \operatorname{ecx}, 0
   mov
                         ; counter of how many bytes we need to print in the end
divideLoop:
   inc
          ecx
                         ; count each byte to print - number of characters
          edx, 0
                         ; empty edx
   mov
   mov
           esi , 10
                         ; mov 10 into esi
                         ; divide eax by esi
   idiv
           esi
          edx, 48
                         ; convert edx to it 's_ascii_representation_-_edx_holds_
   add
       the_remainder_after_a_divide_instruction
___push___edx____;_push_edx_(string_representation_of_an_intger)_onto_
   the_stack
____dec____ecx____;_count_down_each_byte_that_we_put_on_the_stack
___mov___eax,_esp___;_mov_the_stack_pointer_into_eax_for_printing
___call___strprint____;_call_our_string_print_function
___cmp___ex,_0___;_have_we_printed_all_bytes_we_pushed_onto_the_stack?
\verb| Lul| jnz| \verb| Lul| printLoop| \verb| Lul| jump_is| \verb| Lnot_zero_to_the_label_printLoop| |
___pop___esi___esi___;_restore_esi_from_the_value_we_pushed_onto_the_stack_
   at\_the\_start
```

```
___pop____edx____; restore_edx_from_the_value_we_pushed_onto_the_stack_
  at_the_start
___pop___ecx___; restore_ecx_from_the_value_we_pushed_onto_the_stack_
  at_the_start
  _pop____eax____; _restore_eax_from_the_value_we_pushed_onto_the_stack_
  at_the_start
ret
; _void_iprintLF (Integer_number)
; _Integer_printing_function_with_linefeed_(itoa)
iprintLF:
___call___iprint____;_call_our_integer_printing_function
___push___eax___it_while_we_use_
  the _eax _ register _ in _ this _ function
___oAh_is_the_ascii_character_for_a_
  linefeed
 __push___eax____; _push_the_linefeed_onto_the_stack_so_we_can_get_the_
  address
 ___mov____eax,_esp____;_move_the_address_of_the_current_stack_pointer_into_
  eax_for_sprint
call_our_strprint_coll_our_sprint_function
___pop___eax___;_remove_our_linefeed_character_from_the_stack
___pop___eax__before_our_function
  \verb"_was_called"
ret
; Concatenates_two_strings_(EDI_hols_the_reference_of_the_source_string,_ESI_holds_
  the_reference_of_the_dest_string)
cat:
____mov_eax , _ [ edi ]
____mov_[esi],_eax
___inc_edi
____cmp_byte_[edi],_0xA
____jne_copyChar
ret
;Lee_una_cadena_del_tama o_de_n_bytes_especificados_en_edx
strread:
_____mov_edx,_ebx____;ebx_contiene_la_cantidad_
  de_bytes_a_leer
Ludulumovecx, leax ludululululululululululululululi; contiene la direcci n del
  la \verb|_cadena| en \verb|_la| \verb|_cual| \verb|_guardar|
____mov_ebx, _0 ____; Entrada_est ndar
____int_0x80____;Llamada_al_sistema
ret____; Salimos_de_la_
   funci n
;Imprime_una_cadena
strprint:
push edx ____; Guardamos el valor
  _de_los_registros_a_ocupar
____push_ecx
____push_ebx
____push_eax
call strlen .....: Llamamos a strlen
____mov_edx,_eax____;edx_contiene_ahora_la_
  longitud_de_la_cadena
```

direcci n_de_la_cadena_a_imprimir	; eax.contiene.la.
de_la_cadena_a_ecxmov_ebx , _1	;Salida_est_ndar
valor_de_nuestros_registrospop_ecxpop_edx	
funci n	; Salimos .de .la .
; _int_strlen(String_message) ; _String_length_calculation_function strlen:pushebxmovebx,_eax	
nextchar: pzfinished inceax	
LLLLLL finished: LLLLLL subLLLLeax, Lebx LLLLLL popLLLLebx ret	
; Cierra_el_programa exit:	
$\begin{array}{c} \text{constant} & \text{moveax} \;,\; \text{1} \\ \text{constant} & \text{int} \; \text{0x80} \end{array}$	

2.1.11. Punto 13

Programe una apicación 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]. Consejo: revise las instrucciones de ensamblador ADD, SUB, MUL y DIV.

Ensamblamos, enlazamos y ejecutamos el programa y nos aparecerá un menú con las 4 operaciones y una opción de salir, en el ejemplo selecciono la opción 3 (Multiplicación):

```
james@james-Lenovo-ideapad-320-15ABR:~/Documents
OPERATIVOS/1_Parcial/3_Practice/13_Point$ ./Calc
MENU
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit
Choose your option: 3
Enter the first number: 3
Enter the second number: 4
The result is: 12
```

Figura 5:

2.1.12. Código

```
. data
segment
msg1 db 'MENU',0
msg2 db '1._Addition',0
msg3 db '2._Subtraction',0
msg4 db '3._Multiplication',0
msg5 db '4. Division',0
msg6 db '5._Exit',0
msg7 db 'Choose_your_option:_',0
msg8 db 'Enter_the_first_number:_',0
msg9 db 'Enter_the_second_number:_',0
msg10 db 'The_result_is:_',0
msg11 db 'Choose_a_correct_option',0
segment .bss
option resb 5
number1 resb 5
number2 resb 5
segment .text
global _start
_start:
        menu:
                mov eax, msg1
                                                                    ; Print the MENU
                 call strprintLF
                mov eax, msg2
                 {\tt call\ strprint} LF
                mov eax, msg3
                 call strprintLF
                mov eax, msg4
                 {\tt call\ strprintLF}
                mov eax, msg5
                 call strprintLF
```

```
mov eax, msg6
          call strprintLF
         mov eax, msg7
          call strprint
         mov eax, option
                                                                    ; Read the option
         mov ebx, 2
          call strread
         mov eax, option
                                                                    ; Compare the option
         cmp byte [eax], 49
         jz addition
          cmp byte [eax], 50
          jz subtraction
         cmp byte [eax], 51
jz multiplication
         cmp\ byte\ [\,eax\,]\;,\ 52
          jz division
         cmp\ byte\ [\,eax\,]\;,\ 53
          jz _exit
         mov\ eax\ ,\ msg11
          call strprintLF
jmp menu
_exit:
          call exit
addition:
         mov eax, msg8
                                                          ;Enter the first number
         call strprint
         \quad \text{mov } \operatorname{eax} \;, \; \operatorname{number} 1
         mov ebx, 5
          call strread
         mov\ eax\ ,\ msg9
                                                          ; Enter the second number
          call strprint
         mov\ eax\ ,\ number 2
         mov ebx, 5
         call strread
         mov eax, msg10
                                                          ; The result is:
         call strprint
         mov eax, number1
         call atoi
         mov edx, eax
         mov\ eax\ ,\ number 2
          call atoi
         \mathrm{add}\ \mathrm{edx}\ ,\ \mathrm{eax}
         mov eax, edx
          call iprintLF
jmp menu
subtraction:
                                                          ; Enter the first number
        mov eax, msg8
         call strprint
```

```
mov eax, number1
          mov ebx, 5
          call strread
          mov eax, msg9
          call strprint
                                                              ; Enter the second number
          mov\ eax\ ,\ number 2
          mov ebx, 5
          call strread
          mov\ eax\ ,\ msg10
                                                              ; The result is:
          call strprint
          mov\ eax\ ,\ number 1
          call atoi
          \quad \text{mov } \operatorname{edx}\,, \ \operatorname{eax}
          mov eax, number2
          call atoi
          sub edx, eax
          \quad \text{mov } \text{eax} \;, \; \text{edx}
          call iprintLF
jmp menu
multiplication:
          mov eax, msg8
                                                              ; Enter the first number
          call strprint
          mov eax, number1
mov ebx, 5
          call strread
          mov\ eax\ ,\ msg9
          call strprint
                                                              ; Enter the second number
          mov eax, number2
          mov ebx, 5
          call strread
          mov eax, msg10
                                                              ; The result is:
          call strprint
          mov eax, number1
          call atoi
          mov edx, eax
          mov\ eax\ ,\ number 2
          call atoi
          \quad \text{mov } \operatorname{ebx}\,, \ \operatorname{eax}
          mov eax, edx
          mul ebx
          call iprintLF
jmp menu
division:
          mov eax, msg8
                                                              ; Enter the first number
          call strprint
          mov\ eax\ ,\ number 1
          \quad \text{mov } \operatorname{ebx} \,, \quad 5
          call strread
          mov eax, msg9
```

```
call strprint
                                                       ; Enter the second number
               mov\ eax\ ,\ number 2
               mov ebx, 5
               call strread
                                                       :The result is:
               mov eax, msg10
               call strprint
               mov\ eax\ ,\ number 1
               call atoi
               mov edx, eax
               mov eax, number2
               call atoi
               mov ebx, eax
               mov eax, edx
               mov edx. 0
                                                              ; Clean the EDX
                   register to avoid segmentation fault
               div ebx
               call iprintLF
       jmp menu
; int atoi(Integer number)
; Ascii to integer function (atoi)
atoi:
                           ; preserve ebx on the stack to be restored after
   push
       function runs
                           ; preserve ecx on the stack to be restored after
   push
          ecx
       function runs
                           ; preserve edx on the stack to be restored after
   push
          edx
      function runs
                           ; preserve esi on the stack to be restored after
   push
          esi
      function runs
           esi, eax
                           ; move pointer in eax into esi (our number to convert)
   mov
           eax, 0
                           ; initialise eax with decimal value 0
   mov
                           ; initialise ecx with decimal value 0
   mov
           ecx, 0
.multiplyLoop:
        ebx, ebx
                          ; resets both lower and uppper bytes of ebx to be 0
           bl, [esi+ecx] ; move a single byte into ebx register's_lower_half
   mov
___cmp___bl,_48____;_compare_ebx_register's lower half value against ascii
   value 48 (char value 0)
           finished; jump if less than to label finished
   j l
   cmp
           bl, 57
                           ; compare ebx register's_lower_half_value_against_ascii
      _value_57_(char_value_9)
___jg___.finished____;_jump_if_greater_than_to_label_finished
___cmp___bl,_10____;_compare_ebx_register's lower half value against ascii
    value 10 (linefeed character)
    je .finished ; jump if equal to label finished
   cmp
           bl, 0
                           ; compare ebx register's lower half value against.
      decimal_value_0_(end_of_string)
___jz___. finished ____; _jump_if_zero_to_label_finished
___sub___bl,_48___;_convert_ebx_register's lower half to decimal
   representation of ascii value
           eax, ebx
                       ; add ebx to our interger value in eax
   add
                           ; move decimal value 10 into ebx
   mov
           ebx, 10
                           ; multiply eax by ebx to get place value
   mul
           ebx
                           ; increment ecx (our counter register)
   inc
           ecx
   _{\mathrm{jmp}}
           . multiplyLoop
                          ; continue multiply loop
. finished:
   mov
           ebx , 10
                           ; move decimal value 10 into ebx
   div
                           ; divide eax by value in ebx (in this case 10)
           ebx
```

```
; restore esi from the value we pushed onto the stack
                     esi
       pop
              at the start
                                                   ; restore edx from the value we pushed onto the stack
       pop
                     edx
              at the start
                                                   ; restore ecx from the value we pushed onto the stack
       pop
                     ecx
              at the start
                                                  ; restore ebx from the value we pushed onto the stack
                    ebx
       pop
              at the start
ret
;Lee una cadena del tama o de n bytes especificados en edx
strread:
                                                                                                     ; ebx contiene la cantidad
              mov edx, ebx
                     de bytes a leer
              mov ecx, eax
                                                                                                     ; contiene la direcci n de
                     la cadena en la cual guardar
              \quad \text{mov } \operatorname{ebx} \,, \quad 0
                                                                                                                    ;Entrada est ndar
              mov eax, 3
                                                                                                                    ; sys_read
              int 0x80
                                                                                                                    ; Llamada al sistema
ret
                                                                                                                    :Salimos de la
       funci n
; void iprint (Integer number)
   Integer printing function (itoa)
iprint:
                                                  ; preserve eax on the stack to be restored after
      push
                     eax
             function runs
       push
                                                  ; preserve ecx on the stack to be restored after
                    ecx
              function runs
                    edx
                                                  ; preserve edx on the stack to be restored after
       push
              function runs
       push
                    esi
                                                  ; preserve esi on the stack to be restored after
             function runs
                                                  ; counter of how many bytes we need to print in the end
      mov
                     ecx, 0
divideLoop:
                                                  ; count each byte to print - number of characters
      inc
                     ecx
      mov
                     edx, 0
                                                  ; empty edx
                                                  ; mov 10 into esi
                     esi , 10
      mov
       idiv
                     esi
                                                  ; divide eax by esi
       add
                     edx, 48
                                                  ; convert edx to it 's_ascii_representation_-_edx_holds_
              the_remainder_after_a_divide_instruction
___push___edx____;_push_edx_(string_representation_of_an_intger)_onto_
       the \, \_\, stack
\verb| lower = constraint | can the linteger | be divided | lower = can the linteger | be divided | lower = can the linteger | be divided | lower = can the linteger | be divided | lower = can the linteger | lower
____jnz____divideLoop____; _jump_if_not_zero_to_the_label_divideLoop
___dec___ecx___; _count_down_each_byte_that_we_put_on_the_stack
____mov___eax, _esp____; _mov_the_stack_pointer_into_eax_for_printing
___call___strprint___;_call_our_string_print_function
forward
___cmp___ex,_0___;_have_we_printed_all_bytes_we_pushed_onto_the_stack?
___jnz___printLoop____; _jump_is_not_zero_to_the_label_printLoop
___pop___esi____esi____;_restore_esi_from_the_value_we_pushed_onto_the_stack_
      at\_the\_start
___pop____edx____etack_
      at_the_start
   .__pop____ecx____; _restore_ecx_from_the_value_we_pushed_onto_the_stack_
       at_the_start
   ...pop....eax...; restore_eax_from_the_value_we_pushed_onto_the_stack_
       at_the_start
ret
```

```
; _void_iprintLF (Integer_number)
 ; _Integer_printing_function_with_linefeed_(itoa)
iprintLF:
___call___iprint____; _call_our_integer_printing_function
___push___eax____i_push_eax_onto_the_stack_to_preserve_it_while_we_use_
              the _eax _ register _ in _ this _ function
__unov__ueax,_0Ah__uuuu;,_move_0Ah_into_eax_-_0Ah_is_the_ascii_character_for_a_
             linefeed
\verb| Lumpush| \verb| Lumeax| \verb| Lumpush| Lumpush| \verb| Lumpush| Lumpush| \verb| Lumpush| Lu
             address
___mov___eax,_esp___;_move_the_address_of_the_current_stack_pointer_into_
             eax_for_sprint
____call___strprint____;_call_our_sprint_function
___pop____; _remove_our_linefeed_character_from_the_stack
___pop___eax___; restore_the_original_value_of_eax_before_our_function
             _was_called
ret
; _int_strlen(String_message)
; _String _length _calculation _function
strlen:
\verb"---push" \verb----ebx"
___mov___ebx,_eax
nextchar:
___cmp___byte_[eax],_0
___inc___eax
___jmp___nextchar
___sub___eax,_ebx
___pop___ebx
; _void_strprint (String_message)
; _String _printing _function
strprint:
___push___edx
___push___ecx
___push___ebx
___push___eax
___call___strlen
___pop___eax
___mov___ecx,_eax
\_\_\_mov\_\_\_\_ebx, \_1
____mov____eax , _4
\verb"----int----80h"
___pop___ebx
___pop___ecx
___pop___edx
ret
; _void_strprintLF (String_message)
; _String _printing _with _ line _feed _function
strprintLF:
___call___strprint
\verb| uuupushuuueaxuuuuu; pushueaxuontoutheustackutopreservelituwhileuweuuseutheustackutopreservelituwhileuweuuseutheustackutopreservelituwhileuweuuseutheustackutopreservelituwhileuweuuseutheustackutopreservelituwhileuweuuseutheustackutopreservelituwhileuweuuseutheustackutopreservelituwhileuweuuseutheustackutopreservelituumhileuweuuseutheustackutopreservelituumhileuweuuseutheustackutopreservelituumhileuweuuseutheustackutopreservelituumhileuweuuseutheustackutopreservelituumhileuweuuseutheustackutopreservelituumhileuweuuseutheustackutopreservelituumhileuweuuseutheustackutopreservelituumhileuweuuseutheustackutopreservelituumhileuweuuseutheustackutopreservelituumhileuweuuseutheustackutopreservelituumhileuweuuseutheustackutopreservelituumhileuweuuseutheustackutopreservelituumhileuweuuseutheustackutopreservelituumhileuweuuseutheustachutopreservelituumhileuweuuseutheustachutopreservelituumhileuweuuseutheustachutopreservelituumhileuweuuseutheustachutopreservelituumhileuweuuseutheustachutopreservelituumhileuweuuseutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheustachutopreservelituumhileuweutheust
              eax_register_in_this_function
     ___mov____eax,_0Ah____;_move_0Ah_into_eax_-_0Ah_is_the_ascii_character_for_a_
             linefeed
```

```
called
ret ______; return_to_our_program

; Cierra_el_programa
exit:
_____move_eax , ____; push_the_linefeed __onto_the_stack_so_we_can_get_the_
address
_____; push_the_linefeed __onto_the_stack_pointer_into_eax_
for_sprint
_____; restore_sprint_function
_____; remove_our_linefeed __character_from_the_stack
_____; restore_the_original_value_of_eax_before_our_function_was
____;
; Cierra_el_programa
exit:
______; Cierra_el_programa
exit:
______; the content of the cont
```

2.2. Windows

2.2.1. Punto 6

El cambio de Windows a Linux hace más trabajoso lo que ya era algo pesado, pero fuera de eso se trata del mismo funcionamiento.

2.2.2. Punto 8

Lo mismo, se trata de una manera diferente de programar las funciones de escritura y lectura de datos, pero fuera de eso sigue siendo muy parecido a como se programa en Linux.

2.2.3. Punto 9

Programe una aplicación en ensamblador que genere un contador de 0 a 9, mostrando en pantalla el conteo generado. Consejo: revise las instrucciones de ensamblador CMP, JMP, JE, JNE e INC.

Ensamblamos, enlazamos y ejecutamos el programa y nos aparecerá lo siguiente en consola:



Figura 6:

2.2.4. Código

```
segment .bss
handleConsola resd 1
longitudCadena resd 1
caractEscritos resd 1
ultimoArgumento resd 1

segment .text
global _main
extern _GetStdHandle@4
```

```
extern _WriteConsoleA@20
extern _ExitProcess@4
_main:
       mov ecx, 0
        increase:
               mov eax, ecx
               add eax, 48
               push eax
               mov eax, esp
               call strprintLF
               pop eax
               inc ecx
                cmp ecx, 10
                jne increase
                call exit
; void strprintLF (String message)
; String printing with line feed function
strprintLF:
   call
           strprint
                        ; push eax onto the stack to preserve it while we use the
   push
          eax
      eax register in this function
           eax, 0Ah ; move 0Ah into eax - 0Ah is the ascii character for a
       linefeed
                        ; push the linefeed onto the stack so we can get the
    push
           eax
      address
          eax, esp
                       ; move the address of the current stack pointer into eax
   mov
       for sprint
    call
           strprint
                       ; call our sprint function
                        ; remove our linefeed character from the stack
    pop
           eax
           eax
                        ; restore the original value of eax before our function was
   pop
        called
                        ; return to our program
ret
; void strprint (String message)
; String printing function
strprint:
   push
            edx
   push
           ecx
           ebx
   push
   push
           eax
           strlen
    call
       mov [longitudCadena], eax
                                                       ;The length of the string
           to print
   push dword -11
                                                                        ; Get the
       write handler
    call \ \_GetStdHandle@4
   mov [handleConsola], eax
    xor eax, eax
                                                                        ; The last
      argument is arbitrarily 0
    mov eax, 0d
   mov [ultimoArgumento], eax
   pop
                                                                        ; Get the
       address of the string to print into eax
        push dword [ultimoArgumento]
                                                       ; Print the string
       push dword caractEscritos
        push dword [longitudCadena]
```

```
push dword eax
         push dword [handleConsola]
         call _WriteConsoleA@20
              ebx
             ecx
    pop
             edx
    pop
ret
  int strlen (String message)
 String length calculation function
strlen:
    push
             ebx
    mov
             ebx, eax
nextchar:
              byte [eax], 0
    \operatorname{cmp}
              finished
    jz
    inc
             eax
    jmp
              nextchar
finished:
    \operatorname{sub}
              eax, ebx
             ebx
    pop
ret
exit:
         xor eax, eax
         mov eax, 0d
         mov [ultimoArgumento], eax
         push dword [ultimoArgumento]
         call _ExitProcess@4
ret
```

2.2.5. Punto 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 cadena 4. Consejo: revise el uso de los registros índice SI y DI.

Ensamblamos, enlazamos y ejecutamos el programa y nos aparecerá un mensaje solicitando ingresar las tres cadenas respectivas, para después mostrar la cuarta cadena:

```
C:\Users\James\Documents\ESCOM_SEMESTRE_5\2CM9_SISTEMENTER the first string: hola
Enter the second string: soy
Enter the third string: james
The fourth string is: hsjooalym

C:\Users\James\Documents\ESCOM_SEMESTRE_5\2CM9_SISTEMENTER
```

Figura 7:

2.2.6. Código

```
string3 resb 21
string4 resb 61
handleConsola resd 1
longitudCadena resd 1
caractLeidos resd 1
caractEscritos resd 1
ultimoArgumento resd 1
segment .data
msg1 db 'Enter_the_first_string:_',0
msg2 db 'Enter_the_second_string:_',0
msg3 db 'Enter_the_third_string:_',0
msg4 db 'The_fourth_string_is:_',0
segment .text
global _main
extern _GetStdHandle@4
\mathbf{extern} \  \  \, \_WriteConsoleA@20
extern _ReadConsoleA@20
extern _ExitProcess@4
_main:
         mov\ eax\ ,\ msg1
                                                                 ; Move the address of msg1
             into eax
         call strprint
                                                                 ; Print the first message
                                                                 ; Move the address of
         mov eax, string1
             string1 into eax
                                                                 ; Size of bytes to read
         mov ebx, 20d
         call strread
                                                                 ; Call strread
                                                                 ; Print the second message
         mov eax, msg2
         call strprint
         mov\ eax\,,\ string\,2
                                                                 ; Read the second string 2
         mov ebx, 20d
         call strread
         mov eax, msg3
                                                                 ; Print the third message
         call strprint
         mov\ eax\,,\ string3
                                                                 ; Read the third string3
         mov ebx, 20d
         call strread
                                                       ; eax contains the address of
         mov eax, string1
             string1
         mov ebx, string2
                                                        ; ebx contains the address of
             \operatorname{string} 2
         mov\ ecx\,,\ string3
                                                       ; ecx contains the address of
             string3
         mov\ esi\ ,\ string 4
                                                        ; esi contains the address of
             string4
         copiaChar:
                  mov edi, [eax]
                                                       ; edi is used just to hold the value
                        of the string
                                                       ; we copy then the value into the
                  mov [esi], edi
                      desired string (in this case string4)
                  inc esi
                  mov edi, [ebx]
mov [esi], edi
                  inc esi
                  mov edi, [ecx]
mov [esi], edi
                  inc esi
```

```
inc eax
                                                          ; move the pointer in the
                    string
                cmp byte [eax], 0xA
                                                ; check if it has reached the end of
                     the string
                                                          ; in case of, jump to the
                jz finish
                    finish process
                \operatorname{cmp} byte [\operatorname{ebx}], 0xA
                jz finish
                inc ecx
                cmp byte [ecx], 0xA
                jz finish
                jmp copiaChar
                                                 ; in the case any string has reached
                     the end continue
        finish:
                mov eax, msg4
                call strprint
                                                ; put the NULL value into the string
                mov byte [esi], 0
                mov eax, string4
                                                  ; print the string4
                call strprint
                mov eax, 0xA
                push eax
                mov eax, esp
                call strprint
        call exit
                                                                   ; exit program
;Lee una cadena
strread:
                                                          ; Pass the address of the
        mov ecx, eax
           buffer to read to ecx
        push ecx
        push dword -10
                                                          ; Get the read handler
        call _GetStdHandle@4
        mov [handleConsola], eax
        mov [longitudCadena], ebx
                                                 ; Pass the number of bytes to read
                                                          ; The last argument is
        xor eax, eax
            arbitrarily 0
        mov eax, 0d
        mov [ultimoArgumento], eax
        push dword [ultimoArgumento]
                                         ; Calls read
        push dword caractLeidos
        push dword [longitudCadena]
        push dword ecx
        push dword [handleConsola]
        call \ \_ReadConsoleA@20
                                                                   ; Get into ecx the
        pop\ ecx
           address of the string
        dec byte [caractLeidos]
                                                 ; 'Eliminate' the line feed windows
            uses from the counter
        mov ebx, 1d
        jump:
                cmp ebx, [caractLeidos]
                                                 ; If our counter (ebx) is equal to
                   the number of chars read
                jz end
                inc ebx
                inc ecx
```

```
jmp jump
        end:
                 mov byte [ecx], 0xA
                                                               ; Put the real line feed
                      character into the string
                                                                                 :Salimos de
ret
     la funci n
; void strprint(String message)
; String printing function
strprint:
    push
    push
             ecx
    push
             _{\mathrm{ebx}}
    push
             eax
    call
             strlen
                  [longitudCadena], eax
                                                               ;The length of the string
        mov
             to print
    push dword -11
                                                                                 ; Get the
        write handler
    call _GetStdHandle@4
    mov [handleConsola], eax
    xor eax, eax
                                                                                  ; The last
        argument is arbitrarily 0
    mov eax, 0d
    mov [ultimoArgumento], eax
                                                                                  ; Get the
    pop
             eax
        address of the string to print into eax
        push dword [ultimoArgumento]
                                                               ; Print the string
        push dword caractEscritos
        push dword [longitudCadena]
        push dword eax
push dword [handleConsola]
         call _WriteConsoleA@20
    pop
             ebx
    pop
             ecx
             edx
    \operatorname{pop}
{\tt ret}
; int strlen(String message)
; String length calculation function
strlen:
    push
             ebx, eax
    mov
nextchar:
             byte [eax], 0
   cmp
    _{
m j}\,{
m z}
             finished
    inc
             eax
    _{\rm jmp}
             nextchar
finished:
    \operatorname{sub}
             eax, ebx
             ebx
    pop
ret
; Cierra el programa
exit:
        xor eax, eax
        mov eax, 0d
        mov [ultimoArgumento], eax
```

```
push dword [ultimoArgumento]
call _ExitProcess@4
ret
```

2.2.7. Punto 11

Programe una apicación 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.

Ensamblamos, enlazamos y ejecutamos el programa y nos aparecerá un mensaje solicitando ingresar la cadena, para después mostrar su longitud:

```
C:\Users\James\Documents\ESCOM_SEMESTRE_5\2CM9_SISTEMAS_OP
Enter the string: hola mi nombre es james
The length is: 24
C:\Users\James\Documents\ESCOM_SEMESTRE_5\2CM9_SISTEMAS_OP
```

Figura 8:

2.2.8. Código

```
segment .bss
string resb 101
handleConsola\ resd\ 1
longitudCadena resd 1
caractLeidos resd 1
caractEscritos resd 1
ultimoArgumento resd 1
segment .data
msg1 db 'Enter_the_string: _',0
msg2 db 'The_length_is:_',0
segment .txt
global _main
extern _GetStdHandle@4
extern _WriteConsoleA@20
\mathbf{extern} \  \  \, \text{-}ReadConsoleA@20
extern _ExitProcess@4
        mov eax, msg1
        call strprint
                                                              ; print msg
        mov eax, string mov ebx, 100d
                                                               ; string is the buffer
                                                              ;100 bytes to read
        call strread
                                                              ; read the bytes into string
        mov eax, msg2
        call strprint
        mov eax, string
                                                               ; save the address of string
              into eax
         call strlen
                                                                       ; calculate the
             length of string, eax will contain the length
        dec eax
                                                                       ; To not count the '
             \n'
```

```
call iprintLF
                                     call exit
; void iprint(Integer number)
      Integer printing function (itoa)
iprint:
                                                                                                                                ; preserve eax on the stack to be restored after
                 push
                                                      eax
                                   function runs
                  push
                                                                                                                                ; preserve ecx on the stack to be restored after
                                                    ecx
                                   function runs
                  push
                                                     edx
                                                                                                                                ; preserve edx on the stack to be restored after
                                    function runs
                  push
                                                                                                                                ; preserve esi on the stack to be restored after
                                                   esi
                                   function runs
                 mov
                                                     ecx, 0
                                                                                                                               ; counter of how many bytes we need to print in the end
divideLoop:
                                                                                                                               ; count each byte to print - number of characters
                 inc
                                                      ecx
                 mov
                                                     edx, 0
                                                                                                                               ; empty edx
                                                                                                                              ; mov 10 into esi
                 mov
                                                      esi, 10
                  idiv
                                                                                                                              ; divide eax by esi
                                                       esi
                  add
                                                      edx, 48
                                                                                                                               ; convert edx to it 's_ascii_representation_-_edx_holds_
                                   the_remainder_after_a_divide_instruction
      .....bush...
                                               __edx_____;_push_edx_(string_representation_of_an_intger)_onto_
____cmp___eax._0____; _can_the_integer_be_divided_anymore?
___jnz___divideLoop____;_jump_if_not_zero_to_the_label_divideLoop
printLoop:
___dec___ecx___; _count_down_each_byte_that_we_put_on_the_stack
____mov___eax, _esp____; _mov_the_stack_pointer_into_eax_for_printing
___call___strprint____; _call_our_string_print_function
___pop____; remove_last_character_from_the_stack_to_move_esp_
                  forward
\verb| cmp| = ecx , |0| = ecx , |a| = ecx , 
\verb| \_\_\_jnz| = \verb| \_\_printLoop| = \verb| \_\_\_jump\_is| = \verb| not_zero\_to\_the\_label\_printLoop| = \verb| \_\_printLoop| = \verb| \_\_printLoop
___pop___esi____esi____;_restore_esi_from_the_value_we_pushed_onto_the_stack_
                  at_the_start
 ___pop____edx_____; restore_edx_from_the_value_we_pushed_onto_the_stack_
                  at_the_start
 ___pop___ecx____; _restore_ecx_from_the_value_we_pushed_onto_the_stack_
                  at_the_start
\verb| Lumpop = Lumpop = \verb| Lumpop = Lump
                 at_the_start
ret
; _void _iprintLF (Integer _number)
; _Integer_printing_function_with_linefeed_(itoa)
iprintLF:
___call___iprint____;_call_our_integer_printing_function
___push___eax____;_push_eax_onto_the_stack_to_preserve_it_while_we_use_
                 the \verb|_eax|| register \verb|_in|| this \verb|_function|
       ___mov____eax,_0Ah____;_move_0Ah_into_eax_-_0Ah_is_the_ascii_character_for_a_
                 linefeed
___push___eax____;_push_the_linefeed_onto_the_stack_so_we_can_get_the_
                  address
____mov____eax,_esp____;_move_the_address_of_the_current_stack_pointer_into_
                 eax_for_sprint
 ____call___strprint____;_call_our_sprint_function
\verb| Lulpop Lulleax | \verb| Lullpop Lulleax | \verb| Lulleax | 
 ___pop___eax___eax___;_restore_the_original_value_of_eax_before_our_function
                 \_was \_called
ret
```

```
; Lee _una _ cadena
strread:
\verb| lower | Pass\_the\_address\_of\_the\_| \\
   buffer_to_read_to_ecx
push_dword_-10_____; Get_the_read_handler
\verb| \_\_\_\_\_\_ call \_\_GetStdHandle@4|
____mov_[handleConsola],_eax
____mov_[longitudCadena], _ebx _____; Pass_the_number_of_bytes_to_read
____xor_eax , _eax ____ ; The_last _argument_is_
   arbitrarily_0
____mov_eax,_0d
____mov_[ultimoArgumento],_eax
____push_dword_[ultimoArgumento]
____push_dword_caractLeidos
____push_dword_[longitudCadena]
____push_dword_ecx
____push_dword_[handleConsola]
\verb| Luculu | call | LReadConsoleA@20
ret .....; Salimos .de
   _la_funci n
; _void_strprint(String_message)
; _String _printing _function
strprint:
___push___edx
___push___ecx
___push___ebx
___push___eax
___strlen
____mov____[longitudCadena], _eax____; The_length_of_the_string_
  to_print
___push_dword_-11___; Get_the_
   write_handler
\_\_\_\_call\_\_GetStdHandle@4
___mov_[handleConsola], _eax
___xor_eax,_eax___;The_last_
  argument_is_arbitrarily_0
___mov_eax,_0d
___mov_[ultimoArgumento],_eax
___pop___eax___; Get_the_
   address_of_the_string_to_print_into_eax
____push_dword_[ultimoArgumento] ____; Print_the_string
____push_dword_caractEscritos
____push_dword_[longitudCadena]
____push_dword_eax
____push_dword_[handleConsola]
\verb| \_\_\_\_\_ call \_\_WriteConsoleA@20|
___pop___ebx
___pop___ecx
___pop___edx
ret
; _int_strlen(String_message)
; _String _length _calculation _function
strlen:
___push___ebx
___mov___ebx._eax
```

```
nextchar:
---cmp----byte-[eax], -0
___jz___finished
____inc____eax
___jmp___nextchar
finished:
___sub___eax,_ebx
\verb"----pop"----ebx"
ret
; Cierra_el_programa
exit:
____xor_eax,_eax
\verb""" ultimoArgumento"], \verb""" eax"
____push_dword_[ultimoArgumento]
\verb| \_\_\_\_ call \_\_ExitProcess@4|
ret
```

2.2.9. Punto 12

Programe una apicación en ensamblador que concatene diez cadenas (cadena1 hasta cadena10) 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.

Ensamblamos, enlazamos y ejecutamos el programa y nos aparecerá un mensaje solicitando ingresar las cadenas, para después mostrar la cadena concatenada, la cadena concatenada en sentido inverso y su longitud:

```
C:\Users\James\Documents\ESCOM_SEMESTRE_5\2CM9_SISTEMAS_OPERATIVOS\1_Parcial\3_P
Enter the 1 string: hola
Enter the 2 string: mi
Enter the 3 string: nombre
Enter the 4 string: es
Enter the 5 string: computadora
Enter the 6 string: yo
Enter the 7 string: hago
Enter the 8 string: calculos
Enter the 9 string: de
Enter the 10 string: numeros
The final string is: holaminombreescomputadorayohagocalculosdenumeros
The final string inverted is: numerosdecalculoshagoyocomputadoraesnombremihola
The length is: 48
```

Figura 9:

2.2.10. Código

```
        segment .bss

        string1 resb 101

        string2 resb 101

        string3 resb 101

        string5 resb 101

        string6 resb 101

        string7 resb 101

        string8 resb 101

        string9 resb 101

        string10 resb 101

        stringf resb 1001
        ; Final string

        stringfi resb 1001
        ; Final string inverted
```

```
handleConsola resd 1
longitudCadena resd 1
caractLeidos resd 1
caractEscritos resd 1
ultimoArgumento resd 1
segment .data
msg1 db 'Enter_the_1_string:_',0
msg1 db 'Enter_the_1_string:_',0
msg2 db 'Enter_the_2_string:_',0
msg3 db 'Enter_the_3_string:_',0
msg4 db 'Enter_the_4_string:_',0
msg5 db 'Enter_the_5_string:_',0
msg6 db 'Enter_the_6_string:_',0
msg7 db 'Enter_the_7_string:_',0
msg8 db 'Enter_the_8_string:_',0
msg9 db 'Enter_the_9_string:_',0
msg10 db 'Enter_the_10_string: ',0
msg11 db 'The_final_string_is: ',0
msg12 db 'The_final_string_inverted_is:_',0
msg13 db 'The_length_is:_',0
segment .txt
global -main
extern _GetStdHandle@4
{\bf extern \ \_WriteConsoleA@20}
\mathbf{extern} \ _{-} \mathbf{ReadConsoleA@20}
extern _ExitProcess@4
           mov eax, msg1
                                                                             ; First string
           call strprint
           mov eax, string1
           mov\ ebx\ ,\ 100d
           call strread
           mov eax, msg2
                                                                             ; Second string
           call strprint
           call strread
           mov\ eax\ ,\ msg3
                                                                             ; Third string
           call strprint
           mov\ eax\,,\ string3
           mov ebx, 100d
           call strread
           mov eax, msg4
                                                                              ; Fourth string
           call strprint
           mov eax, string4
mov ebx, 100d
           call strread
           mov eax, msg5
                                                                             ; Fifth string
           call strprint
           mov eax, string5
           mov ebx, 100d
           call strread
           mov eax, msg6
                                                                             ; Sexth string
           call strprint
           mov eax, string6
           mov ebx, 100d
```

call strread mov eax, msg7 ; Seventh string call strprint mov eax, string7 mov ebx, 100d call strread ; Eighth string mov eax, msg8 call strprint mov eax, string8 mov ebx, 100d call strread mov eax, msg9 ; Ninth string call strprint mov eax, string9 mov ebx, 100d call strread $mov\ eax\;,\ msg10$; Tenth string call strprint $\ \, \text{mov eax} \,, \ \ \text{string} \, 10 \,$ mov ebx, 100d call strread ; ESI now contains the $mov\ esi\ ,\ stringf$ reference to stringf mov edi, string1 call cat $mov\ edi\,,\ string\,2$ call cat mov edi, string3 call cat mov edi, string4 call cat mov edi, string5 call cat mov edi, string6 call cat mov edi, string7 call cat mov edi, string8 call cat mov edi, string9 call cat mov edi, string10 call cat $mov\ eax\ ,\ msg11$ call strprint mov byte [esi], 0 mov eax, stringf ; Print the stringf call strprint

```
mov esi, stringfi
                                                               ; ESI holds the reference to
              strinf inverted
        mov\ edi\ ,\ string 10
         call cat
        mov edi, string9
        call cat
        mov edi, string8
         call cat
        mov\ edi\ ,\ string 7
        call cat
        mov edi, string6
        call cat
        mov edi, string5
        call cat
        mov edi, string4
        call cat
        mov edi, string3
        call cat
        mov edi, string2
        call cat
        mov edi, string1
        call cat
                                                               ; Print a line feed '\n'
        mov eax, 0xA
        push eax
        mov eax, esp
        call strprint
        pop eax
        mov eax, msg12
        call strprint
        \quad \text{mov byte } \left[\, \text{esi} \, \right], \ 0
                                                               ; Print stringf inverted
        mov eax, stringfi
        call strprint
                                                               ; Print a line feed '\n'
        mov eax, 0xA
        push eax
        mov eax, esp
        call strprint
        pop eax
        mov eax, msg13
        call strprint
        mov\ eax\,,\ stringf
        call strlen
                                                                         ; After calling
             strlen EAX will contain the length of the stringf
         call iprintLF
         call exit
; Concatenates two strings (EDI hols the reference of the source string, ESI holds
    the reference of the dest string)
cat:
         copyChar:
                 \quad \text{mov eax} \;, \; \; [\; \text{edi} \;]
                 mov [esi], eax
```

```
inc edi
                                           inc esi
                                           cmp byte [edi], 0xA
                                           jne copyChar
ret
; void iprint(Integer number)
 ; Integer printing function (itoa)
iprint:
          push
                                                                            ; preserve eax on the stack to be restored after
                                eax
                     function runs
          push
                                                                            ; preserve ecx on the stack to be restored after
                              ecx
                     function runs
          push
                              edx
                                                                            ; preserve edx on the stack to be restored after
                    function runs
          push
                                                                            ; preserve esi on the stack to be restored after
                              esi
                     function runs
                                                                            ; counter of how many bytes we need to print in the end
          mov
                                ecx, 0
divideLoop:
          inc
                                ecx
                                                                           ; count each byte to print - number of characters
                                                                            ; empty edx
          mov
                                edx, 0
                                esi , 10
          mov
                                                                            ; mov 10 into esi
                                                                            ; divide eax by esi
          idiv
                                esi
                                                                           ; convert edx to it 's_ascii_representation_-_edx_holds_
          add
                                edx, 48
                     the \verb|| remainder \verb||| after \verb||| a \verb|| divide \verb||| instruction
___push___edx____;_push_edx_(string_representation_of_an_intger)_onto_
          the_stack
___cmp___eax,_0___;_can_the_integer_be_divided_anymore?
___jnz___divideLoop____; _jump_if_not_zero_to_the_label_divideLoop
printLoop:
\verb| dec | = ecx | = e
____mov___eax,_esp___;_mov_the_stack_pointer_into_eax_for_printing
___call___strprint___;_call_our_string_print_function
___pop___eax____; _remove_last_character_from_the_stack_to_move_esp_
          forward
___cmp___ecx,_0___;_have_we_printed_all_bytes_we_pushed_onto_the_stack?
____jnz___printLoop____; jump_is_not_zero_to_the_label_printLoop
___pop___esi____; restore_esi_from_the_value_we_pushed_onto_the_stack_
          at_the_start
\verb| Lumpop = Lumpop = \verb| Lumpop = L
          at_the_start
     ____ecx____; restore_ecx_from_the_value_we_pushed_onto_the_stack_
          at_the_start
    ___pop____eax___pushed_onto_the_stack_
          at\_the\_start
ret
; _void_iprintLF (Integer_number)
; _Integer_printing_function_with_linefeed_(itoa)
iprintLF:
___call___iprint____;_call_our_integer_printing_function
___push___eax___;_push_eax_onto_the_stack_to_preserve_it_while_we_use_
          the _eax _ register _ in _ this _ function
         _mov____eax,_OAh____;_move_OAh_into_eax_-_OAh_is_the_ascii_character_for_a_
          linefeed
\verb| \_\_\_\_ push \_\_\_\_ eax \_\_\_\_\_\_; \verb| \_push \_ the \_linefeed \_ onto \_ the \_stack \_ so \_ we \_ can \_get \_ the \_
          address
____mov___eax,_esp____;_move_the_address_of_the_current_stack_pointer_into_
          eax_for_sprint
____call___strprint____;_call_our_sprint_function
___pop___eax___;_remove_our_linefeed_character_from_the_stack
```

```
___pop___eax___; restore_the_original_value_of_eax_before_our_function
                     _was_called
; Lee _una _cadena
strread:
unovex, eax undersolution control ; Pass the address of the
                    buffer_to_read_to_ecx
____push_ecx
push_dword_-10_____; Get_the_read_handler
\verb| \_\_\_\_\_ call \_\_GetStdHandle@4|
____mov_[handleConsola],_eax
\verb| Lumber | longitud Cadena|, \verb| lebx | lumber | longitud Cadena|, lumber | lum
Luculus xorueax, Leax Luculus Luculus Luculus ; The last Largument is L
                arbitrarily_0
____mov_eax,_0d
____mov_[ultimoArgumento],_eax
____push_dword_[ultimoArgumento]
____push_dword_caractLeidos
____push_dword_[longitudCadena]
____push_dword_ecx
\verb| \_\_\_\_\_push\_dword\_[handleConsola]|
____call__ReadConsoleA@20
____pop_ecx____; Get_into_ecx_the_
                address_of_the_string
____dec_byte_[caractLeidos]______; 'Eliminate'_the_line_feed_windows_
                    uses\_from\_the\_counter
 ____mov_ebx,_1d
____jump:
_____; If_our_counter_(ebx)_is_equal_to_
                   the_number_of_chars_read
____inc_ebx
____inc_ecx
\verb| lower = | ccx|, lower = |
                   character_into_the_string
{\tt ret} = {\tt less} =
                    _la_funci n
; _void_strprint (String_message)
; _String _printing _function
strprint:
___push___edx
___push___ecx
\verb"----push" \verb----- ebx
___push___eax
____strlen
____mov_ebx,_eax
____xor_eax,_eax
____mov_eax,_ebx
\verb| longitudCadena|, \verb| leax| \verb| longitudCadena|, \verb| leax| \verb| longitudCadena|, \verb| leax| \verb| longitudCadena|, longitu
                     {\tt string\_to\_print}
___push_dword_-11____; Get_the_
                      write_handler
___call__GetStdHandle@4
___mov_[handleConsola], _eax
```

```
\verb"lunuxorleax", \verb"leax" \verb"lunuxorleax"; The \verb"last" \verb"lunuxorleax" |
   argument_is_arbitrarily_0
\verb"---mov-eax", \verb--0d"
\verb""" ultimoArgumento"], \verb""eax"
address_of_the_string_to_print_into_eax
push_dword_[ultimoArgumento] ____; Print_the_string
____push_dword_[longitudCadena]
____push_dword_eax
____push_dword_[handleConsola]
____call__WriteConsoleA@20
___pop___ecx
___pop___edx
; _int_strlen(String_message)
; \_String\_length\_calculation\_function
strlen:
\verb"---push" \verb----ebx"
____mov____ebx,_eax
nextchar:
___jz___finished
\verb"----inc----eax"
___jmp___nextchar
finished:
\verb"---sub-----eax", \verb--ebx"
___pop___ebx
ret
; Cierra_el_programa
exit:
____xor_eax,_eax
____mov_eax,_0d
____mov_[ultimoArgumento],_eax
____push_dword_[ultimoArgumento]
____call__ExitProcess@4
ret
```

2.2.11. Punto 13

Programe una apicación 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]. Consejo: revise las instrucciones de ensamblador ADD, SUB, MUL y DIV.

Ensamblamos, enlazamos y ejecutamos el programa y nos aparecerá un menú con las 4 operaciones y una opción de salir, en el ejemplo selecciono la opción 4 (División):

```
MENU

1. Addition

2. Subtraction

3. Multiplication

4. Division

5. Exit

Choose your option: 4

Enter the first number: 200

Enter the second number: 20

The result is: 10
```

Figura 10:

2.2.12. Código

```
segment
        . data
msg1 db 'MENU',0
msg2 db '1._Addition',0
msg3 db '2._Subtraction',0
msg4 db '3. Multiplication',0
msg5 db '4._Division',0
msg6 db '5._Exit',0
msg7 db 'Choose_your_option:_',0
msg8 db 'Enter_the_first_number:_',0
msg9 db 'Enter_the_second_number:_',0
msg10 db 'The_result_is:_',0
msg11 db 'Choose_a_correct_option',0
segment .bss
option resb 5
number1 resb 5
number2 resb 5
handleConsola resd 1
longitudCadena resd 1
caractLeidos resd 1
caractEscritos resd 1
ultimoArgumento resd 1
segment .text
global _main
extern _GetStdHandle@4
\mathbf{extern} \  \  \, \text{-}WriteConsoleA@20
extern _ReadConsoleA@20
extern _ExitProcess@4
_main:
        menu:
                                                                     ; Print a \n
                mov eax, 0xA
                 push eax
                mov eax, esp
                 call strprint
                 pop eax
```

```
; Print the MENU
         mov eax, msg1
         call strprintLF
         mov\ eax\ ,\ msg2
         call strprintLF
         mov\ eax\ ,\ msg3
         call strprintLF
         mov\ eax\ ,\ msg4
         call strprintLF
         mov\ eax\ ,\ msg5
         call strprintLF
         mov eax, msg6
         call strprintLF
         mov eax, msg7
         call strprint
         \quad \text{mov eax} \;, \; \; \text{option} \\
                                                                   ; Read the option
         mov ebx, 4
         call strread
         mov eax, option
                                                                   ; Compare the option
         cmp byte [eax], 49
         jz addition
         cmp byte [eax], 50
         jz subtraction
         \operatorname{cmp} byte [\operatorname{eax}], 51
         jz multiplication
         cmp\ byte\ [\,eax\,]\;,\ 52
         jz division
         cmp\ byte\ [\,eax\,]\;,\ 53
         jz _exit
         mov\ eax\ ,\ msg11
         call strprintLF
jmp menu
_{\mathtt{exit}} :
         call exit
addition:
         mov eax, msg8
                                                         ; Enter the first number
         call strprint
         mov eax, number1 mov ebx, 5
         call strread
         mov eax, msg9
         call strprint
                                                         ; Enter the second number
         mov eax, number2
         mov ebx, 5
         call strread
                                                         ; The result is:
         mov eax, msg10
         call strprint
         mov eax, number1
         call atoi
```

```
mov edx, eax
         mov eax, number2
          call atoi
          add edx, eax
         \quad \text{mov } \text{eax} \;, \; \text{edx}
          call iprintLF
jmp menu
subtraction:
         mov\ eax\ ,\ msg8
                                                             ; Enter the first number
          call strprint
         mov\ eax\ ,\ number 1
         mov ebx, 5
          call strread
         mov eax, msg9
                                                             ; Enter the second number
          call strprint
         mov\ eax\ ,\ number 2
         mov ebx, 5
          call strread
                                                             ; The result is:
         mov eax, msg10
         call strprint
         mov eax, number1
         call atoi
         mov edx, eax
         mov eax, number2
          call atoi
          sub edx, eax
         mov eax, edx
          call iprintLF
jmp menu
multiplication:
         mov eax, msg8
                                                             ; Enter the first number
          call strprint
         mov eax, number1
         mov ebx, 5
          call strread
         mov\ eax\ ,\ msg9
          call strprint
                                                             ; Enter the second number
         \begin{array}{ll} mov \ eax \ , \ number 2 \\ mov \ ebx \ , \ 5 \end{array}
          call strread
                                                             ; The result is:
         mov eax, msg10
          call strprint
         mov\ eax\ ,\ number 1
          call atoi
         mov edx, eax
          mov\ eax\ ,\ number 2
          call atoi
         \begin{array}{lll} mov & ebx\;, & eax\\ mov & eax\;, & edx \end{array}
          mul ebx
```

```
call iprintLF
        jmp menu
        division:
                                                          :Enter the first number
                mov eax, msg8
                call strprint
                mov\ eax\ ,\ number 1
                mov ebx, 5
                call strread
                mov eax, msg9
                call strprint
                                                          ; Enter the second number
                mov\ eax\ ,\ number 2
                mov ebx. 5
                call strread
                mov eax, msg10
                                                          ; The result is:
                call strprint
                mov eax, number1
                call atoi
                mov edx, eax
                mov eax, number2
                call atoi
                mov ebx, eax
                mov eax, edx
                                                                  ; Clean the EDX
                mov edx, 0
                    register to avoid segmentation fault
                div ebx
                call iprintLF
        jmp menu
; int atoi(Integer number)
; Ascii to integer function (atoi)
atoi:
                             ; preserve ebx on the stack to be restored after
    push
           ebx
       function runs
                             ; preserve ecx on the stack to be restored after
    push ecx
       function runs
    push
           edx
                             ; preserve edx on the stack to be restored after
       function runs
    push
           esi
                             ; preserve esi on the stack to be restored after
       function runs
                             ; move pointer in eax into esi (our number to convert)
           esi, eax
    mov
    mov
            eax, 0
                             ; initialise eax with decimal value 0
    mov
            ecx, 0
                            ; initialise ecx with decimal value 0
.multiplyLoop:
            ebx, ebx ; resets both lower and uppper bytes of ebx to be 0 bl, [esi+ecx] ; move a single byte into ebx register's_lower_half
    xor
    mov
___cmp___bl,_48____;_compare_ebx_register's lower half value against ascii
    value 48 (char value 0)
    j l
            .finished ; jump if less than to label finished
                             ; compare ebx register's_lower_half_value_against_ascii
            bl, 57
    cmp
      _{\text{value}} _{57} _{\text{char}} _{\text{value}} _{9}
___jg____.finished ____; _jump_if_greater_than_to_label_finished
___cmp___bl,_10____;_compare_ebx_register's lower half value against ascii
    value 10 (linefeed character)
        finished; jump if equal to label finished
            bl. 0
                             ; compare ebx register's_lower_half_value_against_
    cmp
       decimal_value_0_(end_of_string)
```

```
____jz____.finished ____; _jump_if_zero_to_label_finished
___sub___bl,_48____;_convert_ebx_register's lower half to decimal
   representation of ascii value
    add
            eax, ebx
                             ; add ebx to our interger value in eax
                             ; move decimal value 10 into ebx
   mov
            ebx, 10
                             ; multiply eax by ebx to get place value
   mul
            ebx
    inc
            ecx
                             ; increment ecx (our counter register)
   jmp
            .multiplyLoop
                            ; continue multiply loop
.finished:
                             ; move decimal value 10 into ebx
            ebx, 10
   mov
    div
            ebx
                             ; divide eax by value in ebx (in this case 10)
                             ; restore esi from the value we pushed onto the stack
            esi
   pop
        at the start
            edx
                             ; restore edx from the value we pushed onto the stack
        at the start
                             ; restore ecx from the value we pushed onto the stack
    pop
            ecx
        at the start
                             ; restore ebx from the value we pushed onto the stack
           ebx
    pop
        at the start
r\,e\,t
; Lee una cadena
strread:
        mov ecx, eax
                                                          ; Pass the address of the
            buffer to read to ecx
        push ecx
        push dword -10
                                                          ; Get the read handler
        call \ \_GetStdHandle@4
        mov [handleConsola], eax
        mov [longitudCadena], ebx
                                                 ; Pass the number of bytes to read
        xor eax, eax
                                                          ;The last argument is
           arbitrarily 0
        mov eax, 0d
        mov [ultimoArgumento], eax
        push dword [ultimoArgumento]
        push dword caractLeidos
        push dword [longitudCadena]
        push dword ecx
push dword [handleConsola]
        call _ReadConsoleA@20
        pop ecx
                                                                  ; Get into ecx the
           address of the string
        dec byte [caractLeidos]
                                                 ; 'Eliminate' the line feed windows
            uses from the counter
        mov ebx, 1d
        jump:
                cmp ebx, [caractLeidos]
                                                 ; If our counter (ebx) is equal to
                    the number of chars read
                jz end
                inc ebx
                inc ecx
                jmp jump
        end:
                mov byte [ecx], 0xA
                                                          ;Put the real line feed
                    character into the string
                                                                           ; Salimos de
ret
     la funci n
```

```
; void iprint (Integer number)
; Integer printing function (itoa)
iprint:
   push
                          ; preserve eax on the stack to be restored after
       function runs
   push
           ecx
                          : preserve ecx on the stack to be restored after
       function runs
   push
          edx
                          ; preserve edx on the stack to be restored after
       function runs
   push
                          ; preserve esi on the stack to be restored after
          esi
       function runs
   mov
           ecx, 0
                          ; counter of how many bytes we need to print in the end
divideLoop:
   inc
                          ; count each byte to print - number of characters
           ecx
                          ; empty edx
   mov
           edx, 0
                          ; mov 10 into esi
   mov
           esi , 10
   idiv
                          ; divide eax by esi
           esi
                          ; convert edx to it 's_ascii_representation_-_edx_holds_
   add
           edx. 48
       the \verb|\_remainder\_| after \verb|\_a\_| divide \verb|\_instruction|
___push___edx___;_push_edx_(string_representation_of_an_intger)_onto_
   the_stack
\verb| constraint = = canthelinteger | below ided | anymore?
___jnz___divideLoop____; _jump_if_not_zero_to_the_label_divideLoop
___dec___ecx___; _count_down_each_byte_that_we_put_on_the_stack
____mov___eax, _esp____; _mov_the_stack_pointer_into_eax_for_printing
___call___strprint___;_call_our_string_print_function
___pop___eax___; remove_last_character_from_the_stack_to_move_esp_
   forward
___cmp___ecx,_0___;_have_we_printed_all_bytes_we_pushed_onto_the_stack?
____jnz___printLoop____; _jump_is_not_zero_to_the_label_printLoop
___pop___esi____; restore_esi_from_the_value_we_pushed_onto_the_stack_
   at the start
___pop___edx___; restore_edx_from_the_value_we_pushed_onto_the_stack_
   at_the_start
___pop____ecx_____; restore_ecx_from_the_value_we_pushed_onto_the_stack_
   at_the_start
 ...pop....eax...; restore_eax_from_the_value_we_pushed_onto_the_stack_
   at\_the\_start
; _void _iprintLF (Integer _number)
; _Integer_printing_function_with_linefeed_(itoa)
iprintLF:
___call___iprint____;_call_our_integer_printing_function
___push___eax____i_push_eax_onto_the_stack_to_preserve_it_while_we_use_
   the \verb""eax" register" \verb""in" \verb""this" \verb""function"
   _mov____eax,_0Ah_____;_move_0Ah_into_eax_-_0Ah_is_the_ascii_character_for_a_
___push___eax____;_push_the_linefeed_onto_the_stack_so_we_can_get_the_
   address
___move_the_address_of_the_current_stack_pointer_into_
   eax_for_sprint
____call___strprint____;_call_our_sprint_function
___pop___eax____; remove_our_linefeed_character_from_the_stack
___pop___eax__before_our_function
   _was_called
ret
; _void_strprint (String_message)
; _String _printing _function
strprint:
___push___edx
```

```
___push___ecx
___push___ebx
___push___eax
____call___strlen
____mov_ebx,_eax
____xor_eax,_eax
____mov_eax,_ebx
____mov_[longitudCadena], _eax ______; The_length_of_the_
              string_to_print
write\_handler
___call__GetStdHandle@4
___mov_[handleConsola],_eax
\verb"conseq" x = \texttt{conseq} = \texttt{
              argument_is_arbitrarily_0
\verb"---mov_eax", \verb--0d"
___mov_[ultimoArgumento],_eax
address_of_the_string_to_print_into_eax
push_dword_[ultimoArgumento]____; Print_the_string
____push_dword_[longitudCadena]
____push_dword_eax
____push_dword_[handleConsola]
\verb| Luculu | call | LWrite \dot{C}onsole A@20
___pop___ebx
___pop___ecx
___pop___edx
ret
; _int_strlen(String_message)
; _String _length _calculation _function
strlen:
___push___ebx
___mov___ebx,_eax
---cmp----byte-[eax], -0
____jz____finished
___inc___eax
___jmp___nextchar
finished:
\_\_\_sub\_\_\_\_eax, \_ebx
___pop___ebx
ret
; _void_strprintLF (String_message)
; _String _printing _with _ line _feed _function
strprintLF:
____call___strprint
\verb| uuupushuuueaxuuuuuu; pushueaxuontoutheustackutopreserveuituwhileuweuuseutheuruuseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuruseutheuru
              eax_register_in_this_function
___oAh_is_the_ascii_character_for_a_
              linefeed
___push___eax____; _push_the_linefeed_onto_the_stack_so_we_can_get_the_
___mov____eax,_esp___;_move_the_address_of_the_current_stack_pointer_into_eax_
              {\tt for\_sprint}
____call___strprint____;_call_our_sprint_function
___pop___eax___; remove_our_linefeed_character_from_the_stack
```

```
called
ret_____; return_to_our_program
;; Cierra_el_programa
exit:
____xor_eax,_eax
___mov_eax,_0d
___mov_[ultimoArgumento],_eax
___push_dword_[ultimoArgumento]
____call__ExitProcess@4
ret
```

3. Análisis Crítico

Está práctica fue como siempre más complicada que la anterior y como siempre proporcionalmente más educativa, es muy interesante analizar y programar las aplicaciones a todos los niveles, desde un uso sencillo de los comandos hasta el nivel de programar en ensamblador llamando a las interrupciones del sistema.

4. Observaciones

El manejo de las cadenas en lenguaje ensamblador es muy parecido a usado en lenguaje C, se tiene que tener cuidado con los marcadores de fin de cadena en Linux y mucho más especialmente en Windows ya que puede generar muchos problemas si no se tiene en cuenta el manejo de estos saltos de línea o "line feed".

5. Conclusión

Programar en el lenguaje ensamblador es sin duda una tarea complicada y más confusa que en cualquier otro lenguaje de programación, pero a su vez nos permite ahorar en gran medida recursos que con otros leguajes muchas veces se desperdician, además de darnos un control casi total de estos recursos aunque eso signifique una responsabilidad mayor para el programador, siendo posible sacar mucha ventaja de ello si se es precavido y organizado.