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

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.

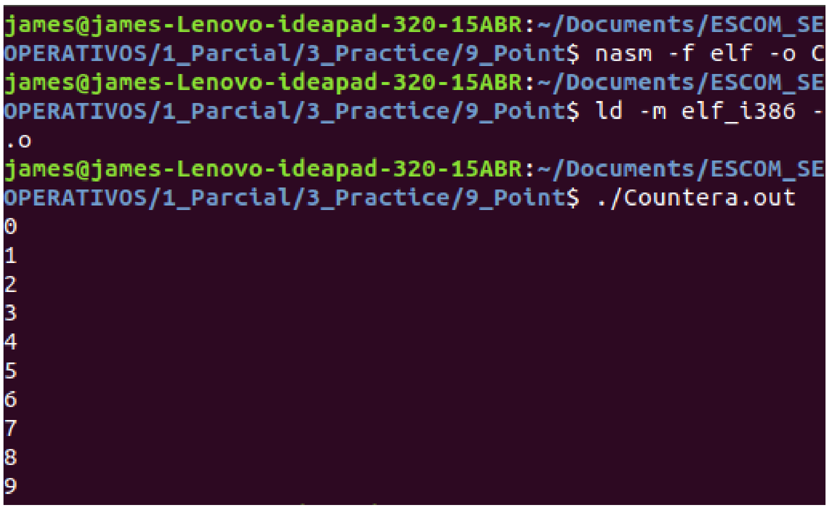
2.1.2. Punto 8

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:

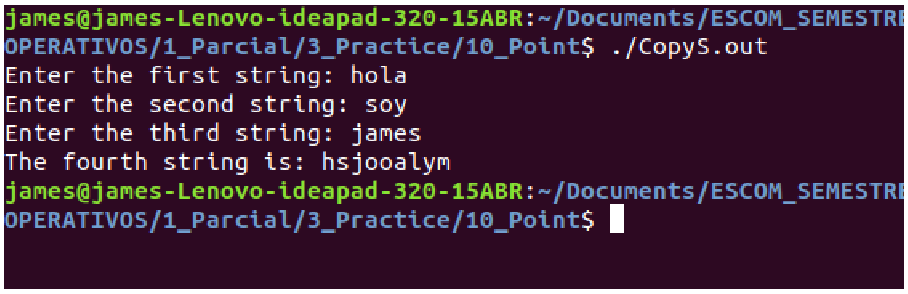
2.1.4. Código

```
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         ; 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


;String printing function
strprint:
    push    edx
    push    ecx
    push    ebx
    push    eax
    call    strlen

    mov     edx, eax
    pop     eax

    mov     ecx, eax
    mov     ebx, 1
    mov     eax, 4
    int     0x80

    pop     ebx
    pop     ecx
    pop     edx
ret

;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
            pop     ebx
ret

exit:
        mov eax, 1                                              ;sys_exit
        int 0x80                                                    ;Llamada al sistema
        ret
```

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:

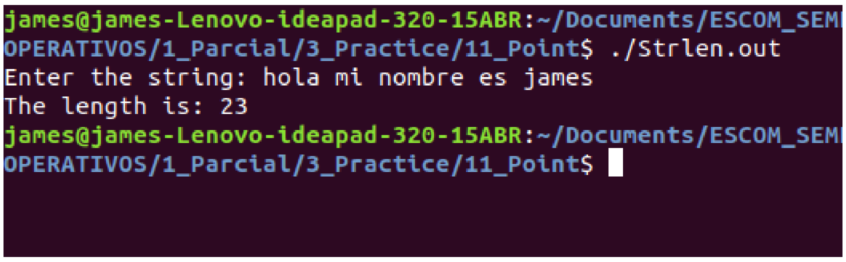
2.1.6. Código

```
segment .bss
string1 resb 21                                         ;Save 1 byte more for the '
    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:
        mov eax, msg1                                   ;Move the address of 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 string2
        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

        mov eax, string1                        ;eax contains the address of
            string1
        mov ebx, string2                        ;ebx contains the address of
            string2
        mov ecx, string3                        ;ecx contains the address of
            string3
        mov esi, string4                        ;esi contains the address of
            string4

        copiaChar:
                mov edi, [eax]                  ;edi is used just to hold the value
                     of the string
                mov [esi], edi                  ;we copy then the value into the
                    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
                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:
        mov edx, ebx                                    ;ebx contiene la cantidad
            de bytes a leer
        mov ecx, eax                                    ;contiene la direcci n de
            la cadena en la cual guardar
        mov ebx, 0                                              ;Entrada est ndar
        mov eax, 3                                              ;sys_read
        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
        pop eax                                                 ;eax contiene la
            direcci n de la cadena a imprimir

        mov ecx, eax                                    ;Le pasamos la direcci n
            de la cadena a ecx
        mov ebx, 1                                              ;Salida est ndar
        mov eax, 4                                              ;sys_write
        int 0x80                                                ;Llamada al sistema

        pop ebx                                                 ;Rescatamos el
            valor de nuestros registros
        pop ecx
        pop edx
ret                                                             ;Salimos de la
    funci n

;Calcula la longitud de una cadena
strlen:
        push ebx                                                ;Guardamos el valor
             de ebx
        mov ebx, eax                                    ;Movemos la direcci n de
            eax en ebx

        nextchar:
                cmp byte [eax], 0                       ;Es el fin de la candena?
                jz finished                                     ;Caso de S
                    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
ret                                                             ;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:

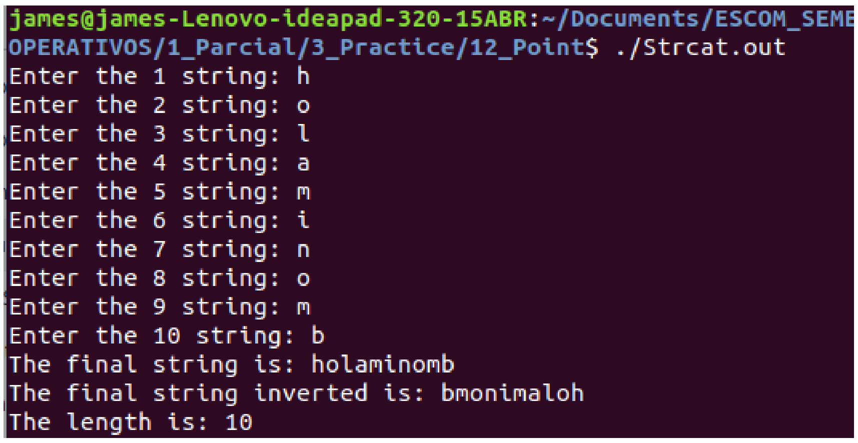
2.1.8. Código

```
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
        call strprint                                   ;print msg

        mov eax, string                                 ;string is the buffer
        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

        dec eax                                              ;To not count the '
            \n'
        call iprintLF

        call exit

;------------------------------------------
; void iprint(Integer number)
; Integer printing function (itoa)
iprint:
    push    eax             ; 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    esi             ; preserve esi on the stack to be restored after
        function runs
    mov      ecx, 0          ; counter of how many bytes we need to print in the end

divideLoop:
    inc      ecx             ; count each byte to print - number of characters
    mov      edx, 0          ; empty edx
    mov      esi, 10         ; mov 10 into esi
    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
␣␣␣␣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␣
    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
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:
␣␣␣␣␣␣␣␣mov␣edx,␣ebx␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;ebx␣contiene␣la␣cantidad␣
    de␣bytes␣a␣leer
␣␣␣␣␣␣␣␣mov␣ecx,␣eax␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;contiene␣la␣direcci n␣de␣
    la␣cadena␣en␣la␣cual␣guardar
␣␣␣␣␣␣␣␣mov␣ebx,␣0␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;Entrada␣est ndar
␣␣␣␣␣␣␣␣mov␣eax,␣3␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;sys␣read
␣␣␣␣␣␣␣␣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
        pop eax                                                 ;eax contiene la
    direcci n de la cadena a imprimir

        mov ecx, eax                                    ;Le pasamos la direcci n
    de la cadena a ecx
        mov ebx, 1                                              ;Salida est ndar
        mov eax, 4                                              ;sys_write
        int 0x80                                                ;Llamada al sistema

        pop ebx                                                 ;Rescatamos el
    valor de nuestros registros
        pop ecx
        pop edx
ret                                                             ;Salimos de la
    funci n

;------------------------------------------
; 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
            pop     ebx
ret

 ;------------------------------------------
;Cierra el programa
exit:
        mov eax, 1                                              ;sys_exit
        int 0x80                                                ;Llamada al sistema
ret
```

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 final string inverted is: bmonimaloh
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: ',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 _start

_start:
        mov eax, msg1                               ;First string
        call strprint

        mov eax, string1
        mov ebx, 100d
        call strread

        mov eax, msg2                           ;Second string
        call strprint

        mov eax, string2
        mov ebx, 100d
        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

        mov eax, msg8                           ;Eighth string
        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

        mov eax, string10
        mov ebx, 100d
        call strread
```

```
mov esi, stringf                                ;ESI now contains the
    reference to stringf

mov edi, string1
call cat

mov edi, string2
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                               ;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, string5
call cat

mov edi, string4
call cat

mov edi, string3
call cat

mov edi, string2
call cat
```

```
        mov edi, string1
        call cat

        mov eax, 0xA                                    ;Print a line feed '\n'
        push eax
        mov eax, esp
        call strprint
        pop eax

        mov eax, msg12
        call strprint

        mov byte [esi], 0                               ;Print stringf inverted
        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:
    push    eax             ; 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    esi             ; preserve esi on the stack to be restored after
        function runs
    mov     ecx, 0          ; counter of how many bytes we need to print in the end

divideLoop:
    inc     ecx             ; count each byte to print - number of characters
    mov     edx, 0          ; empty edx
    mov     esi, 10         ; mov 10 into esi
    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
␣␣␣␣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␣
    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
```

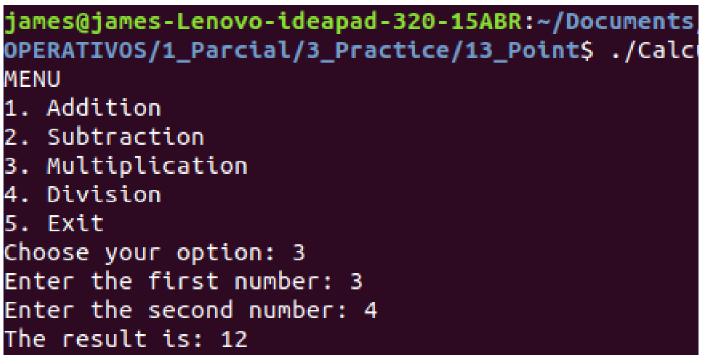
```
␣␣␣␣pop␣␣␣␣␣edx␣␣␣␣␣␣␣␣␣␣␣␣␣;␣restore␣edx␣from␣the␣value␣we␣pushed␣onto␣the␣stack␣
    at␣the␣start
␣␣␣␣pop␣␣␣␣␣ecx␣␣␣␣␣␣␣␣␣␣␣␣␣;␣restore␣ecx␣from␣the␣value␣we␣pushed␣onto␣the␣stack␣
    at␣the␣start
␣␣␣␣pop␣␣␣␣␣eax␣␣␣␣␣␣␣␣␣␣␣␣␣;␣restore␣eax␣from␣the␣value␣we␣pushed␣onto␣the␣stack␣
    at␣the␣start
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


;——————————————————————————————————————————
;Concatenates␣two␣strings␣(EDI␣hols␣the␣reference␣of␣the␣source␣string,␣ESI␣holds␣
    the␣reference␣of␣the␣dest␣string)
cat:
␣␣␣␣␣␣␣␣copyChar:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣mov␣eax,␣[edi]
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣mov␣[esi],␣eax

␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣inc␣edi
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣inc␣esi

␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣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
␣␣␣␣␣␣␣␣mov␣ecx,␣eax␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;contiene␣la␣direcci n␣de␣
    la␣cadena␣en␣la␣cual␣guardar
␣␣␣␣␣␣␣␣mov␣ebx,␣0␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;Entrada␣est ndar
␣␣␣␣␣␣␣␣mov␣eax,␣3␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;sys_read
␣␣␣␣␣␣␣␣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
```

```
␣␣␣␣␣␣␣␣pop␣eax␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;eax␣contiene␣la␣
    direcci n␣de␣la␣cadena␣a␣imprimir

␣␣␣␣␣␣␣␣mov␣ecx,␣eax␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;Le␣pasamos␣la␣direcci n␣
    de␣la␣cadena␣a␣ecx
␣␣␣␣␣␣␣␣mov␣ebx,␣1␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;Salida␣est ndar
␣␣␣␣␣␣␣␣mov␣eax,␣4␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;sys_write
␣␣␣␣␣␣␣␣int␣0x80␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;Llamada␣al␣sistema

␣␣␣␣␣␣␣␣pop␣ebx␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;Rescatamos␣el␣
    valor␣de␣nuestros␣registros
␣␣␣␣␣␣␣␣pop␣ecx
␣␣␣␣␣␣␣␣pop␣edx
ret␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;Salimos␣de␣la␣
    funci n

;------------------------------------------------
;␣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
␣␣␣␣␣␣␣␣␣␣␣␣pop␣␣␣␣␣ebx
ret

;------------------------------------------------
;Cierra␣el␣programa
exit:
␣␣␣␣␣␣␣␣mov␣eax,␣1␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;sys_exit
␣␣␣␣␣␣␣␣int␣0x80␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣;Llamada␣al␣sistema
ret
```

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

```
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

segment .text
global _start

_start:
        menu:
                mov eax, msg1                                   ;Print the MENU
                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

                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

                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, number2
                call atoi
                add edx, eax

                mov eax, edx
                call iprintLF

        jmp menu

        subtraction:
                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, 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

                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, number2
                call atoi
                mov ebx, eax
                mov eax, edx

                mul ebx

                call iprintLF

        jmp menu

        division:
                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, 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:
    push     ebx              ; preserve ebx 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     esi              ; preserve esi on the stack to be restored after
        function runs
    mov      esi, eax         ; move pointer in eax into esi (our number to convert)
    mov      eax, 0           ; initialise eax with decimal value 0
    mov      ecx, 0           ; initialise ecx with decimal value 0

.multiplyLoop:
    xor      ebx, ebx         ; resets both lower and uppper bytes of ebx to be 0
    mov      bl, [esi+ecx]    ; move a single byte into ebx register's␣lower␣half
␣␣␣␣cmp␣␣␣␣␣bl,␣48␣␣␣␣␣␣␣␣␣␣;␣compare␣ebx␣register's lower half value against ascii
     value 48 (char value 0)
    jl       .finished        ; jump if less than to label finished
    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
    add      eax, ebx         ; add ebx to our interger value in eax
    mov      ebx, 10          ; move decimal value 10 into ebx
    mul      ebx              ; multiply eax by ebx to get place value
    inc      ecx              ; increment ecx (our counter register)
    jmp      .multiplyLoop    ; continue multiply loop

.finished:
    mov      ebx, 10          ; move decimal value 10 into ebx
    div      ebx              ; divide eax by value in ebx (in this case 10)
    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     ebx             ; restore ebx from the value we pushed onto the stack
        at the start
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
        mov ecx, eax                                    ;contiene la direcci n de
            la cadena en la cual guardar
        mov ebx, 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:
    push    eax             ; 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    esi             ; preserve esi on the stack to be restored after
        function runs
    mov     ecx, 0          ; counter of how many bytes we need to print in the end

divideLoop:
    inc     ecx             ; count each byte to print - number of characters
    mov     edx, 0          ; empty edx
    mov     esi, 10         ; mov 10 into esi
    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
␣␣␣␣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␣
    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
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

;------------------------------------------
; 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
    pop     ebx
ret

;------------------------------------------
; void strprint(String message)
; String printing function
strprint:
    push    edx
    push    ecx
    push    ebx
    push    eax
    call    strlen

    mov     edx, eax
    pop     eax

    mov     ecx, eax
    mov     ebx, 1
    mov     eax, 4
    int     80h

    pop     ebx
    pop     ecx
    pop     edx
ret

;------------------------------------------
; void strprintLF(String message)
; String printing with line feed function
strprintLF:
    call    strprint

    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                     ; return to our program

;------------------------------------------------
; Cierra el programa
exit:
        mov eax, 1                                                ; sys_exit
        int 0x80                                                  ; Llamada al sistema
ret
```

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.

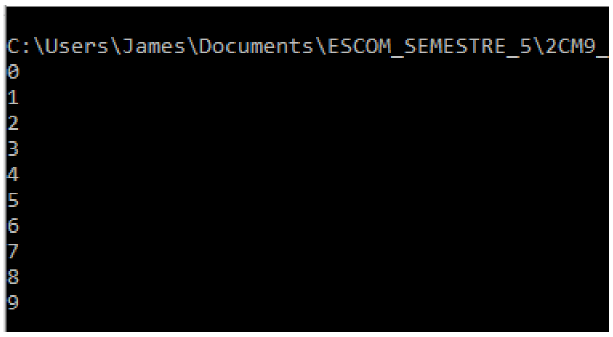
2.2.2. Punto 8

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:



```
C:\Users\James\Documents\ESCOM_SEMESTRE_5\2CM9_
0
1
2
3
4
5
6
7
8
9
```

Figura 6:

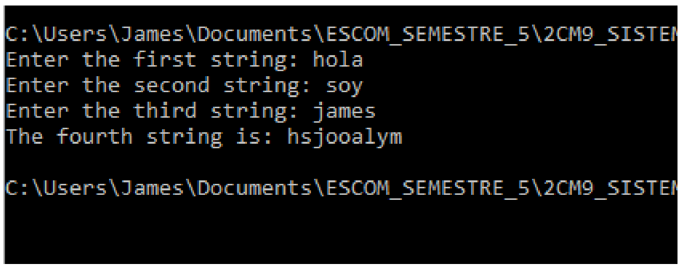
2.2.4. Código

```
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         ; 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                     ; return to our program

;------------------------------------------
; void strprint(String message)
; String printing function
strprint:
    push    edx
    push    ecx
    push    ebx
    push    eax
    call    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]
        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
    pop     ebx
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_SISTEM
Enter 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_SISTEM
```

Figura 7:

2.2.6. Código

```
segment .bss
string1 resb 21                                    ;Save 1 byte more for the '
    0' (null)
string2 resb 21
```

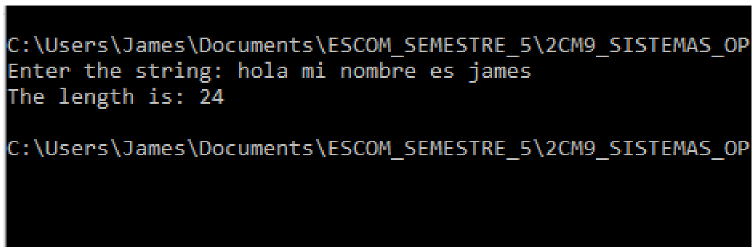
```
string3 resb 21
string4 resb 61

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
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

        mov eax, string1                                ;Move the address of
            string1 into eax
        mov ebx, 20d                                    ;Size of bytes to read
        call strread                                    ;Call strread

        mov eax, msg2                                   ;Print the second message
        call strprint

        mov eax, string2                                ;Read the second string2
        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

        mov eax, string1                        ;eax contains the address of
            string1
        mov ebx, string2                        ;ebx contains the address of
            string2
        mov ecx, string3                        ;ecx contains the address of
            string3
        mov esi, string4                        ;esi contains the address of
            string4

        copiaChar:
                mov edi, [eax]                  ;edi is used just to hold the value
                     of the string
                mov [esi], edi                  ;we copy then the value into the
                    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
                jz finish                               ;in case of, jump to the
                    finish process

                inc ebx
                cmp byte [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

                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:
        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]     ;Calls read
        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
ret                                                             ;Salimos de
     la funci n

;--------------------------------------------------
; void strprint(String message)
; String printing function
strprint:
    push    edx
    push    ecx
    push    ebx
    push    eax
    call    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]
        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
    pop     ebx
ret

;Cierra el programa
exit:
        xor eax, eax
        mov eax, 0d
        mov [ultimoArgumento], eax
        push dword [ultimoArgumento]
        call _ExitProcess@4
ret
```

2.2.7. 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
extern _ReadConsoleA@20
extern _ExitProcess@4

_main:
        mov eax, msg1
        call strprint                           ;print msg

        mov eax, string                         ;string is the buffer
        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

        dec eax                                         ;To not count the '
            \n'
        call iprintLF

        call exit

;------------------------------------------
; void iprint(Integer number)
; Integer printing function (itoa)
iprint:
    push    eax             ; 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    esi             ; preserve esi on the stack to be restored after
        function runs
    mov     ecx, 0          ; counter of how many bytes we need to print in the end

divideLoop:
    inc     ecx             ; count each byte to print - number of characters
    mov     edx, 0          ; empty edx
    mov     esi, 10         ; mov 10 into esi
    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
␣␣␣␣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␣
    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
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

;------------------------------------------
```

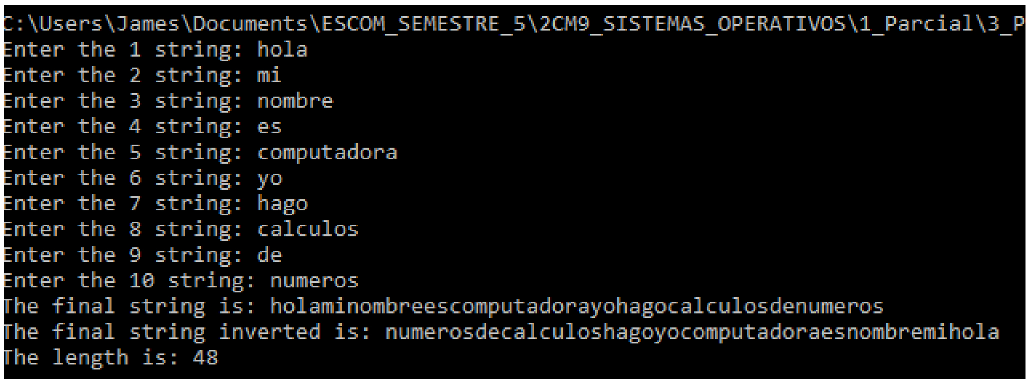
```
;Lee una cadena
strread:
        mov ecx, eax                                    ;Pass the address of the 
    buffer to read to ecx

        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
ret                                                                     ;Salimos de
     la funci n

;------------------------------------------------------------
; void strprint(String message)
; String printing function
strprint:
    push    edx
    push    ecx
    push    ebx
    push    eax
    call    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]
        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
␣␣␣␣pop␣␣␣␣␣ebx
ret

;---------------------------------------------------
;Cierra␣el␣programa
exit:
␣␣␣␣␣␣␣␣xor␣eax,␣eax
␣␣␣␣␣␣␣␣mov␣eax,␣0d
␣␣␣␣␣␣␣␣mov␣[ultimoArgumento],␣eax
␣␣␣␣␣␣␣␣push␣dword␣[ultimoArgumento]
␣␣␣␣␣␣␣␣call␣␣ExitProcess@4
ret
```

2.2.9. 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
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
```

```
handleConsola resd 1
longitudCadena resd 1
caractLeidos resd 1
caractEscritos resd 1
ultimoArgumento resd 1

segment .data
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
extern _WriteConsoleA@20
extern _ReadConsoleA@20
extern _ExitProcess@4

_main:
        mov eax, msg1                                   ;First string
        call strprint

        mov eax, string1
        mov ebx, 100d
        call strread

        mov eax, msg2                                   ;Second string
        call strprint

        mov eax, string2
        mov ebx, 100d
        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

mov eax, msg8                                   ;Eighth string
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

mov eax, string10
mov ebx, 100d
call strread

mov esi, stringf                                ;ESI now contains the
    reference to stringf

mov edi, string1
call cat

mov edi, string2
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                               ;Print the stringf
mov eax, stringf
call strprint
```

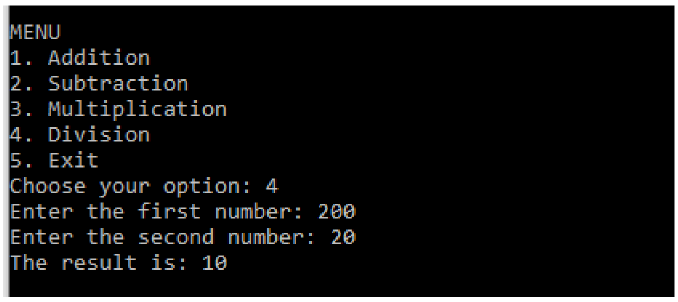
```
        mov esi, stringfi                                    ;ESI holds the reference to
             strinf inverted

        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, string5
        call cat

        mov edi, string4
        call cat

        mov edi, string3
        call cat

        mov edi, string2
        call cat

        mov edi, string1
        call cat

        mov eax, 0xA                                         ;Print a line feed '\n'
        push eax
        mov eax, esp
        call strprint
        pop eax

        mov eax, msg12
        call strprint

        mov byte [esi], 0                                    ;Print stringf inverted
        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

;------------------------------------------------
;Concatenates two strings (EDI hols the reference of the source string, ESI holds
    the reference of the dest string)
cat:
        copyChar:
                mov eax, [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    eax             ; 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    esi             ; preserve esi on the stack to be restored after
        function runs
    mov     ecx, 0          ; counter of how many bytes we need to print in the end

divideLoop:
    inc     ecx             ; count each byte to print - number of characters
    mov     edx, 0          ; empty edx
    mov     esi, 10         ; mov 10 into esi
    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
␣␣␣␣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␣
    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
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
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
ret                                                                     ;Salimos de
     la funci n

;—————————————————————————————————————————————
; 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

    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]
        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
    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
extern _WriteConsoleA@20
extern _ReadConsoleA@20
extern _ExitProcess@4

_main:
        menu:
                mov eax, 0xA                                    ;Print a \n
                push eax
                mov eax, esp
                call strprint
                pop eax
                mov eax, msg1                                       ;Print the MENU
                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

                mov eax, 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

                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

                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, number2
                call atoi
                add edx, eax

                mov eax, edx
                call iprintLF

        jmp menu

        subtraction:
                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, 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

                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, number2
                call atoi
                mov ebx, eax
                mov eax, edx

                mul ebx
```

```
                call iprintLF

        jmp menu

        division:
                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, 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:
    push    ebx             ; preserve ebx 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    esi             ; preserve esi on the stack to be restored after
        function runs
    mov     esi, eax        ; move pointer in eax into esi (our number to convert)
    mov     eax, 0          ; initialise eax with decimal value 0
    mov     ecx, 0          ; initialise ecx with decimal value 0

.multiplyLoop:
    xor     ebx, ebx        ; resets both lower and uppper bytes of ebx to be 0
    mov     bl, [esi+ecx]   ; move a single byte into ebx register's␣lower␣half
␣␣␣␣cmp␣␣␣␣␣bl,␣48␣␣␣␣␣␣␣␣␣␣;␣compare␣ebx␣register's lower half value against ascii
     value 48 (char value 0)
    jl      .finished       ; jump if less than to label finished
    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
    add     eax, ebx        ; add ebx to our interger value in eax
    mov     ebx, 10         ; move decimal value 10 into ebx
    mul     ebx             ; multiply eax by ebx to get place value
    inc     ecx             ; increment ecx (our counter register)
    jmp     .multiplyLoop   ; continue multiply loop

.finished:
    mov     ebx, 10         ; move decimal value 10 into ebx
    div     ebx             ; divide eax by value in ebx (in this case 10)
    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     ebx             ; restore ebx from the value we pushed onto the stack
        at the start
ret

;------------------------------------------------
;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
ret                                                             ;Salimos de
    la funci n
```

```
;------------------------------------------
; void iprint(Integer number)
; Integer printing function (itoa)
iprint:
    push    eax             ; 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    esi             ; preserve esi on the stack to be restored after
        function runs
    mov     ecx, 0          ; counter of how many bytes we need to print in the end

divideLoop:
    inc     ecx             ; count each byte to print - number of characters
    mov     edx, 0          ; empty edx
    mov     esi, 10         ; mov 10 into esi
    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
    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
    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
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

;------------------------------------------
; 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

    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]
        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
    pop     ebx
ret

;------------------------------------------
; void strprintLF(String message)
; String printing with line feed function
strprintLF:
    call    strprint

    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                    ; 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.