Ministerul Educaţiei și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work 5:

Elaborat:
st. gr. FAF-221 Cuzmin Simion

Verificat:

asist. Univ. Voitcovschi Vladislav

Purpose of the work:

1. Write an interactive menu that allows the user to choose from the 10 processes.
2. Write the code for each of the 10 processes. Each process must be written cyclically so that the program always returns to the interactive menu after a process is completed.
3. Ensure that your program is well-commented and structured clearly so that it is easy to understand and modify
4. Test the program to ensure that it works correctly and that the user can choose any of the 10 processes

1. Concatening two strings :

This assembly code concatenates two strings entered by the user and prints the combined result. It prompts for the first string, reads it character by character until Enter is pressed, then repeats the process for the second string. After removing newline characters and null-terminating both strings, it prints them sequentially with a newline between them. Finally, it exits using system calls for input/output and program termination.

```
                                section .data
                    prompt1:      db "Enter first string: ", 0
                                size1: equ $-prompt1
                    prompt2:      db "Enter second string: ", 0
                                size2: equ $-prompt2
                output        db "The concatenated string is: ", 0
                                size3: equ $-output
                      output2     db 10 ; Newline character
                                size4: equ $-output2


                                section .bss
 string1:           resb 100              ; Buffer to hold the first input string
string2:            resb 100             ; Buffer to hold the second input string
  temp:           resb 1                          ; Temporary buffer for reading
                                 characters
     len1:           resb 1                          ; Length of first string
```

```
len2:           resb 1                          ; Length of second string
j:              resb 1                                  ; Loop variable
i:              resb 1                                  ; Loop variable
d0:             resb 1                                ; Temporary variable
d1:             resb 1                                ; Temporary variable

                                section .text
                                global concatening

                                ; Constants for system calls
                                SYS_READ equ 3        ; sys_read
                                SYS_WRITE equ 4       ; sys_write
                                SYS_EXIT equ 1        ; sys_exit


                                concatening:

                        ; Prompt the user to enter the first string
                                mov eax, SYS_WRITE
                                    mov ebx, 1
                                mov ecx, prompt1
                                 mov edx, size1
                                    int 0x80

                                ; Read the first string
        mov ebx, string1    ; Address where the first string will be stored
          mov byte[len1], 0 ; Initialize the length of the first string

                                    reading1:
mov [rsp-8], rbx            ; Save the address where the character will be stored
                                mov eax, SYS_READ    ; sys_read
                        mov ebx, 0           ; File descriptor 0 (stdin)
         mov ecx, temp        ; Address where the character will be stored
                            mov edx, 1           ; Read one character
                                    int 0x80

                                ; Loop to read the first string
  mov rbx, [rsp-8]              ; Restore the address where the character will be
                                        stored
                            mov al, byte[temp] ; Get the character
```

```
                    mov byte[ebx], al  ; Store the character

            inc byte[len1]    ; Increment the length of the first string
             inc ebx           ; Move to the next position in the buffer

                        ; Check for end of line (Enter key)
        cmp byte[temp], 10 ; Compare the read character with newline character
              jne reading1       ; Jump back to reading1 if not newline

                                end_reading1:
                  dec ebx          ; Remove the newline character
               mov byte[ebx],0    ; Null terminate the first string
            dec byte[len1]     ; Decrement the length of the first string

                    ; Prompt the user to enter the second string
                          mov eax, SYS_WRITE  ; sys_write
                   mov ebx, 1          ; File descriptor 1 (stdout)
                     mov ecx, prompt2    ; Address of the prompt
                      mov edx, size2      ; Length of the prompt
                                    int 0x80

                               ; Read the second string
          mov ebx, string2   ; Address where the second string will be stored
            mov byte[len2], 0  ; Initialize the length of the second string

                                    reading2:
mov [rsp-8], rbx           ; Save the address where the character will be stored
                              mov eax, SYS_READ   ; sys_read
                    mov ebx, 0          ; File descriptor 0 (stdin)
           mov ecx, temp       ; Address where the character will be stored
                        mov edx, 1          ; Read one character
                                    int 0x80

                             ; Loop to read the second string
  mov rbx, [rsp-8]           ; Restore the address where the character will be
                                         stored
                         mov al, byte[temp] ; Get the character
                        mov byte[ebx], al  ; Store the character

            inc byte[len2]    ; Increment the length of the second string
```

```
        inc ebx           ; Move to the next position in the buffer

                ; Check for end of line (Enter key)
cmp byte[temp], 10 ; Compare the read character with newline character
       jne reading2       ; Jump back to reading2 if not newline

                        end_reading2:
            dec ebx          ; Remove the newline character
         mov byte[ebx],0    ; Null terminate the second string
     dec byte[len2]     ; Decrement the length of the second string

                        concatenate:
          ; Prompt the user with the concatenated string message
                    mov eax, SYS_WRITE  ; sys_write
              mov ebx, 1        ; File descriptor 1 (stdout)
                mov ecx, output    ; Address of the message
                mov edx, size3     ; Length of the message
                              int 0x80

                         ; Print first string
                              mov eax, 4
                              mov ebx, 1
                            mov ecx, string1
                           movzx edx, byte[len1]
                              int 0x80

                         ; Print second string
                              mov eax, 4
                              mov ebx, 1
                            mov ecx, string2
                           movzx edx, byte[len2]
                              int 0x80

                       ; Print new line character
                              mov eax, 4
                              mov ebx, 1
                            mov ecx, output2
                             mov edx, size4
                              int 0x80
```

```
                                ; Exit program
                                  exit:
                    syscall        ; invoke the system call
                                     ret
```

2. Calculating length of a string:

This assembly code prompts the user to input a string, reads the input, calculates its length, and prints the length. It achieves this by first displaying a prompt message asking for a string input, then reading the user's input string. Subsequently, it iterates through the input string to count the characters, excluding the null terminator. After converting the length to ASCII and storing it, the code prints a predefined message indicating the length of the string, followed by the length itself. Finally, it prints newline characters for formatting before exiting the program.

```
                              section .data

                      prompt      db "Enter a string: ", 0
              newline     db 0x0A, 0x0D, 0     ; Newline characters
                           length_str  db "Length: ", 0
             buffer_size equ 1000             ; Maximum buffer size

                             section .bss
          input_string resb 100   ; Buffer to hold the input string
length1      resb 1               ; Variable to hold the length of the string

                            section .text
                              global length

                               length:
                             ; Print prompt
                               mov eax, 4
```

```
                                mov ebx, 1
                               mov ecx, prompt
                                 mov edx, 15
                                  int 0x80

                             ; Read input string
                    mov eax, 3          ; Read system call
    mov ebx, 0          ; File descriptor for stdin (standard input)
              mov ecx, input_string   ; Buffer to read into
        mov edx, buffer_size    ; Maximum number of bytes to read
                                 int 0x80

                           ; Find length of string
          mov esi, input_string   ; Pointer to the input string
            mov ecx, 0              ; Initialize counter to 0

                              check_length:
                         ; Check for null terminator
                              cmp byte [esi], 0
                                 je end_loop

                            ; Increment counter
                                  inc ecx

                          ; Move to next character
                                  inc esi
                               jmp check_length

                                end_loop:
      ; Subtract 1 from the counter to exclude the null terminator
                                  dec ecx

                     ; Convert length to ASCII and print
                                mov eax, ecx
                                add eax, '0'
                               mov [length1], al

                             ; Print length string
                                  mov eax, 4
                                  mov ebx, 1
```

```
                                mov ecx, length_str
                                    mov edx, 7
                                     int 0x80

                                  ; Print length
                                    mov eax, 4
                                    mov ebx, 1
                                 mov ecx, length1
                                    mov edx, 1
                                     int 0x80

                                  ; Print newline
                                    mov eax, 4
                                    mov ebx, 1
                                 mov ecx, newline
                                    mov edx, 2
                                     int 0x80

                                  ; Exit program
                        syscall         ; invoke the system call
                                           ret
```

3. Inverting a string:

This assembly code defines a function named r3 that reads a string from stdin, reverses it, and prints the reversed string to stdout. It starts by reading the input string into x_buffer, then finds its length. Next, it reverses the string by copying characters from the end of x_buffer to the beginning of y_buffer. After reversing, it prints the reversed string using SYS_WRITE system call. Additionally, it prints a debugging message confirming the successful printing of the reversed string. Finally, the program exits.

```
section .bss
    x_buffer: resb 255    ; Allocate 255 bytes for input buffer
    y_buffer: resb 255    ; Allocate 255 bytes for output buffer

section .text
    global r3
```

```
    ; Constants for system calls
    SYS_READ equ 3        ; sys_read
    SYS_WRITE equ 4       ; sys_write
    SYS_EXIT equ 1       ; sys_exit

r3:
    ; Read input string from stdin
    mov eax, SYS_READ        ; sys_read
    mov ebx, 0               ; filedescriptor stdin
    mov ecx, x_buffer        ; buffer to store the input
    mov edx, 255             ; maximum number of bytes to read
    int 0x80                 ; interrupt to call kernel

    ; Find the length of the input string
    xor edi, edi             ; Clear edi for use as a counter
find_length:
    cmp byte [x_buffer + edi], 0  ; Check for null terminator
    je reverse_string        ; If null terminator found, go to reverse_string
    inc edi                  ; Move to the next character
    jmp find_length          ; Repeat the loop to find length

reverse_string:
    ; Reverse the input string
    mov esi, edi             ; esi points to the end of the input string
    mov edi, y_buffer        ; edi points to the beginning of the output buffer
reverse_loop:
    test esi, esi            ; Check if esi has reached the beginning of the input
string
    jz print_output          ; If yes, jump to print_output

    dec esi                  ; Move to the previous character
    mov al, [x_buffer + esi]  ; Load character from input buffer
    mov [edi], al            ; Store character to output buffer
    inc edi                  ; Move to the next position in output buffer
    jmp reverse_loop         ; Repeat the loop

print_output:
    ; Write the reversed string to stdout
    mov eax, SYS_WRITE       ; sys_write
```

```
    mov ebx, 1              ; filedescriptor stdout
    mov ecx, y_buffer       ; buffer to write
    mov edx, edi            ; length of the reversed string
    int 0x80                ; interrupt to call kernel

    ; Debugging output - Reversed string printed successfully
    mov eax, SYS_WRITE      ; sys_write
    mov ebx, 1              ; filedescriptor stdout
    mov ecx, debug_msg      ; buffer containing debug message
    mov edx, debug_msg_len  ; length of debug message
    int 0x80                ; interrupt to call kernel

    ; Exit program
    syscall         ; invoke the system call
    ret

section .data
    debug_msg db 0xA, "Reversed string printed successfully.", 0xA, 0
    debug_msg_len equ $ - debug_msg
    newline db 0xA, 0       ; Newline character
```

4. Checking if a string is a palindrome:

This assembly code prompts the user to input a string, reads the input, checks if the string is a palindrome, and then prints a corresponding message. It starts by displaying a prompt message asking for a string input and then reads the input string. Afterward, it iterates through the string to check if it's a palindrome, comparing characters from the start and end of the string until they meet in the middle. Depending on whether the string is a palindrome or not, it prints one of two predefined messages. Finally, it prints a newline character and exits the program.

```
section .data
    msg1: db "Enter the string : "
    size1: equ $-msg1
    msg2: db " - This is a pallindrome "
    size2: equ $-msg2
    msg3: db " - This is not a pallindrome "
```

```
    size3: equ $-msg3
    newline: db 10

section .bss
    string: resb 50
    temp: resb 1
    len:  resb 1
    j: resb 1
    i: resb 1

section .text           ; Text section declaration (code)
    global cd        ; Define the entry point of the program as _start
      ; Constants for system calls
    SYS_READ equ 3       ; sys_read
    SYS_WRITE equ 4      ; sys_write
    SYS_EXIT equ 1      ; sys_exit
cd:

    ; Prompt the user to enter the first string
    mov eax, SYS_WRITE
    mov ebx, 1
    mov ecx, msg1
    mov edx, size1
    int 0x80

    ; Read the first string
    mov ebx, string   ; Address where the first string will be stored
    mov byte[len], 0 ; Initialize the length of the first string

reading:
    mov [rsp-8], rbx          ; Save the address where the character will be stored
    mov eax, SYS_READ   ; sys_read
    mov ebx, 0          ; File descriptor 0 (stdin)
    mov ecx, temp       ; Address where the character will be stored
    mov edx, 1          ; Read one character
    int 0x80

    ; Loop to read the first string
    mov rbx, [rsp-8]           ; Restore the address where the character will be
stored
```

```
    mov al, byte[temp] ; Get the character
    mov byte[ebx], al  ; Store the character

    inc byte[len]    ; Increment the length of the first string
    inc ebx          ; Move to the next position in the buffer

    ; Check for end of line (Enter key)
    cmp byte[temp], 10 ; Compare the read character with newline character
    jne reading       ; Jump back to reading1 if not newline

end_reading1:
    dec ebx          ; Remove the newline character
    mov byte[ebx],0    ; Null terminate the first string
    dec byte[len]     ; Decrement the length of the first string

    ;Printing the string....
    mov eax, 4            ; System call for sys_write (print)
    mov ebx, 1            ; File descriptor 1 (stdout)
    mov ecx, string       ; Load the address of string into ecx
    movzx edx, byte[len] ; Load the string length into edx
    int 0x80               ; Call kernel

    mov byte[i], 0        ; Initialize loop variable i to 0
    mov al, byte[len]     ; Load the string length into al
    mov byte[j], al       ; Copy the string length to j
    sub byte[j], 1        ; Decrement j by 1

; Loop to check if the string is a palindrome
palindrome:
    mov eax, string       ; Load the address of string into eax
    mov ebx, string       ; Load the address of string into ebx
    movzx ecx, byte[i]    ; Load i into ecx
    add eax, ecx          ; Move the address in eax by i
    movzx ecx, byte[j]    ; Load j into ecx
    add ebx, ecx          ; Move the address in ebx by j
    mov cl, byte[eax]     ; Load the byte at address eax into cl
    mov ch, byte[ebx]     ; Load the byte at address ebx into ch
    cmp cl, ch            ; Compare the bytes
    jne not_palindrome    ; Jump to not_palindrome if not equal
```

```
    inc byte[i]             ; Increment i
    dec byte[j]             ; Decrement j
    mov al, byte[i]         ; Load i into al
    mov ah, byte[j]         ; Load j into ah
    cmp al,ah               ; Compare al and ah
    jl palindrome           ; Jump to palindrome if al is less than ah


    mov eax, 4              ; System call for sys_write (print)
    mov ebx, 1              ; File descriptor 1 (stdout)
    mov ecx, msg2           ; Load the address of msg2 into ecx
    mov edx, size2          ; Load the length of msg2 into edx
    int 0x80                ; Call kernel
    jmp exit                ; Jump to exit

not_palindrome:
    mov eax, 4              ; System call for sys_write (print)
    mov ebx, 1              ; File descriptor 1 (stdout)
    mov ecx, msg3           ; Load the address of msg3 into ecx
    mov edx, size3          ; Load the length of msg3 into edx
    int 0x80                 ; Call kernel


exit:
    ; Print newline
    mov eax, 4
    mov ebx, 1
    mov ecx, newline
    mov edx, 2
    int 0x80

    syscall          ; invoke the system call
    ret
```

5. Checking whether a number is odd or even:

This assembly code prompts the user to input a number, reads the input, checks if the number is odd or even, and prints a corresponding message. It begins by displaying a message

prompting for a number input, then reads the input number. After that, it converts the input from ASCII to an integer. Following this, it divides the number by 2 to determine if it's odd or even based on the remainder. Depending on whether the remainder is zero (indicating an even number) or not (indicating an odd number), it prints one of two predefined messages. Finally, it prints a newline character and exits the program.

```
section .data
    msg1: db "Enter the number : "
    size1: equ $-msg1
    msg2: db "The number in odd  "
    size2: equ $-msg2
    msg3: db "The number in even  "
    size3: equ $-msg3
    newline db 10, 0


section .bss
    num: resb 32


section .text
    global odd_checker

    ; Constants for system calls
    SYS_READ equ 0                  ; sys_read
    SYS_WRITE equ 1                 ; sys_write
    SYS_EXIT equ 60                 ; sys_exit

odd_checker:
    ; Display the message
    mov eax, SYS_WRITE               ; sys_write
    mov edi, 1                       ; file descriptor 1 is stdout
    mov esi, msg1                    ; address of the message
    mov edx, size1                   ; message length
    syscall

    ; Read the number
    mov eax, SYS_READ                ; sys_read
    mov edi, 0                       ; file descriptor 0 is stdin
```

```
    mov esi, num                        ; address of the number
    mov edx, 10                         ; maximum number of bytes to read
    syscall

convert_to_integer:
    ; Convert the number to an integer
    xor eax, eax                        ; clear eax
    xor ecx, ecx                        ; clear ecx
    mov esi, num                        ; address of the number
    .convert_to_integer_loop:
    movzx ebx, byte [esi + ecx]         ; load the character
    test ebx, ebx                       ; check if the character is null
    jz division                      ; if the character is null, convert the number
    sub ebx, '0'                        ; convert the character to a number
    imul eax, eax, 10                   ; multiply eax by 10
    add eax, ebx                        ; add the number to eax
    inc ecx                             ; move to the next character
    jmp .convert_to_integer_loop

division:
    ; Divide the number by 2
    mov ebx, 2                          ; divisor
    xor edx, edx                        ; clear edx
    mov eax, [num]                ; load the dividend
    div ebx                             ; divide eax by ebx, quotient in eax,
remainder in edx
    mov [num], eax                ; store the quotient in num_integer

    ; Check if the number is odd or even
    test edx, edx                       ; check if the remainder is 0
    jz even                              ; if the remainder is 0, the number is even
    jmp odd                             ; if the remainder is not 0, the number is
odd


even:
    ; Display the message
    mov eax, SYS_WRITE                  ; sys_write
    mov edi, 1                          ; file descriptor 1 is stdout
    mov esi, msg3                       ; address of the message
```

```
    mov edx, size3                      ; message length
    syscall
    jmp exit

odd:
    ; Display the message
    mov eax, SYS_WRITE                  ; sys_write
    mov edi, 1                          ; file descriptor 1 is stdout
    mov esi, msg2                       ; address of the message
    mov edx, size2                      ; message length
    syscall
    jmp exit

exit:
    ; printing new line
    mov eax, SYS_WRITE                  ; sys_write
    mov edi, 1                          ; file descriptor 1 is stdout
    mov esi, newline                    ; address of the newline
    mov edx, 1                          ; message length
    syscall

    ; Exit program
    syscall         ; invoke the system call
    ret
```

6. Determining the larger of two numbers:

This assembly code prompts the user to input two numbers, compares them, and then prints a message indicating which number is bigger or if they are equal. It starts by asking for and reading two numbers from the user. Then, it converts each input from ASCII to an integer. After converting, it compares the numbers. Depending on whether the first number is greater,

the second number is greater, or they are equal, it prints one of three predefined messages. Finally, it prints a newline character and exits the program.

```
section .data
    msg1: db "Enter first number : "
    size1: equ $-msg1
    msg2: db "Enter second number :  "
    size2: equ $-msg2
    msg3: db "First number is bigger ! "
    size3: equ $-msg3
    msg4: db "Second number is bigger ! "
    size4: equ $-msg4
    msg5: db "The numbers are equal ! "
    size5: equ $-msg5
    newline db 10, 0


section .bss
    num: resb 32
    num2: resb 32


section .text
    global bin

    ; Constants for system calls
    SYS_READ equ 0                  ; sys_read
    SYS_WRITE equ 1                 ; sys_write
    SYS_EXIT equ 60                 ; sys_exit

bin:
    ; Display the message
    mov eax, SYS_WRITE               ; sys_write
    mov edi, 1                       ; file descriptor 1 is stdout
    mov esi, msg1                    ; address of the message
    mov edx, size1                   ; message length
    syscall
```

```
    ; Read the number
    mov eax, SYS_READ                   ; sys_read
    mov edi, 0                          ; file descriptor 0 is stdin
    mov esi, num                        ; address of the number
    mov edx, 10                         ; maximum number of bytes to read
    syscall

    ; Display the message
    mov eax, SYS_WRITE                  ; sys_write
    mov edi, 1                          ; file descriptor 1 is stdout
    mov esi, msg2                       ; address of the message
    mov edx, size2                      ; message length
    syscall

    ; Read the number
    mov eax, SYS_READ                   ; sys_read
    mov edi, 0                          ; file descriptor 0 is stdin
    mov esi, num2                        ; address of the number
    mov edx, 10                         ; maximum number of bytes to read
    syscall

convert_to_integer:
    ; Convert the number to an integer
    xor eax, eax                        ; clear eax
    xor ecx, ecx                        ; clear ecx
    mov esi, num                        ; address of the number
    .convert_to_integer_loop:
    movzx ebx, byte [esi + ecx]         ; load the character
    test ebx, ebx                       ; check if the character is null
    jz comparing                      ; if the character is null, convert the number
    sub ebx, '0'                        ; convert the character to a number
    imul eax, eax, 10                   ; multiply eax by 10
    add eax, ebx                        ; add the number to eax
    inc ecx                             ; move to the next character
    jmp .convert_to_integer_loop

convert_to_integer2:
    ; Convert the number to an integer
    xor eax, eax                        ; clear eax
    xor ecx, ecx                        ; clear ecx
```

```
    mov esi, num2                          ; address of the number
    .convert_to_integer_loop2:
    movzx ebx, byte [esi + ecx]            ; load the character
    test ebx, ebx                          ; check if the character is null
    jz comparing                         ; if the character is null, convert the number
    sub ebx, '0'                           ; convert the character to a number
    imul eax, eax, 10                      ; multiply eax by 10
    add eax, ebx                           ; add the number to eax
    inc ecx                                ; move to the next character
    jmp .convert_to_integer_loop2

comparing:

    ; Compare the numbers
    mov eax, [num]                         ; load the first number
    mov ebx, [num2]                        ; load the second number
    cmp eax, ebx                           ; compare the numbers
    jg first_is_bigger                     ; if the first number is bigger
    jl second_is_bigger                    ; if the second number is bigger
    je equal                               ; if the numbers are equal

first_is_bigger:
    ; Display the message
    mov eax, SYS_WRITE                   ; sys_write
    mov edi, 1                           ; file descriptor 1 is stdout
    mov esi, msg3                        ; address of the message
    mov edx, size3                       ; message length
    syscall
    jmp exit

second_is_bigger:
    ; Display the message
    mov eax, SYS_WRITE                   ; sys_write
    mov edi, 1                           ; file descriptor 1 is stdout
    mov esi, msg4                        ; address of the message
    mov edx, size4                       ; message length
    syscall
    jmp exit

equal:
```

```
    ; Display the message
    mov eax, SYS_WRITE                  ; sys_write
    mov edi, 1                          ; file descriptor 1 is stdout
    mov esi, msg5                       ; address of the message
    mov edx, size5                      ; message length
    syscall

exit:
    ; printing new line
    mov eax, SYS_WRITE                   ; sys_write
    mov edi, 1                           ; file descriptor 1 is stdout
    mov esi, newline                     ; address of the newline
    mov edx, 1                           ; message length
    syscall

    ; Exit program
    syscall         ; invoke the system call
    ret
```

7. Addition 2 numbers:

This assembly code is a calculator program that prompts the user to enter two numbers and then calculates their sum. It defines several messages for user guidance and initializes variables. The code reads input characters for each number, converts them to integers, and constructs the numbers. After summing up the numbers, it reverses the result and prints it. The program then terminates after printing the sum and a newline character.

```
section .data                ; Data section
    msg1: db "Calculator for 4 digits maximum ", 0   ; Define a string message
    size1: equ $ - msg1                 ; Calculate the size of msg1
    newline1: db 0x0A, 0x0D, 0x00    ; Define a newline character
    size_newline1: equ $ - newline1   ; Calculate the size of newline

    msg2: db "Enter first number : ", 0   ; Define a string message
    size2: equ $ - msg2                 ; Calculate the size of msg1
    newline2: db 0x0A, 0x0D, 0x00    ; Define a newline character
    size_newline2: equ $ - newline2   ; Calculate the size of newline

    msg3: db "Enter second number : ", 0   ; Define a string message
```

```
    size3: equ $ - msg3                 ; Calculate the size of msg1
    newline3: db 0x0A, 0x0D, 0x00     ; Define a newline character
    size_newline3: equ $ - newline3    ; Calculate the size of newline

    msg4: db "The sum is : ", 0    ; Define a string message
    size4: equ $ - msg4                 ; Calculate the size of msg1
    newline4: db 0x0A, 0x0D, 0x00     ; Define a newline character
    size_newline4: equ $ - newline4    ; Calculate the size of newline

section .bss                ; Uninitialized data section
    x_buffer: resd 1        ; Reserve space for x_buffer (1 dword)
    x: resd 16              ; Reserve space for x (16 dwords)
    y: resd 16              ; Reserve space for y (16 dwords)
    p: resd 1               ; Reserve space for p (1 dword)
    cnt: resd 1             ; Reserve space for cnt (1 dword)
    ok: resd 1              ; Reserve space for ok (1 dword)
    zero: resd 1            ; Reserve space for zero (1 dword)

section .text               ; Code section
    global add          ; Entry point for the program
add:
    mov eax, 4              ; Set eax to 4 (syscall number for write)
    mov ebx, 1              ; Set ebx to 1 (file descriptor for stdout)
    mov ecx, msg1           ; Set ecx to point to msg1
    mov edx, size1          ; Set edx to the size of msg1
    int 0x80                ; Call kernel

    mov eax, 4              ; Set eax to 4 (syscall number for write)
    mov ebx, 1              ; Set ebx to 1 (file descriptor for stdout)
    mov ecx, newline1        ; Set ecx to point to newline
    mov edx, size_newline1   ; Set edx to the size of newline
    int 0x80                ; Call kernel

    xor eax, eax            ; Set eax to 0

    mov [x], eax            ; Move the value of eax into memory location x
    mov [cnt], eax          ; Move the value of eax into memory location cnt
    mov [ok], eax           ; Move the value of eax into memory location ok

    mov eax, 10             ; Move the value 10 into eax
```

```
    mov [p], eax            ; Move the value of eax into memory location p

    mov eax, 0x30           ; Move the ASCII value for '0' into eax
    mov [zero], eax         ; Move the value of eax into memory location zero
first_label:
    mov eax, 4              ; Set eax to 4 (syscall number for write)
    mov ebx, 1              ; Set ebx to 1 (file descriptor for stdout)
    mov ecx, msg2           ; Set ecx to point to msg2
    mov edx, size2          ; Set edx to the size of msg2
    int 0x80                ; Call kernel

    mov eax, 4              ; Set eax to 4 (syscall number for write)
    mov ebx, 1              ; Set ebx to 1 (file descriptor for stdout)
    mov ecx, newline2        ; Set ecx to point to newline
    mov edx, size_newline2   ; Set edx to the size of newline
    int 0x80                ; Call kernel

et_citire:                 ; Loop label for reading input

    xor edx, edx            ; Set edx to 0
    mov [x_buffer], edx     ; Move the value of edx into memory location x_buffer

    mov eax, 0x3            ; Set eax to 3 (syscall number for read)
    xor ebx, ebx            ; Clear ebx
    mov ecx, x_buffer       ; Set ecx to point to x_buffer
    mov edx, 0x1            ; Set edx to 1 (number of bytes to read)
    int 0x80                ; Call kernel

    mov eax, 0x30           ; Move the ASCII value for '0' into eax

    cmp eax, [x_buffer]     ; Compare the value in x_buffer with the ASCII value for
'0'
    jle verif_mai_mic       ; Jump to verif_mai_mic if less than or equal
    jmp second_label         ; Jump to et_citire_y otherwise

verif_mai_mic:             ; Label for checking if input is greater than or equal to
'0'
    mov eax, 0x39           ; Move the ASCII value for '9' into eax
    cmp eax, [x_buffer]     ; Compare the value in x_buffer with the ASCII value for
'9'
```

```
    jge et_construire_numar ; Jump to et_construire_numar if greater than or equal

jmp second_label            ; Jump to et_citire_y otherwise

et_construire_numar:        ; Label for constructing a number from input
    mov ecx, [p]            ; Move the value at memory location p into ecx
    xor edx, edx            ; Clear edx
    xor eax, eax            ; Clear eax

    mov eax, [x_buffer]     ; Move the value in x_buffer into eax
    sub eax, 0x30           ; Subtract the ASCII value for '0' from eax
    mov ebx, eax            ; Move the result into ebx
    mov eax, [x]            ; Move the value at memory location x into eax
    mul ecx                 ; Multiply eax by ecx
    add eax, ebx            ; Add ebx to eax

    mov [x], eax            ; Move the result into memory location x

    jmp et_citire           ; Jump to et_citire to continue reading input

second_label:
    mov eax, 4              ; Set eax to 4 (syscall number for write)
    mov ebx, 1              ; Set ebx to 1 (file descriptor for stdout)
    mov ecx, msg3           ; Set ecx to point to msg2
    mov edx, size3          ; Set edx to the size of msg2
    int 0x80                ; Call kernel

    mov eax, 4              ; Set eax to 4 (syscall number for write)
    mov ebx, 1              ; Set ebx to 1 (file descriptor for stdout)
    mov ecx, newline3        ; Set ecx to point to newline
    mov edx, size_newline3   ; Set edx to the size of newline
    int 0x80                ; Call kernel

    jmp et_citire_y      ; Jump to et_citire_y to read the second number


et_citire_y:                ; Label for reading input for y
    xor eax, eax            ; Clear eax
    mov [y], eax            ; Move the value of eax into memory location y
```

```
et_citire_y_1:              ; Loop label for reading y input
    xor edx, edx            ; Clear edx
    mov [x_buffer], edx     ; Move the value of edx into memory location x_buffer

    mov eax, 0x3            ; Set eax to 3 (syscall number for read)
    mov ebx, 0x0            ; Clear ebx
    mov ecx, x_buffer       ; Set ecx to point to x_buffer
    mov edx, 0x1            ; Set edx to 1 (number of bytes to read)
    int 0x80                ; Call kernel

    mov eax, 0x30           ; Move the ASCII value for '0' into eax

    cmp eax, [x_buffer]     ; Compare the value in x_buffer with the ASCII value for
'0'
    jle verif_mai_mic_y     ; Jump to verif_mai_mic_y if less than or equal
    jmp continue            ; Jump to continue otherwise

verif_mai_mic_y:            ; Label for checking if input is greater than or equal
to '0'
    mov eax, 0x39           ; Move the ASCII value for '9' into eax
    cmp eax, [x_buffer]     ; Compare the value in x_buffer with the ASCII value for
'9'
    jge et_construire_numar_y ; Jump to et_construire_numar_y if greater than or
equal

jmp continue                ; Jump to continue otherwise

et_construire_numar_y:      ; Label for constructing a number from input for y
    mov ecx, [p]            ; Move the value at memory location p into ecx
    xor edx, edx            ; Clear edx
    xor eax, eax            ; Clear eax

    mov eax, [x_buffer]     ; Move the value in x_buffer into eax
    sub eax, 0x30           ; Subtract the ASCII value for '0' from eax
    mov ebx, eax            ; Move the result into ebx
    mov eax, [y]            ; Move the value at memory location y into eax
    mul ecx                 ; Multiply eax by ecx
    add eax, ebx            ; Add ebx to eax

    mov [y], eax            ; Move the result into memory location y
```

```
    jmp et_citire_y_1       ; Jump to et_citire_y_1 to continue reading input for y

continue:                   ; Label for continuing after reading input for y
    mov eax, [x]            ; Move the value at memory location x into eax
    add [y], eax            ; Add eax to the value at memory location y

    xor eax, eax            ; Clear eax
    mov [x], eax            ; Move the value of eax into memory location x

et_oglindit:                ; Label for reversing y
    xor edx, edx            ; Clear edx
    mov [x_buffer], edx     ; Move the value of edx into memory location x_buffer

    mov eax, [y]            ; Move the value at memory location y into eax
    mov ecx, 10             ; Move the value 10 into ecx
    div ecx                 ; Divide eax by ecx

    cmp edx, 0              ; Compare the remainder with 0
    mov ebx, 1              ; Move the value 1 into ebx
    mov [ok], ebx           ; Move the value of ebx into memory location ok
    je zero_in_coada        ; Jump to zero_in_coada if equal

    jmp resume              ; Jump to resume otherwise

zero_in_coada:              ; Label for handling zero in the queue
    xor ebx, ebx            ; Clear ebx
    cmp ebx, [ok]           ; Compare the value in ok with ebx
    je inc_cnt              ; Jump to inc_cnt if equal

    jmp resume              ; Jump to resume otherwise

inc_cnt:                    ; Label for incrementing cnt
    mov ebx, [cnt]          ; Move the value at memory location cnt into ebx
    inc ebx                 ; Increment ebx
    mov [cnt], ebx          ; Move the value of ebx into memory location cnt

resume:                     ; Label for resuming operations
    mov [x_buffer], edx     ; Move the value of edx into memory location x_buffer
    mov [y], eax            ; Move the value of eax into memory location y
```

```
    xor edx, edx            ; Clear edx
    mov eax, [x]            ; Move the value at memory location x into eax
    mul ecx                 ; Multiply eax by ecx
    add eax, [x_buffer]     ; Add the value at memory location x_buffer to eax

    mov [x], eax            ; Move the result into memory location x

    mov eax, [y]            ; Move the value at memory location y into eax
    cmp eax, 0              ; Compare the value in eax with 0
    jne et_oglindit         ; Jump to et_oglindit if not equal

third_label:
    mov eax, 4              ; Set eax to 4 (syscall number for write)
    mov ebx, 1              ; Set ebx to 1 (file descriptor for stdout)
    mov ecx, msg4           ; Set ecx to point to msg2
    mov edx, size4          ; Set edx to the size of msg2
    int 0x80                ; Call kernel



et_print:                   ; Label for printing the result
    xor edx, edx            ; Clear edx
    mov [x_buffer], edx     ; Move the value of edx into memory location x_buffer

    mov eax, [x]            ; Move the value at memory location x into eax
    mov ecx, 10             ; Move the value 10 into ecx
    div ecx                 ; Divide eax by ecx

    add edx, 0x30           ; Add the ASCII value for '0' to edx
    mov [x_buffer], edx     ; Move the value of edx into memory location x_buffer

    mov [x], eax            ; Move the value of eax into memory location x

    mov eax, 0x4            ; Set eax to 4 (syscall number for write)
    mov ebx, 0x1            ; Set ebx to 1 (file descriptor for stdout)
    mov ecx, x_buffer       ; Set ecx to point to x_buffer
    mov edx, 0x1            ; Set edx to 1 (number of bytes to write)
    int 0x80                ; Call kernel
```

```
    mov eax, [x]              ; Move the value at memory location x into eax
    cmp eax, 0                ; Compare the value in eax with 0
    je afis_zero              ; Jump to afis_zero if equal

    jmp et_print              ; Jump to et_print otherwise

afis_zero:                    ; Label for printing zero
    xor ebx, ebx              ; Clear ebx
    cmp [cnt], ebx            ; Compare the value in cnt with ebx
    je terminate              ; Jump to terminate if equal

    mov eax, 0x4              ; Set eax to 4 (syscall number for write)
    mov ebx, 0x1              ; Set ebx to 1 (file descriptor for stdout)
    mov ecx, zero             ; Set ecx to point to zero
    mov edx, 0x1              ; Set edx to 1 (number of bytes to write)
    int 0x80                  ; Call kernel

    mov eax, [cnt]            ; Move the value at memory location cnt into eax
    dec eax                   ; Decrement eax
    mov [cnt], eax            ; Move the value of eax into memory location cnt

    jmp afis_zero             ; Jump to afis_zero

terminate:                    ; Label for terminating the program
    mov eax, 4                ; Set eax to 4 (syscall number for write)
    mov ebx, 1                ; Set ebx to 1 (file descriptor for stdout)
    mov ecx, newline4          ; Set ecx to point to newline
    mov edx, size_newline4    ; Set edx to the size of newline
    int 0x80                  ; Call kernel

    syscall                   ; Exit the program
    ret                       ; Return from the program
```

8. Factorial:

This assembly code defines a function to calculate the factorial of a number. It prompts the user to input a number, reads the input, and then calls a function (atoi3) to convert the input

string to an integer. After obtaining the number, it calculates its factorial using a loop. The result is then returned for printing. The atoi3 function, utilized for string to integer conversion, iterates over the characters of the input string, converts them to integers, and accumulates the result. Finally, it returns the converted integer value.

```
; factorial of a number

section .data
    ; Messages
    msg1 db 'Enter a number to calculate its factorial: ', 0
    lmsg1 equ $ - msg1

    msg2 db 'Result: ', 0
    lmsg2 equ $ - msg2

    nlinea db 10, 0
    lnlinea equ $ - nlinea

    format_int db "Result: %ld", 10   ; Format string for printing integer

section .bss
    num resb 8
    result resb 64
    input resb 256     ; Buffer to store user input

section .text
    global ex1

ex1:
    ; Print message 1
    mov rax, 1
    mov rdi, 1
    mov rsi, msg1
    mov rdx, lmsg1
    syscall

    ; Read user input
    mov rax, 0                  ; sys_read syscall number
    mov rdi, 0                  ; stdin file descriptor
```

```
    mov rsi, input              ; Address of the input buffer
    mov rdx, 256                ; Maximum number of bytes to read
    syscall

    ; Call atoi3 function to convert input string to integer
    mov rdi, input              ; Pass the address of input string
    call atoi3                   ; Call the atoi3 function
    mov [num], rax              ; Store the result in num

    ; Calculate factorial
    mov rax, 1                  ; Initialize result to 1
    mov rcx, [num]              ; Load the input number

.factorial_loop:
    imul rax, rcx               ; Multiply result by current number
    dec rcx                     ; Decrement current number
    jnz .factorial_loop         ; Repeat until current number is zero

mov rsi, rax                ; Pass the result to be printed
ret

section .text
extern printf
global atoi3

atoi3:
    mov rax, 0              ; Set initial total to 0

convert:
    movzx rsi, byte [rdi]   ; Get the current character
    test rsi, rsi           ; Check for \0
    je done

    cmp rsi, 48             ; Anything less than 0 is invalid
    jl done

    cmp rsi, 57             ; Anything greater than 9 is invalid
    jg done

    sub rsi, 48             ; Convert from ASCII to decimal
```

```
    imul rax, 10            ; Multiply total by 10
    add rax, rsi            ; Add current digit to total

    inc rdi                 ; Get the address of the next character
    jmp convert

done:
    ret                     ; Return total
```

9. Generating random string:

This assembly code prompts the user to input a length, reads the input, and converts it to an integer using a function (atoi). Then, it generates a random string of the specified length using a loop that iterates over the charset defined in the data section. The random string is stored in memory and printed to the console, followed by a newline character. The atoi function converts a string of digits to an integer.

```
section .data
    charset db
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789,;.+-="', 0

    ; Messages
    msg1 db 'Enter length: ', 0
    lmsg1 equ $ - msg1

    str_len equ 64   ; Maximum length of the generated string
    str: times str_len db 0
    nlinea db 10, 0
    lnlinea equ $ - nlinea

section .bss
    len resb 4
    input resb 256     ; Buffer to store user input

section .text
    global p

p:
    ; Print message 1
```

```
    mov rax, 1
    mov rdi, 1
    mov rsi, msg1
    mov rdx, lmsg1
    syscall

    ; Read user input
    mov rax, 0                  ; sys_read syscall number
    mov rdi, 0                  ; stdin file descriptor
    mov rsi, input              ; Address of the input buffer
    mov rdx, 256                ; Maximum number of bytes to read
    syscall

    ; Convert input string to integer
    mov rdi, input              ; Address of the input buffer
    call atoi                   ; Call the atoi function
    mov [len], eax             ; Store the result in len

    ; Generate random string
    mov ecx, eax                ; Set loop counter to the length entered by the user
    lea rdi, [str]

generate_string:
    rdtsc                       ; Get timestamp for random seed
    and eax, 94                 ; Limit to size of charset
    lea rsi, [charset]
    movzx eax, byte [rsi+rax]   ; Get a random character from charset
    stosb                       ; Store character in str
    loop generate_string        ; Repeat until the desired length is reached

    ; Print the string
    mov eax, 1
    mov edi, eax
    lea rsi, [str]
    mov edx, [len]              ; Print only the specified length
    syscall

    ; Print newline
    mov rax, 1
    mov rdi, 1
```

```
    mov rsi, nlinea
    mov rdx, lnlinea
    syscall

ret

atoi:
    xor rax, rax                ; Set initial total to 0

convert:
    movzx rsi, byte [rdi]       ; Get the current character
    test rsi, rsi               ; Check for \0
    je done

    cmp rsi, 48                 ; Anything less than '0' is invalid
    jl done

    cmp rsi, 57                 ; Anything greater than '9' is invalid
    jg done

    sub rsi, 48                 ; Convert from ASCII to decimal
    imul rax, 10                ; Multiply total by 10
    add rax, rsi                ; Add current digit to total

    inc rdi                     ; Get the address of the next character
    jmp convert

done:
    ret                         ; Return total
```

10.Division:

This assembly code prompts the user to enter two numbers, reads the inputs, and converts them to integers using the atoi2 function. Then, it divides the first number by the second and stores the result. Finally, it returns the result to be printed. The atoi2 function converts a string of digits to an integer.

```
section .data
    ; Messages
```

```
    msg1 db 'Enter the first number: ', 0
    lmsg1 equ $ - msg1

    msg2 db 'Enter the second number: ', 0
    lmsg2 equ $ - msg2

    msg3 db 'Result: ', 0
    lmsg3 equ $ - msg3

    nlinea db 10, 0
    lnlinea equ $ - nlinea

    format_int db "Result: %ld", 10   ; Format string for printing integer

section .bss
    num1 resb 8
    num2 resb 8
    result resb 16
    input1 resb 256     ; Buffer to store user input
    input2 resb 256     ; Buffer to store user input

section .text
    global c

c:
    ; Print message 1
    mov rax, 1
    mov rdi, 1
    mov rsi, msg1
    mov rdx, lmsg1
    syscall

    ; Read user input for first number
    mov rax, 0                  ; sys_read syscall number
    mov rdi, 0                  ; stdin file descriptor
    mov rsi, input1             ; Address of the input buffer
    mov rdx, 256                ; Maximum number of bytes to read
    syscall

    ; Print message 2
```

```
    mov rax, 1
    mov rdi, 1
    mov rsi, msg2
    mov rdx, lmsg2
    syscall

    ; Read user input for second number
    mov rax, 0                  ; sys_read syscall number
    mov rdi, 0                  ; stdin file descriptor
    mov rsi, input2             ; Address of the input buffer
    mov rdx, 256                ; Maximum number of bytes to read
    syscall

    ; Call atoi2 function to convert input strings to integers
    mov rdi, input1             ; Pass the address of the first input string
    call atoi2                   ; Call the atoi2 function
    mov [num1], rax             ; Store the result in num1

    mov rdi, input2             ; Pass the address of the second input string
    call atoi2                   ; Call the atoi2 function
    mov [num2], rax             ; Store the result in num2

    ; Divide first number by the second
    mov rax, [num1]             ; Load the first number
    mov rbx, [num2]             ; Load the second number
    xor rdx, rdx                ; Clear RDX for the division operation
    idiv rbx                    ; Divide RDX:RAX by the second number (RBX)
    mov [result], rax           ; Store the result in result


    mov rax, [result]           ; Pass the integer to be printed

ret

section .text
extern printf
global atoi2

atoi2:
    mov rax, 0              ; Set initial total to 0
```

```
convert:
    movzx rsi, byte [rdi]    ; Get the current character
    test rsi, rsi            ; Check for \0
    je done

    cmp rsi, 48              ; Anything less than 0 is invalid
    jl done

    cmp rsi, 57              ; Anything greater than 9 is invalid
    jg done

    sub rsi, 48              ; Convert from ASCII to decimal
    imul rax, 10             ; Multiply total by 10
    add rax, rsi             ; Add current digit to total

    inc rdi                  ; Get the address of the next character
    jmp convert

done:
    ret                      ; Return total
```

MAIN:

This assembly code presents a menu to the user, prompting them to select a process from 1 to 10 or enter 0 to exit. The user's choice is read, converted to an integer using the atoi function, and stored in the num variable. Based on the user's choice, the corresponding process is executed by calling the respective function. If an invalid choice is made, an error message is displayed, and the menu is shown again. Finally, if the user chooses to exit (0), the program terminates.

```
section .data
    newline db 10, 0
    choice_prompt db "Select a process from 1 to 10, or enter 0 to exit:", 10
    menu db "1. Concatening two strings", 10
         db "2. Calculating length of a string", 10
         db "3. Inverting a string", 10
         db "4. Checking if a string is a palindrome", 10
```

```
        db "5. Checking whether a number is odd or even", 10
        db "6. Determining the larger of two numbers", 10
        db "7. Addition 2 numbers", 10
        db "8. Factorial", 10
        db "9. Generating random string", 10
        db "10.Division", 10
        db "0. Exit", 10
        db "Enter your choice : ", 0
    invalid_choice db "Invalid choice. Please try again.", 10
    choice db 0, 0, 0, 0, 0 ; 4 bytes for input and 1 byte for null-terminator
    num dq 0

    format_int db "Result: %ld", 10, 0   ; Format string for printing integer
    format_float db "Result: %f", 10   ; Format string for printing floating point
number

section .text
    global main
    extern concatening, length, r3, cd, odd_checker, bin, add, ex1, p, c,
atoi,printf

main:
    ; display menu
    mov rax, 1           ; syscall number for sys_write
    mov rdi, 1           ; file descriptor 1 (stdout)
    mov rsi, choice_prompt
    mov rdx, 363      ; message length
    syscall

    ; read choice from user
    mov rax, 0           ; syscall number for sys_read
    mov rdi, 0           ; file descriptor 0 (stdin)
    mov rsi, choice
    mov rdx, 4           ; number of bytes to read
    syscall

    ; null-terminate the string
    mov byte [rsi+rdx], 0

    ; Call atoi function to convert input string to integer
```

```
    mov rdi, choice   ; Pass the address of input string
    call atoi               ; Call the atoi function
    mov [num], rax        ; Store the result in num

    ; perform the chosen process
    cmp qword [num], 0
    je exit_program
    cmp qword [num], 1
    je call_ex1
    cmp qword [num], 2
    je call_ex2
    cmp qword [num], 3
    je call_ex3
    cmp qword [num], 4
    je call_ex4
    cmp qword [num], 5
    je call_ex5
    cmp qword [num], 6
    je call_ex6
    cmp qword [num], 7
    je call_ex7
    cmp qword [num], 8
    je call_ex8
    cmp qword [num], 9
    je call_ex9
    cmp qword [num], 10
    je call_ex10

    ; invalid choice
    mov rax, 1           ; syscall number for sys_write
    mov rdi, 1           ; file descriptor 1 (stdout)
    mov rsi, invalid_choice
    mov rdx, 32          ; message length
    syscall
    jmp main

exit_program:
    ; exit program
    mov rax, 60          ; syscall number for sys_exit
    xor rdi, rdi         ; exit code 0
```

```
    syscall

call_ex1:
    call concatening
    jmp main
call_ex2:
    call length
    jmp main
call_ex3:
    call r3
    jmp main
call_ex4:
    call cd
    jmp main
call_ex5:
    call odd_checker
    jmp main
call_ex6:
    call bin
    jmp main
call_ex7:
    call add
    jmp main
call_ex8:
    call ex1
    mov rdi, format_int        ; Pass the format string for printing integer
    mov rsi, rax             ; Pass the integer to be printed
    xor rax, rax               ; Clear RAX register for syscall number (sys_write)
    call printf                ; Call printf function
    jmp main
call_ex9:
    call p
    jmp main

call_ex10:
    call c
    mov rdi, format_int        ; Pass the format string for printing integer
    mov rsi, rax             ; Pass the integer to be printed
    xor rax, rax               ; Clear RAX register for syscall number (sys_write)
    call printf                ; Call printf function
```

```
jmp main
```

Result:

```
russian_12@DESKTOP-77U3BH7:~$ code .
russian_12@DESKTOP-77U3BH7:~$ cd arhlab/Lab4
russian_12@DESKTOP-77U3BH7:~/arhlab/Lab4$ ./m2
Select a process from 1 to 10, or enter 0 to exit:
1. Concatening two strings
2. Calculating length of a string
3. Inverting a string
4. Checking if a string is a palindrome
5. Checking whether a number is odd or even
6. Determining the larger of two numbers
7. Addition 2 numbers
8. Factorial
9. Generating random string
10.Division
0. Exit
Enter your choice : 1
Enter first string: ee
Enter second string: rr
The concatenated string is: eerr
Select a process from 1 to 10, or enter 0 to exit:
1. Concatening two strings
2. Calculating length of a string
3. Inverting a string
4. Checking if a string is a palindrome
5. Checking whether a number is odd or even
6. Determining the larger of two numbers
7. Addition 2 numbers
8. Factorial
9. Generating random string
10.Division
0. Exit
Enter your choice : 2
Enter a string:eee
Length:3
Select a process from 1 to 10, or enter 0 to exit:
1. Concatening two strings
2. Calculating length of a string
3. Inverting a string
4. Checking if a string is a palindrome
5. Checking whether a number is odd or even
6. Determining the larger of two numbers
7. Addition 2 numbers
8. Factorial
9. Generating random string
10.Division
0. Exit
```

```
Enter your choice : 3
tyu

uyt
Reversed string printed successfully.
Select a process from 1 to 10, or enter 0 to exit:
1. Concatening two strings
2. Calculating length of a string
3. Inverting a string
4. Checking if a string is a palindrome
5. Checking whether a number is odd or even
6. Determining the larger of two numbers
7. Addition 2 numbers
8. Factorial
9. Generating random string
10.Division
0. Exit
Enter your choice : 4
Enter the string : rryrr
rryrr - This is a pallindrome
Select a process from 1 to 10, or enter 0 to exit:
1. Concatening two strings
2. Calculating length of a string
3. Inverting a string
4. Checking if a string is a palindrome
5. Checking whether a number is odd or even
6. Determining the larger of two numbers
7. Addition 2 numbers
8. Factorial
9. Generating random string
10.Division
0. Exit
Enter your choice : 5
Enter the number : 7
The number in odd

Select a process from 1 to 10, or enter 0 to exit:
1. Concatening two strings
2. Calculating length of a string
3. Inverting a string
4. Checking if a string is a palindrome
5. Checking whether a number is odd or even
6. Determining the larger of two numbers
7. Addition 2 numbers
8. Factorial
9. Generating random string
10.Division
0. Exit
```

```
Enter your choice : 6
Enter first number : 45
Enter second number :  32
First number is bigger !

Select a process from 1 to 10, or enter 0 to exit:
1. Concatening two strings
2. Calculating length of a string
3. Inverting a string
4. Checking if a string is a palindrome
5. Checking whether a number is odd or even
6. Determining the larger of two numbers
7. Addition 2 numbers
8. Factorial
9. Generating random string
10.Division
0. Exit
Enter your choice : 7
Calculator for 4 digits maximum
Enter first number :
34
Enter second number :
23
The sum is : 57
Select a process from 1 to 10, or enter 0 to exit:
1. Concatening two strings
2. Calculating length of a string
3. Inverting a string
4. Checking if a string is a palindrome
5. Checking whether a number is odd or even
6. Determining the larger of two numbers
7. Addition 2 numbers
8. Factorial
9. Generating random string
10.Division
0. Exit
Enter your choice : 8
Enter a number to calculate its factorial: 5
Result: 120
```

```
Enter your choice : 8
Enter a number to calculate its factorial: 5
Result: 120
Select a process from 1 to 10, or enter 0 to exit:
1. Concatening two strings
2. Calculating length of a string
3. Inverting a string
4. Checking if a string is a palindrome
5. Checking whether a number is odd or even
6. Determining the larger of two numbers
7. Addition 2 numbers
8. Factorial
9. Generating random string
10.Division
0. Exit
Enter your choice : 9
Enter length: 2
ge
Select a process from 1 to 10, or enter 0 to exit:
1. Concatening two strings
2. Calculating length of a string
3. Inverting a string
4. Checking if a string is a palindrome
5. Checking whether a number is odd or even
6. Determining the larger of two numbers
7. Addition 2 numbers
8. Factorial
9. Generating random string
10.Division
0. Exit
Enter your choice : 10
Enter the first number: 34
Enter the second number: 2
Result: 17
Select a process from 1 to 10, or enter 0 to exit:
1. Concatening two strings
2. Calculating length of a string
3. Inverting a string
4. Checking if a string is a palindrome
5. Checking whether a number is odd or even
6. Determining the larger of two numbers
7. Addition 2 numbers
8. Factorial
9. Generating random string
10.Division
0. Exit
Enter your choice : 0
```

Conclusion:

In conclusion, the program involves various functionalities, including string manipulation, arithmetic operations, and input/output handling. Each code segment defines specific tasks, such as concatenating strings, calculating string length, inverting strings, checking for palindromes, determining odd/even numbers, performing arithmetic operations, calculating factorials, generating random strings, and performing divisions. The program presents a menu to the user, allowing them to select one of these functionalities or exit the program. Input is read from the user, converted to appropriate data types if necessary, and then processed accordingly. Error handling is implemented to deal with invalid user choices, ensuring robustness and usability. Overall, the program demonstrates a comprehensive approach to assembly language programming, covering a range of fundamental tasks and providing a user-friendly interface.