Ministerul Educaţiei și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work 5:

Introduction in NASM

Elaborat:

st. gr. FAF-221 Berco Andrei

Verificat:

asist. Univ. Voitcovschi Vladislav

Chişinău - 2024

Purpose of the work:

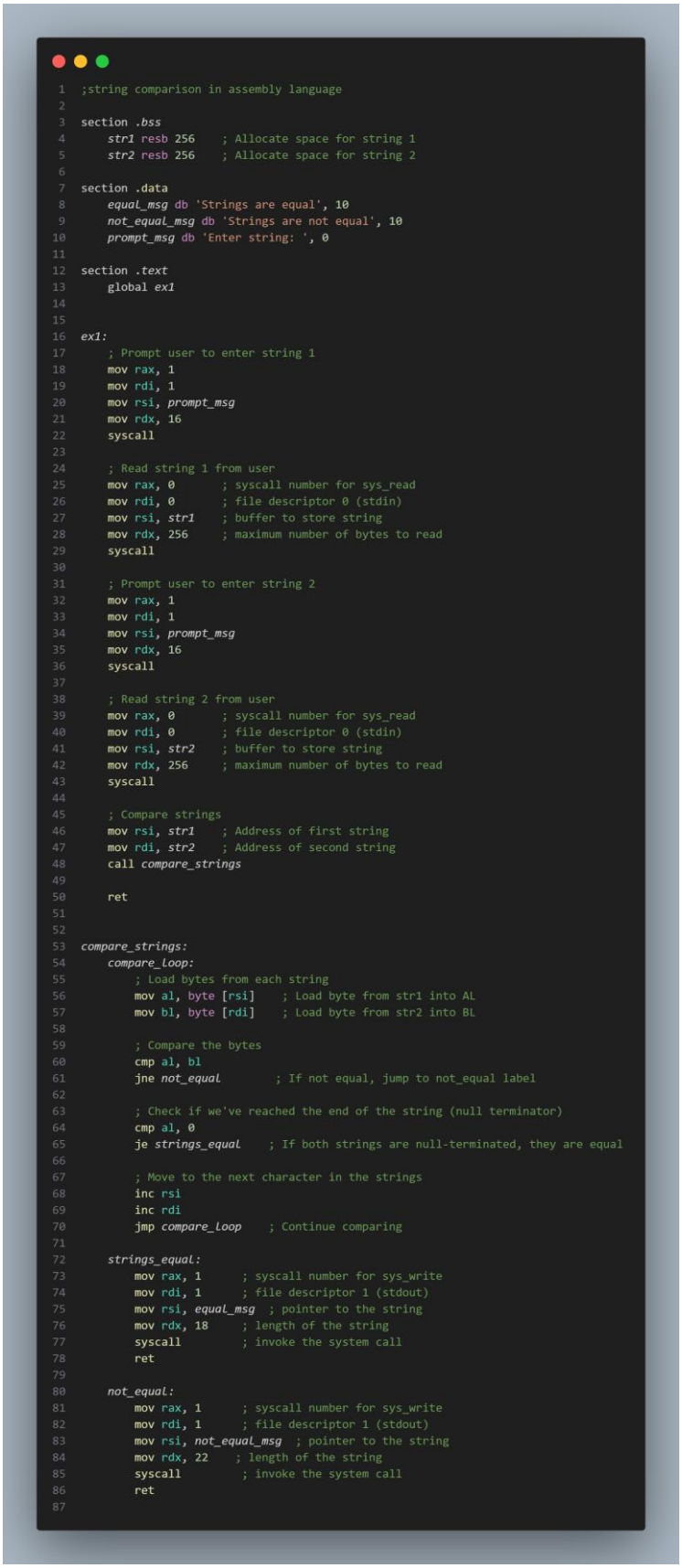
1. Write an interactive menu that allows the user to choose from the 10 processes.

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

Exercise 2: Comparing 2 strings



```
;string comparison in assembly language

section .bss
    str1 resb 256    ; Allocate space for string 1
    str2 resb 256    ; Allocate space for string 2

section .data
    equal_msg db 'Strings are equal', 10
    not_equal_msg db 'Strings are not equal', 10
    prompt_msg db 'Enter string: ', 0

section .text
    global ex1


ex1:
    ; Prompt user to enter string 1
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt_msg
    mov rdx, 16
    syscall

    ; Read string 1 from user
    mov rax, 0       ; syscall number for sys_read
    mov rdi, 0       ; file descriptor 0 (stdin)
    mov rsi, str1    ; buffer to store string
    mov rdx, 256     ; maximum number of bytes to read
    syscall

    ; Prompt user to enter string 2
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt_msg
    mov rdx, 16
    syscall

    ; Read string 2 from user
    mov rax, 0       ; syscall number for sys_read
    mov rdi, 0       ; file descriptor 0 (stdin)
    mov rsi, str2    ; buffer to store string
    mov rdx, 256     ; maximum number of bytes to read
    syscall

    ; Compare strings
    mov rsi, str1    ; Address of first string
    mov rdi, str2    ; Address of second string
    call compare_strings

    ret


compare_strings:
    compare_loop:
        ; Load bytes from each string
        mov al, byte [rsi]    ; Load byte from str1 into AL
        mov bl, byte [rdi]    ; Load byte from str2 into BL

        ; Compare the bytes
        cmp al, bl
        jne not_equal        ; If not equal, jump to not_equal label

        ; Check if we've reached the end of the string (null terminator)
        cmp al, 0
        je strings_equal    ; If both strings are null-terminated, they are equal

        ; Move to the next character in the strings
        inc rsi
        inc rdi
        jmp compare_loop    ; Continue comparing

    strings_equal:
        mov rax, 1      ; syscall number for sys_write
        mov rdi, 1      ; file descriptor 1 (stdout)
        mov rsi, equal_msg  ; pointer to the string
        mov rdx, 18     ; length of the string
        syscall         ; invoke the system call
        ret

    not_equal:
        mov rax, 1      ; syscall number for sys_write
        mov rdi, 1      ; file descriptor 1 (stdout)
        mov rsi, not_equal_msg  ; pointer to the string
        mov rdx, 22    ; length of the string
        syscall         ; invoke the system call
        ret

```

Figure 1: Code for ex2

Explanation:

This assembly code facilitates string comparison by prompting the user to input two strings. It allocates memory space for both strings and displays a prompt message for each input. After the user provides the strings, the code compares them byte by byte until it either finds a mismatch or reaches the end of one of the strings.

Within the `compare_strings` subroutine, the code iterates through the characters of the strings, loading bytes from each string and comparing them. If a byte mismatch occurs, it jumps to the `not_equal` label, indicating that the strings are not equal. If the code reaches the end of both strings without finding any mismatches, it concludes that the strings are equal and proceeds to output "Strings are equal". If a string reaches its null terminator while the other does not, it also implies inequality.

The output messages "Strings are equal" and "Strings are not equal" are displayed accordingly, and the program returns after displaying the appropriate message.

Exercise 9: Inverting string

```
;invert string

section .data
    msg times 64 db 0
    max equ 64
    prompt_msg db 'Enter string: ', 0
        nlinea db 10, 0
    lnlinea equ $ - nlinea


section .text
    global ex2

ex2:
    ; Prompt user to enter string 1
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt_msg
    mov rdx, 16
    syscall

    ; Read
    mov rdx,max
    mov rsi,msg
    mov rdi,0
    mov rax,0
    syscall

    mov rcx,rax     ; Copy the string for later
    mov rdi,msg     ; Set RDI and RSI to point at message
    mov rsi,msg     ;
    add rdi,rax     ; RDI should point at last character in message
    dec rdi         ;
    shr rax,1       ; Divide length by 2

    .loop           ; Begin loop:
    mov bl,[rsi]    ; Swap the characters using 8 bit registers
    mov bh,[rdi]    ;
    mov [rsi],bh    ;
    mov [rdi],bl    ;
    inc rsi         ; Increment rsi (which is a pointer)
    dec rdi         ; Decrement rdi (also a pointer)
    dec rax         ; Decrement our counter
    jnz .loop       ; If our counter isn't zero, keep looping

    ; Write
    .write
    mov rdx,rcx ; nbytes
    mov rsi,msg ; *msg
    mov rdi,1 ; stream
    mov rax,1 ; syscall
    syscall

        ; Print newline
    mov rax, 1
    mov rdi, 1
    mov rsi, nlinea
    mov rdx, lnlinea
    syscall

ret
```

Figure 2: Code for ex9

Explanation:

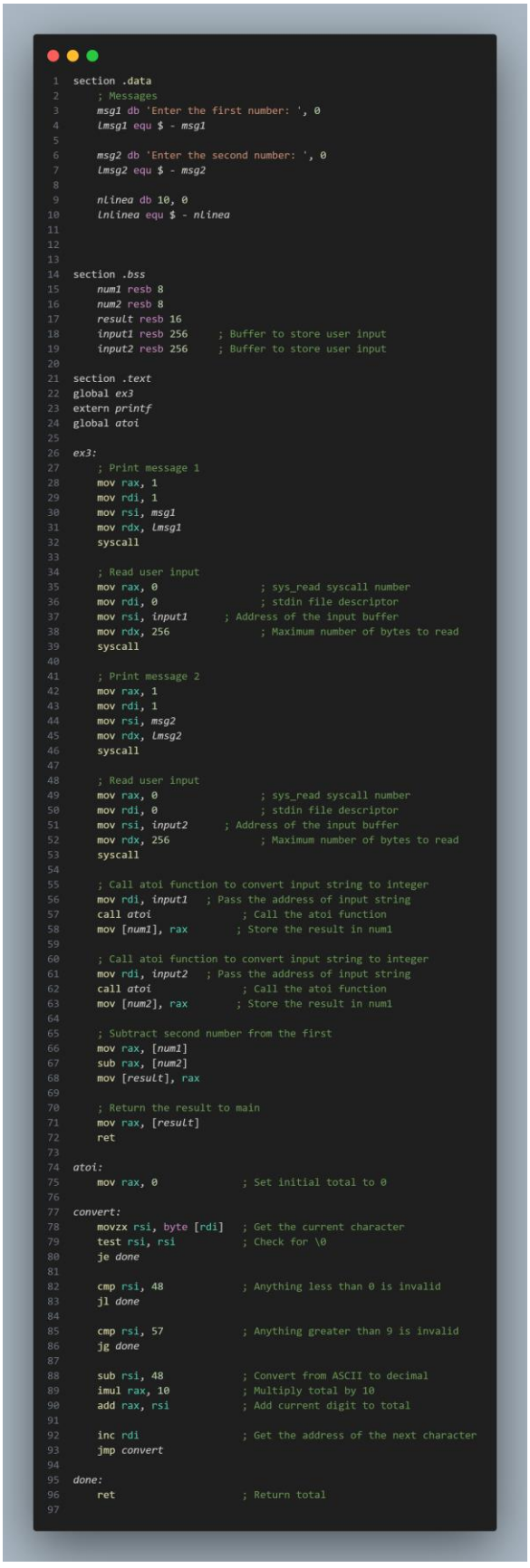
This assembly code is designed to invert a string entered by the user. Initially, it prompts the user to input a string and then reads it. The entered string is stored in memory, and its length is calculated.

The main logic of string inversion is implemented in the `.loop` section. It utilizes two pointers, `rsi` and `rdi`, to traverse the string from both ends towards the center. Within the loop, it swaps characters using 8-bit registers, effectively reversing the string. This process continues until the pointers meet at the middle of the string.

After the inversion, the code writes the inverted string back to the standard output. Finally, it prints a newline character to ensure proper formatting and returns.

Overall, this code efficiently reverses a string using assembly language principles, showcasing low-level manipulation of data in memory.

Exercise 2(math): Substraction of 2 numbers



```
section .data
    ; Messages
    msg1 db 'Enter the first number: ', 0
    lmsg1 equ $ - msg1

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

    nlinea db 10, 0
    lnlinea equ $ - nlinea



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 ex3
extern printf
global atoi

ex3:
    ; 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, 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
    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 atoi function to convert input string to integer
    mov rdi, input1   ; Pass the address of input string
    call atoi               ; Call the atoi function
    mov [num1], rax        ; Store the result in num1

    ; Call atoi function to convert input string to integer
    mov rdi, input2   ; Pass the address of input string
    call atoi               ; Call the atoi function
    mov [num2], rax        ; Store the result in num1

    ; Subtract second number from the first
    mov rax, [num1]
    sub rax, [num2]
    mov [result], rax

    ; Return the result to main
    mov rax, [result]
    ret

atoi:
    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

```

Figure 3: Code for ex2 (math)

Explanation:

This assembly code takes two numbers as input from the user, subtracts the second number from the first, and returns the result. It contains a data section defining messages for user prompts and memory space for storing numbers and input buffers.

In the text section, the main routine (`ex3`) prompts the user for input, reads the entered numbers, converts them to integers using the `atoi` function, subtracts the second number from the first, and returns the result.

The `atoi` function converts a string to an integer by iterating through each character, converting ASCII characters to their respective integer values, and accumulating the total. It stops when it encounters a null terminator and returns the total.

Overall, the code demonstrates input handling, string-to-integer conversion, arithmetic operations, and value return in assembly language.

Exercise 19: Converting a string to an integer

```
;string to int conversion

section .bss
    input_buffer resb 256     ; Buffer to store user input

section .data
    input_prompt db "Enter the string to convert: ", 0  ; Prompt string
    format_int db "Coverted integer: %ld", 10   ; Format string for printing integer
section .text
    global _start

ex4:

    ; Display prompt message
    mov rax, 1                  ; syscall number for sys_write
    mov rdi, 1                  ; file descriptor 1 (stdout)
    mov rsi, input_prompt             ; pointer to the prompt message
    mov rdx, 30                 ; length of the prompt message
    syscall                     ; invoke the system call

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

    ; Call atoi1 function to convert input string to integer
    mov rdi, input_buffer   ; Pass the address of input string
    call atoi1               ; Call the atoi1 function


    ret

section .text
extern printf

global atoi1
global ex4

atoi1:
    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

```

Figure 4: Code for ex19

Explanation:

This assembly code is designed to convert a string input by the user into an integer. It contains a data section defining memory space for storing user input and strings for prompting and formatting.

In the text section, the main routine (`ex4`) displays a prompt asking the user to input a string, reads the user input, and then calls the `atoi1` function to perform the conversion.

The `atoi1` function converts the input string to an integer by iterating through each character, converting ASCII characters to their corresponding integer values, and accumulating the total. It stops when encountering a null terminator and returns the total.

Overall, this code demonstrates a simple approach to string-to-integer conversion in assembly language.

Exercise 4 (math): Dividing 2 numbers

```
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 ex5

ex5:
    ; 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
```

Figure 5: Code for ex4 math

Explanation:

This assembly code manages the input of two numbers from the user, divides the first number by the second, and presents the result.

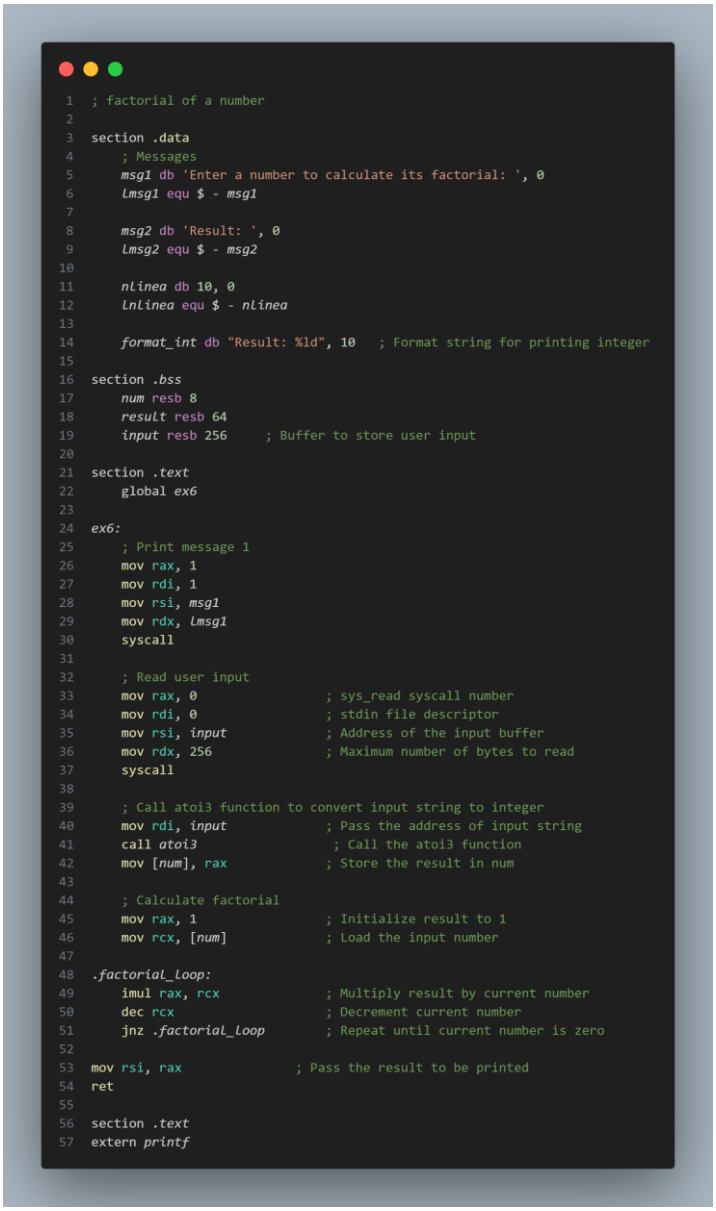
In the data section, it defines messages prompting the user for the numbers and for displaying the result, along with a newline character for formatting and a format string for printing integers.

Within the text section, the `ex5` routine displays prompts for both numbers, reads the input, converts the strings to integers using the `atoi2` function, performs the division operation, and stores the result. Afterward, it returns.

The `atoi2` function converts a string to an integer. It iterates through each character of the string, converting ASCII characters to integers, and accumulates the total, which is then returned.

In summary, this code demonstrates input handling, string-to-integer conversion, arithmetic operations, and result presentation in assembly language.

Exercise 6 (math): Calculating the factorial of a number



```
; 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 ex6

ex6:
    ; 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
```

Figure 6: Code for ex6 math

Explanation:

This assembly code calculates the factorial of a number entered by the user. Here's how it works:

In the data section, it defines messages for prompting the user to enter a number and for displaying the result, along with newline characters for formatting and a format string for printing integers.

Within the text section, the `ex6` routine:

- Displays the prompt message for the number, reads the input, and converts it to an integer using the `atoi3` function.

- Stores the result in the `num` variable.

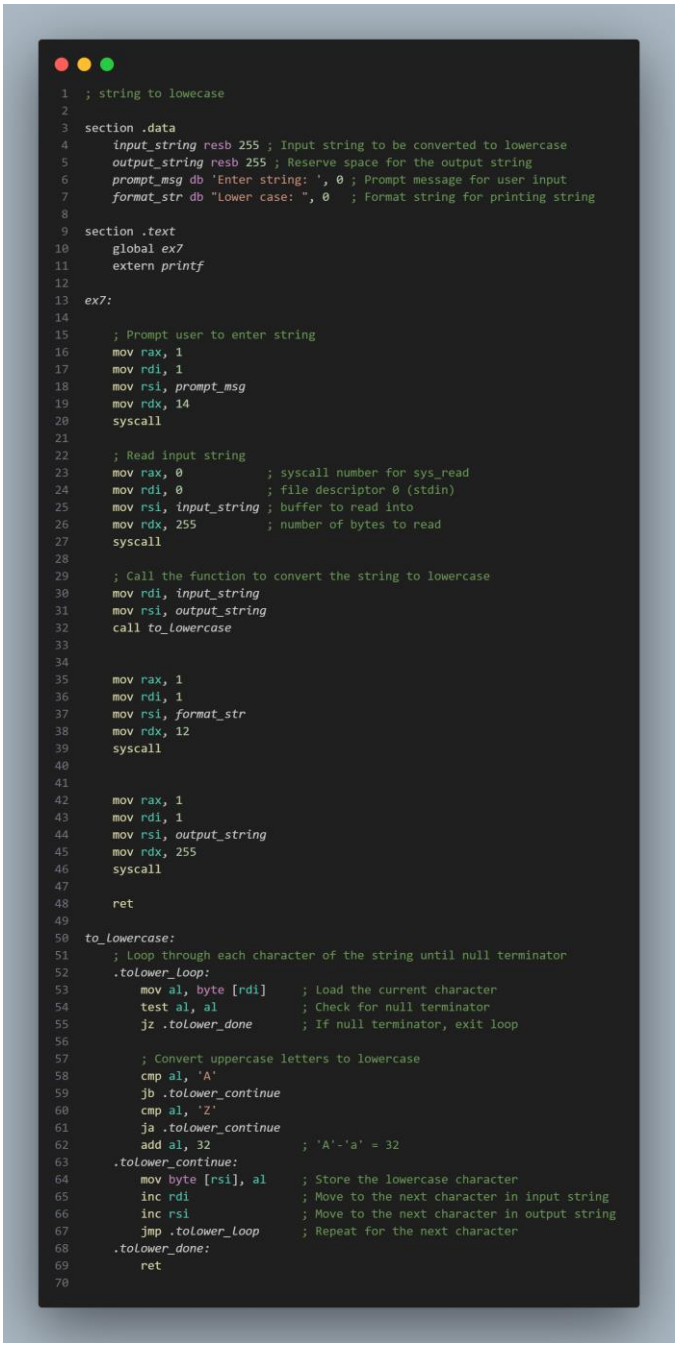
- Calculates the factorial of the input number using a loop, multiplying the current result by the current number and decrementing the number until it reaches zero.

- Returns the factorial result.

The `atoi3` function converts a string to an integer by iterating through each character of the string, converting ASCII characters to integers, and accumulating the total, which is then returned.

In summary, this code demonstrates input handling, string-to-integer conversion, factorial calculation, and result presentation in assembly language.

Exercise 6 (string): Converting a string to lower case



```
; string to lowecase

section .data
    input_string resb 255 ; Input string to be converted to lowercase
    output_string resb 255 ; Reserve space for the output string
    prompt_msg db 'Enter string: ', 0 ; Prompt message for user input
    format_str db "Lower case: ", 0   ; Format string for printing string

section .text
    global ex7
    extern printf

ex7:

    ; Prompt user to enter string
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt_msg
    mov rdx, 14
    syscall

    ; Read input string
    mov rax, 0            ; syscall number for sys_read
    mov rdi, 0            ; file descriptor 0 (stdin)
    mov rsi, input_string ; buffer to read into
    mov rdx, 255          ; number of bytes to read
    syscall

    ; Call the function to convert the string to lowercase
    mov rdi, input_string
    mov rsi, output_string
    call to_lowercase


    mov rax, 1
    mov rdi, 1
    mov rsi, format_str
    mov rdx, 12
    syscall


    mov rax, 1
    mov rdi, 1
    mov rsi, output_string
    mov rdx, 255
    syscall

    ret

to_lowercase:
    ; Loop through each character of the string until null terminator
    .tolower_loop:
        mov al, byte [rdi]     ; Load the current character
        test al, al            ; Check for null terminator
        jz .tolower_done       ; If null terminator, exit loop

        ; Convert uppercase letters to lowercase
        cmp al, 'A'
        jb .tolower_continue
        cmp al, 'Z'
        ja .tolower_continue
        add al, 32             ; 'A'-'a' = 32
    .tolower_continue:
        mov byte [rsi], al     ; Store the lowercase character
        inc rdi                ; Move to the next character in input string
        inc rsi                ; Move to the next character in output string
        jmp .tolower_loop      ; Repeat for the next character
    .tolower_done:
        ret

```

Figure 7: Code for ex6 string

Explanation:

This assembly code converts a string entered by the user into lowercase characters. Here's how it functions:

In the data section, it reserves memory for the input string, the output string, and defines a prompt message for user input, along with a format string for displaying the converted string.

Within the text section, the `ex7` routine:

- Prompts the user to input a string.

- Reads the input string.

- Calls the `to_lowercase` function to convert the string to lowercase.

- Prints the format string indicating the converted string.

- Displays the converted string.

The `to_lowercase` function:

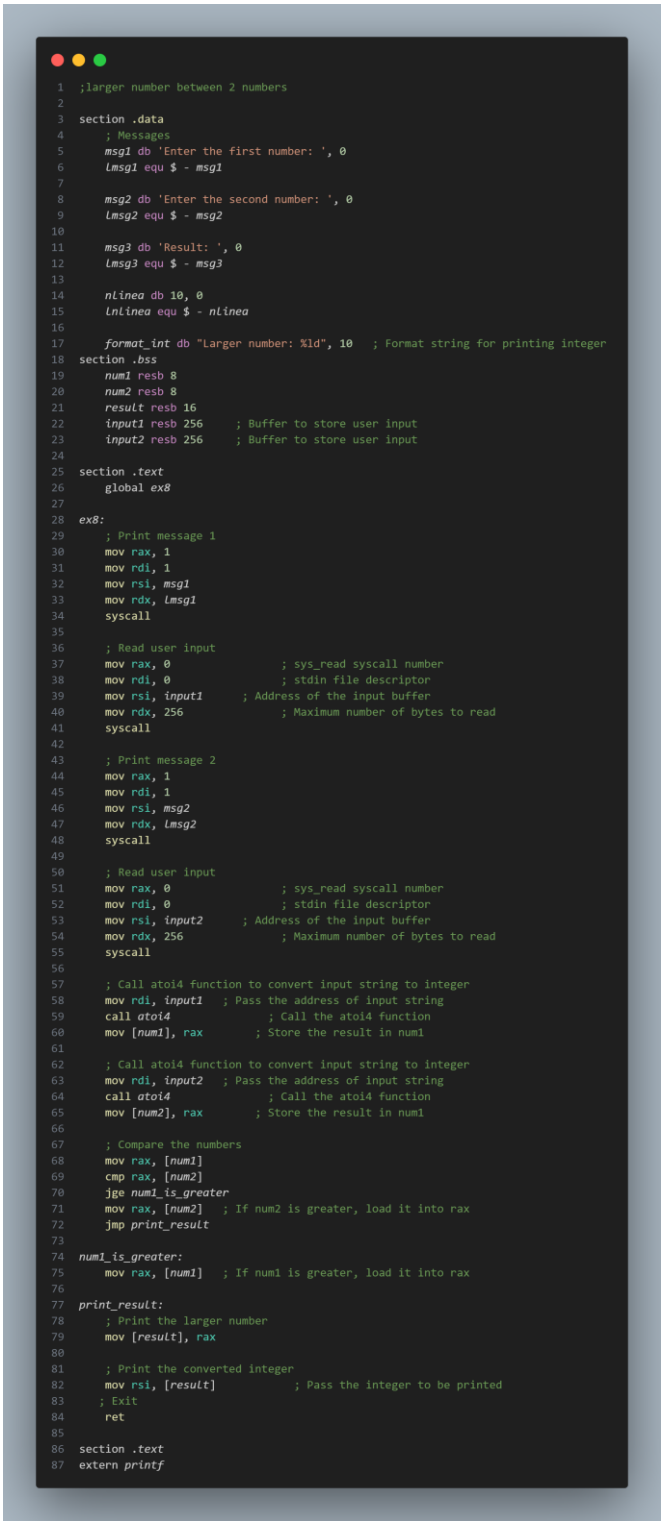
- Iterates through each character of the input string until it encounters a null terminator.

- Converts uppercase letters to lowercase by adding 32 to their ASCII values.

- Stores the lowercase characters in the output string.

Overall, this code demonstrates the conversion of a string to lowercase characters in assembly language.

Exercise 15 (math): Deterining the larger of 2 numbers



```
;larger number between 2 numbers

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 "Larger number: %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 ex8

ex8:
    ; 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, 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
    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 atoi4 function to convert input string to integer
    mov rdi, input1   ; Pass the address of input string
    call atoi4              ; Call the atoi4 function
    mov [num1], rax       ; Store the result in num1

    ; Call atoi4 function to convert input string to integer
    mov rdi, input2   ; Pass the address of input string
    call atoi4              ; Call the atoi4 function
    mov [num2], rax       ; Store the result in num1

    ; Compare the numbers
    mov rax, [num1]
    cmp rax, [num2]
    jge num1_is_greater
    mov rax, [num2]   ; If num2 is greater, load it into rax
    jmp print_result

num1_is_greater:
    mov rax, [num1]   ; If num1 is greater, load it into rax

print_result:
    ; Print the larger number
    mov [result], rax

    ; Print the converted integer
    mov rsi, [result]           ; Pass the integer to be printed
   ; Exit
    ret

section .text
extern printf
```

Figure 8: Code for ex15 math

Explanation:

This assembly code determines the larger of two numbers entered by the user. Here's how it works:

In the data section, it defines messages prompting the user to enter the two numbers, a message for displaying the result, newline characters for formatting, and a format string for printing integers.

Within the text section, the `ex8` routine:

- Displays the prompts for both numbers, reads the input, and converts them to integers using the `atoi4` function.

- Compares the two numbers and selects the larger one.

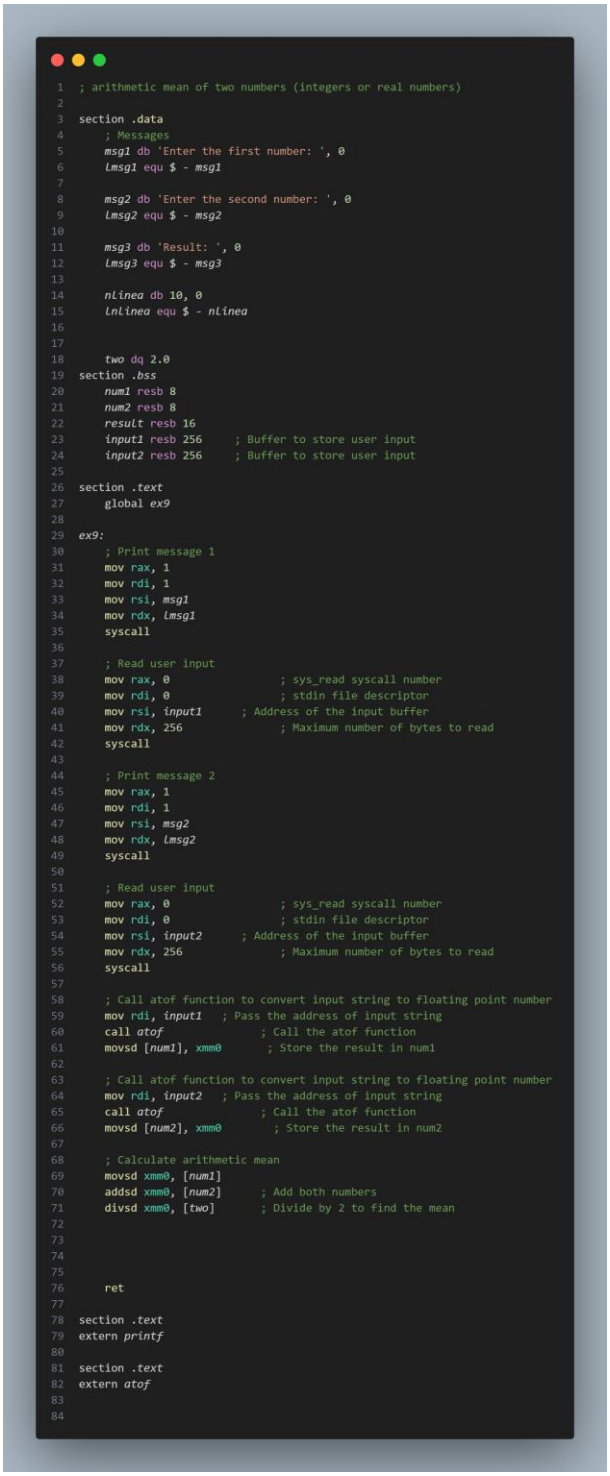
- Stores the result in the `result` variable.

- Returns.

The `atoi4` function converts a string to an integer. It iterates through each character of the string, converting ASCII characters to integers, and accumulating the total, which is then returned.

In summary, this code demonstrates input handling, string-to-integer conversion, comparison, and result presentation in assembly language.

Exercise 17 (math): Determining the arithmetic mean of two numbers



```
; arithmetic mean of two numbers (integers or real numbers)

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


    two dq 2.0
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 ex9

ex9:
    ; 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, 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
    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 atof function to convert input string to floating point number
    mov rdi, input1   ; Pass the address of input string
    call atof               ; Call the atof function
    movsd [num1], xmm0       ; Store the result in num1

    ; Call atof function to convert input string to floating point number
    mov rdi, input2   ; Pass the address of input string
    call atof               ; Call the atof function
    movsd [num2], xmm0        ; Store the result in num2

    ; Calculate arithmetic mean
    movsd xmm0, [num1]
    addsd xmm0, [num2]      ; Add both numbers
    divsd xmm0, [two]       ; Divide by 2 to find the mean



    ret

section .text
extern printf

section .text
extern atof
```

Figure 9: Code for ex17 math

Explanation:

This assembly code computes the arithmetic mean of two numbers input by the user, which can be integers or real numbers.

In the data section, it defines messages prompting the user to enter the numbers, a message for displaying the result, newline characters for formatting, and a constant `two` initialized with the value 2.0.

In the text section, the `ex9` routine:

- Prompts the user to enter the first number, reads the input, and converts it to a floating-point number using the `atof` function.

- Prompts for the second number, reads the input, and converts it to a floating-point number as well.

- Calculates the arithmetic mean by adding both numbers, dividing the result by 2, and storing the mean in the `xmm0` register.

- Returns.

The `atof` function converts a string to a floating-point number.

Overall, this code demonstrates input handling, string-to-floating-point conversion, arithmetic operations, and result presentation in assembly language.

Exercise 13 (string): Generating a random string

```
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 ex10

ex10:
    ; 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
```

Figure 10: Code for ex13 string

Explanation:

This assembly code generates a random string of a specified length entered by the user. Here's how it operates:

In the data section, it initializes a character set containing alphanumeric and special characters, along with messages for prompting the user to enter the desired string length, a maximum string length, and newline characters for formatting.

Within the text section, the `ex10` routine:

- Prompts the user to enter the desired length of the string.

- Reads the user input, converts it to an integer using the `atoi` function, and stores the result.

- Generates a random string of the specified length by selecting characters from the charset based on a timestamp for random seed.

- Prints the generated string of the specified length.

- Prints a newline character for formatting.

he `atoi` function converts a string to an integer by iterating through each character of the string, converting ASCII characters to integers, and accumulating the total, which it then returns.

In summary, this code demonstrates input handling, string-to-integer conversion, random string generation, and result presentation in assembly language.

Main menu:

```
section .data
    choice_prompt db "Select a process from 1 to 10, or enter 0 to exit:", 10
    menu db "1. Compare strings", 10
         db "2. Inverting a string", 10
         db "3. Substraction of 2 numbers", 10
         db "4. String to int conversion", 10
         db "5. Division of 2 numbers", 10
         db "6. Factorial of a given number", 10
         db "7. String to lower case", 10
         db "8. Larger of 2 numbers", 10
         db "9. Arithmetic mean of 2 integer/real numbers", 10
         db "10. Generate a random string", 10
         db "0. Exit", 10
    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 ex1, ex2, ex3, ex4, ex5, ex6, ex7, ex8, ex9, ex10, 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, 334      ; 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 ex1
    jmp main

call_ex2:
    call ex2
    jmp main

call_ex3:
    call ex3
    ; Print the converted integer
    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_ex4:
    call ex4
     ; Print the converted integer
    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_ex5:
    call ex5
    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_ex6:
    call ex6
    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_ex7:
    call ex7
    jmp main

call_ex8:
    call ex8
    mov rdi, format_int     ; Pass the format string for printing integer
    xor rax, rax            ; Clear RAX register for syscall number (sys_write)
    call printf             ; Call printf function
    jmp main

call_ex9:
    call ex9
     ; Print the result
    mov rdi, format_float
    mov rax, 1
    call printf
    jmp main

call_ex10:
    call ex10
    jmp main
```

Figure 11: Code for main menu

Explanation:

This assembly code presents a menu to the user, allowing them to choose a process to execute from a list of ten options. It handles the input, performs the chosen operation, and displays the result.

In the data section, it defines messages for prompting the user and for displaying an invalid choice message, along with format strings for printing integers and floating-point numbers.

Within the text section, the `main` routine:

- Displays the menu prompt, reads the user's choice, and converts it to an integer.

- Based on the choice, it calls the corresponding subroutine (`ex1` to `ex10`) to execute the selected operation.

- If the choice is invalid, it displays an error message and prompts the user again.

The `ex1` to `ex10` subroutines represent different processes the user can select. After executing the chosen process, the program returns to the main menu prompt.

Overall, this code demonstrates a menu-driven program in assembly language, where the user selects an operation, the program performs it, and then returns to the menu for further actions.

Screenshots:

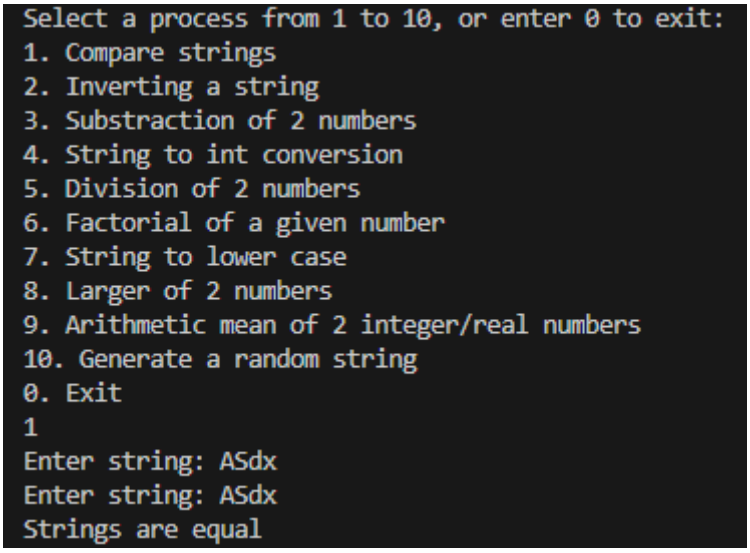


Figure 12: String comparing

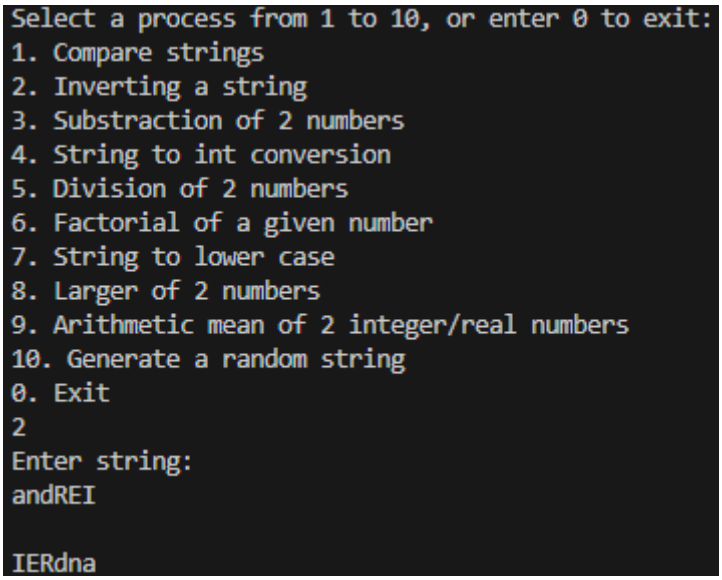


Figure 13: String inversion

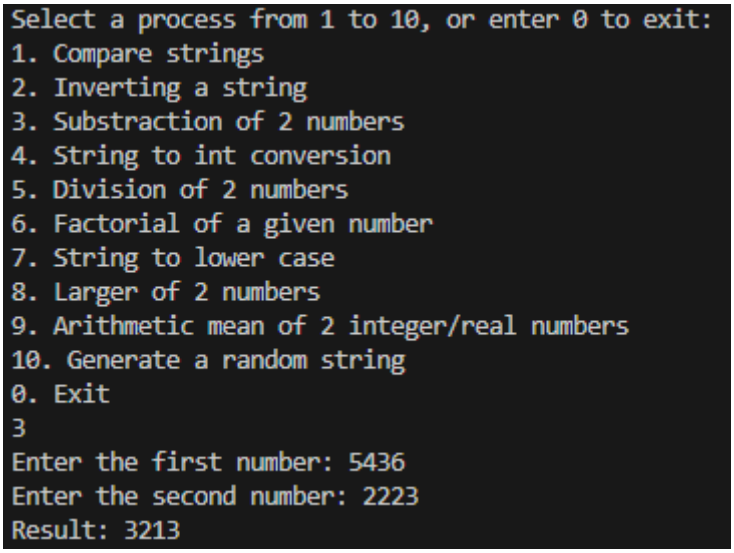


Figure 14: Subtraction of 2 numbers

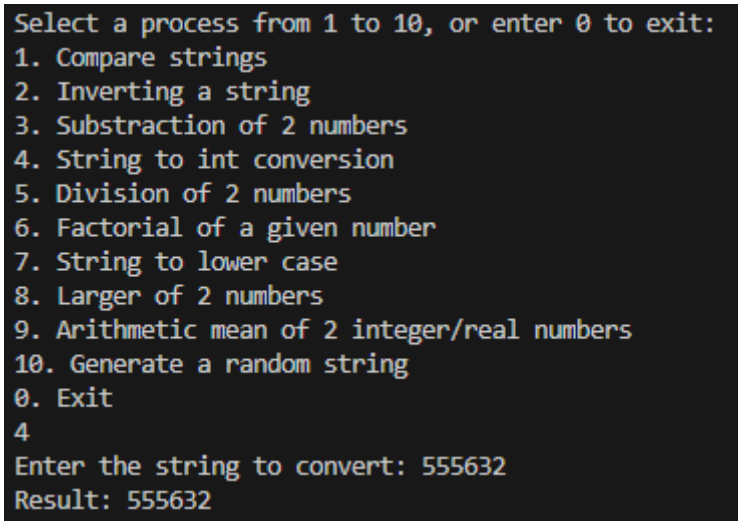


Figure 15: String to int conversion

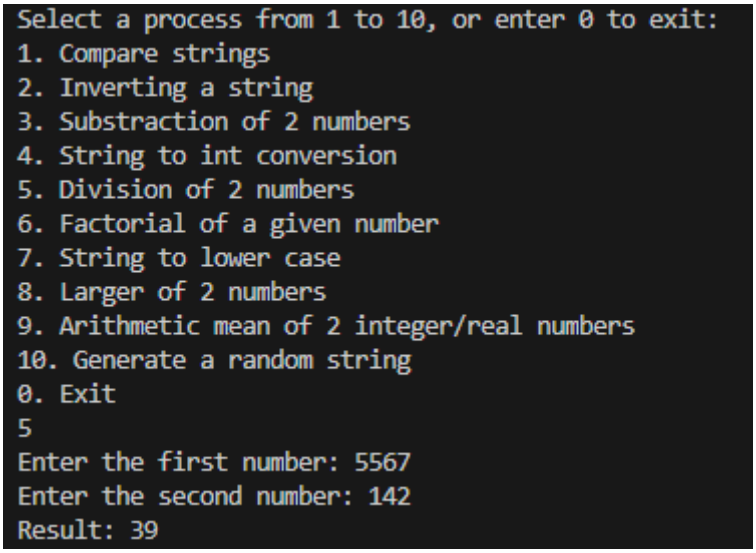


Figure 16: Division of 2 numbers

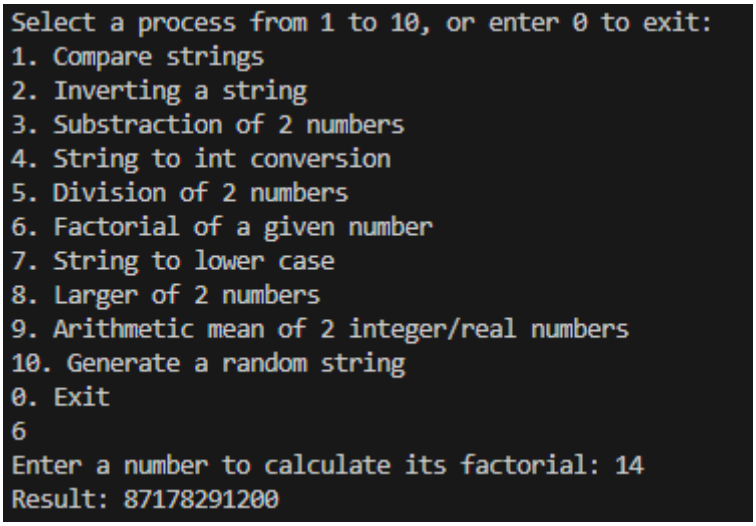


Figure 17: Factorial of a number

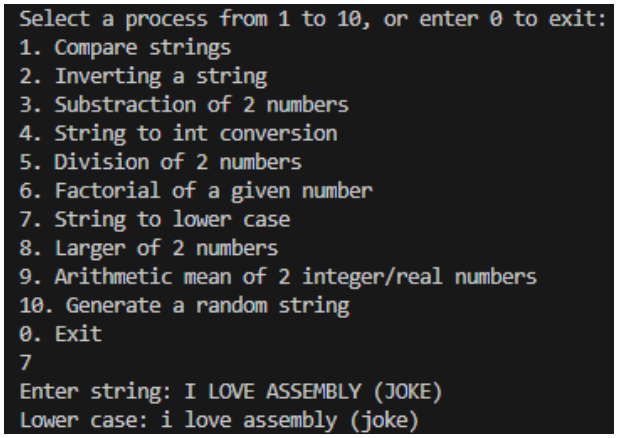


Figure 18: String to lower case

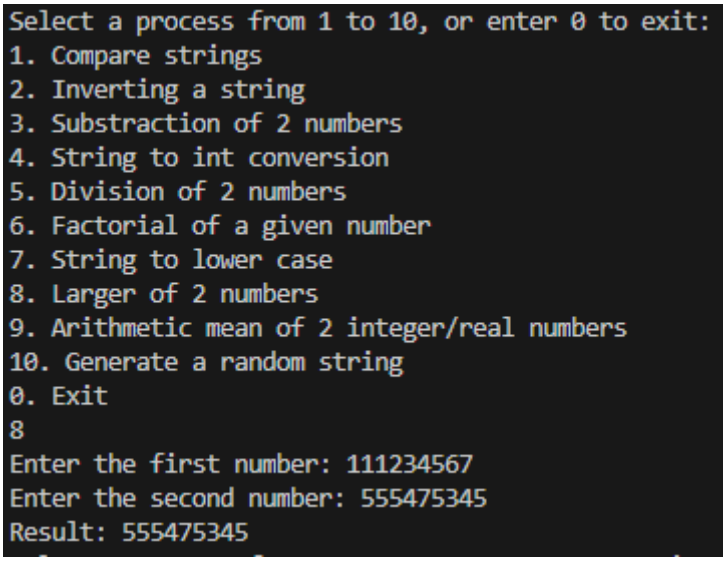


Figure 19: Larger of 2 numbers

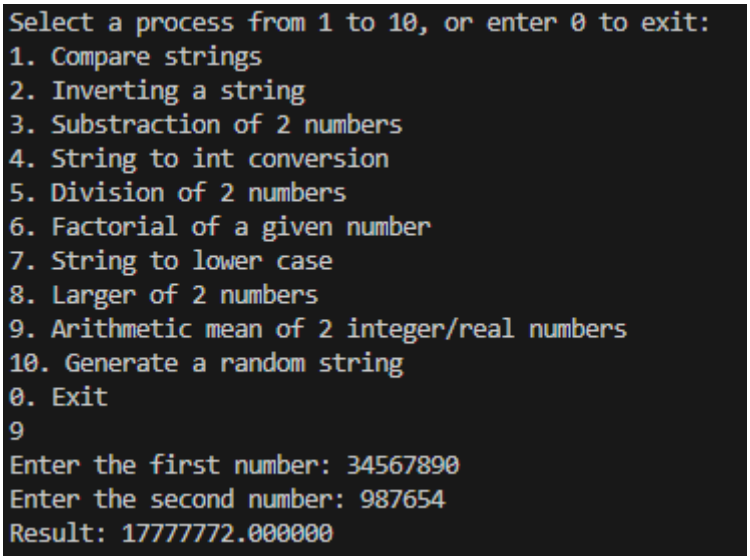


Figure 20: Arithmetic mean of 2 numbers

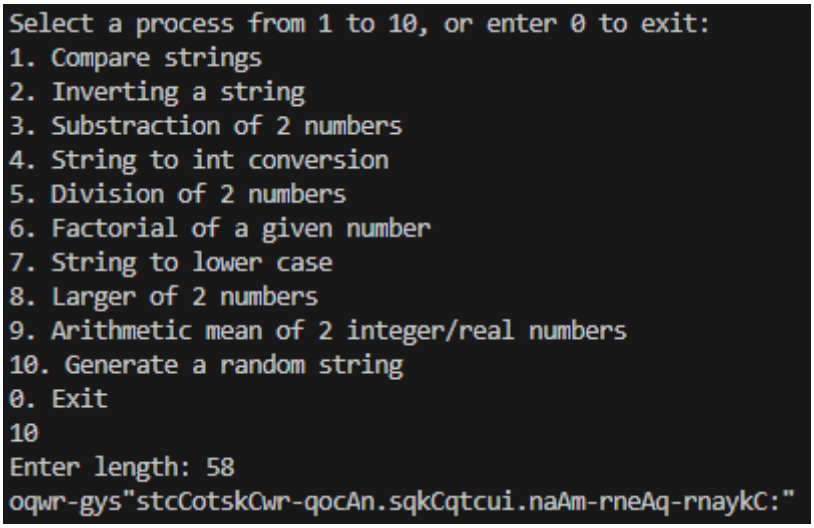


Figure 21: Random string

Conclusions:

In conclusion, these assembly language exercises provided valuable insight into low-level programming concepts and techniques. Through these exercises, we explored various fundamental operations such as string manipulation, integer conversion, arithmetic calculations, and user input handling.

Each exercise offered a unique challenge, ranging from basic string operations like comparison and inversion to more complex tasks such as factorial

calculation and menu-driven program implementation. By dissecting and understanding the intricacies of each exercise, we gained a deeper understanding of how assembly language operates at the hardware level.

Furthermore, these exercises underscored the importance of efficiency and optimization in low-level programming. We learned how to leverage CPU registers effectively, minimize memory usage, and streamline code execution to achieve optimal performance.

Overall, these exercises served as a stepping stone for delving deeper into the realm of assembly language programming. They provided a solid foundation upon which to build more advanced skills and tackle increasingly complex programming tasks in the future.

Refernces:

1. GitHub page: https://github.com/KaBoomKaBoom/Computer-Architecture-Labs.git