

Ministerul Educației 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

Chișinău - 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

```

1 ;string comparison in assembly language
2
3 section .bss
4     str1 resb 256    ; Allocate space for string 1
5     str2 resb 256    ; Allocate space for string 2
6
7 section .data
8     equal_msg db 'Strings are equal', 10
9     not_equal_msg db 'Strings are not equal', 10
10    prompt_msg db 'Enter string: ', 0
11
12 section .text
13     global ex1
14
15 ex1:
16     ; Prompt user to enter string 1
17     mov rax, 1
18     mov rdi, 1
19     mov rsi, prompt_msg
20     mov rdx, 16
21     syscall
22
23     ; Read string 1 from user
24     mov rax, 0        ; syscall number for sys_read
25     mov rdi, 0        ; file descriptor 0 (stdin)
26     mov rsi, str1     ; buffer to store string
27     mov rdx, 256      ; maximum number of bytes to read
28     syscall
29
30     ; Prompt user to enter string 2
31     mov rax, 1
32     mov rdi, 1
33     mov rsi, prompt_msg
34     mov rdx, 16
35     syscall
36
37     ; Read string 2 from user
38     mov rax, 0        ; syscall number for sys_read
39     mov rdi, 0        ; file descriptor 0 (stdin)
40     mov rsi, str2     ; buffer to store string
41     mov rdx, 256      ; maximum number of bytes to read
42     syscall
43
44     ; Compare strings
45     mov rsi, str1     ; Address of first string
46     mov rdi, str2     ; Address of second string
47     call compare_strings
48
49     ret
50
51
52
53 compare_strings:
54     compare_loop:
55         ; Load bytes from each string
56         mov al, byte [rsi]    ; Load byte from str1 into AL
57         mov bl, byte [rdi]    ; Load byte from str2 into BL
58
59         ; Compare the bytes
60         cmp al, bl
61         jne not_equal        ; If not equal, jump to not_equal label
62
63         ; Check if we've reached the end of the string (null terminator)
64         cmp al, 0
65         je strings_equal    ; If both strings are null-terminated, they are equal
66
67         ; Move to the next character in the strings
68         inc rsi
69         inc rdi
70         jmp compare_loop    ; Continue comparing
71
72 strings_equal:
73     mov rax, 1        ; syscall number for sys_write
74     mov rdi, 1        ; file descriptor 1 (stdout)
75     mov rsi, equal_msg ; pointer to the string
76     mov rdx, 18       ; length of the string
77     syscall           ; invoke the system call
78     ret
79
80 not_equal:
81     mov rax, 1        ; syscall number for sys_write
82     mov rdi, 1        ; file descriptor 1 (stdout)
83     mov rsi, not_equal_msg ; pointer to the string
84     mov rdx, 22       ; length of the string
85     syscall           ; invoke the system call
86     ret
87

```

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

```

1  ;invert string
2
3  section .data
4      msg times 64 db 0
5      max equ 64
6      prompt_msg db 'Enter string: ', 0
7      nlinea db 10, 0
8      lnlinea equ $ - nlinea
9
10
11 section .text
12     global ex2
13
14 ex2:
15     ; Prompt user to enter string 1
16     mov rax, 1
17     mov rdi, 1
18     mov rsi, prompt_msg
19     mov rdx, 16
20     syscall
21
22     ; Read
23     mov rdx, max
24     mov rsi, msg
25     mov rdi, 0
26     mov rax, 0
27     syscall
28
29     mov rcx, rax ; Copy the string for later
30     mov rdi, msg ; Set RDI and RSI to point at message
31     mov rsi, msg ;
32     add rdi, rax ; RDI should point at last character in message
33     dec rdi ;
34     shr rax, 1 ; Divide length by 2
35
36     .loop ; Begin loop:
37     mov bl, [rsi] ; Swap the characters using 8 bit registers
38     mov bh, [rdi] ;
39     mov [rsi], bh ;
40     mov [rdi], bl ;
41     inc rsi ; Increment rsi (which is a pointer)
42     dec rdi ; Decrement rdi (also a pointer)
43     dec rax ; Decrement our counter
44     jnz .loop ; If our counter isn't zero, keep looping
45
46     ; Write
47     .write
48     mov rdx, rcx ; nbytes
49     mov rsi, msg ; *msg
50     mov rdi, 1 ; stream
51     mov rax, 1 ; syscall
52     syscall
53
54     ; Print newline
55     mov rax, 1
56     mov rdi, 1
57     mov rsi, nlinea
58     mov rdx, lnlinea
59     syscall
60
61     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

```

1  section .data
2      ; Messages
3      msg1 db 'Enter the first number: ', 0
4      lmsg1 equ $ - msg1
5
6      msg2 db 'Enter the second number: ', 0
7      lmsg2 equ $ - msg2
8
9      nlinea db 10, 0
10     llinea equ $ - nlinea
11
12
13
14 section .bss
15     num1 resb 8
16     num2 resb 8
17     result resb 16
18     input1 resb 256 ; Buffer to store user input
19     input2 resb 256 ; Buffer to store user input
20
21 section .text
22 global ex3
23 extern printf
24 global atoi
25
26 ex3:
27     ; Print message 1
28     mov rax, 1
29     mov rdi, 1
30     mov rsi, msg1
31     mov rdx, lmsg1
32     syscall
33
34     ; Read user input
35     mov rax, 0 ; sys_read syscall number
36     mov rdi, 0 ; stdin file descriptor
37     mov rsi, input1 ; Address of the input buffer
38     mov rdx, 256 ; Maximum number of bytes to read
39     syscall
40
41     ; Print message 2
42     mov rax, 1
43     mov rdi, 1
44     mov rsi, msg2
45     mov rdx, lmsg2
46     syscall
47
48     ; Read user input
49     mov rax, 0 ; sys_read syscall number
50     mov rdi, 0 ; stdin file descriptor
51     mov rsi, input2 ; Address of the input buffer
52     mov rdx, 256 ; Maximum number of bytes to read
53     syscall
54
55     ; Call atoi function to convert input string to integer
56     mov rdi, input1 ; Pass the address of input string
57     call atoi ; Call the atoi function
58     mov [num1], rax ; Store the result in num1
59
60     ; Call atoi function to convert input string to integer
61     mov rdi, input2 ; Pass the address of input string
62     call atoi ; Call the atoi function
63     mov [num2], rax ; Store the result in num2
64
65     ; Subtract second number from the first
66     mov rax, [num1]
67     sub rax, [num2]
68     mov [result], rax
69
70     ; Return the result to main
71     mov rax, [result]
72     ret
73
74 atoi:
75     mov rax, 0 ; Set initial total to 0
76
77 convert:
78     movzx rsi, byte [rdi] ; Get the current character
79     test rsi, rsi ; Check for \0
80     je done
81
82     cmp rsi, 48 ; Anything less than 0 is invalid
83     jl done
84
85     cmp rsi, 57 ; Anything greater than 9 is invalid
86     jg done
87
88     sub rsi, 48 ; Convert from ASCII to decimal
89     imul rax, 10 ; Multiply total by 10
90     add rax, rsi ; Add current digit to total
91
92     inc rdi ; Get the address of the next character
93     jmp convert
94
95 done:
96     ret ; Return total
97

```

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


```

1 ;string to int conversion
2
3 section .bss
4     input_buffer resb 256     ; Buffer to store user input
5
6 section .data
7     input_prompt db "Enter the string to convert: ", 0 ; Prompt string
8     format_int db "Converted integer: %ld", 10 ; Format string for printing integer
9 section .text
10    global _start
11
12    ex4:
13
14    ; Display prompt message
15    mov rax, 1 ; syscall number for sys_write
16    mov rdi, 1 ; file descriptor 1 (stdout)
17    mov rsi, input_prompt ; pointer to the prompt message
18    mov rdx, 30 ; length of the prompt message
19    syscall ; invoke the system call
20
21    ; Read user input
22    mov rax, 0 ; sys_read syscall number
23    mov rdi, 0 ; stdin file descriptor
24    mov rsi, input_buffer ; Address of the input buffer
25    mov rdx, 256 ; Maximum number of bytes to read
26    syscall
27
28    ; Call atoi function to convert input string to integer
29    mov rdi, input_buffer ; Pass the address of input string
30    call atoi ; Call the atoi function
31
32
33    ret
34
35 section .text
36 extern printf
37
38 global atoi
39 global ex4
40
41 atoi:
42     mov rax, 0 ; Set initial total to 0
43
44 convert:
45     movzx rsi, byte [rdi] ; Get the current character
46     test rsi, rsi ; Check for \0
47     je done
48
49     cmp rsi, 48 ; Anything less than 0 is invalid
50     jl done
51
52     cmp rsi, 57 ; Anything greater than 9 is invalid
53     jg done
54
55     sub rsi, 48 ; Convert from ASCII to decimal
56     imul rax, 10 ; Multiply total by 10
57     add rax, rsi ; Add current digit to total
58
59     inc rdi ; Get the address of the next character
60     jmp convert
61
62 done:
63     ret ; Return total
64

```

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 `atoi` function to perform the conversion.

The `atoi` 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

```

1 section .data
2     ; Messages
3     msg1 db 'Enter the first number: ', 0
4     lmsg1 equ $ - msg1
5
6     msg2 db 'Enter the second number: ', 0
7     lmsg2 equ $ - msg2
8
9     msg3 db 'Result: ', 0
10    lmsg3 equ $ - msg3
11
12    nlinea db 10, 0
13    lnlinea equ $ - nlinea
14
15    format_int db "Result: %ld", 10    ; Format string for printing integer
16
17 section .bss
18     num1 resb 8
19     num2 resb 8
20     result resb 16
21     input1 resb 256    ; Buffer to store user input
22     input2 resb 256    ; Buffer to store user input
23
24 section .text
25     global ex5
26
27 ex5:
28     ; Print message 1
29     mov rax, 1
30     mov rdi, 1
31     mov rsi, msg1
32     mov rdx, lmsg1
33     syscall
34
35     ; Read user input for first number
36     mov rax, 0          ; sys_read syscall number
37     mov rdi, 0          ; stdin file descriptor
38     mov rsi, input1     ; Address of the input buffer
39     mov rdx, 256        ; Maximum number of bytes to read
40     syscall
41
42     ; Print message 2
43     mov rax, 1
44     mov rdi, 1
45     mov rsi, msg2
46     mov rdx, lmsg2
47     syscall
48
49     ; Read user input for second number
50     mov rax, 0          ; sys_read syscall number
51     mov rdi, 0          ; stdin file descriptor
52     mov rsi, input2     ; Address of the input buffer
53     mov rdx, 256        ; Maximum number of bytes to read
54     syscall
55
56     ; Call atoi2 function to convert input strings to integers
57     mov rdi, input1     ; Pass the address of the first input string
58     call atoi2          ; Call the atoi2 function
59     mov [num1], rax      ; Store the result in num1
60
61     mov rdi, input2     ; Pass the address of the second input string
62     call atoi2          ; Call the atoi2 function
63     mov [num2], rax      ; Store the result in num2
64
65     ; Divide first number by the second
66     mov rax, [num1]      ; Load the first number
67     mov rbx, [num2]      ; Load the second number
68     xor rdx, rdx         ; Clear RDX for the division operation
69     idiv rbx             ; Divide RAX:RDX by the second number (RBX)
70     mov [result], rax    ; Store the result in result
71
72     mov rax, [result]    ; Pass the integer to be printed
73
74     ret
75
76
77 section .text
78     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

```

1  ; factorial of a number
2
3  section .data
4      ; Messages
5      msg1 db 'Enter a number to calculate its factorial: ', 0
6      lmsg1 equ $ - msg1
7
8      msg2 db 'Result: ', 0
9      lmsg2 equ $ - msg2
10
11     nlinea db 10, 0
12     lnlinea equ $ - nlinea
13
14     format_int db "Result: %ld", 10 ; Format string for printing integer
15
16 section .bss
17     num resb 8
18     result resb 64
19     input resb 256 ; Buffer to store user input
20
21 section .text
22     global ex6
23
24 ex6:
25     ; Print message 1
26     mov rax, 1
27     mov rdi, 1
28     mov rsi, msg1
29     mov rdx, lmsg1
30     syscall
31
32     ; Read user input
33     mov rax, 0 ; sys_read syscall number
34     mov rdi, 0 ; stdin file descriptor
35     mov rsi, input ; Address of the input buffer
36     mov rdx, 256 ; Maximum number of bytes to read
37     syscall
38
39     ; Call atoi3 function to convert input string to integer
40     mov rdi, input ; Pass the address of input string
41     call atoi3 ; Call the atoi3 function
42     mov [num], rax ; Store the result in num
43
44     ; Calculate factorial
45     mov rax, 1 ; Initialize result to 1
46     mov rcx, [num] ; Load the input number
47
48     .factorial_loop:
49     imul rax, rcx ; Multiply result by current number
50     dec rcx ; Decrement current number
51     jnz .factorial_loop ; Repeat until current number is zero
52
53     mov rsi, rax ; Pass the result to be printed
54     ret
55
56 section .text
57     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

```

1 ; string to lowercase
2
3 section .data
4     input_string resb 255 ; Input string to be converted to lowercase
5     output_string resb 255 ; Reserve space for the output string
6     prompt_msg db 'Enter string: ', 0 ; Prompt message for user input
7     format_str db "Lower case: ", 0 ; Format string for printing string
8
9 section .text
10    global ex7
11    extern printf
12
13 ex7:
14
15    ; Prompt user to enter string
16    mov rax, 1
17    mov rdi, 1
18    mov rsi, prompt_msg
19    mov rdx, 14
20    syscall
21
22    ; Read input string
23    mov rax, 0 ; syscall number for sys_read
24    mov rdi, 0 ; file descriptor 0 (stdin)
25    mov rsi, input_string ; buffer to read into
26    mov rdx, 255 ; number of bytes to read
27    syscall
28
29    ; Call the function to convert the string to lowercase
30    mov rdi, input_string
31    mov rsi, output_string
32    call to_lowercase
33
34
35    mov rax, 1
36    mov rdi, 1
37    mov rsi, format_str
38    mov rdx, 12
39    syscall
40
41
42    mov rax, 1
43    mov rdi, 1
44    mov rsi, output_string
45    mov rdx, 255
46    syscall
47
48    ret
49
50 to_lowercase:
51    ; Loop through each character of the string until null terminator
52    .toLower_Loop:
53        mov al, byte [rdi] ; Load the current character
54        test al, al ; Check for null terminator
55        jz .toLower_done ; If null terminator, exit loop
56
57        ; Convert uppercase letters to lowercase
58        cmp al, 'A'
59        jb .toLower_continue
60        cmp al, 'Z'
61        ja .toLower_continue
62        add al, 32 ; 'A'-'a' = 32
63    .toLower_continue:
64        mov byte [rsi], al ; Store the lowercase character
65        inc rdi ; Move to the next character in input string
66        inc rsi ; Move to the next character in output string
67        jmp .toLower_Loop ; Repeat for the next character
68    .toLower_done:
69        ret
70

```

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): Determining the larger of 2 numbers


```

1 ;larger number between 2 numbers
2
3 section .data
4     ; Messages
5     msg1 db 'Enter the first number: ', 0
6     lmsg1 equ $ - msg1
7
8     msg2 db 'Enter the second number: ', 0
9     lmsg2 equ $ - msg2
10
11    msg3 db 'Result: ', 0
12    lmsg3 equ $ - msg3
13
14    nlinea db 10, 0
15    lnlinea equ $ - nlinea
16
17    format_int db "Larger number: %ld", 10    ; Format string for printing integer
18 section .bss
19    num1 resb 8
20    num2 resb 8
21    result resb 16
22    input1 resb 256    ; Buffer to store user input
23    input2 resb 256    ; Buffer to store user input
24
25 section .text
26     global ex8
27
28 ex8:
29     ; Print message 1
30     mov rax, 1
31     mov rdi, 1
32     mov rsi, msg1
33     mov rax, lmsg1
34     syscall
35
36     ; Read user input
37     mov rax, 0    ; sys_read syscall number
38     mov rdi, 0    ; stdin file descriptor
39     mov rsi, input1    ; Address of the input buffer
40     mov rax, 256    ; Maximum number of bytes to read
41     syscall
42
43     ; Print message 2
44     mov rax, 1
45     mov rdi, 1
46     mov rsi, msg2
47     mov rax, lmsg2
48     syscall
49
50     ; Read user input
51     mov rax, 0    ; sys_read syscall number
52     mov rdi, 0    ; stdin file descriptor
53     mov rsi, input2    ; Address of the input buffer
54     mov rax, 256    ; Maximum number of bytes to read
55     syscall
56
57     ; Call atoi4 function to convert input string to integer
58     mov rdi, input1    ; Pass the address of input string
59     call atoi4    ; Call the atoi4 function
60     mov [num1], rax    ; Store the result in num1
61
62     ; Call atoi4 function to convert input string to integer
63     mov rdi, input2    ; Pass the address of input string
64     call atoi4    ; Call the atoi4 function
65     mov [num2], rax    ; Store the result in num1
66
67     ; Compare the numbers
68     mov rax, [num1]
69     cmp rax, [num2]
70     jge num1_is_greater
71     mov rax, [num2]    ; If num2 is greater, load it into rax
72     jmp print_result
73
74 num1_is_greater:
75     mov rax, [num1]    ; If num1 is greater, load it into rax
76
77 print_result:
78     ; Print the larger number
79     mov [result], rax
80
81     ; Print the converted integer
82     mov rsi, [result]    ; Pass the integer to be printed
83     ; Exit
84     ret
85
86 section .text
87 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

```

1 ; arithmetic mean of two numbers (integers or real numbers)
2
3 section .data
4 ; Messages
5 msg1 db 'Enter the first number: ', 0
6 lmsg1 equ $ - msg1
7
8 msg2 db 'Enter the second number: ', 0
9 lmsg2 equ $ - msg2
10
11 msg3 db 'Result: ', 0
12 lmsg3 equ $ - msg3
13
14 nlinea db 10, 0
15 llinea equ $ - nlinea
16
17
18 two dq 2.0
19 section .bss
20 num1 resb 8
21 num2 resb 8
22 result resb 16
23 input1 resb 256 ; Buffer to store user input
24 input2 resb 256 ; Buffer to store user input
25
26 section .text
27 global ex9
28
29 ex9:
30 ; Print message 1
31 mov rax, 1
32 mov rdi, 1
33 mov rsi, msg1
34 mov rdx, lmsg1
35 syscall
36
37 ; Read user input
38 mov rax, 0 ; sys_read syscall number
39 mov rdi, 0 ; stdin file descriptor
40 mov rsi, input1 ; Address of the input buffer
41 mov rdx, 256 ; Maximum number of bytes to read
42 syscall
43
44 ; Print message 2
45 mov rax, 1
46 mov rdi, 1
47 mov rsi, msg2
48 mov rdx, lmsg2
49 syscall
50
51 ; Read user input
52 mov rax, 0 ; sys_read syscall number
53 mov rdi, 0 ; stdin file descriptor
54 mov rsi, input2 ; Address of the input buffer
55 mov rdx, 256 ; Maximum number of bytes to read
56 syscall
57
58 ; Call atof function to convert input string to floating point number
59 mov rdi, input1 ; Pass the address of input string
60 call atof ; Call the atof function
61 movsd [num1], xmm0 ; Store the result in num1
62
63 ; Call atof function to convert input string to floating point number
64 mov rdi, input2 ; Pass the address of input string
65 call atof ; Call the atof function
66 movsd [num2], xmm0 ; Store the result in num2
67
68 ; Calculate arithmetic mean
69 movsd xmm0, [num1]
70 addsd xmm0, [num2] ; Add both numbers
71 divsd xmm0, [two] ; Divide by 2 to find the mean
72
73
74
75
76 ret
77
78 section .text
79 extern printf
80
81 section .text
82 extern atof
83
84

```

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

```

1  section .data
2  charset db 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789,;+~=' , 0
3
4  ; Messages
5  msg1 db 'Enter length: ', 0
6  lmsg1 equ $ - msg1
7
8  str_len equ 64 ; Maximum length of the generated string
9  str: times str_len db 0
10 nlinea db 10, 0
11 llinea equ $ - nlinea
12
13 section .bss
14 len resb 4
15 input resb 256 ; Buffer to store user input
16
17 section .text
18 global ex10
19
20 ex10:
21 ; Print message 1
22 mov rax, 1
23 mov rdi, 1
24 mov rsi, msg1
25 mov rdx, lmsg1
26 syscall
27
28 ; Read user input
29 mov rax, 0 ; sys_read syscall number
30 mov rdi, 0 ; stdin file descriptor
31 mov rsi, input ; Address of the input buffer
32 mov rdx, 256 ; Maximum number of bytes to read
33 syscall
34
35 ; Convert input string to integer
36 mov rdi, input ; Address of the input buffer
37 call atoi ; Call the atoi function
38 mov [len], eax ; Store the result in len
39
40 ; Generate random string
41 mov ecx, eax ; Set loop counter to the length entered by the user
42 lea rdi, [str]
43
44 generate_string:
45 rdtsc ; Get timestamp for random seed
46 and eax, 94 ; Limit to size of charset
47 lea rsi, [charset]
48 movzx eax, byte [rsi+rax] ; Get a random character from charset
49 stosb ; Store character in str
50 loop generate_string ; Repeat until the desired length is reached
51
52 ; Print the string
53 mov eax, 1
54 mov edi, eax
55 lea rsi, [str]
56 mov edx, [len] ; Print only the specified length
57 syscall
58
59 ; Print newline
60 mov rax, 1
61 mov rdi, 1
62 mov rsi, nlinea
63 mov rdx, llinea
64 syscall
65
66 ret
67
68 atoi:
69 xor rax, rax ; Set initial total to 0
70
71 convert:
72 movzx rsi, byte [rdi] ; Get the current character
73 test rsi, rsi ; Check for \0
74 je done
75
76 cmp rsi, 48 ; Anything less than '0' is invalid
77 jl done
78
79 cmp rsi, 57 ; Anything greater than '9' is invalid
80 jg done
81
82 sub rsi, 48 ; Convert from ASCII to decimal
83 imul rax, 10 ; Multiply total by 10
84 add rax, rsi ; Add current digit to total
85
86 inc rdi ; Get the address of the next character
87 jmp convert
88
89 done:
90 ret ; Return total
91

```

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.

The ``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:

```

1 section .data
2 choice_prompt db "Select a process from 1 to 10, or enter 0 to exit:", 10
3 menu db "1. Compare strings", 10
4      db "2. Inverting a string", 10
5      db "3. Subtraction of 2 numbers", 10
6      db "4. String to int conversion", 10
7      db "5. Division of 2 numbers", 10
8      db "6. Factorial of a given number", 10
9      db "7. String to lower case", 10
10     db "8. Larger of 2 numbers", 10
11     db "9. Arithmetic mean of 2 integer/real numbers", 10
12     db "10. Generate a random string", 10
13     db "0. Exit", 10
14 invalid_choice db "Invalid choice, please try again.", 10
15 choice db 0, 0, 0, 0, 0, 0 ; 4 bytes for input and 1 byte for null-terminator
16 num db 0
17
18 format_int db "Result: %d", 10, 0 ; Format string for printing integer
19 format_float db "Result: %f", 10 ; Format string for printing floating point number
20
21 section .text
22 global main
23 extern exit, exit, exit, exit, exit, exit, exit, exit, exit, atoi, printf
24
25 main:
26     ; display menu
27     mov rax, 1 ; syscall number for sys_write
28     mov rdi, 1 ; file descriptor 1 (stdout)
29     mov rsi, choice_prompt
30     mov rdx, 334 ; message length
31     syscall
32
33     ; read choice from user
34     mov rax, 0 ; syscall number for sys_read
35     mov rdi, 0 ; file descriptor 0 (stdin)
36     mov rsi, choice
37     mov rdx, 4 ; Number of bytes to read
38     syscall
39
40     ; null-terminate the string
41     mov byte [rsi+3], 0
42
43     ; Call atoi function to convert input string to integer
44     mov rdi, choice ; Pass the address of input string
45     call atoi ; Call the atoi function
46     mov [num], rax ; Store the result in num
47
48     ; perform the chosen process
49     cmp word [num], 0
50     je exit_program
51     cmp word [num], 1
52     je call_exit
53     cmp word [num], 2
54     je call_exit
55     cmp word [num], 3
56     je call_exit
57     cmp word [num], 4
58     je call_exit
59     cmp word [num], 5
60     je call_exit
61     cmp word [num], 6
62     je call_exit
63     cmp word [num], 7
64     je call_exit
65     cmp word [num], 8
66     je call_exit
67     cmp word [num], 9
68     je call_exit
69     cmp word [num], 10
70     je call_exit
71
72     ; Invalid choice
73     mov rax, 1 ; syscall number for sys_write
74     mov rdi, 1 ; file descriptor 1 (stdout)
75     mov rsi, invalid_choice
76     mov rdx, 32 ; message length
77     syscall
78     jmp main
79
80 exit_program:
81     ; exit program
82     mov rax, 0 ; syscall number for sys_exit
83     mov rdi, 0 ; exit code 0
84     syscall
85
86 call_exit:
87     call exit
88     jmp main
89
90 call_exit:
91     call exit
92     jmp main
93
94 call_exit:
95     call exit
96
97     ; Print the converted integer
98     mov rdi, format_int ; Pass the format string for printing integer
99     mov rsi, num ; Pass the integer to be printed
100    xor rax, rax ; Clear RAX register for syscall number (sys_write)
101    call printf ; Call printf function
102    jmp main
103
104 call_exit:
105    ; Print the converted integer
106    mov rdi, format_int ; Pass the format string for printing integer
107    mov rsi, num ; Pass the integer to be printed
108    xor rax, rax ; Clear RAX register for syscall number (sys_write)
109    call printf ; Call printf function
110    jmp main
111
112 call_exit:
113    ; Pass the format string for printing integer
114    mov rdi, format_int ; Pass the integer to be printed
115    mov rsi, num ; Clear RAX register for syscall number (sys_write)
116    xor rax, rax ; Call printf function
117    call printf ; Call printf function
118    jmp main
119
120 call_exit:
121    ; Pass the format string for printing integer
122    mov rdi, format_int ; Pass the integer to be printed
123    mov rsi, num ; Clear RAX register for syscall number (sys_write)
124    xor rax, rax ; Call printf function
125    call printf ; Call printf function
126    jmp main
127
128 call_exit:
129    ; Pass the format string for printing integer
130    mov rdi, format_int ; Pass the integer to be printed
131    mov rsi, num ; Clear RAX register for syscall number (sys_write)
132    xor rax, rax ; Call printf function
133    call printf ; Call printf function
134    jmp main
135
136 call_exit:
137    ; Print the result
138    mov rdi, format_float
139    mov rsi, 1
140    xor rax, rax ; Clear RAX register for syscall number (sys_write)
141    call printf ; Call printf function
142    jmp main
143
144 call_exit:
145    ; Print the result
146    mov rdi, format_float
147    mov rsi, 1
148    xor rax, rax ; Clear RAX register for syscall number (sys_write)
149    call printf ; Call printf function
150    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:

```
Select a process from 1 to 10, or enter 0 to exit:
1. Compare strings
2. Inverting a string
3. Substraction of 2 numbers
4. String to int conversion
5. Division of 2 numbers
6. Factorial of a given number
7. String to lower case
8. Larger of 2 numbers
9. Arithmetic mean of 2 integer/real numbers
10. Generate a random string
0. Exit
1
Enter string: ASdx
Enter string: ASdx
Strings are equal
```

Figure 12: String comparing

```
Select a process from 1 to 10, or enter 0 to exit:
1. Compare strings
2. Inverting a string
3. Substraction of 2 numbers
4. String to int conversion
5. Division of 2 numbers
6. Factorial of a given number
7. String to lower case
8. Larger of 2 numbers
9. Arithmetic mean of 2 integer/real numbers
10. Generate a random string
0. Exit
2
Enter string:
andREI

IERdna
```

Figure 13: String inversion

```
Select a process from 1 to 10, or enter 0 to exit:
1. Compare strings
2. Inverting a string
3. Substraction of 2 numbers
4. String to int conversion
5. Division of 2 numbers
6. Factorial of a given number
7. String to lower case
8. Larger of 2 numbers
9. Arithmetic mean of 2 integer/real numbers
10. Generate a random string
0. Exit
3
Enter the first number: 5436
Enter the second number: 2223
Result: 3213
```

Figure 14: Subtraction of 2 numbers

```
Select a process from 1 to 10, or enter 0 to exit:
1. Compare strings
2. Inverting a string
3. Substraction of 2 numbers
4. String to int conversion
5. Division of 2 numbers
6. Factorial of a given number
7. String to lower case
8. Larger of 2 numbers
9. Arithmetic mean of 2 integer/real numbers
10. Generate a random string
0. Exit
4
Enter the string to convert: 555632
Result: 555632
```

Figure 15: String to int conversion

```
Select a process from 1 to 10, or enter 0 to exit:
1. Compare strings
2. Inverting a string
3. Substraction of 2 numbers
4. String to int conversion
5. Division of 2 numbers
6. Factorial of a given number
7. String to lower case
8. Larger of 2 numbers
9. Arithmetic mean of 2 integer/real numbers
10. Generate a random string
0. Exit
5
Enter the first number: 5567
Enter the second number: 142
Result: 39
```

Figure 16: Division of 2 numbers

```
Select a process from 1 to 10, or enter 0 to exit:
1. Compare strings
2. Inverting a string
3. Substraction of 2 numbers
4. String to int conversion
5. Division of 2 numbers
6. Factorial of a given number
7. String to lower case
8. Larger of 2 numbers
9. Arithmetic mean of 2 integer/real numbers
10. Generate a random string
0. Exit
6
Enter a number to calculate its factorial: 14
Result: 87178291200
```

Figure 17: Factorial of a number

```
Select a process from 1 to 10, or enter 0 to exit:
1. Compare strings
2. Inverting a string
3. Substraction of 2 numbers
4. String to int conversion
5. Division of 2 numbers
6. Factorial of a given number
7. String to lower case
8. Larger of 2 numbers
9. Arithmetic mean of 2 integer/real numbers
10. Generate a random string
0. Exit
7
Enter string: I LOVE ASSEMBLY (JOKE)
Lower case: i love assembly (joke)
```

Figure 18: String to lower case

```
Select a process from 1 to 10, or enter 0 to exit:
1. Compare strings
2. Inverting a string
3. Substraction of 2 numbers
4. String to int conversion
5. Division of 2 numbers
6. Factorial of a given number
7. String to lower case
8. Larger of 2 numbers
9. Arithmetic mean of 2 integer/real numbers
10. Generate a random string
0. Exit
8
Enter the first number: 111234567
Enter the second number: 555475345
Result: 555475345
```

Figure 19: Larger of 2 numbers

```
Select a process from 1 to 10, or enter 0 to exit:
1. Compare strings
2. Inverting a string
3. Substraction of 2 numbers
4. String to int conversion
5. Division of 2 numbers
6. Factorial of a given number
7. String to lower case
8. Larger of 2 numbers
9. Arithmetic mean of 2 integer/real numbers
10. Generate a random string
0. Exit
9
Enter the first number: 34567890
Enter the second number: 987654
Result: 17777772.000000
```

Figure 20: Arithmetic mean of 2 numbers

```
Select a process from 1 to 10, or enter 0 to exit:
1. Compare strings
2. Inverting a string
3. Substraction of 2 numbers
4. String to int conversion
5. Division of 2 numbers
6. Factorial of a given number
7. String to lower case
8. Larger of 2 numbers
9. Arithmetic mean of 2 integer/real numbers
10. Generate a random string
0. Exit
10
Enter length: 58
oqwr-gys"stcCotskCwr-qocAn.sqkCqtcui.naAm-rneAq-rnaykC:"
```

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.

References:

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