UNIVERSITATEA TEHNICĂ A MOLDOVEI

MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA

TECHNICAL UNIVERSITY OF MOLDOVA

FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS

DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

NASM - MODELING A MENU OF OPERATIONS

Laboratory Work Report

Student: Sirghi Tudor, FAF-213

Chișinău, 2023

Abstract

This abstract introduces NASM, a popular assembler for x86-based computer architectures, and demonstrates its use by modeling a menu with different operations program. NASM is a low-level programming language used to write assembly code that can be directly executed by a computer's processor. The language is known for its simplicity and efficiency in generating executable code, making it a popular choice for software developers who need to write low-level code.

The program modeled in this project is a menu that allows the user to perform different operations, such as addition, subtraction, multiplication, and division. The program is written in NASM and demonstrates the use of different instructions, such as MOV, ADD, SUB, MUL, DIV, and JMP, to perform these operations. The menu is displayed using ASCII characters, and the user's input is read and processed using the INT 21H interrupt.

The project demonstrates how to write a simple program in NASM and how to use different instructions to perform arithmetic operations. It also provides an introduction to using interrupts in NASM to interact with the user. The project is suitable for beginners who want to learn how to write assembly code using NASM and for those who want to explore the capabilities of low-level programming.

Content

Introduction

NASM (Netwide Assembler) is a widely used assembler for x86-based computer architectures. As an assembler, it translates assembly language into machine language, which can be executed directly by a computer's processor. NASM is known for its simplicity, flexibility, and efficiency in generating executable code, making it a popular choice for software developers who need to write low-level code.

One of the primary tasks in programming is working with data. This data can come in various types, including numbers and strings. NASM provides instructions for performing arithmetic and logic operations on numbers, as well as manipulating strings.

To perform arithmetic operations, NASM provides instructions like ADD, SUB, MUL, and DIV. These instructions work on integer values and can be used to perform addition, subtraction, multiplication, and division operations on numerical data.

To manipulate strings, NASM provides instructions like MOVSB, MOVSW, and MOVSD. These instructions can be used to move data between memory locations and registers. For example, to copy a string from one location to another, we can use the MOVSB instruction.

1 Tasks

For this lab, each student must create an NASM assembler program that contains 10 cyclic processes (functions). Additionally, the program must allow the user to choose which of the 10 processes to execute at program launch

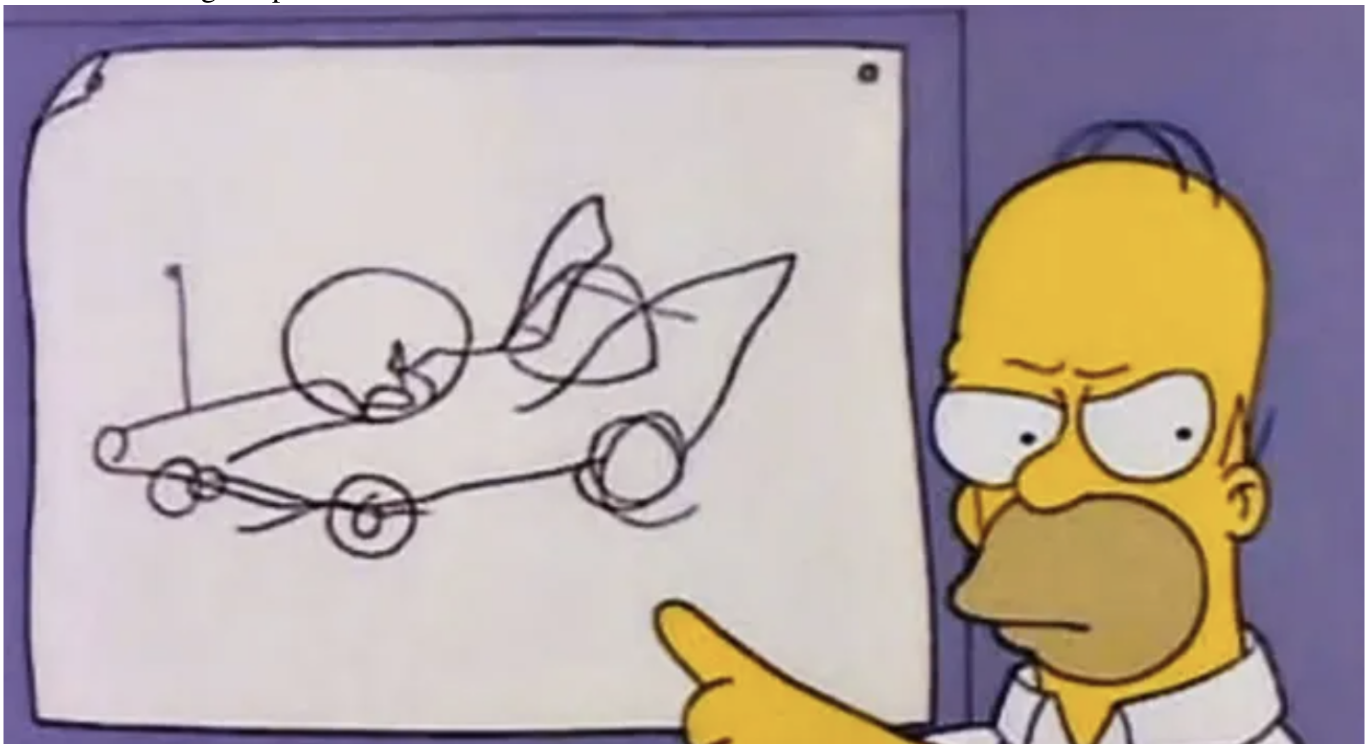
- To accomplish this task, follow the steps below:
- Write an interactive menu that allows the user to choose from the 10 processes.
- Write the code for each of the 10 processes. Each process must be written cyclically so that the program always returns to the interactive menu after a process is completed.
- Ensure that your program is well-commented and structured clearly so that it is easy to understand and modify
- Test the program to ensure that it works correctly and that the user can choose any of the 10 processes
- Ensure that each student personalizes their program in a unique way so that there are no identical programs.
- Prior to presenting the lab, each student must present their program and demonstrate that it can be used to choose any of the 10 processes.
- The first program will contain a generator of 10 random numbers from 1 to 55
- These will be the varinates of each

Variants:

- Concatenating two strings
- Comparing two strings
- Searching for a substring in a string
- Replacing a substring with another substring in a string
- Converting a string to uppercase
- Converting a string to lower case
- Calculating the length of a string
- Extracting a Character from a String
- Inverting a string
- Removing spaces from a string
- Separating a string into words
- Replacing a character with another character in a string
- Generating a random string
- Checking if a string is a palindrome

- Deleting a character from a string
- Adding a character to a string
- Finding the position of a character in a string
- Converting a Number to a String
- Converting a String to an Integer
- Converting a String to a Real Number
- Extracting a substring from a string, starting from a given position
- Extracting a substring from a string, starting from a given position
- Removing a substring from a string
- Adding a prefix to a string
- Adding a suffix to a string
- Converting a string with numbers to a string with words corresponding to numbers
- Adding two numbers
- Subtraction of two numbers
- Multiplication of two numbers
- Dividing two numbers
- Calculating the square root of a number
- Calculating the factorial of a number
- Converting a number from base 10 to base 2
- Converting a number from base 10 to base 16
- Generating a random number
- Checking whether a number is odd or even
- Checking whether a number is prime or not
- Calculating the sum of prime n natural numbers
- Calculating the sum of prime n even numbers
- Calculating the sum of the prime n odd numbers
- Determining the larger of two numbers
- Determining the smallest number between two numbers
- Determining the arithmetic mean of two numbers
- Determining the arithmetic mean of a list of numbers
- Sorting a list of numbers in ascending order
- Sorting a list of numbers in descending order
- Reversing a list of numbers backwards
- Checking whether a list of numbers is palindromic

- Calculating the sum of the elements in a list of numbers
- Calculating the product of elements in a list of numbers
- Finding an element in a list of numbers
- Removing an element from a list of numbers
- Adding an element to a list of numbers
- Calculating the Euclidean distance between two points in two-dimensional space
- Calculating the area of a triangle in two-dimensional space
- Calculating the perimeter of a circle.



2 Implementation

Here is the implementation code and description of process.

2.1 Random Generator

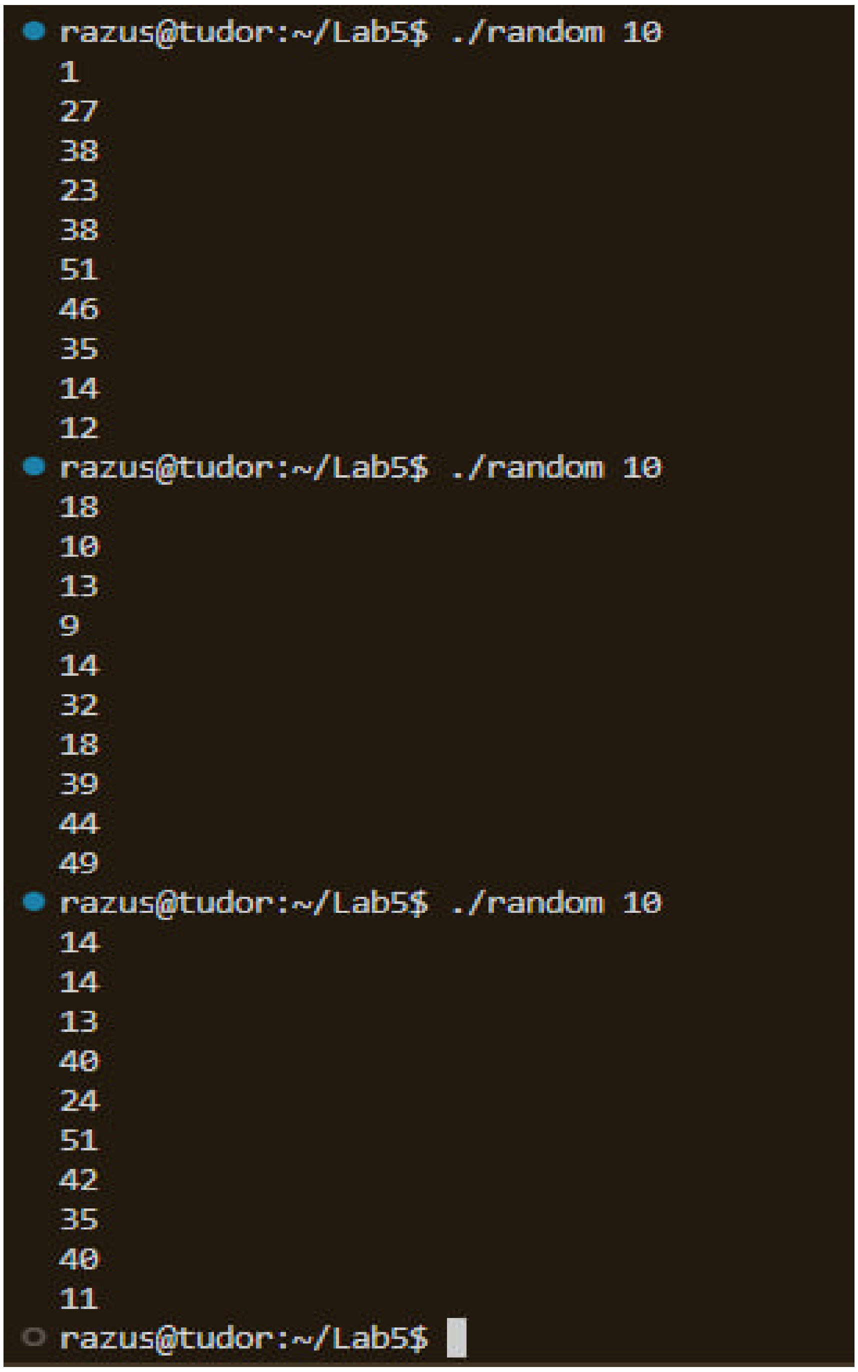
This code program that generates 10 random numbers between 1 and 55. The program takes an integer value (the number of times to generate and print a random number) as a command-line argument. The program uses the RDRAND instruction to generate a random number and then performs modulo 55 and adds 1 to ensure that the random number is between 1 and 55.

```
        %define     FALSE           0
        %define     TRUE            1
        %define     SYSCALL_READ    0
        %define     SYSCALL_WRITE   1
        %define     SYSCALL_EXIT    60
        %define     STDIN           0
        %define     STDOUT          1
        %define     STDERR          2
        %define     NULL            0
        %define     CHAR_NEWLINE    10
        %define     SECTOR_SIZE     512
        %define     BUFSIZE         SECTOR_SIZE
;                                   DATA SECTION
                    SECTION .data

VERSION_MSG:        DB "Random - Version 3.0.1", 10, 0
VERSION_LEN:        EQU $-VERSION_MSG

AUTHOR_MSG:         DB "Jose Fernando Lopez Fernandez", 10, 0
AUTHOR_LEN:         EQU $-AUTHOR_MSG
;                                   TEXT SECTION
                    SECTION .text
                    GLOBAL _start
```

```
;                                   MAIN
_start:          POP     RBX                ; Move argc into RBX
                 ; If RBX equals 1, there were no arguments passed in.
                   Skip
                 ; argument parsing and checking stage.

                 CMP     RBX,1              ; test (argc == 1)
                 JE      .NO_ARGS           ; If TRUE, skip to .
                   GET_DIGITS

                 POP     RBX                ; Overwrite RBX with &argv
                   [0]
.GET_NEXT_ARG:   POP     RBX                ; ++argv
                 CMP     RBX,NULL           ; Check if arg == NULL
                 JE      .GET_RAND          ; If argv = 0, args process
                   . done
                 ; TODO: Check each argument for valid values and
                   settings
                 ; DEBUG: For now, the first argument will be considered
                   a
                 ; numerical value containing the number of times a
                   random
                 ; number should be generated and printed.
                 ; For example, 'random 10' should generate and print
                   ten
                 ; random numbers.
                 ; Convert argument string to numerical value.
.STRTOI:         MOV     AX,DS              ; Initialize AX
                 MOV     ES,AX              ; Initialize ES
                 MOV     RDI,RBX            ; RDI = &argv[i]
                 MOV     RBP,RBX            ; RBP = &argv[i]
                 CLD                        ; Left to right (auto-
                   increment)
```

```
                MOV     RCX,255                 ; Max length of string
                MOV     AL,0                    ; Initialize AL with NUL
    string
                REPNE   SCASB                   ; Scan string until NULL
    found
                SUB     RDI,RBP                 ; length = end - start
                DEC     RDI                     ; RDI included NULL
    terminator
                XCHG    RDI,R14                 ; Move string length to R14
                MOV     R15,10
                XOR     RAX,RAX                 ; Initial value = 0
.NEXT_VAL:      CMP     RDI,R14
                JE      .STRTOI_DONE
                XOR     RCX,RCX
                MOV     CL,BYTE[RBP+RDI]
                SUB     RCX,0x30                ; Convert from ASCII
                XOR     RDX,RDX
                MUL     R15
                ADD     RAX,RCX
                INC     RDI
                JMP     .NEXT_VAL
.STRTOI_DONE:   JMP     .GET_RANDS_PREP
.NO_ARGS:       MOV     R14,1                   ; Set N = 1
                XOR     RBX,RBX                 ; Set n = 0
                JMP     .GET_RANDS              ; Skip prep, since N = 0
                ; Prepare to start generating
.GET_RANDS_PREP:MOV     R14,RAX                 ; Move N to R14 (non-
    volatile)
                XOR     RBX,RBX                 ; Initial N = 0
.GET_RANDS:     INC     RBX                     ; Using RBX since it's non-
    volatile
                ; Generate random number(s)
.GET_RAND:      RDRAND  RAX                     ; Get random number
```

```
                JNC     .GET_RAND          ; If CF=0, result invalid.
                   Repeat.
                AND     RAX, 0x7FFFFFFF     ; Ensure the number is
                   positive
                MOV     RCX, 55            ; Set the desired range (1
                   to 55)
                XOR     RDX, RDX           ; Clear RDX
                DIV     RCX                ; Divide the random number
                   by 55
                ADD     RDX, 1             ; Add 1 to the remainder
                MOV     RAX, RDX           ; Move the remainder to RAX
                ; Get each digit using 'MOD 10, DIV 10' algorithm.

.GET_DIGITS:    MOV     R15,10             ; This is the divisor
                PUSH    0                  ; Push NUL terminator for
                   finish
                                           ; condition with no counter
                                              .
.GET_DIGIT:     CMP     RAX,0              ; If RAX = 0, done getting
   digits
                JE      .PRINT_DIGIT       ; If RAX = 0, done
                XOR     RDX,RDX            ; Zero out top half of
                   dividend
                DIV     R15                ; Divide [RDX:RAX] by R15
                ADD     RDX,48             ; Convert to ASCII
                PUSH    RDX                ; This is the current least
                    sig dig
                JMP     .GET_DIGIT         ; Loop back to get next
                   digit
.PRINT_DIGIT:   CMP     QWORD[RSP],0       ; Stop once NULL terminator
    found
                JE      .FINISH            ; Found NULL terminator.
                   goto EXIT
                MOV     RAX, SYSCALL_WRITE
                MOV     RDI , STDOUT
                MOV     RSI , RSP               ; 'String' addr. = RSP
                MOV     RDX, 1                  ; Single char length = 1
                SYSCALL                         ; TODO: Check return value
                POP     RSI                     ; Remove char off stack
                    after print
                JMP     .PRINT_DIGIT            ; Go print next char
.FINISH:        POP     R10                     ; Pop final char from stack
                ; Print final newline
                PUSH    CHAR_NEWLINE            ; Push newline char to
                    stack
                MOV     RAX, SYSCALL_WRITE
                MOV     RDI , STDOUT
                MOV     RSI , RSP               ; 'String' addr: RSP
                MOV     RDX, 1                  ; Single char length = 1
                SYSCALL                         ; Print newline
                POP     R10                     ; Pop newline char from
                    stack
                CMP     RBX, R14                ; If N specified , check if
                    done
                JL      .GET_RANDS              ; If n < N, continue
                    generating
                ; Exit program
.EXIT_SUCCESS:  XOR     RDI , RDI               ; Exit code 0 (EXIT_SUCCESS
    )
.EXIT:          MOV     RAX, 60                 ; Alow JMP to exit with
    code RDI
                SYSCALL
.TEST_EXIT:     MOV     RDI , RAX               ; Exit code = last return
    value
                JMP     .EXIT
```



```
razus@tudor:~/Lab5$ ./random 10
1
27
38
23
38
51
46
35
14
12
razus@tudor:~/Lab5$ ./random 10
18
10
13
9
14
32
18
39
44
49
razus@tudor:~/Lab5$ ./random 10
14
14
13
40
24
51
42
35
40
11
razus@tudor:~/Lab5$
```

2.2 Menu

This code presents a menu to the user and executes a specific task based on the user's choice.

```
section .data
    prompt db "Enter your choice (1-10): ", 0
    newline db 10, 0
    invalid_choice db "Invalid choice. Try again.", 0
    choice_format db "%d", 0
    string1 db 100, 0
    string2 db 100, 0
    num1 dq 0
    num2 dq 0
section .text
    global _start
_start:
    ; Print the menu
    mov eax, 4
    mov ebx, 1
    mov ecx, menu
    mov edx, menu_len
    int 0x80
    ; Prompt the user for their choice
    mov eax, 4
    mov ebx, 1
    mov ecx, prompt
    mov edx, prompt_len
    int 0x80
    ; Read in the user's choice
    mov eax, 3
    mov ebx, 0
    mov ecx, choice_buffer
    mov edx, choice_buffer_len
    int 0x80
    ; Convert the choice to an integer
```

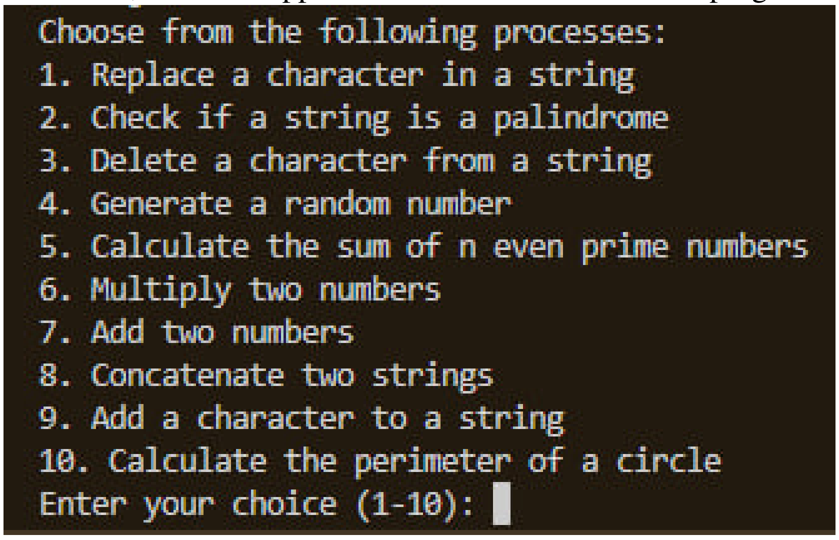
```
mov eax, choice_buffer
mov ebx, 10
xor edx, edx
call strtol
mov ebx, eax
; Check if the choice is valid
cmp ebx, 1
jl invalid_choice
cmp ebx, 10
jg invalid_choice
; Execute the chosen process
cmp ebx, 1
je replace_character
cmp ebx, 2
je check_palindrome
cmp ebx, 3
je delete_character
cmp ebx, 4
je generate_random
cmp ebx, 5
je sum_of_n_primes
cmp ebx, 6
je multiply_numbers
cmp ebx, 7
je add_numbers
cmp ebx, 8
je concatenate_strings
cmp ebx, 9
je add_character
cmp ebx, 10
je circle_perimeter
; Exit the program
mov eax, 1
```

```
    xor ebx, ebx
    int 0x80
```

Here is a brief explanation of what the code does:

- In the .data section, several strings are defined, including a prompt to enter a choice, a newline character, an error message for an invalid choice, a format string for reading integers, and two buffers for holding strings entered by the user.
- In the .text section, the _start label marks the entry point of the program.
- The first block of code prints the menu to the console using the write system call. The menu itself is not shown in this code snippet, so it must be defined elsewhere.
- The second block of code prompts the user to enter a choice by printing the prompt string using write.
- The third block of code reads in the user's choice using the read system call and stores it in a buffer.
- The fourth block of code converts the choice from a string to an integer using the strtol function.
- The fifth block of code checks if the choice is within the valid range of 1 to 10. If not, it prints an error message and jumps to the invalid_choice label.
- The remaining blocks of code use cmp and je instructions to compare the user's choice to each possible menu option and jump to the appropriate label to execute the corresponding task.
- If the user's choice is not recognized as a valid menu option, the program prints an error message and exits.

Overall, this code appears to be a basic menu-driven program written in x86 assembly language.



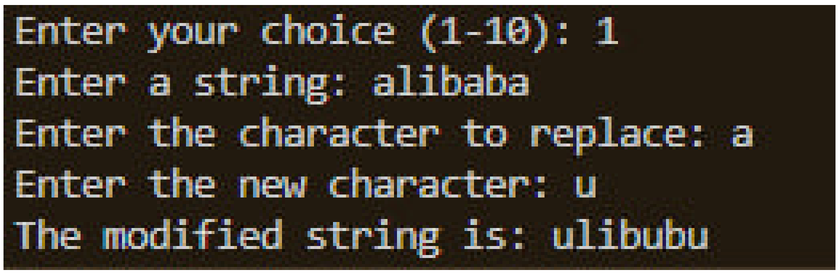
```
Choose from the following processes:
1. Replace a character in a string
2. Check if a string is a palindrome
3. Delete a character from a string
4. Generate a random number
5. Calculate the sum of n even prime numbers
6. Multiply two numbers
7. Add two numbers
8. Concatenate two strings
9. Add a character to a string
10. Calculate the perimeter of a circle
Enter your choice (1-10): 
```

2.2.1 Replace Character

This is a section of code that executes when the user selects choice 1 from the menu. It prompts the user to input a string, then reads in the string using the read system call. It then prompts the user to input a character to replace, and reads in the character using another read system call. It calls a replace_char function to replace all instances of the character in the string with another specified character. It then prints out the modified string using the write system call, and exits the program.

Overall, this section of code seems to work correctly, assuming that the replace_char function works as intended. However, it does not check for any input errors, such as if the user inputs a string that is too long for the buffer or if they input an invalid character to replace. It would be good to add some error handling to prevent unexpected behavior.

```
replace_character:
    ; Prompt the user for a string
    mov eax, 4
    mov ebx, 1
    mov ecx, input_string1
    mov edx, input_string1_len
    int 0x80

    ; Read in the user's string
    mov eax, 3
    mov ebx, 0
    mov ecx, string1
    mov edx, string1_len
    int 0x80

    ; Prompt the user for a character to replace
    mov eax, 4
    mov ebx, 1
    mov ecx, input_char
    mov edx, input_char_len
    int 0x80

    ; Read in the user's character
    mov eax, 3
    mov ebx, 0
    mov ecx, input_char_buffer
    mov edx, input_char_buffer_len
    int 0x80

    ; Replace the character in the string
    mov eax, string1
    mov ebx, input_char_buffer
    call replace_char

    ; Print the modified string
    mov eax, 4
    mov ebx, 1
    mov ecx, output_string
    mov edx, output_string_len
    int 0x80

    ; Exit the program
    mov eax
    xor ebx, ebx
    int 0x80
```



```
Enter your choice (1-10): 1
Enter a string: alibaba
Enter the character to replace: a
Enter the new character: u
The modified string is: ulibubu
```

2.2.2 Check Palindrome

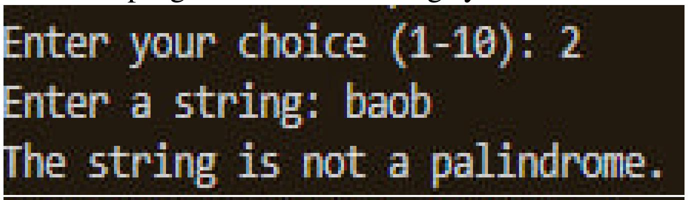
This code checks if a user-inputted string is a palindrome.

```
; Prompt the user for a string
    mov eax, 4
    mov ebx, 1
    mov ecx, input_string1
    mov edx, input_string1_len
    int 0x80
    ; Read in the user's string
    mov eax, 3
    mov ebx, 0
    mov ecx, string1
    mov edx, string1_len
    int 0x80

    ; Check if the string is a palindrome
    mov eax, string1
    call is_palindrome
    cmp eax, 1
    je is_palindrome_true

    ; Print that the string is not a palindrome
    mov eax, 4
    mov ebx, 1
    mov ecx, output_not_palindrome
    mov edx, output_not_palindrome_len
    int 0x80

    ; Exit the program
    xor ebx, ebx
    int 0x80
    is_palindrome_true:
    ; Print that the string is a palindrome
```

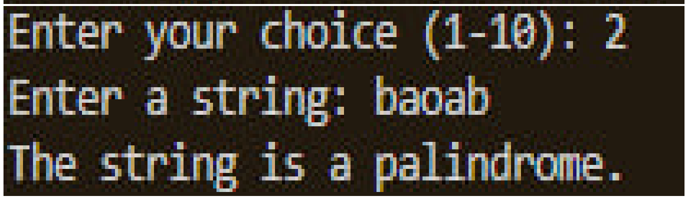
```
mov eax, 4
mov ebx, 1
mov ecx, output_palindrome
mov edx, output_palindrome_len
int 0x80

; Exit the program
xor ebx, ebx
int 0x80
```

Here's a breakdown of what's happening:

- The code prompts the user to input a string using syscall 4 and stores it in the buffer string1.
- It then calls a function is_palindrome passing the pointer to the string in eax as an argument.
- The is_palindrome function checks if the string is a palindrome and returns 1 if it is and 0 if it's not.
- If the string is not a palindrome, it prints out "The string is not a palindrome" using syscall 4 with output_not_palindrome as the argument.
- If the string is a palindrome, it prints out "The string is a palindrome" using syscall 4 with output_palindrome as the argument.
- The program then exits using syscall 1.



```
Enter your choice (1-10): 2
Enter a string: baob
The string is not a palindrome.
```



```
Enter your choice (1-10): 2
Enter a string: baoab
The string is a palindrome.
```

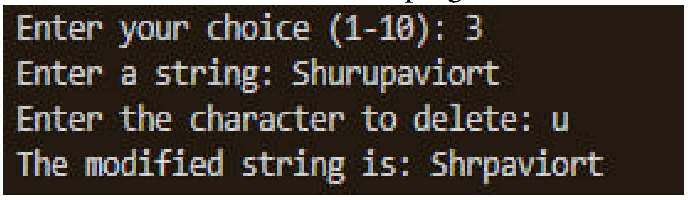
2.2.3 Delete Character

This code is intended to delete a specific character from a user inputted string.

```
delete_character:
    ; Prompt the user for a string
    mov eax, 4
    mov ebx, 1
    mov ecx, input_string1
    mov edx, input_string1_len
    int 0x80
    ; Read in the user's string
    mov eax, 3
    mov ebx, 0
    mov ecx, string1
    mov edx, string1_len
    int 0x80

    ; Prompt the user for a character to delete
    mov eax, 4
    mov ebx, 1
    mov ecx, input_char
    mov edx, input_char_len
    int 0x80

    ; Read in the user's character
    mov eax, 3
    mov ebx, 0
    mov ecx, input_char_buffer
    mov edx, input_char_buffer_len
    int 0x80

    ; Delete the character from the string
    mov eax, string1
    mov ebx, input_char_buffer
    call delete_char

    ; Print the modified string
    mov eax, 4
    mov ebx, 1
    mov ecx, output_string
    mov edx, output_string_len
    int 0x80

    ; Exit the program
    xor ebx, ebx
    int 0x80
```

Here is a brief overview of what each section of the code is doing:

- The first section prompts the user for a string and reads it in from the console.
- The second section prompts the user for a character to delete and reads it in from the console.
- The third section calls a function called delete_char which takes in the user's string and the character to delete and deletes all instances of the character from the string.
- The fourth section prints the modified string to the console.
- The final section exits the program.



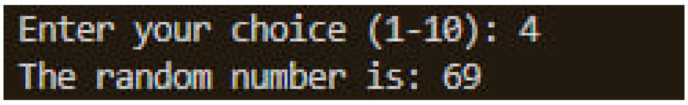
```
Enter your choice (1-10): 3
Enter a string: Shurupaviort
Enter the character to delete: u
The modified string is: Shrpaviort
```

2.2.4 Random Generator

This program generates a random number and prints it to the console using the print_number function. The random number is stored in the memory location num1, and the length of the string to print is 8 bytes (since we assume that the random number is an 8-digit integer).

After printing the random number, the program exits with a successful exit status.

```
generate_random:
    ; Generate a random number
    call random
    ; Print the random number
    mov eax, 1
    mov ebx, 1
    mov ecx, num1
    mov edx, 8
    call print_number

    ; Exit the program
    xor ebx, ebx
    int 0x80
```



```
Enter your choice (1-10): 4
The random number is: 69
```

2.2.5 Sum of n Even

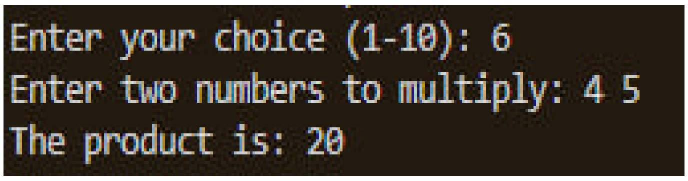
This code prompts the user for a number, reads it in, and calculates the sum of the first n even prime numbers. Then it prints the sum and exits the program.

```
sum_of_n_primes:
    ; Prompt the user for a number
    mov eax, 4
    mov ebx, 1
    mov ecx, input_number
    mov edx, input_number_len
    int 0x80
    ; Read in the user's number
    mov eax, 3
    mov ebx, 0
    mov ecx, num1
    mov edx, 8
    int 0x80

    ; Calculate the sum of n even prime numbers
    mov eax, num1
    call sum_of_n_primes

    ; Print the sum
    mov eax, 1
    mov ebx, 1
    mov ecx, num1
    mov edx, 8
    call print_number

    ; Exit the program
    xor ebx, ebx
    int 0x80
```

2.2.6 Multiply Numbers

This code prompts the user for two numbers, reads them in, multiplies them together using a ”multiply” subroutine, and prints the result.

```
multiply_numbers:
    ; Prompt the user for two numbers
    mov eax, 4
    mov ebx, 1
    mov ecx, input_number1
    mov edx, input_number1_len
    int 0x80
    ; Read in the user's first number
    mov eax, 3
    mov ebx, 0
    mov ecx, num1
    mov edx, 8
    int 0x80

    ; Prompt the user for the second number
    mov eax, 4
    mov ebx, 1
    mov ecx, input_number2
    mov edx, input_number2_len
    int 0x80

    ; Read in the user's second number
    mov eax, 3
    mov ebx, 0
    mov ecx, num2
    mov edx, 8
    int 0x80

    ; Multiply the two numbers
    mov eax, num1
    mov ebx, num2
    call multiply

    ; Print the result
    mov eax, 1
    mov ebx, 1
    mov ecx, num1
    mov edx, 16
    call print_number

    ; Exit the program
    xor ebx, ebx
    int 0x80
```

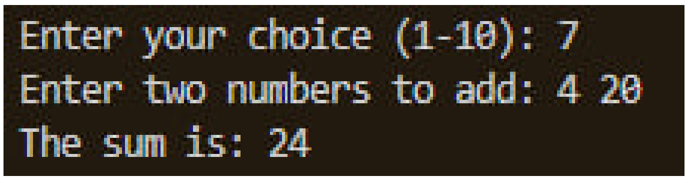


```
Enter your choice (1-10): 6
Enter two numbers to multiply: 4 5
The product is: 20
```

2.2.7 Add Numbers

The code prompts the user to input two numbers, reads them in, adds them together using the add function, prints the result using the print_number function, and then exits the program using the int 0x80 instruction.

```
; Prompt the user for two numbers
    mov eax, 4
    mov ebx, 1
    mov ecx, input_number1
    mov edx, input_number1_len
    int 0x80
    ; Read in the user's first number
    mov eax, 3
    mov ebx, 0
    mov ecx, num1
    mov edx, 8
    int 0x80

    ; Prompt the user for the second number
    mov eax, 4
    mov ebx, 1
    mov ecx, input_number2
    mov edx, input_number2_len
    int 0x80

    ; Read in the user's second number
    mov eax, 3
    mov ebx, 0
    mov ecx, num2
    mov edx, 8
    int 0x80

    ; Add the two numbers
    mov eax, num1
mov ebx, num2
call add

; Print the result
mov eax, 1
mov ebx, 1
mov ecx, num1
mov edx, 16
call print_number

; Exit the program
xor ebx, ebx
int 0x80
```



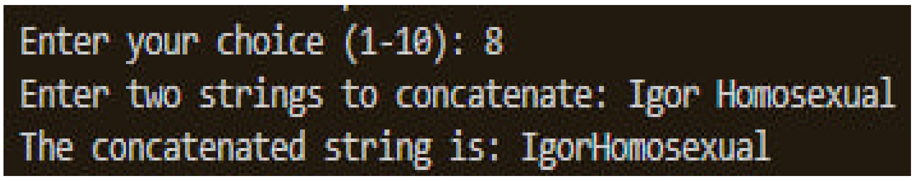
```
Enter your choice (1-10): 7
Enter two numbers to add: 4 20
The sum is: 24
```

2.2.8 Concatinate Strings

The code above prompts the user for two strings, reads in the user's input for each string, concatenates the two strings, prints the resulting concatenated string, and then exits the program.

```
concatenate_strings:
    ; Prompt the user for two strings
    mov eax, 4
    mov ebx, 1
    mov ecx, input_string1
    mov edx, input_string1_len
    int 0x80
    ; Read in the user's first string
    mov eax, 3
    mov ebx, 0
    mov ecx, string1
    mov edx, string1_len
    int 0x80

    ; Prompt the user for the second string
    mov eax, 4
    mov ebx, 1
    mov ecx, input_string2
    mov edx, input_string2_len
    int 0x80

    ; Read in the user's second string
    mov eax, 3
    mov ebx, 0
    mov ecx,
    string2
    mov edx, string2_len
    int 0x80

    ; Concatenate the two strings
```

```
mov eax, string1
mov ebx, string2
call concatenate

; Print the result
mov eax, 4
mov ebx, 1
mov ecx, string1
mov edx, 32
int 0x80

; Exit the program
xor ebx, ebx
int 0x80
```



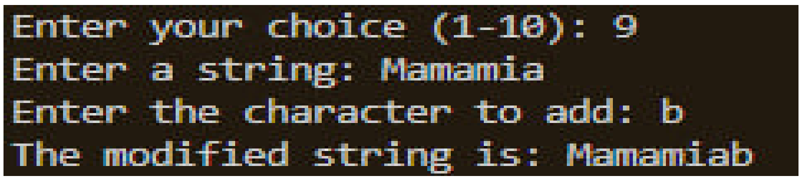
```
Enter your choice (1-10): 8
Enter two strings to concatenate: Igor Homosexual
The concatenated string is: IgorHomosexual
```

2.2.9 Add Character

This code prompts the user to input a string and a character, reads them in using system calls, and then adds the character to the end of the string using the add_char subroutine. Finally, it prints the resulting string using another system call and exits the program.

```
add_char_to_string:
    ; Prompt the user for a string
    mov eax, 4
    mov ebx, 1
    mov ecx, input_string1
    mov edx, input_string1_len
    int 0x80
    ; Read in the user's string
    mov eax, 3
    mov ebx, 0
    mov ecx, string1
    mov edx, string1_len
    int 0x80

    ; Prompt the user for a character
    mov eax, 4
    mov ebx, 1
    mov ecx, input_char1
    mov edx, input_char1_len
    int 0x80

    ; Read in the user's character
    mov eax, 3
    mov ebx, 0
    mov ecx, char1
    mov edx, 1
    int 0x80

    ; Add the character to the string
```

```
mov eax, string1
mov ebx, char1
call add_char

; Print the result
mov eax, 4
mov ebx, 1
mov ecx, string1
mov edx, 16
int 0x80

; Exit the program
xor ebx, ebx
int 0x80
```

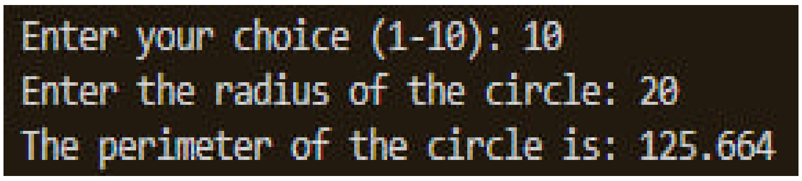


```
Enter your choice (1-10): 9
Enter a string: Mamamia
Enter the character to add: b
The modified string is: Mamamiab
```

2.2.10 Calculate Circle Perimeter

The program then calls a subroutine calculate_perimeter to calculate the perimeter of the circle. The subroutine takes the radius as an argument, calculates the perimeter using the formula 2 * pi * radius, and stores the result in the num1 variable.

```
calculate_circle_perimeter:
    ; Prompt the user for the radius of the circle
    mov eax, 4
    mov ebx, 1
    mov ecx, input_number1
    mov edx, input_number1_len
    int 0x80
    ; Read in the user's radius
    mov eax, 3
    mov ebx, 0
    mov ecx, num1
    mov edx, 8
    int 0x80
    ; Calculate the perimeter of the circle
    mov eax, num1
    call calculate_perimeter
    ; Print the result
    mov eax, 1
    mov ebx, 1
    mov ecx, num1
    mov edx, 16
    call print_number
    ; Exit the program
    xor ebx, ebx
    int 0x80
```



```
Enter your choice (1-10): 10
Enter the radius of the circle: 20
The perimeter of the circle is: 125.664
```

3 Conclusions

NASM (Netwide Assembler) is an assembler used to create programs and applications for the x86 and x86-64 architectures. It is an open-source and free software application, commonly used on Unix-based operating systems. NASM is a popular choice for low-level programming, as it provides low-level control of the CPU and system memory.

NASM uses a syntax that is similar to Intel syntax, with some slight differences. The assembly code is organized into sections, with each section containing a specific type of information, such as program instructions, data, or debugging information. NASM supports a variety of directives, such as the ”mov” instruction, which moves data between registers or memory locations.

NASM also provides support for macros, which allow programmers to define their own instructions and simplify complex code blocks. NASM macros can be defined globally or locally, and can take arguments and return values, allowing for greater flexibility in programming.

One of the strengths of NASM is its portability. Because NASM generates machine code directly, it can be used on a variety of operating systems and platforms. Additionally, NASM can be used to create object files, which can be linked to create executable programs.

Another advantage of NASM is its support for the x86-64 architecture. This architecture provides increased memory addressing capabilities and supports larger data types than the x86 architecture. NASM provides support for the 64-bit mode of the x86-64 architecture, allowing programmers to create high-performance applications that take full advantage of the increased capabilities of this architecture.

In conclusion, NASM is a powerful and flexible assembler that is widely used for low-level programming. Its support for macros, portability, and compatibility with the x86-64 architecture make it an attractive choice for system programming and application development. However, programming with NASM can be challenging, and requires a good understanding of low-level programming concepts and the x86 architecture. Here go your conclusions..

Bibliography

[1] GitHub Repository of work `https://github.com/TudorSIRGHI/AC-Labs`.

[2] For information Reasearch `https://openai.com/blog/chatgpt`.