UNIVERSITY OF CHITTAGONG

Department of Computer Science & Engineering

Program: **B.Sc.** (Engineering)
Session: 2022–2023
4th Semester

Assembly Codes Report

Course Title: *Microprocessors and Embedded Systems Lab*
Course Code: *CSE-416*

Submitted To

Dr. Atiqur Rahman
Associate Professor
Department of Computer Science & Engineering
University of Chittagong

Submitted By

Subha Shesgin
ID: 23701036
Department of Computer Science & Engineering

Date of Submission: 5th November 2025

Contents

Assembly Language Programming Assignment

1 Problem 1: Print ”Hello World!”

1.1 Problem Statement

Write an assembly language program that prints the string ”Hello World!”

1.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```
section .text

_start:
    mov rax, 1              ; syscall number for write
    mov rdi, 1              ; file descriptor: stdout
    mov rsi, message        ; address of the string
    mov rdx, message_length ; length of the string
    syscall                 ; invoke the syscall

    ; Exit gracefully
    mov rax, 60             ; syscall number for exit
    xor rdi, rdi            ; exit code 0
    syscall

section .data

message: db "Hello World!", 0xA
message_length: equ $ - message
```

1.3 Sample Output

```
Hello World!
```

2 Problem 2: Compare two numbers

2.1 Problem Statement

Write an assembly language program that takes two numbers as input from the user and shows which number is greater.

2.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```
; compare_numbers.asm
; Takes two integers (positive or negative) as input,
; compares them, and prints which one is greater or if they are equal.

section .bss
    num1    resb 16
    num2    resb 16

section .data
    msg1 db "Enter first number: ",0
    len_msg1 equ $-msg1

    msg2 db "Enter second number: ",0
    len_msg2 equ $-msg2

    greater db "First number is greater",10,0
    len_greater equ $-greater

    smaller db "Second number is greater",10,0
    len_smaller equ $-smaller

    equal db "Both numbers are equal",10,0
    len_equal equ $-equal

section .text
    global _start

_start:
    ; --- Ask for first number ---
    mov rax, 1          ; sys_write
    mov rdi, 1
    mov rsi, msg1
    mov rdx, len_msg1
    syscall

    ; --- Read first number ---
    mov rax, 0          ; sys_read
    mov rdi, 0
    mov rsi, num1
    mov rdx, 16
    syscall

    ; --- Convert to integer ---
    mov rsi, num1
    call str_to_int
    mov rbx, rax        ; store first number in rbx

    ; --- Ask for second number ---
    mov rax, 1
    mov rdi, 1
    mov rsi, msg2
    mov rdx, len_msg2
    syscall

    ; --- Read second number ---
    mov rax, 0
    mov rdi, 0
    mov rsi, num2
    mov rdx, 16
    syscall

    ; --- Convert to integer ---
    mov rsi, num2
    call str_to_int
    mov rcx, rax        ; store second number in rcx

    ; --- Compare ---
    cmp rbx, rcx
    jg first_greater
    jl second_greater

equal_case:
    mov rax, 1
    mov rdi, 1
    mov rsi, equal
    mov rdx, len_equal
    syscall
    jmp end_prog

first_greater:
    mov rax, 1
    mov rdi, 1
    mov rsi, greater
    mov rdx, len_greater
    syscall
    jmp end_prog

second_greater:
    mov rax, 1
    mov rdi, 1
    mov rsi, smaller
    mov rdx, len_smaller
    syscall

end_prog:
    mov rax, 60         ; sys_exit
    xor rdi, rdi
    syscall

; ----------------------------------------
; str_to_int: converts string in RSI to integer in RAX
; Handles optional leading ’-’
; Stops at newline or null terminator
str_to_int:
    xor rax, rax        ; result = 0
    xor rcx, rcx
    mov r8, 1           ; sign = +1

    ; Check for leading minus sign
    mov al, byte [rsi]
    cmp al, ’-’
    jne .parse_digits
    mov r8, -1
    inc rsi

.parse_digits:
    xor rax, rax        ; clear result
.next_char:
    mov cl, byte [rsi]
    cmp cl, 10          ; newline?
    je .done
    cmp cl, 0
    je .done
    sub cl, ’0’
    cmp cl, 9
    ja .done            ; non-digit, stop
    imul rax, rax, 10
    movzx rcx, cl       ; zero-extend digit
    add rax, rcx
    inc rsi
    jmp .next_char

.done:
    ; Apply sign
    cmp r8, 1
    je .ret
    neg rax
.ret:
    ret
```

2.3 Sample Input

```
Enter first number:  5
Enter second number:  6
```

2.4 Sample Output

```
Second number is greater
```

3 Problem 3: Sum of two numbers

3.1 Problem Statement

Write an assembly language program that takes two numbers as input from the user , prints them and shows the sum of those two numbers.

3.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```
section .data
    ; Prompt messages
    prompt1 db  "Enter first number: ", 0
    prompt1_len equ $ - prompt1 - 1  ; Subtract 1 to exclude null terminator

    prompt2 db  "Enter second number: ", 0
    prompt2_len equ $ - prompt2 - 1

    ; Result messages
    num1_msg db  "First number entered: ", 0
    num1_msg_len equ $ - num1_msg - 1

    num2_msg db  "Second number entered: ", 0
    num2_msg_len equ $ - num2_msg - 1

    sum_msg db  "The sum of ", 0
    sum_msg_len equ $ - sum_msg - 1

    and_msg db  " and ", 0
    and_msg_len equ $ - and_msg - 1

    is_msg  db  " is: ", 0
    is_msg_len equ $ - is_msg - 1

    newline db  10
    newline_len equ $ - newline

section .bss
    num1    resq 1          ; Space for first number
    num2    resq 1          ; Space for second number
    result  resq 1          ; Space for result
    buffer  resb 21         ; Buffer for input (20 digits + null)
    input_len resq 1        ; Length of input

section .text
    global _start

_start:
    ; Prompt for first number
    mov     rax, 1          ; sys_write
    mov     rdi, 1          ; stdout
    mov     rsi, prompt1
    mov     rdx, prompt1_len
    syscall

    ; Read first number
    call    read_number
    mov     [num1], rax     ; Store first number

    ; Display what was entered
    mov     rax, 1          ; sys_write
    mov     rdi, 1          ; stdout
    mov     rsi, num1_msg
    mov     rdx, num1_msg_len
    syscall

    mov     rax, [num1]
    call    print_number
    call    print_newline

    ; Prompt for second number
    mov     rax, 1          ; sys_write
    mov     rdi, 1          ; stdout
    mov     rsi, prompt2
    mov     rdx, prompt2_len
    syscall

    ; Read second number
    call    read_number
    mov     [num2], rax     ; Store second number

    ; Display what was entered
    mov     rax, 1          ; sys_write
    mov     rdi, 1          ; stdout
    mov     rsi, num2_msg
    mov     rdx, num2_msg_len
    syscall

    mov     rax, [num2]
    call    print_number
    call    print_newline

    ; Add the two numbers
    mov     rax, [num1]
    mov     rbx, [num2]
    add     rax, rbx
    mov     [result], rax

    ; Display the complete result message
    mov     rax, 1          ; sys_write
    mov     rdi, 1          ; stdout
    mov     rsi, sum_msg
    mov     rdx, sum_msg_len
    syscall

    ; Display first number in result
    mov     rax, [num1]
    call    print_number

    mov     rax, 1          ; sys_write
    mov     rdi, 1          ; stdout
    mov     rsi, and_msg
    mov     rdx, and_msg_len
    syscall

    ; Display second number in result
    mov     rax, [num2]
    call    print_number

    mov     rax, 1          ; sys_write
    mov     rdi, 1          ; stdout
    mov     rsi, is_msg
    mov     rdx, is_msg_len
    syscall

    ; Display the result
    mov     rax, [result]
    call    print_number
    call    print_newline

    ; Exit program
    mov     rax, 60         ; sys_exit
    xor     rdi, rdi        ; exit code 0
    syscall

; Function to read a number from stdin
read_number:
    ; Read input from stdin
    mov     rax, 0          ; sys_read
    mov     rdi, 0          ; stdin
    mov     rsi, buffer
    mov     rdx, 20         ; max length
    syscall

    ; Check if we got any input
    cmp     rax, 0
    jle     .no_input

    mov     [input_len], rax ; Save length

    ; Convert string to integer
    mov     rsi, buffer     ; Pointer to string
    mov     rcx, rax        ; Length
    xor     rax, rax        ; Clear result
    xor     rbx, rbx        ; Clear temporary
    xor     rdx, rdx        ; Clear digit

.convert_loop:
    mov     bl, [rsi]       ; Get current character
    cmp     bl, 10          ; Check for newline
    je      .done
    cmp     bl, 13          ; Check for carriage return
    je      .done
    cmp     bl, '0'         ; Validate digit
    jb      .skip
    cmp     bl, '9'
    ja      .skip

    sub     bl, '0'         ; Convert ASCII to digit
    imul    rax, 10         ; Multiply current result by 10
    add     rax, rbx        ; Add new digit

.skip:
    inc     rsi
    loop    .convert_loop

.done:
    ret

.no_input:
    xor     rax, rax        ; Return 0 if no input
    ret

; Function to print a number from RAX
print_number:
    push    rax             ; Save number
    push    rdi
    push    rsi
    push    rdx

    mov     rdi, buffer
    call    int_to_string

    ; Calculate string length
    mov     rsi, rdi        ; Start of string (returned from int_to_string)
    mov     rdx, buffer
    add     rdx, 21         ; End of buffer
    sub     rdx, rsi        ; RDX = length

    ; Write the number
    mov     rax, 1          ; sys_write
    mov     rdi, 1          ; stdout
    syscall

    pop     rdx
    pop     rsi
    pop     rdi
    pop     rax             ; Restore number
    ret

; Function to convert integer to string
; Input: RAX = number
; Output: RDI = pointer to start of string in buffer
int_to_string:
    push    rbx
    push    rdx
    push    rsi

    mov     rbx, 10         ; Base 10
    mov     rdi, buffer
    add     rdi, 20         ; Point to end of buffer
    mov     byte [rdi], 0   ; Null terminator

    test    rax, rax        ; Check if number is zero
    jnz     .not_zero
    dec     rdi
    mov     byte [rdi], '0'
    jmp     .done

.not_zero:
    ; Handle positive numbers

.convert_loop:
    dec     rdi
    xor     rdx, rdx        ; Clear RDX for division
    div     rbx             ; RAX = quotient, RDX = remainder
    add     dl, '0'         ; Convert to ASCII
    mov     [rdi], dl
    test    rax, rax        ; Check if quotient is zero
    jnz     .convert_loop

.done:
    ; RDI now points to the start of the string
    pop     rsi
    pop     rdx
    pop     rbx
    ret

; Function to print newline
print_newline:
    mov     rax, 1          ; sys_write
    mov     rdi, 1          ; stdout
    mov     rsi, newline
    mov     rdx, newline_len
    syscall
    ret
```

3.3 Sample Input

```
Enter first number:  12
Enter second number:  10
```

3.4 Sample Output

```
First number entered:  12
Second number entered:  10
The sum of 12 and 10 is:  22
```

4 Problem 4: Swap two numbers

4.1 Problem Statement

Write an assembly language program that takes two numbers as input from the user and then swaps them.

4.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```
section .data
    msg1 db "Enter first number: ",0
    msg1_len equ $-msg1
    msg2 db "Enter second number: ",0
    msg2_len equ $-msg2
    outmsg db "After swapping: ",0
    outmsg_len equ $-outmsg
    newline db 10

section .bss
    inbuf1  resb 10         ; buffer for first number
    inbuf2  resb 10         ; buffer for second number
    num1    resq 1          ; store first number (8 bytes)
    num2    resq 1          ; store second number (8 bytes)
    numbuf  resb 20         ; buffer for printing numbers

section .text
global _start

_start:
    ; -------- Print "Enter first number:" --------
    mov rax, 1
    mov rdi, 1
    mov rsi, msg1
    mov rdx, msg1_len
    syscall

    ; -------- Read first number --------
    mov rax, 0
    mov rdi, 0
    mov rsi, inbuf1
    mov rdx, 10
    syscall

    ; -------- Convert first number string to integer --------
    xor r12, r12        ; r12 = num1
    mov rsi, inbuf1
.parse1:
    mov al, [rsi]
    cmp al, ’0’
    jb .done1
    cmp al, '9'
    ja .done1
    imul r12, r12, 10
    movzx rdx, al
    sub rdx, '0'
    add r12, rdx
    inc rsi
    jmp .parse1
.done1:
    mov [num1], r12

    ; -------- Print "Enter second number:" --------
    mov rax, 1
    mov rdi, 1
    mov rsi, msg2
    mov rdx, msg2_len
    syscall

    ; -------- Read second number --------
    mov rax, 0
    mov rdi, 0
    mov rsi, inbuf2
    mov rdx, 10
    syscall

    ; -------- Convert second number string to integer --------
    xor r12, r12        ; r12 = num2
    mov rsi, inbuf2
.parse2:
    mov al, [rsi]
    cmp al, '0'
    jb .done2
    cmp al, '9'
    ja .done2
    imul r12, r12, 10
    movzx rdx, al
    sub rdx, '0'
    add r12, rdx
    inc rsi
    jmp .parse2
.done2:
    mov [num2], r12

    ; -------- Swap numbers --------
    mov rax, [num1]
    mov rbx, [num2]
    mov [num1], rbx
    mov [num2], rax

    ; -------- Print output message --------
    mov rax, 1
    mov rdi, 1
    mov rsi, outmsg
    mov rdx, outmsg_len
    syscall

    ; -------- Print num1 --------
    mov rax, [num1]
    mov rsi, numbuf
    call int_to_string
    mov rax, 1
    mov rdi, 1
    mov rdx, r13          ; r13 = length of string returned
    syscall

    ; -------- Print space --------
    mov rax, 1
    mov rdi, 1
    mov rsi, newline
    mov rdx, 1
    syscall

    ; -------- Print num2 --------
    mov rax, [num2]
    mov rsi, numbuf
    call int_to_string
    mov rax, 1
    mov rdi, 1
    mov rdx, r13
    syscall

    ; -------- Print newline --------
    mov rax, 1
    mov rdi, 1
    mov rsi, newline
    mov rdx, 1
    syscall

    ; -------- Exit --------
    mov rax, 60
    xor rdi, rdi
    syscall

; -------- Function: int_to_string --------
; Converts rax integer into decimal string stored at rsi
; Returns length in r13
int_to_string:
    mov rbx, 10          ; divisor
    xor r12, r12          ; digit counter
    lea rdi, [rsi+19]     ; point to end of buffer
    mov byte [rdi], 0     ; null terminator
.convert_loop:
    xor rdx, rdx
    div rbx                   ; rax / 10, quotient in rax, remainder in rdx
    add dl, '0'
    dec rdi
    mov [rdi], dl
    inc r12
    cmp rax, 0
    jne .convert_loop
    mov r13, r12              ; length of string
    mov rsi, rdi              ; pointer to start of string
    ret
```

4.3 Sample Input

```
Enter first number: 4
Enter second number: 7
```

4.4 Sample Output

```
After swapping: 7 4
```

5 Problem 5: Fibonacci Series 01

5.1 Problem Statement

Write an assembly language program that takes a number as input from the user and checks if the number is in a fibonacci series or not.

5.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```
; fibo64.asm
section .bss
    inbuf   resb 32

section .data
    prompt  db "Enter a number: ",0
    lenp    equ $-prompt
    msgfib  db "Fibonacci",10,0
    lenfib  equ $-msgfib
    msgnot  db "Not Fibonacci",10,0
    lennot  equ $-msgnot

```

```
section .text
    global _start

_start:
    ; ---- Prompt ----
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt
    mov rdx, lenp
    syscall

    ; ---- Read input ----
    mov rax, 0
    mov rdi, 0
    mov rsi, inbuf
    mov rdx, 32
    syscall

    ; ---- Parse ASCII -> integer in r12 ----
    mov rsi, inbuf
    xor r12, r12
.parse:
    mov al, [rsi]
    cmp al, ’0’
    jb   .done
    cmp al, ’9’
    ja   .done
    imul r12, r12, 10
    movzx rdx, al
    sub rdx, ’0’
    add r12, rdx
    inc rsi
    jmp .parse
.done:

    ; ---- Edge case: if n < 0, Not Fibonacci ----
    cmp r12, 0
    jl   print_not
    cmp r12, 0
    je   print_fib        ; 0 is Fibonacci

    ; ---- Fibonacci loop ----
    xor r13, r13          ; f0 = 0
    mov r14, 1            ; f1 = 1

fib_loop:
    cmp r13, r12
    je   print_fib        ; n matches Fibonacci
    cmp r13, r12
    ja   print_not        ; n exceeded Fibonacci

    ; next = f0 + f1
    mov rax, r13
    add rax, r14
    mov r13, r14          ; f0 = f1
    mov r14, rax          ; f1 = next
    jmp fib_loop

; ---- Print Fibonacci ----
print_fib:
    mov rax, 1
    mov rdi, 1
    mov rsi, msgfib
    mov rdx, lenfib
    syscall
    jmp exit

; ---- Print Not Fibonacci ----
print_not:
    mov rax, 1
    mov rdi, 1
    mov rsi, msgnot
    mov rdx, lennot
    syscall

; ---- Exit ----
exit:
    mov rax, 60
    xor rdi, rdi
    syscall
```

5.3 Sample Input 1

```
Enter a number: 3
```

5.4 Sample Output 1

```
Fibonacci
```

5.5 Sample Input 2

```
Enter a number: 4
```

5.6 Sample Output 2

```
Not Fibonacci
```

6 Problem 6: Fibonacci Series 02

6.1 Problem Statement

Write an assembly language program that takes a number as input from the user and prints the fibonacci series for that many numbers.

6.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```
section .bss
    inbuf   resb 32

section .data
    prompt  db "Enter number of terms: ",0
    lenp    equ $-prompt
    space   db " ",0
    newline db 10

section .text
    global _start

_start:
    ; ---- Prompt ----
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt
    mov rdx, lenp
    syscall

    ; ---- Read input ----
    mov rax, 0
    mov rdi, 0
    mov rsi, inbuf
    mov rdx, 32
    syscall

    ; ---- Parse ASCII -> integer in r12 ----
    mov rsi, inbuf
    xor r12, r12
.parse:
    mov al, [rsi]
    cmp al, '0'
    jb .done
    cmp al, '9'
    ja .done
    imul r12, r12, 10
    movzx rdx, al
    sub rdx, '0'
    add r12, rdx
    inc rsi
    jmp .parse
.done:

    ; ---- Edge case: if n == 0, exit ----
    test r12, r12
    jz exit

    ; ---- Fibonacci variables ----
    xor r13, r13         ; f0 = 0
    mov r14, 1           ; f1 = 1
    xor r15, r15         ; counter = 0

fib_loop:
    cmp r15, r12
    jge exit             ; printed n terms, exit

    ; ---- print f0 ----
    mov rdi, r13
    call print_num

    ; print space
    mov rax, 1
    mov rdi, 1
    mov rsi, space
    mov rdx, 1
    syscall

    ; ---- next Fibonacci ----
    mov rax, r13
    add rax, r14
    mov r13, r14         ; f0 = f1
    mov r14, rax         ; f1 = next

    inc r15
    jmp fib_loop

; ---- Print number in rdi as decimal ----
print_num:
    mov rax, rdi
    mov rcx, 10
    mov rbx, rsp
    sub rsp, 32
    mov rsi, rsp
    add rsi, 32

.convert:
    xor rdx, rdx
    div rcx
    add dl, '0'
    dec rsi
    mov [rsi], dl
    test rax, rax
    jnz .convert

    mov rax, 1
    mov rdi, 1
    mov rdx, rsp
    add rdx, 32
    sub rdx, rsi
    mov rsi, rsi
    syscall

    mov rsp, rbx
    ret

exit:
    mov rax, 60
    xor rdi, rdi
    syscall
```

6.3 Sample Input

```
Enter number of terms: 7
```

6.4 Sample Output

```
0 1 1 2 3 5 8
```

7 Problem 7: Primality Check

7.1 Problem Statement

Write an assembly language program that takes a number as input from the user and checks if the number is prime or not.

7.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```
section .data
    msg db "Enter a number: ",0
    msg_len equ $-msg
    prime_msg db "Number is prime",0
    prime_msg_len equ $-prime_msg
    not_prime_msg db "Number is not prime",0
    not_prime_msg_len equ $-not_prime_msg
    newline db 10

section .bss
    inbuf resb 10       ; buffer for input
    num   resq 1        ; number to check

section .text
global _start

_start:
    ; -------- Print prompt --------
    mov rax, 1
    mov rdi, 1
    mov rsi, msg
    mov rdx, msg_len
    syscall

    ; -------- Read input --------
    mov rax, 0
    mov rdi, 0
    mov rsi, inbuf
    mov rdx, 10
    syscall

    ; -------- Convert string to integer --------
    xor r12, r12        ; r12 = number
    mov rsi, inbuf
.parse:
    mov al, [rsi]
    cmp al, '0'
    jb .done_parse
    cmp al, '9'
    ja .done_parse
    imul r12, r12, 10
    movzx rdx, al
    sub rdx, '0'
    add r12, rdx
    inc rsi
    jmp .parse
.done_parse:
    mov [num], r12     ; store number

    ; -------- Prime check --------
    mov rax, [num]      ; rax = number to check
    cmp rax, 2
    jb .not_prime       ; numbers less than 2 are not prime
    je .prime           ; 2 is prime

    mov rbx, 2          ; divisor = 2
.check_loop:
    mov rdx, 0
    mov rcx, rax
    div rbx              ; rax / rbx, quotient in rax, remainder in rdx
    cmp rdx, 0
    je .not_prime         ; divisible -> not prime
    inc rbx
    mov rax, [num]
    cmp rbx, rax
    jl .check_loop

.prime:
    ; print prime_msg
    mov rax, 1
    mov rdi, 1
    mov rsi, prime_msg
    mov rdx, prime_msg_len
    syscall
    jmp .done

.not_prime:
    ; print not_prime_msg
    mov rax, 1
    mov rdi, 1
    mov rsi, not_prime_msg
    mov rdx, not_prime_msg_len
    syscall

.done:
    ; print newline
    mov rax, 1
    mov rdi, 1
    mov rsi, newline
    mov rdx, 1
    syscall

    ; -------- Exit --------
    mov rax, 60
    xor rdi, rdi
    syscall
```

7.3 Sample Input 01

```
Enter a number: 4
```

7.4 Sample Output 01

```
Number is not prime
```

7.5 Sample Input 02

```
Enter a number: 7
```

7.6 Sample Output 02

```
Number is prime
```

8 Problem 8: Pattern : Left triangle

8.1 Problem Statement

Write an assembly language program that takes a number as input from the user and forms a left triangle pattern of that many rows.

8.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```
global _start

section .data
star: db '*'
nl:   db 10
prompt: db "Enter number of rows: ",0
plen: equ $-prompt

section .bss
inbuf: resb 64
```

```
section .text
_start:
    ; print prompt
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt
    mov rdx, plen
    syscall

    ; read input
    mov rax, 0
    mov rdi, 0
    mov rsi, inbuf
    mov rdx, 64
    syscall

    ; parse digits
    xor rax, rax
    mov rsi, inbuf

.parse_loop:
    mov bl, [rsi]
    cmp bl,'0'
    jl .parsed
    cmp bl,'9'
    jg .parsed
    imul rax, rax, 10
    sub bl,'0'
    movzx rbx, bl
    add rax, rbx
    inc rsi
    jmp .parse_loop

.parsed:
    mov r8, rax         ; N safely
    mov r9, 1           ; outer loop counter i = 1

.outer_loop:
    cmp r9, r8
    jg .exit

    mov rbx, r9         ; inner loop counter j = i
.inner_loop:
    cmp rbx, 0
    je .newline

    mov rax, 1
    mov rdi, 1
    mov rsi, star
    mov rdx, 1
    syscall

    dec rbx
    jmp .inner_loop

.newline:
    mov rax, 1
    mov rdi, 1
    mov rsi, nl
    mov rdx, 1
    syscall

    inc r9
    jmp .outer_loop

.exit:
    mov rax, 60
    xor rdi, rdi
    syscall
```

8.3 Sample Input

```
Enter number of rows:  5
```

8.4 Sample Output

```
*
**
***
****
*****
```

9 Problem 9: Pattern : Pyramid

9.1 Problem Statement

Write an assembly language program that takes a number as input from the user and forms a Pyramid pattern of that many rows.

9.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```
section .bss
    inbuf   resb 32  ; reserve 32 bytes of memory in buffer not in executable file

section .data       ; the initialized data section. Anything you define here gets stored in the executable
    prompt  db "Enter number: "
    prompt_len equ $-prompt
    star    db "*"
    space   db " "
    newline db 10

section .text
    global _start

_start:
    ; write prompt
    mov rax, 1                ; sys_write
    mov rdi, 1                ; stdout
    mov rsi, prompt
    mov rdx, prompt_len
    syscall

    ; read input
    mov rax, 0                ; sys_read
    mov rdi, 0                ; stdin
    mov rsi, inbuf
    mov rdx, 32
    syscall                   ; rax = bytes read (unused here)

    ; parse unsigned integer from ASCII in inbuf -> r12
    xor r12, r12              ; r12 = total rows
    mov rsi, inbuf
.parse:
    mov al, [rsi]
    cmp al, '0' ; compae with 0
    jb  parsed
    cmp al, '9'
    ja  parsed
    imul r12, r12, 10
    movzx rdx, al
    sub rdx, '0'
    add r12, rdx
    inc rsi
    jmp .parse

parsed:
    ; if rows == 0, just exit
    test r12, r12
    jz exit

    mov r13, 1              ; current row = 1
row_loop:
    cmp r13, r12
    jg  exit

    ; spaces = r12 - r13
    mov rbx, r12
    sub rbx, r13
    call print_spaces

    ; stars = 2*r13 - 1
    mov rbx, r13
    shl rbx, 1
    sub rbx, 1
    call print_stars

    ; newline
    mov rax, 1
    mov rdi, 1
    mov rsi, newline
    mov rdx, 1
    syscall

    inc r13
    jmp row_loop

; rbx = count; prints that many spaces
print_spaces:
    test rbx, rbx
    jle .ps_done
.ps_loop:
    mov rax, 1
    mov rdi, 1
    mov rsi, space
    mov rdx, 1
    syscall
    dec rbx
    jg  .ps_loop
.ps_done:
    ret

; rbx = count; prints that many stars
print_stars:
    test rbx, rbx
    jle .pt_done
.pt_loop:
    mov rax, 1
    mov rdi, 1
    mov rsi, star
    mov rdx, 1
    syscall
    dec rbx
    jg  .pt_loop
.pt_done:
    ret

exit:
    mov rax, 60             ; sys_exit
    xor rdi, rdi
    syscall
```

9.3 Sample Input

```
Enter number: 6
```

9.4 Sample Output

```
     *
    ***
   *****
  *******
 *********
***********
```

10 Problem 10: GCD

10.1 Problem Statement

Write an assembly language program that takes two numbers as input from the user and finds the GCD of those two numbers.

10.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```

global _start

section .data
prompt1 db "Enter first number: ",0
plen1   equ $-prompt1
prompt2 db "Enter second number: ",0
plen2   equ $-prompt2
msg     db "GCD is: ",0
msg_len equ $-msg
```

```
nl       db 10

section .bss
buf1 resb 32
buf2 resb 32

section .text
_start:
    ; --- Prompt first number ---
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt1
    mov rdx, plen1
    syscall

    mov rax, 0
    mov rdi, 0
    mov rsi, buf1
    mov rdx, 32
    syscall
    mov rsi, buf1
    call strip_newline
    call atoi
    mov r8, rax       ; store first number

    ; --- Prompt second number ---
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt2
    mov rdx, plen2
    syscall

    mov rax, 0
    mov rdi, 0
    mov rsi, buf2
    mov rdx, 32
    syscall
    mov rsi, buf2
    call strip_newline
    call atoi
    mov r9, rax       ; store second number

    ; --- Euclidean GCD ---
    mov rax, r8       ; a
    mov rbx, r9       ; b

gcd_loop:
    cmp rbx, 0
    je gcd_done
    xor rdx, rdx       ; zero RDX BEFORE div
    div rbx
    mov rax, rbx
    mov rbx, rdx
    jmp gcd_loop

gcd_done:
    mov r10, rax       ; save GCD safely

    ; --- Print "GCD is: " ---
    mov rax, 1
    mov rdi, 1
    mov rsi, msg
    mov rdx, msg_len
    syscall

    ; --- Convert GCD to string ---
    mov rax, r10       ; use saved GCD
    mov rbx, 10
    lea rsi, [rsp-20]
    mov rcx, 0

convert_loop:
    xor rdx, rdx
    div rbx
    add dl, '0'
    dec rsi
    mov [rsi], dl
    inc rcx
    test rax, rax
    jnz convert_loop

    ; --- Print GCD string ---
    mov rax, 1
    mov rdi, 1
    mov rdx, rcx
    syscall

    ; --- Print newline ---
    mov rax, 1
    mov rdi, 1
    mov rsi, nl
    mov rdx, 1
    syscall

    ; --- Exit ---
    mov rax, 60
    xor rdi, rdi
    syscall

; --- atoi function ---
; Converts string in RSI to integer in RAX
atoi:
    xor rax, rax
.next_digit:
    mov bl, [rsi]
    cmp bl,'0'
    jl .done
    cmp bl,'9'
    jg .done
    imul rax, rax, 10
    sub bl,'0'
    movzx rbx, bl
    add rax, rbx
    inc rsi
    jmp .next_digit
.done:
    ret

; --- strip_newline function ---
; Replaces first newline (ASCII 10) with null terminator
strip_newline:
    mov rbx, rsi
.strip_loop:
    cmp byte [rbx], 0
    je .strip_done
    cmp byte [rbx], 10
    jne .next_char
    mov byte [rbx], 0
    jmp .strip_done
.next_char:
    inc rbx
    jmp .strip_loop
.strip_done:
    ret
```

10.3 Sample Input

```
Enter first number:  10 Enter second number:  12
```

10.4 Sample Output

```
GCD is:  2
```

11 Problem 11: LCM

11.1 Problem Statement

Write an assembly language program that takes two numbers as input from the user and finds the LCM of those two numbers.

11.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```

section .bss
    num1 resb 20
    num2 resb 20
    result resb 20

section .data
    prompt1 db "Enter first number: ",0
    prompt1_len equ $-prompt1
    prompt2 db "Enter second number: ",0
    prompt2_len equ $-prompt2
    newline db 10,0

section .text
    global _start

_start:
    ; --- Read first number ---
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt1
    mov rdx, prompt1_len
    syscall

    mov rax, 0
    mov rdi, 0
    mov rsi, num1
    mov rdx, 20
    syscall

    mov rsi, num1
    call str2int
    mov rbx, r8            ; store first number

    ; --- Read second number ---
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt2
    mov rdx, prompt2_len
    syscall

    mov rax, 0
    mov rdi, 0
    mov rsi, num2
    mov rdx, 20
    syscall

    mov rsi, num2
    call str2int
    mov rcx, r8               ; second number

    ; --- Compute GCD ---
    mov r9, rbx               ; keep copy of first number
    mov r10, rcx              ; keep copy of second number
gcd_loop:
    cmp rcx, 0
    je gcd_done
    mov rax, rbx
    xor rdx, rdx
    div rcx
    mov rbx, rcx
    mov rcx, rdx
    jmp gcd_loop
gcd_done:
    mov r8, rbx               ; GCD in r8

    ; --- Compute LCM = (a * b) / GCD ---
    mov rax, r9               ; rax = first number
    mov rbx, r10              ; rbx = second number
    mul rbx                   ; rdx:rax = rax * rbx
    mov rbx, r8               ; GCD
    xor rdx, rdx              ; clear rdx for division
    div rbx                   ; rax = LCM
    mov r8, rax               ; store LCM in r8

    ; --- Convert LCM to string ---
    mov rdi, result
    call int2str

    ; --- Print LCM ---
    mov rax, 1
    mov rdi, 1
    mov rsi, result
    mov rdx, 20
    syscall

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

    ; Exit
    mov rax, 60
    xor rdi, rdi
    syscall

; -----------------------------
; Convert string to integer (r8)
; Input: rsi = pointer to string
; Output: r8 = integer
str2int:
    xor r8, r8
.next_digit:
    mov al, byte [rsi]
    cmp al, 10
    je .done
    sub al, '0'
    imul r8, r8, 10
    add r8, rax
    inc rsi
    jmp .next_digit
.done:
    ret

; -----------------------------
; Convert integer in r8 to string at rdi
int2str:
    mov rax, r8
    mov rcx, 0
    mov rbx, 10
.reverse_loop:
    xor rdx, rdx
    div rbx
    add dl, '0'
    push rdx
    inc rcx
    cmp rax, 0
    jne .reverse_loop

.print_loop:
    pop rax
    mov [rdi], al
    inc rdi
    dec rcx
    cmp rcx, 0
    jne .print_loop
    mov byte [rdi], 0
    ret
```

11.3 Sample Input

```
Enter first number:  9
Enter second number:  13
```

11.4 Sample Output

```
LCM : 117
```

12 Problem 12: Average

12.1 Problem Statement

Write an assembly language program that takes three numbers as input from the user and finds the average of those two numbers.

12.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```

; NASM x86-64 Assembly - Average of 3 numbers
; Assemble: nasm -f elf64 avg.asm -o avg.o
; Link: ld avg.o -o avg
; Run: ./avg

section .bss
    num1 resb 10
    num2 resb 10
    num3 resb 10
    result resb 10

section .data
    prompt1 db "Enter first number: ", 0
    prompt1_len equ $-prompt1
    prompt2 db "Enter second number: ", 0
    prompt2_len equ $-prompt2
    prompt3 db "Enter third number: ", 0
    prompt3_len equ $-prompt3
    out_msg db "Average is: ", 0
    out_len equ $-out_msg

section .text
    global _start

_start:
    ; --- Read first number ---
    mov rax, 1          ; sys_write
    mov rdi, 1          ; stdout
    mov rsi, prompt1
    mov rdx, prompt1_len
    syscall

    mov rax, 0          ; sys_read
    mov rdi, 0          ; stdin
    mov rsi, num1
    mov rdx, 10
    syscall

    call str2int
    mov rbx, rax        ; store first number in rbx

    ; --- Read second number ---
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt2
    mov rdx, prompt2_len
    syscall

    mov rax, 0
    mov rdi, 0
    mov rsi, num2
    mov rdx, 10
    syscall

    call str2int
    add rbx, rax        ; sum += second number

    ; --- Read third number ---
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt3
    mov rdx, prompt3_len
    syscall

    mov rax, 0
    mov rdi, 0
    mov rsi, num3
    mov rdx, 10
    syscall

    call str2int
    add rbx, rax        ; sum += third number

    ; --- Calculate average ---
    mov rax, rbx
    mov rcx, 3
    cqo                 ; extend rax to rdx:rax
    idiv rcx            ; rax = rax / 3, remainder in rdx

    ; --- Convert integer to string ---
    mov rdi, result
    call int2str

    ; --- Print output ---
    mov rax, 1
    mov rdi, 1
    mov rsi, out_msg
    mov rdx, out_len
    syscall

    mov rax, 1
    mov rdi, 1
    mov rsi, result
    mov rdx, 10
    syscall

    ; --- Exit ---
    mov rax, 60
    xor rdi, rdi
    syscall

; --- Subroutine: string to integer ---
str2int:
    xor rax, rax
    xor rcx, rcx
.next_char:
    mov cl, byte [rsi]
    cmp cl, 10        ; newline
    je .done
    cmp cl, 0
    je .done
    sub cl, '0'
    imul rax, rax, 10
    add rax, rcx
    inc rsi
    jmp .next_char
.done:
    ret

; --- Subroutine: integer to string ---
int2str:
    mov rbx, 10
    xor rcx, rcx        ; digit counter
.next_digit:
    xor rdx, rdx
    div rbx
    add dl, '0'
    push rdx
    inc rcx
    test rax, rax
    jnz .next_digit

    mov rdi, rdi        ; destination buffer
.print_digit:
    pop rax
    mov [rdi], al
    inc rdi
    loop .print_digit
    mov byte [rdi], 10  ; newline
    ret
```

12.3 Sample Input

```
Enter first number: 12
Enter second number: 13
Enter third number: 17
```

12.4 Sample Output

```
Average is: 14
```

13 Problem 13: Factorial

13.1 Problem Statement

Write an assembly language program that takes a number as input from the user and finds the factorial of that number.

13.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```
section .bss
    num resb 20
    result resb 50      ; to store factorial as string (big enough)

section .data
    prompt db "Enter a number: ",0
    prompt_len equ $-prompt
    newline db 10,0

```

```
section .text
    global _start

_start:
    ; --- Prompt user ---
    mov rax, 1
    mov rdi, 1
    mov rsi, prompt
    mov rdx, prompt_len
    syscall

    ; --- Read input ---
    mov rax, 0
    mov rdi, 0
    mov rsi, num
    mov rdx, 20
    syscall

    ; --- Convert string to integer ---
    mov rsi, num
    call str2int
    mov rbx, r8       ; n in rbx

    ; --- Compute factorial ---
    mov rax, 1        ; factorial accumulator in rax
    cmp rbx, 0
    je print_result   ; 0! = 1

fact_loop:
    imul rax, rbx     ; rax = rax * rbx
    dec rbx
    cmp rbx, 0
    jne fact_loop

print_result:
    mov r8, rax       ; store factorial in r8
    mov rdi, result
    call int2str

    ; --- Print factorial ---
    mov rax, 1
    mov rdi, 1
    mov rsi, result
    mov rdx, 50
    syscall

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

    ; Exit
    mov rax, 60
    xor rdi, rdi
    syscall

; -----------------------------
; Convert string to integer (r8)
; Input: rsi = pointer to string
; Output: r8 = integer
str2int:
    xor r8, r8
.next_digit:
    mov al, byte [rsi]
    cmp al, 10
    je .done
    sub al, ’0’
    imul r8, r8, 10
    add r8, rax
    inc rsi
    jmp .next_digit
.done:
    ret

; -----------------------------
; Convert integer in r8 to string at rdi
int2str:
    mov rax, r8
    mov rcx, 0
    mov rbx, 10
.reverse_loop:
    xor rdx, rdx
    div rbx
    add dl, ’0’
    push rdx
    inc rcx
    cmp rax, 0
    jne .reverse_loop

.print_loop:
    pop rax
    mov [rdi], al
    inc rdi
    dec rcx
    cmp rcx, 0
    jne .print_loop
    mov byte [rdi], 0
    ret
```

13.3 Sample Input

```
Enter a number:  5
```

13.4 Sample Output

```
120
```

14 Problem 14: Matrix Addition

14.1 Problem Statement

Write an assembly language program that takes two 3X3 matrices as input from the user and performs the matrix addition operation among those two matrices.

14.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```
section .data
    ; Matrix dimensions
    rows equ 3
    cols equ 3
    matrix_size equ rows * cols

    ; Prompt messages
    msg_matrix1 db "Enter elements for Matrix 1 (3x3):", 10, 0
    msg_matrix2 db 10, "Enter elements for Matrix 2 (3x3):", 10, 0
    msg_result db 10, "Sum of matrices:", 10, 0
    space db " ", 0
    newline db 10, 0

    ; Buffers
    number_buffer times 12 db 0
    input_buffer times 100 db 0  ; Larger buffer for row input

section .bss
    matrix1 resd 9      ; Reserve space for 3x3 matrix (9 doublewords)
    matrix2 resd 9
    result resd 9

section .text
    global _start

```

```
; Function to read a row of integers
read_row:
    push rbx
    push rcx
    push rdx
    push rsi
    push rdi
    push r8
    push r9

    ; Read entire line
    mov rax, 0          ; sys_read
    mov rdi, 0          ; stdin
    mov rsi, input_buffer
    mov rdx, 100
    syscall

    mov rsi, input_buffer  ; pointer to input
    mov rdi, r8           ; matrix pointer
    mov r9, 0             ; numbers read counter

parse_loop:
    ; Skip whitespace
    cmp byte [rsi], ' '
    je skip_space
    cmp byte [rsi], 9    ; tab
    je skip_space
    cmp byte [rsi], 10   ; newline
    je parse_done
    cmp byte [rsi], 0    ; null terminator
    je parse_done

    ; Convert number
    mov rax, 0
    mov rcx, 0           ; negative flag
    mov bl, byte [rsi]
    cmp bl, '-'
    jne convert_digit
    mov rcx, 1
    inc rsi

convert_digit:
    movzx rbx, byte [rsi]
    cmp rbx, '0'
    jl number_done
    cmp rbx, '9'
    jg number_done
    sub rbx, '0'
    imul rax, 10
    add rax, rbx
    inc rsi
    jmp convert_digit

number_done:
    test rcx, rcx
    jz store_number
    neg rax

store_number:
    mov [rdi], eax      ; store in matrix
    add rdi, 4
    inc r9

    ; Check if we've read enough numbers
    cmp r9, cols
    jge parse_done

    jmp parse_loop

skip_space:
    inc rsi
    jmp parse_loop

parse_done:
    pop r9
    pop r8
    pop rdi
    pop rsi
    pop rdx
    pop rcx
    pop rbx
    ret

; Function to read a matrix
read_matrix:
    push r8
    push r9

    mov r8, rdi        ; matrix pointer
    mov r9, 0          ; row counter

read_row_loop:
    call read_row      ; read one row

    add r8, cols * 4   ; move to next row
    inc r9
    cmp r9, rows
    jl read_row_loop

    pop r9
    pop r8
    ret
```

```

; Function to print integer
print_int:
    push rax
    push rbx
    push rcx
    push rdx
    push rsi
    push rdi

    mov rdi, number_buffer + 11
    mov byte [rdi], 0
    mov rbx, 10

    test eax, eax
    jns convert_loop
    neg eax
    push rax
    mov al, ’-’
    call print_char
    pop rax

convert_loop:
    xor rdx, rdx
    div rbx
    add dl, ’0’
    dec rdi
    mov [rdi], dl
    test rax, rax
    jnz convert_loop

    ; Print the number
    mov rsi, rdi
    mov rdx, number_buffer + 11
    sub rdx, rsi
    mov rax, 1
    mov rdi, 1
    syscall

    pop rdi
    pop rsi
    pop rdx
    pop rcx
    pop rbx
    pop rax
    ret

; Function to print single character
print_char:
    push rax
    push rdi
    push rsi
    push rdx

    mov [number_buffer], al
    mov rax, 1
    mov rdi, 1
    mov rsi, number_buffer
    mov rdx, 1
    syscall

    pop rdx
    pop rsi
    pop rdi
    pop rax
    ret

; Function to print string
print_string:
    push rax
    push rdi
    push rsi
    push rdx
    push rcx

    mov rsi, rax        ; string pointer
    mov rdx, 0          ; length counter

str_len_loop:
    cmp byte [rsi + rdx], 0
    je str_len_done
    inc rdx
    jmp str_len_loop

str_len_done:
    mov rax, 1          ; sys_write
    mov rdi, 1          ; stdout
    syscall

    pop rcx
    pop rdx
    pop rsi
    pop rdi
    pop rax
    ret

_start:
    ; Read Matrix 1
    mov rax, msg_matrix1
    call print_string
    mov rdi, matrix1
    call read_matrix

    ; Read Matrix 2
    mov rax, msg_matrix2
    call print_string
    mov rdi, matrix2
    call read_matrix

    ; Calculate sum of matrices
    mov rcx, 0
sum_loop:
    mov eax, [matrix1 + rcx * 4]
    add eax, [matrix2 + rcx * 4]
    mov [result + rcx * 4], eax
    inc rcx
    cmp rcx, matrix_size
    jl sum_loop

    ; Print result message
    mov rax, msg_result
    call print_string

    ; Print result matrix using nested loops
    mov r8, 0           ; row counter
row_loop:
    mov r9, 0           ; column counter
col_loop:
    ; Calculate index = row * cols + column
    mov rax, r8         ; row
    mov rbx, cols
    mul rbx             ; row * cols
    add rax, r9         ; + column

    ; Print element
    mov eax, [result + rax * 4]
    call print_int

    ; Print space (except after last column)
    inc r9
    cmp r9, cols
    jge no_space

    mov al, ' '
    call print_char
    jmp col_loop

no_space:
    ; Print newline after each row
    mov al, 10
    call print_char

    inc r8
    cmp r8, rows
    jl row_loop

    ; Exit program
    mov rax, 60
    xor rdi, rdi
    syscall
```

14.3 Sample Input

```
Enter elements for Matrix 1 (3x3):
1 2 3
4 5 6
7 8 9

Enter elements for Matrix 2 (3x3):
9 8 7
6 5 4
3 2 1
```

14.4 Sample Output

```
Sum of matrices:
10 10 10
10 10 10
10 10 10
```

15 Problem 15: Matrix Subtraction

15.1 Problem Statement

Write an assembly language program that takes two 3X3 matrices as input from the user and performs the matrix subtraction operation among those two matrices.

15.2 Solution

The following NASM 64-bit assembly code implements the following solution:

```



section .data
```

```
    ; Matrix dimensions
    rows equ 3
    cols equ 3
    matrix_size equ rows * cols

    ; Prompt messages
    msg_matrix1 db "Enter elements for Matrix 1 (3x3):", 10, 0
    msg_matrix2 db 10, "Enter elements for Matrix 2 (3x3):", 10, 0
    msg_result db 10, "Difference of matrices (Matrix1 - Matrix2):", 10, 0
    space db " ", 0
    newline db 10, 0

    ; Buffers
    number_buffer times 12 db 0
    input_buffer times 100 db 0  ; Larger buffer for row input

section .bss
    matrix1 resd 9      ; Reserve space for 3x3 matrix (9 doublewords)
    matrix2 resd 9
    result resd 9

section .text
    global _start

; Function to read a row of integers
read_row:
    push rbx
    push rcx
    push rdx
    push rsi
    push rdi
    push r8
    push r9

    ; Read entire line
    mov rax, 0          ; sys_read
    mov rdi, 0          ; stdin
    mov rsi, input_buffer
    mov rdx, 100
    syscall

    mov rsi, input_buffer  ; pointer to input
    mov rdi, r8           ; matrix pointer
    mov r9, 0             ; numbers read counter

parse_loop:
    ; Skip whitespace
    cmp byte [rsi], ’ ’
    je skip_space
    cmp byte [rsi], 9    ; tab
    je skip_space
    cmp byte [rsi], 10   ; newline
    je parse_done
    cmp byte [rsi], 0    ; null terminator
    je parse_done

    ; Convert number
    mov rax, 0
    mov rcx, 0           ; negative flag
    mov bl, byte [rsi]
    cmp bl, '-'
    jne convert_digit
    mov rcx, 1
    inc rsi

convert_digit:
    movzx rbx, byte [rsi]
    cmp rbx, '0'
    jl number_done
    cmp rbx, '9'
    jg number_done
    sub rbx, '0'
    imul rax, 10
    add rax, rbx
    inc rsi
    jmp convert_digit

number_done:
    test rcx, rcx
    jz store_number
    neg rax

store_number:
    mov [rdi], eax      ; store in matrix
    add rdi, 4
    inc r9

    ; Check if we've read enough numbers
    cmp r9, cols
    jge parse_done

    jmp parse_loop

skip_space:
    inc rsi
    jmp parse_loop

parse_done:
    pop r9
    pop r8
    pop rdi
    pop rsi
    pop rdx
    pop rcx
    pop rbx
    ret

; Function to read a matrix
read_matrix:
    push r8
    push r9

    mov r8, rdi        ; matrix pointer
    mov r9, 0          ; row counter

read_row_loop:
    call read_row      ; read one row

    add r8, cols * 4   ; move to next row
    inc r9
    cmp r9, rows
    jl read_row_loop

    pop r9
    pop r8
    ret

; Function to print integer
print_int:
    push rax
    push rbx
    push rcx
    push rdx
    push rsi
    push rdi

    mov rdi, number_buffer + 11
    mov byte [rdi], 0
    mov rbx, 10

    test eax, eax
    jns convert_loop
    neg eax
    push rax
    mov al, '-'
    call print_char
    pop rax

convert_loop:
    xor rdx, rdx
    div rbx
    add dl, ’0’
    dec rdi
    mov [rdi], dl
    test rax, rax
    jnz convert_loop

    ; Print the number
    mov rsi, rdi
    mov rdx, number_buffer + 11
    sub rdx, rsi
    mov rax, 1
    mov rdi, 1
    syscall

    pop rdi
    pop rsi
    pop rdx
    pop rcx
    pop rbx
    pop rax
    ret

; Function to print single character
print_char:
    push rax
    push rdi
    push rsi
    push rdx

    mov [number_buffer], al
    mov rax, 1
    mov rdi, 1
    mov rsi, number_buffer
    mov rdx, 1
    syscall

    pop rdx
    pop rsi
    pop rdi
    pop rax
    ret

; Function to print string
print_string:
    push rax
    push rdi
    push rsi
    push rdx
    push rcx

    mov rsi, rax         ; string pointer
    mov rdx, 0           ; length counter

str_len_loop:
    cmp byte [rsi + rdx], 0
    je str_len_done
    inc rdx
    jmp str_len_loop

str_len_done:
    mov rax, 1           ; sys_write
    mov rdi, 1           ; stdout
    syscall

    pop rcx
    pop rdx
    pop rsi
    pop rdi
    pop rax
    ret

_start:
    ; Read Matrix 1
    mov rax, msg_matrix1
    call print_string
    mov rdi, matrix1
    call read_matrix

    ; Read Matrix 2
    mov rax, msg_matrix2
    call print_string
    mov rdi, matrix2
    call read_matrix

    ; Calculate DIFFERENCE of matrices (Matrix1 - Matrix2)
    mov rcx, 0
subtract_loop:
    mov eax, [matrix1 + rcx * 4]   ; Load from Matrix1
    sub eax, [matrix2 + rcx * 4]   ; Subtract Matrix2
    mov [result + rcx * 4], eax    ; Store result
    inc rcx
    cmp rcx, matrix_size
    jl subtract_loop

    ; Print result message
    mov rax, msg_result
    call print_string

    ; Print result matrix using nested loops
    mov r8, 0           ; row counter
row_loop:
    mov r9, 0           ; column counter
col_loop:
    ; Calculate index = row * cols + column
    mov rax, r8         ; row
    mov rbx, cols
    mul rbx             ; row * cols
    add rax, r9         ; + column

    ; Print element
    mov eax, [result + rax * 4]
    call print_int

    ; Print space (except after last column)
    inc r9
    cmp r9, cols
    jge no_space

    mov al, ' '
    call print_char
    jmp col_loop

no_space:
    ; Print newline after each row
    mov al, 10
    call print_char

    inc r8
    cmp r8, rows
    jl row_loop

    ; Exit program
    mov rax, 60
    xor rdi, rdi
    syscall
```

15.3 Sample Input

```
Enter elements for Matrix 1 (3x3):
10 11 12
13 14 15
16 17 18

Enter elements for Matrix 2 (3x3):
9 8 7
6 5 4
3 2 1
```

15.4 Sample Output

```
Difference of matrices (Matrix1 - Matrix2):
1 3 5
7 9 11
13 15 17
```