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
```

```
92      mov rdx, len_smaller
93      syscall
94
95  end_prog:
96      mov rax, 60          ; sys_exit
97      xor rdi, rdi
98      syscall
99
100 ; ---------------------------------------
101 ; str_to_int: converts string in RSI to integer in RAX
102 ; Handles optional leading '-'
103 ; Stops at newline or null terminator
104 str_to_int:
105     xor rax, rax         ; result = 0
106     xor rcx, rcx
107     mov r8, 1            ; sign = +1
108
109     ; Check for leading minus sign
110     mov al, byte [rsi]
111     cmp al, '-'
112     jne .parse_digits
113     mov r8, -1
114     inc rsi
115
116 .parse_digits:
117     xor rax, rax         ; clear result
118 .next_char:
119     mov cl, byte [rsi]
120     cmp cl, 10           ; newline?
121     je .done
122     cmp cl, 0
123     je .done
124     sub cl, '0'
125     cmp cl, 9
126     ja .done             ; non-digit, stop
127     imul rax, rax, 10
128     movzx rcx, cl        ; zero-extend digit
129     add rax, rcx
130     inc rsi
131     jmp .next_char
132
133 .done:
134     ; Apply sign
135     cmp r8, 1
136     je .ret
137     neg rax
138 .ret:
139     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
```

```
77      syscall
78
79      mov     rax, [num2]
80      call    print_number
81      call    print_newline
82
83      ; Add the two numbers
84      mov     rax, [num1]
85      mov     rbx, [num2]
86      add     rax, rbx
87      mov     [result], rax
88
89      ; Display the complete result message
90      mov     rax, 1          ; sys_write
91      mov     rdi, 1          ; stdout
92      mov     rsi, sum_msg
93      mov     rdx, sum_msg_len
94      syscall
95
96      ; Display first number in result
97      mov     rax, [num1]
98      call    print_number
99
100     mov     rax, 1          ; sys_write
101     mov     rdi, 1          ; stdout
102     mov     rsi, and_msg
103     mov     rdx, and_msg_len
104     syscall
105
106     ; Display second number in result
107     mov     rax, [num2]
108     call    print_number
109
110     mov     rax, 1          ; sys_write
111     mov     rdi, 1          ; stdout
112     mov     rsi, is_msg
113     mov     rdx, is_msg_len
114     syscall
115
116     ; Display the result
117     mov     rax, [result]
118     call    print_number
119     call    print_newline
120
121     ; Exit program
122     mov     rax, 60         ; sys_exit
123     xor     rdi, rdi        ; exit code 0
124     syscall
125
126 ; Function to read a number from stdin
127 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
```

```
179     push    rdx
180
181     mov     rdi, buffer
182     call    int_to_string
183
184     ; Calculate string length
185     mov     rsi, rdi        ; Start of string (returned from
            int_to_string)
186     mov     rdx, buffer
187     add     rdx, 21         ; End of buffer
188     sub     rdx, rsi        ; RDX = length
189
190     ; Write the number
191     mov     rax, 1          ; sys_write
192     mov     rdi, 1          ; stdout
193     syscall
194
195     pop     rdx
196     pop     rsi
197     pop     rdi
198     pop     rax             ; Restore number
199     ret
200
201 ; Function to convert integer to string
202 ; Input: RAX = number
203 ; Output: RDI = pointer to start of string in buffer
204 int_to_string:
205     push    rbx
206     push    rdx
207     push    rsi
208
209     mov     rbx, 10         ; Base 10
210     mov     rdi, buffer
211     add     rdi, 20         ; Point to end of buffer
212     mov     byte [rdi], 0   ; Null terminator
213
214     test    rax, rax        ; Check if number is zero
215     jnz     .not_zero
216     dec     rdi
217     mov     byte [rdi], '0'
218     jmp     .done
219
220 .not_zero:
221     ; Handle positive numbers
222
223 .convert_loop:
224     dec     rdi
225     xor     rdx, rdx        ; Clear RDX for division
226     div     rbx             ; RAX = quotient, RDX = remainder
227     add     dl, '0'         ; Convert to ASCII
228     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
```

```
93      mov rdi, 1
94      mov rsi, outmsg
95      mov rdx, outmsg_len
96      syscall
97
98      ; -------- Print num1 --------
99      mov rax, [num1]
100     mov rsi, numbuf
101     call int_to_string
102     mov rax, 1
103     mov rdi, 1
104     mov rdx, r13          ; r13 = length of string returned
105     syscall
106
107     ; -------- Print space --------
108     mov rax, 1
109     mov rdi, 1
110     mov rsi, newline
111     mov rdx, 1
112     syscall
113
114     ; -------- Print num2 --------
115     mov rax, [num2]
116     mov rsi, numbuf
117     call int_to_string
118     mov rax, 1
119     mov rdi, 1
120     mov rdx, r13
121     syscall
122
123     ; -------- Print newline --------
124     mov rax, 1
125     mov rdi, 1
126     mov rsi, newline
127     mov rdx, 1
128     syscall
129
130     ; -------- Exit --------
131     mov rax, 60
132     xor rdi, rdi
133     syscall
134
135 ; -------- Function: int_to_string --------
136 ; Converts rax integer into decimal string stored at rsi
137 ; Returns length in r13
138 int_to_string:
139     mov rbx, 10           ; divisor
140     xor r12, r12          ; digit counter
141     lea rdi, [rsi+19]     ; point to end of buffer
142     mov byte [rdi], 0     ; null terminator
143 .convert_loop:
```

```
144     xor rdx, rdx
145     div rbx                 ; rax / 10, quotient in rax, remainder
            in rdx
146     add dl, '0'
147     dec rdi
148     mov [rdi], dl
149     inc r12
150     cmp rax, 0
151     jne .convert_loop
152     mov r13, r12            ; length of string
153     mov rsi, rdi            ; pointer to start of string
154     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:

```
1  ; fibo64.asm
2  section .bss
3      inbuf   resb 32
4
5  section .data
6      prompt  db "Enter a number: ",0
7      lenp    equ $-prompt
8      msgfib  db "Fibonacci",10,0
9      lenfib  equ $-msgfib
10     msgnot  db "Not Fibonacci",10,0
11     lennot  equ $-msgnot
12
```

```
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:
```

```
88      xor rdx, rdx
89      div rcx
90      add dl, '0'
91      dec rsi
92      mov [rsi], dl
93      test rax, rax
94      jnz .convert
95
96      mov rax, 1
97      mov rdi, 1
98      mov rdx, rsp
99      add rdx, 32
100     sub rdx, rsi
101     mov rsi, rsi
102     syscall
103
104     mov rsp, rbx
105     ret
106
107 exit:
108     mov rax, 60
109     xor rdi, rdi
110     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
```

```
42      movzx rdx, al
43      sub rdx, '0'
44      add r12, rdx
45      inc rsi
46      jmp .parse
47  .done_parse:
48      mov [num], r12      ; store number
49
50      ; -------- Prime check --------
51      mov rax, [num]      ; rax = number to check
52      cmp rax, 2
53      jb .not_prime       ; numbers less than 2 are not prime
54      je .prime           ; 2 is prime
55
56      mov rbx, 2          ; divisor = 2
57  .check_loop:
58      mov rdx, 0
59      mov rcx, rax
60      div rbx              ; rax / rbx, quotient in rax, remainder
            in rdx
61      cmp rdx, 0
62      je .not_prime         ; divisible -> not prime
63      inc rbx
64      mov rax, [num]
65      cmp rbx, rax
66      jl .check_loop
67
68  .prime:
69      ; print prime_msg
70      mov rax, 1
71      mov rdi, 1
72      mov rsi, prime_msg
73      mov rdx, prime_msg_len
74      syscall
75      jmp .done
76
77  .not_prime:
78      ; print not_prime_msg
79      mov rax, 1
80      mov rdi, 1
81      mov rsi, not_prime_msg
82      mov rdx, not_prime_msg_len
83      syscall
84
85  .done:
86      ; print newline
87      mov rax, 1
88      mov rdi, 1
89      mov rsi, newline
90      mov rdx, 1
91      syscall
```

```
92
93      ; -------- Exit --------
94      mov rax, 60
95      xor rdi, rdi
96      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:

```
1  global _start
2
3  section .data
4  star: db '*'
5  nl:   db 10
6  prompt: db "Enter number of rows: ",0
7  plen: equ $-prompt
8
9  section .bss
10 inbuf: resb 64
```

```
11
12 section .text
13 _start:
14     ; print prompt
15     mov rax, 1
16     mov rdi, 1
17     mov rsi, prompt
18     mov rdx, plen
19     syscall
20
21     ; read input
22     mov rax, 0
23     mov rdi, 0
24     mov rsi, inbuf
25     mov rdx, 64
26     syscall
27
28     ; parse digits
29     xor rax, rax
30     mov rsi, inbuf
31
32 .parse_loop:
33     mov bl, [rsi]
34     cmp bl,'0'
35     jl .parsed
36     cmp bl,'9'
37     jg .parsed
38     imul rax, rax, 10
39     sub bl,'0'
40     movzx rbx, bl
41     add rax, rbx
42     inc rsi
43     jmp .parse_loop
44
45 .parsed:
46     mov r8, rax         ; N safely
47     mov r9, 1           ; outer loop counter i = 1
48
49 .outer_loop:
50     cmp r9, r8
51     jg .exit
52
53     mov rbx, r9         ; inner loop counter j = i
54 .inner_loop:
55     cmp rbx, 0
56     je .newline
57
58     mov rax, 1
59     mov rdi, 1
60     mov rsi, star
61     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
```

```
49
50      mov r13, 1              ; current row = 1
51  row_loop:
52      cmp r13, r12
53      jg  exit
54
55      ; spaces = r12 - r13
56      mov rbx, r12
57      sub rbx, r13
58      call print_spaces
59
60      ; stars = 2*r13 - 1
61      mov rbx, r13
62      shl rbx, 1
63      sub rbx, 1
64      call print_stars
65
66      ; newline
67      mov rax, 1
68      mov rdi, 1
69      mov rsi, newline
70      mov rdx, 1
71      syscall
72
73      inc r13
74      jmp row_loop
75
76  ; rbx = count; prints that many spaces
77  print_spaces:
78      test rbx, rbx
79      jle .ps_done
80  .ps_loop:
81      mov rax, 1
82      mov rdi, 1
83      mov rsi, space
84      mov rdx, 1
85      syscall
86      dec rbx
87      jg  .ps_loop
88  .ps_done:
89      ret
90
91  ; rbx = count; prints that many stars
92  print_stars:
93      test rbx, rbx
94      jle .pt_done
95  .pt_loop:
96      mov rax, 1
97      mov rdi, 1
98      mov rsi, star
99      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:

```
1
2  section .bss
3      num1 resb 20
4      num2 resb 20
5      result resb 20
6
7  section .data
8      prompt1 db "Enter first number: ",0
9      prompt1_len equ $-prompt1
10     prompt2 db "Enter second number: ",0
11     prompt2_len equ $-prompt2
12     newline db 10,0
13
14 section .text
15     global _start
16
17 _start:
18     ; --- Read first number ---
19     mov rax, 1
20     mov rdi, 1
21     mov rsi, prompt1
22     mov rdx, prompt1_len
23     syscall
24
25     mov rax, 0
26     mov rdi, 0
27     mov rsi, num1
28     mov rdx, 20
29     syscall
30
31     mov rsi, num1
32     call str2int
33     mov rbx, r8            ; store first number
34
35     ; --- Read second number ---
36     mov rax, 1
37     mov rdi, 1
38     mov rsi, prompt2
39     mov rdx, prompt2_len
40     syscall
41
```

```
    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
```

```
93
94      ; Exit
95      mov rax, 60
96      xor rdi, rdi
97      syscall
98
99  ; ------------------------------
100 ; Convert string to integer (r8)
101 ; Input: rsi = pointer to string
102 ; Output: r8 = integer
103 str2int:
104     xor r8, r8
105 .next_digit:
106     mov al, byte [rsi]
107     cmp al, 10
108     je .done
109     sub al, '0'
110     imul r8, r8, 10
111     add r8, rax
112     inc rsi
113     jmp .next_digit
114 .done:
115     ret
116
117 ; ------------------------------
118 ; Convert integer in r8 to string at rdi
119 int2str:
120     mov rax, r8
121     mov rcx, 0
122     mov rbx, 10
123 .reverse_loop:
124     xor rdx, rdx
125     div rbx
126     add dl, '0'
127     push rdx
128     inc rcx
129     cmp rax, 0
130     jne .reverse_loop
131
132 .print_loop:
133     pop rax
134     mov [rdi], al
135     inc rdi
136     dec rcx
137     cmp rcx, 0
138     jne .print_loop
139     mov byte [rdi], 0
140     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:

```
1
2  ; NASM x86-64 Assembly - Average of 3 numbers
3  ; Assemble: nasm -f elf64 avg.asm -o avg.o
4  ; Link: ld avg.o -o avg
5  ; Run: ./avg
6
7  section .bss
8      num1 resb 10
9      num2 resb 10
10     num3 resb 10
11     result resb 10
12
13 section .data
14     prompt1 db "Enter first number: ", 0
15     prompt1_len equ $-prompt1
16     prompt2 db "Enter second number: ", 0
17     prompt2_len equ $-prompt2
18     prompt3 db "Enter third number: ", 0
19     prompt3_len equ $-prompt3
20     out_msg db "Average is: ", 0
21     out_len equ $-out_msg
22
23 section .text
24     global _start
25
26 _start:
```

```
27      ; --- Read first number ---
28      mov rax, 1          ; sys_write
29      mov rdi, 1          ; stdout
30      mov rsi, prompt1
31      mov rdx, prompt1_len
32      syscall
33
34      mov rax, 0          ; sys_read
35      mov rdi, 0          ; stdin
36      mov rsi, num1
37      mov rdx, 10
38      syscall
39
40      call str2int
41      mov rbx, rax        ; store first number in rbx
42
43      ; --- Read second number ---
44      mov rax, 1
45      mov rdi, 1
46      mov rsi, prompt2
47      mov rdx, prompt2_len
48      syscall
49
50      mov rax, 0
51      mov rdi, 0
52      mov rsi, num2
53      mov rdx, 10
54      syscall
55
56      call str2int
57      add rbx, rax        ; sum += second number
58
59      ; --- Read third number ---
60      mov rax, 1
61      mov rdi, 1
62      mov rsi, prompt3
63      mov rdx, prompt3_len
64      syscall
65
66      mov rax, 0
67      mov rdi, 0
68      mov rsi, num3
69      mov rdx, 10
70      syscall
71
72      call str2int
73      add rbx, rax        ; sum += third number
74
75      ; --- Calculate average ---
76      mov rax, rbx
77      mov rcx, 3
```

```
78      cqo                     ; extend rax to rdx:rax
79      idiv rcx                ; rax = rax / 3, remainder in rdx
80
81      ; --- Convert integer to string ---
82      mov rdi, result
83      call int2str
84
85      ; --- Print output ---
86      mov rax, 1
87      mov rdi, 1
88      mov rsi, out_msg
89      mov rdx, out_len
90      syscall
91
92      mov rax, 1
93      mov rdi, 1
94      mov rsi, result
95      mov rdx, 10
96      syscall
97
98      ; --- Exit ---
99      mov rax, 60
100     xor rdi, rdi
101     syscall
102
103 ; --- Subroutine: string to integer ---
104 str2int:
105     xor rax, rax
106     xor rcx, rcx
107 .next_char:
108     mov cl, byte [rsi]
109     cmp cl, 10          ; newline
110     je .done
111     cmp cl, 0
112     je .done
113     sub cl, '0'
114     imul rax, rax, 10
115     add rax, rcx
116     inc rsi
117     jmp .next_char
118 .done:
119     ret
120
121 ; --- Subroutine: integer to string ---
122 int2str:
123     mov rbx, 10
124     xor rcx, rcx          ; digit counter
125 .next_digit:
126     xor rdx, rdx
127     div rbx
128     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
```

```
77      jmp convert_digit
78
79  number_done:
80      test rcx, rcx
81      jz store_number
82      neg rax
83
84  store_number:
85      mov [rdi], eax      ; store in matrix
86      add rdi, 4
87      inc r9
88
89      ; Check if we've read enough numbers
90      cmp r9, cols
91      jge parse_done
92
93      jmp parse_loop
94
95  skip_space:
96      inc rsi
97      jmp parse_loop
98
99  parse_done:
100     pop r9
101     pop r8
102     pop rdi
103     pop rsi
104     pop rdx
105     pop rcx
106     pop rbx
107     ret
108
109 ; Function to read a matrix
110 read_matrix:
111     push r8
112     push r9
113
114     mov r8, rdi         ; matrix pointer
115     mov r9, 0           ; row counter
116
117 read_row_loop:
118     call read_row       ; read one row
119
120     add r8, cols * 4    ; move to next row
121     inc r9
122     cmp r9, rows
123     jl read_row_loop
124
125     pop r9
126     pop r8
127     ret
```

```
128
129 ; Function to print integer
130 print_int:
131     push rax
132     push rbx
133     push rcx
134     push rdx
135     push rsi
136     push rdi
137
138     mov rdi, number_buffer + 11
139     mov byte [rdi], 0
140     mov rbx, 10
141
142     test eax, eax
143     jns convert_loop
144     neg eax
145     push rax
146     mov al, '-'
147     call print_char
148     pop rax
149
150 convert_loop:
151     xor rdx, rdx
152     div rbx
153     add dl, '0'
154     dec rdi
155     mov [rdi], dl
156     test rax, rax
157     jnz convert_loop
158
159     ; Print the number
160     mov rsi, rdi
161     mov rdx, number_buffer + 11
162     sub rdx, rsi
163     mov rax, 1
164     mov rdi, 1
165     syscall
166
167     pop rdi
168     pop rsi
169     pop rdx
170     pop rcx
171     pop rbx
172     pop rax
173     ret
174
175 ; Function to print single character
176 print_char:
177     push rax
178     push rdi
```

```
179     push rsi
180     push rdx
181
182     mov [number_buffer], al
183     mov rax, 1
184     mov rdi, 1
185     mov rsi, number_buffer
186     mov rdx, 1
187     syscall
188
189     pop rdx
190     pop rsi
191     pop rdi
192     pop rax
193     ret
194
195 ; Function to print string
196 print_string:
197     push rax
198     push rdi
199     push rsi
200     push rdx
201     push rcx
202
203     mov rsi, rax        ; string pointer
204     mov rdx, 0          ; length counter
205
206 str_len_loop:
207     cmp byte [rsi + rdx], 0
208     je str_len_done
209     inc rdx
210     jmp str_len_loop
211
212 str_len_done:
213     mov rax, 1          ; sys_write
214     mov rdi, 1          ; stdout
215     syscall
216
217     pop rcx
218     pop rdx
219     pop rsi
220     pop rdi
221     pop rax
222     ret
223
224 _start:
225     ; Read Matrix 1
226     mov rax, msg_matrix1
227     call print_string
228     mov rdi, matrix1
229     call read_matrix
```

```
230
231     ; Read Matrix 2
232     mov rax, msg_matrix2
233     call print_string
234     mov rdi, matrix2
235     call read_matrix
236
237     ; Calculate sum of matrices
238     mov rcx, 0
239 sum_loop:
240     mov eax, [matrix1 + rcx * 4]
241     add eax, [matrix2 + rcx * 4]
242     mov [result + rcx * 4], eax
243     inc rcx
244     cmp rcx, matrix_size
245     jl sum_loop
246
247     ; Print result message
248     mov rax, msg_result
249     call print_string
250
251     ; Print result matrix using nested loops
252     mov r8, 0           ; row counter
253 row_loop:
254     mov r9, 0           ; column counter
255 col_loop:
256     ; Calculate index = row * cols + column
257     mov rax, r8         ; row
258     mov rbx, cols
259     mul rbx             ; row * cols
260     add rax, r9         ; + column
261
262     ; Print element
263     mov eax, [result + rax * 4]
264     call print_int
265
266     ; Print space (except after last column)
267     inc r9
268     cmp r9, cols
269     jge no_space
270
271     mov al, ' '
272     call print_char
273     jmp col_loop
274
275 no_space:
276     ; Print newline after each row
277     mov al, 10
278     call print_char
279
280     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
```

```
5       ; Matrix dimensions
6       rows equ 3
7       cols equ 3
8       matrix_size equ rows * cols
9
10      ; Prompt messages
11      msg_matrix1 db "Enter elements for Matrix 1 (3x3):", 10, 0
12      msg_matrix2 db 10, "Enter elements for Matrix 2 (3x3):", 10,
            0
13      msg_result db 10, "Difference of matrices (Matrix1 -
            Matrix2):", 10, 0
14      space db " ", 0
15      newline db 10, 0
16
17      ; Buffers
18      number_buffer times 12 db 0
19      input_buffer times 100 db 0  ; Larger buffer for row input
20
21  section .bss
22      matrix1 resd 9      ; Reserve space for 3x3 matrix (9
            doublewords)
23      matrix2 resd 9
24      result resd 9
25
26  section .text
27      global _start
28
29  ; Function to read a row of integers
30  read_row:
31      push rbx
32      push rcx
33      push rdx
34      push rsi
35      push rdi
36      push r8
37      push r9
38
39      ; Read entire line
40      mov rax, 0          ; sys_read
41      mov rdi, 0          ; stdin
42      mov rsi, input_buffer
43      mov rdx, 100
44      syscall
45
46      mov rsi, input_buffer  ; pointer to input
47      mov rdi, r8           ; matrix pointer
48      mov r9, 0             ; numbers read counter
49
50  parse_loop:
51      ; Skip whitespace
52      cmp byte [rsi], ' '
```

```
53      je skip_space
54      cmp byte [rsi], 9     ; tab
55      je skip_space
56      cmp byte [rsi], 10    ; newline
57      je parse_done
58      cmp byte [rsi], 0     ; null terminator
59      je parse_done
60
61      ; Convert number
62      mov rax, 0
63      mov rcx, 0            ; negative flag
64      mov bl, byte [rsi]
65      cmp bl, '-'
66      jne convert_digit
67      mov rcx, 1
68      inc rsi
69
70  convert_digit:
71      movzx rbx, byte [rsi]
72      cmp rbx, '0'
73      jl number_done
74      cmp rbx, '9'
75      jg number_done
76      sub rbx, '0'
77      imul rax, 10
78      add rax, rbx
79      inc rsi
80      jmp convert_digit
81
82  number_done:
83      test rcx, rcx
84      jz store_number
85      neg rax
86
87  store_number:
88      mov [rdi], eax        ; store in matrix
89      add rdi, 4
90      inc r9
91
92      ; Check if we've read enough numbers
93      cmp r9, cols
94      jge parse_done
95
96      jmp parse_loop
97
98  skip_space:
99      inc rsi
100     jmp parse_loop
101
102 parse_done:
103     pop r9
```

```
104     pop r8
105     pop rdi
106     pop rsi
107     pop rdx
108     pop rcx
109     pop rbx
110     ret
111 
112 ; Function to read a matrix
113 read_matrix:
114     push r8
115     push r9
116 
117     mov r8, rdi         ; matrix pointer
118     mov r9, 0           ; row counter
119 
120 read_row_loop:
121     call read_row       ; read one row
122 
123     add r8, cols * 4    ; move to next row
124     inc r9
125     cmp r9, rows
126     jl read_row_loop
127 
128     pop r9
129     pop r8
130     ret
131 
132 ; Function to print integer
133 print_int:
134     push rax
135     push rbx
136     push rcx
137     push rdx
138     push rsi
139     push rdi
140 
141     mov rdi, number_buffer + 11
142     mov byte [rdi], 0
143     mov rbx, 10
144 
145     test eax, eax
146     jns convert_loop
147     neg eax
148     push rax
149     mov al, '-'
150     call print_char
151     pop rax
152 
153 convert_loop:
154     xor rdx, rdx
```

```
155     div rbx
156     add dl, '0'
157     dec rdi
158     mov [rdi], dl
159     test rax, rax
160     jnz convert_loop
161 
162     ; Print the number
163     mov rsi, rdi
164     mov rdx, number_buffer + 11
165     sub rdx, rsi
166     mov rax, 1
167     mov rdi, 1
168     syscall
169 
170     pop rdi
171     pop rsi
172     pop rdx
173     pop rcx
174     pop rbx
175     pop rax
176     ret
177 
178 ; Function to print single character
179 print_char:
180     push rax
181     push rdi
182     push rsi
183     push rdx
184 
185     mov [number_buffer], al
186     mov rax, 1
187     mov rdi, 1
188     mov rsi, number_buffer
189     mov rdx, 1
190     syscall
191 
192     pop rdx
193     pop rsi
194     pop rdi
195     pop rax
196     ret
197 
198 ; Function to print string
199 print_string:
200     push rax
201     push rdi
202     push rsi
203     push rdx
204     push rcx
205 
```

```
206     mov rsi, rax        ; string pointer
207     mov rdx, 0          ; length counter
208
209 str_len_loop:
210     cmp byte [rsi + rdx], 0
211     je str_len_done
212     inc rdx
213     jmp str_len_loop
214
215 str_len_done:
216     mov rax, 1          ; sys_write
217     mov rdi, 1          ; stdout
218     syscall
219
220     pop rcx
221     pop rdx
222     pop rsi
223     pop rdi
224     pop rax
225     ret
226
227 _start:
228     ; Read Matrix 1
229     mov rax, msg_matrix1
230     call print_string
231     mov rdi, matrix1
232     call read_matrix
233
234     ; Read Matrix 2
235     mov rax, msg_matrix2
236     call print_string
237     mov rdi, matrix2
238     call read_matrix
239
240     ; Calculate DIFFERENCE of matrices (Matrix1 - Matrix2)
241     mov rcx, 0
242 subtract_loop:
243     mov eax, [matrix1 + rcx * 4]  ; Load from Matrix1
244     sub eax, [matrix2 + rcx * 4]  ; Subtract Matrix2
245     mov [result + rcx * 4], eax   ; Store result
246     inc rcx
247     cmp rcx, matrix_size
248     jl subtract_loop
249
250     ; Print result message
251     mov rax, msg_result
252     call print_string
253
254     ; Print result matrix using nested loops
255     mov r8, 0           ; row counter
256 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
```