**COAL ASSIGNMENT # 2**  
**Q1)**  
Defines a stack using a block of memory (stack\_space). It then provides simple procedures (push, pop, and print\_stack) to perform stack operations. The program pushes two elements onto the stack, prints the stack, pops an element, and prints the updated stack. Finally, it exits the program.  
  
**CODE;**  
  
section .data

stack\_space resb 100

section .text

global \_start

\_start:

mov eax, 10

call push

mov eax, 20

call push

call print\_stack

call pop

call print\_stack

mov eax, 1

xor ebx, ebx

int 0x80

push:

push eax

ret

pop:

pop eax

ret

print\_stack:

mov eax, 4

mov ebx, 1

mov ecx, stack\_space

mov edx, 100

int 0x80

mov eax, 4

mov ebx, 1

mov ecx, newline

mov edx, 1

int 0x80

ret

section .bss

newline resb 1   
  
  
**Q2)**   
This code defines three procedures (print\_space, print\_star, and print\_newline) to print a space, a star, and a newline character, respectively.  
  
**CODE;**section .data

pyramid\_height equ 5

section .text

global \_start

\_start:

mov ecx, pyramid\_height

outer\_loop:

mov ebx, ecx

mov eax, ecx

inner\_loop\_spaces:

cmp eax, ebx

jge print\_stars

call print\_space

inc eax

jmp inner\_loop\_spaces

print\_stars:

mov eax, ecx

inner\_loop\_stars:

cmp eax, pyramid\_height

jge newline

; Print a star

call print\_star

inc eax

jmp inner\_loop\_stars

newline:

call print\_newline

dec ecx

cmp ecx, 0

jg outer\_loop

; Exit the program

mov eax, 1

xor ebx, ebx ; status: 0

int 0x80

print\_space:

mov eax, 4

mov ebx, 1

mov ecx, space

mov edx, 1

int 0x80

ret

print\_star:

mov eax, 4

mov ebx, 1

mov ecx, star

mov edx, 1

int 0x80

ret

print\_newline:

mov eax, 4

mov ebx, 1

mov ecx, newline

mov edx, 1

int 0x80

ret

section .bss

space resb 1

star resb 1

newline resb 1

**Q3)**This program calculates and prints the Fibonacci series from 1 to 20. Adjust the ecx register in the mov ecx, 20 line to control the number of Fibonacci numbers to print  
  
**CODE;**

section .data

newline db 0xA

section .text

global \_start

\_start:

mov eax, 1

mov ebx, 0

mov ecx, 20

fibonacci\_loop:

call print\_number

add eax, ebx

mov ebx, eax

cmp ecx, 1

je last\_number

call print\_comma\_space

last\_number:

dec ecx

cmp ecx, 0

jg fibonacci\_loop

mov eax, 1

xor ebx, ebx

int 0x80

print\_number:

mov eax, ebx

call print\_integer

call print\_newline

ret

print\_integer:

mov ecx, 10

mov edi, 0

.repeat:

xor edx, edx

div ecx

add dl, '0'

mov [buffer + edi], dl

inc edi

test eax, eax

jnz .repeat

.print\_digits:

dec edi

cmp edi, 0

jl .end\_print

mov eax, 4

mov ebx, 1

lea ecx, [buffer + edi]

mov edx, 1

int 0x80

jmp .print\_digits

.end\_print:

ret

print\_comma\_space:

mov eax, 4

mov ebx, 1

mov ecx, comma\_space

mov edx, 2

int 0x80

ret

print\_newline:

mov eax, 4

mov ebx, 1

mov ecx, newline

mov edx, 1

int 0x80

ret

section .bss

buffer resb 11

comma\_space db ',', ' '  
  
  
**Q4)**This program takes a character as input, checks if it is a vowel or consonant, and prints the result accordingly. The input character is read from the user, and the program then checks each character against the vowels.  
**CODE;**

section .data

prompt db 'Enter a character: ', 0

result\_vowel db 'The entered character is a vowel.', 0

result\_consonant db 'The entered character is a consonant.', 0

result\_invalid db 'Invalid input. Please enter a valid character.', 0

newline db 0xA

section .bss

char resb 1

section .text

global \_start

\_start:

mov eax, 4

mov ebx, 1

mov ecx, prompt

mov edx, 23

int 0x80

mov eax, 3

mov ebx, 0

mov ecx, char

mov edx, 1

int 0x80

mov al, [char]

call is\_vowel

test al, al

jnz print\_vowel

call is\_consonant

test al, al

jnz print\_consonant

mov eax, 4

mov ebx, 1

mov ecx, result\_invalid

mov edx, 44

int 0x80

mov eax, 1

xor ebx, ebx

int 0x80

print\_vowel:

mov eax, 4

mov ebx, 1

mov ecx, result\_vowel

mov edx, 35

int 0x80

mov eax, 1

xor ebx, ebx

int 0x80

print\_consonant:

mov eax, 4

mov ebx, 1

mov ecx, result\_consonant

mov edx, 47

int 0x80

mov eax, 1

xor ebx, ebx

int 0x80

is\_vowel:

cmp al, 'a'

je is\_vowel\_found

cmp al, 'e'

je is\_vowel\_found

cmp al, 'i'

je is\_vowel\_found

cmp al, 'o'

je is\_vowel\_found

cmp al, 'u'

je is\_vowel\_found

cmp al, 'A'

je is\_vowel\_found

cmp al, 'E'

je is\_vowel\_found

cmp al, 'I'

je is\_vowel\_found

cmp al, 'O'

je is\_vowel\_found

cmp al, 'U'

je is\_vowel\_found

jmp is\_not\_vowel

is\_vowel\_found:

mov al, 1

ret

is\_not\_vowel:

xor al, al

ret

is\_consonant:

call is\_vowel

test al, al

jnz is\_not\_consonant

cmp al, 0

jz is\_consonant\_found

is\_not\_consonant:

xor al, al

ret

is\_consonant\_found:

mov al, 1

ret

**Q5)**This program reads a non-negative integer from the user, calculates its factorial, and then displays the result. The ascii\_to\_int and int\_to\_ascii subroutines are used for converting between ASCII and integer representations.  
  
**CODE;**section .data

prompt db 'Enter a non-negative integer: ', 0

result\_format db 'The factorial of %d is: %d', 0

newline db 0xA

section .bss

input resd 1 ; Reserve space for the input

result resd 1 ; Reserve space for the result

section .text

global \_start

\_start:

mov eax, 4

mov ebx, 1

mov ecx, prompt

mov edx, 31

int 0x80

mov eax, 3

mov ebx, 0

mov ecx, input

mov edx, 4

int 0x80

mov eax, input

call ascii\_to\_int

mov ebx, eax

mov eax, 1

factorial\_loop:

imul eax, eax, ebx

dec ebx

cmp ebx, 0

jg factorial\_loop

mov ebx, eax

mov eax, 0

mov ecx, result

mov edx, 37

call int\_to\_ascii

add ecx, 8

mov byte [ecx], 0

mov eax, 4

mov ebx, 1

mov edx, 36

int 0x80

mov eax, 1

xor ebx, ebx

int 0x80

ascii\_to\_int:

xor ebx, ebx

xor ecx, ecx

.convert\_loop:

movzx edx, byte [eax + ecx]

sub edx, '0'

imul ebx, ebx, 10

add ebx, edx

inc ecx

cmp ecx, 4

jl .convert\_loop

ret

int\_to\_ascii:

mov ecx, 10

mov edi, 0

.convert\_loop:

xor edx, edx

div ecx

add dl, '0'

dec ecx

mov [edi + ecx], dl

test eax, eax

jnz .convert\_loop

ret  
  
**Q6)**

This program finds and displays the maximum and minimum values within the given array. The int\_to\_ascii subroutine is used to convert integers to ASCII digits for display.  
  
**CODE;**section .data

arr1 dd 10, 2, 8, 9, 26

arr\_size equ ($ - arr1) / 4

max\_format db 'Maximum value in the array: %d', 0

min\_format db 'Minimum value in the array: %d', 0

newline db 0xA

section .text

global \_start

\_start:

mov eax, [arr1]

mov ebx, 1

mov ecx, arr\_size

find\_max\_loop:

cmp eax, [arr1 + ebx \* 4]

jge skip\_update\_max

mov eax, [arr1 + ebx \* 4]

skip\_update\_max:

inc ebx

loop find\_max\_loop

mov ebx, eax

mov eax, 0

mov ecx, max\_format

mov edx, 35

call int\_to\_ascii

add ecx, 8

mov byte [ecx], 0

mov eax, 4

mov ebx, 1

mov edx, 34

int 0x80

mov eax, [arr1]

mov ebx, 1

mov ecx, arr\_size

find\_min\_loop:

cmp eax, [arr1 + ebx \* 4]

jle skip\_update\_min

mov eax, [arr1 + ebx \* 4]

skip\_update\_min:

inc ebx

loop find\_min\_loop

mov ebx, eax

mov eax, 0

mov ecx, min\_format

mov edx, 35

call int\_to\_ascii

add ecx, 8

mov byte [ecx], 0

mov eax, 4

mov ebx, 1

mov edx, 34

int 0x80

mov eax, 1

xor ebx, ebx

int 0x80

int\_to\_ascii:

mov ecx, 10

mov edi, 0

.convert\_loop:

xor edx, edx

div ecx

add dl, '0'

dec ecx

mov [edi + ecx], dl

test eax, eax

jnz .convert\_loop

ret