# NAME: IMRAN KHAN DEPARTMENT : COMPUTER SCIENCE

# ROLL NO:1802035

# SUBMITTED TO: SIR SAAD

Q:NO:1:-

DIFFERENCE IN EACH MODE OPERATION :-

SOLUTION:

|  |  |
| --- | --- |
| PROTECTED MODE | REAL MODE |
| In Protected mode CPU allocates a certain memory to a specific program and the program cannot address to a memory location other  than allowed  to it by CPU. | In Real mode every program is an administrator that can access all the memory. |
| The protected mode allows 4GB and more than 4GB of memory locations. | The real mode can address upto 1MB of addressable memory. |
| A sub-mode of protected mode is virtual-8086, that is its special case. | Real mode does not allow multitasking and paging that is very important in memory management |
| All instructions and features are available in protected mode. | This mode is useful if a program requires direct access to system memory and hardware devices. |
|  |  |

SYSTEM MANAGEMENT MODE**:-**

System management mode (SMM) provides an operating system with a mechanism for implementing functions such as power management and system security. These functions are usually implemented by computer manufacturers who customize the processor for a particular system setup.

QNO:2:-

**Q.2.** Symbolic Integer Constants Write a program that defines symbolic constants for all seven days of the week. Create an array of variable that uses the symbols as initializers?

|  |  |
| --- | --- |
| **1** | .386 |
| **2** | **.** model flat, stdcall |
| **3** | .stack 4096 |
| **4** | ExitProcess proto, dwExitCode:dWord |
| **5** | .data |
| **6** | Sun=0 |
| **7** | Mon=1 |
| **8** | Tue=2 |
| **9** | Wed=3 |
| **10** | Thu=4 |
| **11** | Fri=5 |
| **12** | Sat=6 |
| **13** | Array Byte Sun,Mon,Tue,Wed,Thu,Fri,Sat  Mov eax,offset Array; |
| **14** | .code |
| **15** | Main proc |
| **16** | Invoke ExitProcess,0 |
| **17** | Main endp |
| **18** | End main |

FLOW CHART:-

MOVE OFFSET ARRAY

END

STORE IN ARRAY

ASSIGN CONSTANT

START

**Q.3. Listing File for AddTwoSum Generate a listing file for the AddTwoSum program and write a description of the machine code bytes generated for each instruction. You might have to guess at some of the meanings of the byte values?**

***Microsoft (R) Macro Assembler Version 14.16.27034.0 10/25/19 04:34:10***

***Source.asm Page 1 - 1***

***; AddTwoSum.asm - Chapter 3 example.***

***.386***

***.model flat,stdcall***

***.stack 4096***

***ExitProcess proto,dwExitCode:dword***

***00000000 .data***

***00000000 00000000 sum dword 0***

***00000000 .code***

***00000000 main proc***

***00000000 B8 00000005 mov eax,5***

***00000005 83 C0 06 add eax,6***

***00000008 A3 00000000 R mov sum,eax***

***invoke ExitProcess,0***

***00000014 main endp***

***end main***

***Microsoft (R) Macro Assembler Version 14.16.27034.0 10/25/19 04:34:10***

***Source.asm Symbols 2 - 1***

***Segments and Groups:***

***N a m e Size Length Align Combine Class***

***FLAT . . . . . . . . . . . . . . GROUP***

***STACK . . . . . . . . . . . . . 32 Bit 00001000 DWord Stack 'STACK'***

***\_DATA . . . . . . . . . . . . . 32 Bit 00000004 DWord Public 'DATA'***

***\_TEXT . . . . . . . . . . . . . 32 Bit 00000014 DWord Public 'CODE'***

***Procedures, parameters, and locals:***

***N a m e Type Value Attr***

***ExitProcess . . . . . . . . . . P Near 00000000 FLAT Length= 00000000 External STDCALL***

***main . . . . . . . . . . . . . . P Near 00000000 \_TEXT Length= 00000014 Public STDCALL***

***Symbols:***

***N a m e Type Value Attr***

***@CodeSize . . . . . . . . . . . Number 00000000h***

***@DataSize . . . . . . . . . . . Number 00000000h***

***@Interface . . . . . . . . . . . Number 00000003h***

***@Model . . . . . . . . . . . . . Number 00000007h***

***@code . . . . . . . . . . . . . Text \_TEXT***

***@data . . . . . . . . . . . . . Text FLAT***

***@fardata? . . . . . . . . . . . Text FLAT***

***@fardata . . . . . . . . . . . . Text FLAT***

***@stack . . . . . . . . . . . . . Text FLAT***

***sum . . . . . . . . . . . . . . DWord 00000000 \_DATA***

***0 Warnings***

***0 Errors***

## DESCRIPTON:-

***First 7 lines contains no executable instruction so compiler directly compiled by assembler.***

***Line 9 indicates the starting of code.***

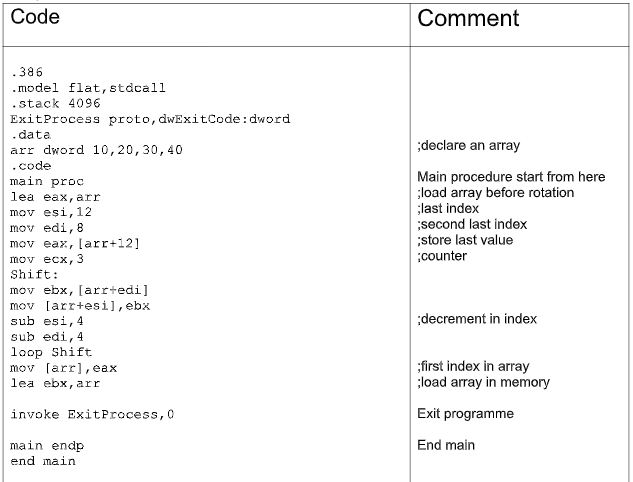
***Line 11 contains first executable command”mov”.***

***Line 12 contains also executable command having an offset 0000005,which has 5 byte distance from starting of programme.***

***Line 14 contain invoke directive.***

**Line 15-16**: These two lines is inserted into our code. Because the INVOKE directive causes the assembler to generate the PUSH and CALL statements shown on lines 15 and 16.

**Q.4. Using a loop and indexed addressing, write code that rotates the members of a 32-bit integer array forward one position. The value at the end of the array must wrap around to the first position. For example, the array [10,20,30,40] would be transformed into [40,10,20,30]?**

******

FALSE

TRUE

SET LAST IN FIRST PLACE OF ARRAY

IF ECX=0

END

START

OVE OFFSET TO ESI

GET ADDRESS OF @ND IN ESI

**Q.5. Write a procedure that produces N values in the Fibonacci number series and stores them in an array of doubleword. Input parameters should be a pointer to an array of doubleword, a counter of the number of values to generate. Write a test program that calls your procedure, passing N = 47. The first value in the array will be 1, and the last value will be 2,971,215,073?**

.386

.model flat,stdcall

.stack 4096

ExitProcess proto,dwExitCode:dword

.data

N dword 47

array dword 47 dup(?)

counter dword 0

sum dword 0

.code

main proc

mov ecx, N

call generate\_Fabo

lea eax, myarray

invoke ExitProcess,0

main endp

generate\_Fabonachi proc

pushad

mov eax, 0

mov ebx, 1

mov esi, offset array

Loop1:

add eax, ebx

mov sum, eax

mov [esi], eax

add esi, 4

inc counter

mov eax, ebx

mov ebx, sum

loop loop1

popad

ret

generate\_Fabonachi endp

end main

**Result:**

In this code we are generating a fabonacii sequence and storing them in array. We pass the NUMBER(counter) and a base address into the function and as a result it will produce a Fabonaci sequence. Below is a screenshot of memory in which all the numbers is displayed.

You can increase or decrease the value by changing the input number(N).

FLOWCHART:-

END

END

GENERATING FABINACHI SEQUENCE

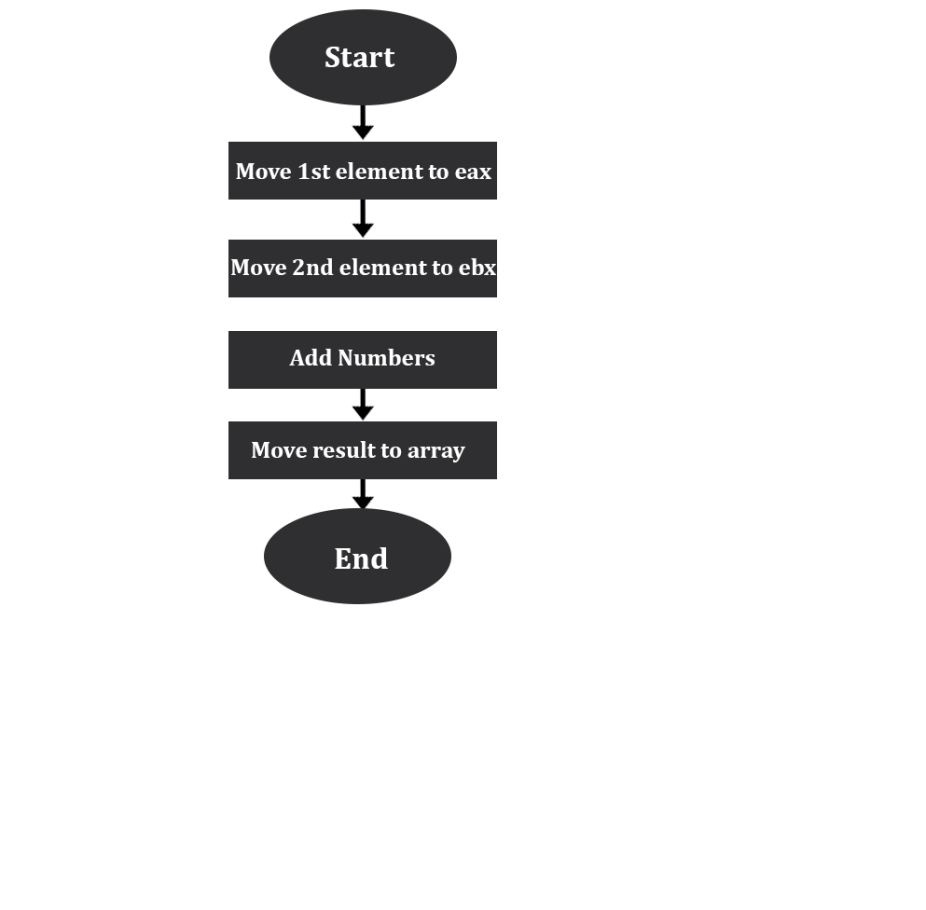
IF ECX=0

STORE COUNTS

STORE OFFSET IN ESI REGISTER

STRAT

Sub procedure flowchart:-

****

**Q.6. Read the section “6.11.1 Suggestions for Testing Your Code” carefully and attempt the following programming exercises accordingly?**

and ax,00FFh

or ax,0FF00h

xor eax,0FFFFFFFFh

test eax,1 ; (low bit set if eax is odd)

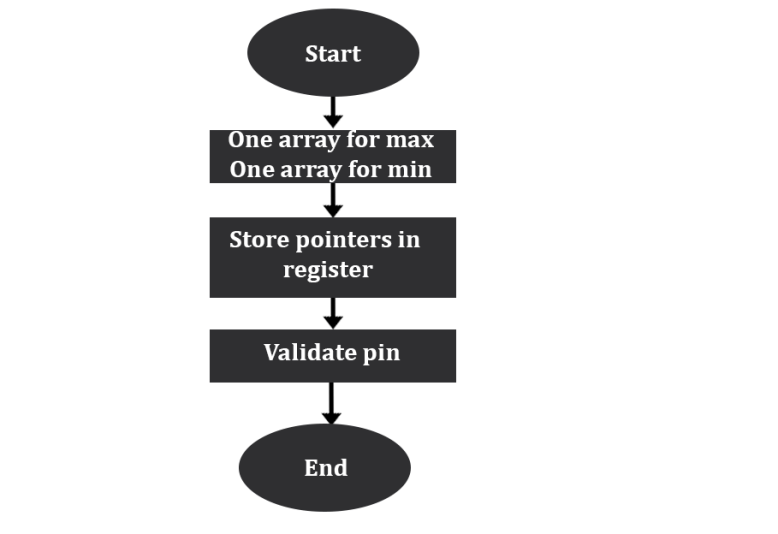
or al,00100000b

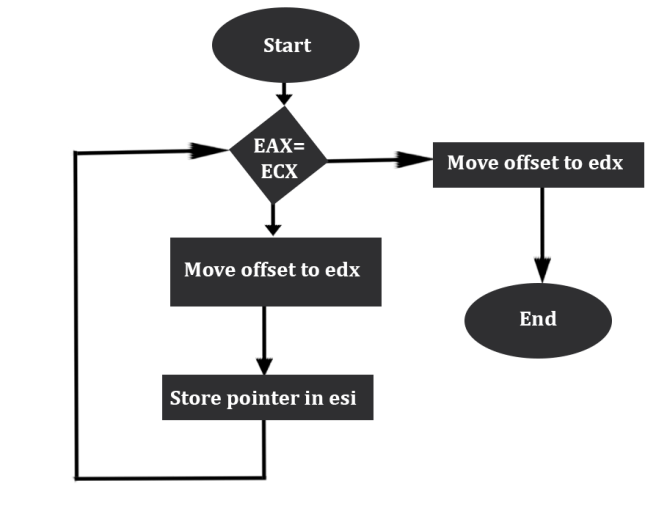
**Q.7. Banks use a Personal Identification Number (PIN) to uniquely identify each customer. Let us assume that our bank has a specified range of acceptable values for each digit in its customers’ 5-digit PINs. The table shown below contains the acceptable ranges, where digits are numbered from left to right in the PIN. Then we can see that the PIN 52413 is valid. But the PIN 43534 is invalid because the first digit is out of range. Similarly, 64535 is invalid because of its last digit. Your task is to create a procedure named Validate\_PIN that receives a pointer to an array of byte containing a 5-digit PIN. Declare two arrays to hold the minimum and maximum range values, and use these arrays to validate each digit of the PIN that was passed to the procedure. If any digit is found to be outside its valid range, immediately return the digit’s position (between 1 and 5) in the EAX register. If the entire PIN is valid, return 0 in EAX. Preserve all other register values between calls to the procedure. Write a test program that calls Validate\_PIN at least four times, using both valid and invalid byte arrays. By running the program in a debugger, verify that the return value in EAX after each procedure call is valid. Or, if you prefer to use the book’s library, you can display "Valid" or "Invalid" on the console after each procedure call.**

|  |
| --- |
| **; Task no 7**  **include c:\\Irvine\irvine32.inc**  **.386**  **.model flat,stdcall**  **.stack 4096**  **ExitProcess proto,dwExitCode:dword**  **.data**  **user\_array byte 5,2,4,1,3 ;User Array**  **min byte 5,2,4,1,3 ;Min values array**  **max byte 9,5,8,4,6 ;Max values array**  **.code**  **main proc**  **mov ebp, 0**  **mov eax, 0**  **mov edx, 0**  **call Validate\_PIN ;you can pass any array**  **mov eax, 0**  **invoke ExitProcess,0**  **main endp**  **;----------PROC-----------;**  **;------Validate\_PIN ------;**  **;-------------------------;**  **Validate\_PIN PROC**  **mov ecx, 5 ;This Loop will call the function 5 times**  **L1:**  **mov al, byte ptr [user\_array+edx]**  **mov bl, min[ebp] ;This will check Lower limit**  **cmp al, bl**  **jl stop**  **mov bl, max[ebp] ;This will check Higher limit**  **cmp al,bl**  **jg stop**  **inc ebp**  **inc edx**  **mov eax, 0**  **loop L1**  **ret**  **stop: ; If codition is vielated**  **mov eax, 1 ; It will jump to stop**  **ret**  **Validate\_PIN endp**  **end main** |

Result:

In this program, we decide the PIN is correct or incorrect. We are passing an array that contains a PIN and we have two arrays containing the lower and higher limits of PIN validity. If any of the digits fall out of the range then it returns “0” immediately.

****

****

**Q.8. Encryption Using Rotate Operations**

**Write a procedure that performs simple encryption by rotating each plaintext byte a varying**

**number of positions in different directions. For example, in the following array that represents**

**the encryption key, a negative value indicates a rotation to the left and a positive value indicates**

**a rotation to the right. The integer in each position indicates the magnitude of the rotation:**

**key BYTE -2, 4, 1, 0, -3, 5, 2, -4, -4, 6**

**Your procedure should loop through a plaintext message and align the key to the first 10 bytes of**

**the message. Rotate each plaintext byte by the amount indicated by its matching key array value.**

**Then, align the key to the next 10 bytes of the message and repeat the process. Write a program**

**that tests your encryption procedure by calling it twice, with different data sets.**

|  |
| --- |
| INCLUDE C:\Irvine\Irvine32.inc  .386  .model flat, stdcall  .stack 4096  ExitProcess proto, dwExitCode:dword  ;data here  .data  text byte "Mystring", 0 ;PLANE TEXT GIVEN BY USER  sizeoftext=($ - text) ;defines size of array  key BYTE 2, -4, -4, 6 ;defines an array of keys  newMSG byte "Encrypted String -> : ", 0  tempKEY dword 0 ;defines a variable  ;code goes here  .code  main proc  call encryptMSG ;calls encryptMSG function  lea eax, text ;moves address of text array into eax  invoke ExitProcess, 0  main endp  ;procedure which rotates according a given key  encryptMSG proc  mov edx, 0  mov esi, 0  mov ecx, 10 ;moves to number of keys  mov edi, 0  mainLOOP: ;main loop of the proc  pushad  mov ecx, sizeoftext  sub ecx, 1  mov al, [key+edi]  cmp al, 0  jl negKEY  posKEY:  pushad  movzx ecx, [key+edi]  L1:  movsx eax, byte ptr [text+esi]  ror [text+esi], 1 ;rotates right  loop L1  popad  inc esi  loop posKEY  jmp newiter ;jump to end of this iteration  negKEY:  pushad  movzx ecx, [key+edi]  NOT ecx  inc ecx  mov tempKEY, ecx  movzx ecx, byte ptr tempKEY  L2:  ;movsx eax, byte ptr [text+esi]  rol [text+esi], 1 ;rotates left  loop L2 ;loops for key  movsx eax, byte ptr [text+esi]  popad  inc esi  loop negKEY ;loops to the next alphabet of the string  newiter:  mov edx, offset newMSG ;moves offset to edx  call WriteString ;calls writestring proc  mov edx, offset text  call WriteString  call Crlf ;calls new line  call Crlf ;calls new line  popad  inc edi  loop mainLOOP  ret ;returns to the main procedure  encryptMSG endp  end main |

**Q.9 Create a procedure named Extended\_Sub that subtracts two binary integers of arbitrary size.**

**The storage size of the two integers must be the same, and their size must be a multiple of 32**

**bits. Write a test program that passes several pairs of integers, each at least 10 bytes long.**

**.386**

**.model flat,stdcall**

**.stack 4096**

**ExitProcess proto,dwExitCode:dword**

**.data**

**val1 word 100101110101010b**

**val2 word 001011110111101b**

**.code**

**main proc**

**mov ax, val1**

**mov ax, val2**

**call Extended\_Sub**

**invoke ExitProcess,0**

**main endp**

**Extended\_Sub PROC ;Extended sub procedure**

**mov ax, val1**

**mov bx, val2**

**sbb ax, bx ;Extended Subtraction with borrow**

**ret**

**Extended\_Sub ENDP**

**end main**