Mini Project – A0091525H

ROHIT MUKHERJEE

Introduction

The aim of the project is to write a program in 8086 assembly using the Turbo Assembler (TASM) to perform manipulations on a particular array. These manipulations are as follows

1) Count the total number of elements in an array and display the count
2) Find numbers less than a particular threshold value specified, count the number of elements less than the threshold and then display them
3) Find all the odd numbers in the array and then calculate their average correct to one decimal place
4) Identify all non-prime numbers in the array and display their total count and the numbers themselves

Software used: TASM package, Notepad++, Turbo Debugger2

Problems encountered

- Memory management manually using CPU registers AX,BX,CX,DX and reusing registers (saving context by pushing register values into the stack)
- Common Errors such as comparing and working with same size registers (Eg: 8 bit with 8 bit, 16 bit with 16 bit)
- Displaying floating point numbers
- Error in jump statement due to too many lines of code between target section and jmp statement
- Getting comfortable with the flow of control in assembly programs, the use of interrupts and macros.

Solutions

- Memory management problems were solved by clearing registers using XOR between different code segments for different questions
- Use of the stack for pushing before macros/segments and then popping afterwards maintained data fidelity in the CPU registers
- After writing the first question, comparing registers of equal size became habit and this prevented errors such as “illegal expression”
- Implementing the division followed by remainder multiplication followed by division again (division algorithm) using registers allowed me to print floating point numbers (Q3)

- Using branched assembly selection statements allowed me to prevent jump statements from going out of reach

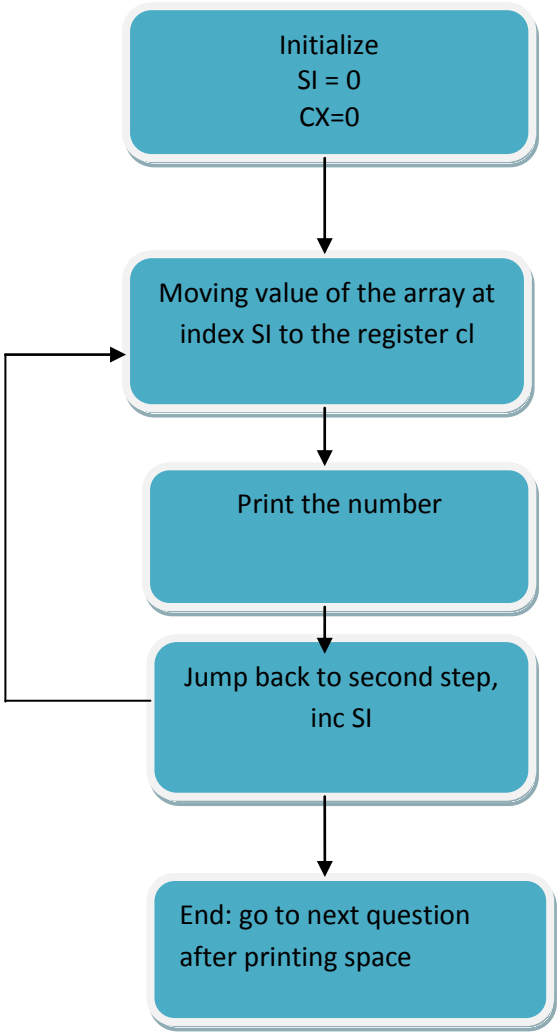
Learning Outcomes

- Understanding how high level languages are different from low level languages and how high level compilers such as g++ and gcc work
- Understanding the bridge between hardware and software and how the microprocessor is integrated with hardware and software simultaneously
- Assembly programming skill

Algorithm used:

1) Counting the number of elements in the array and displaying them

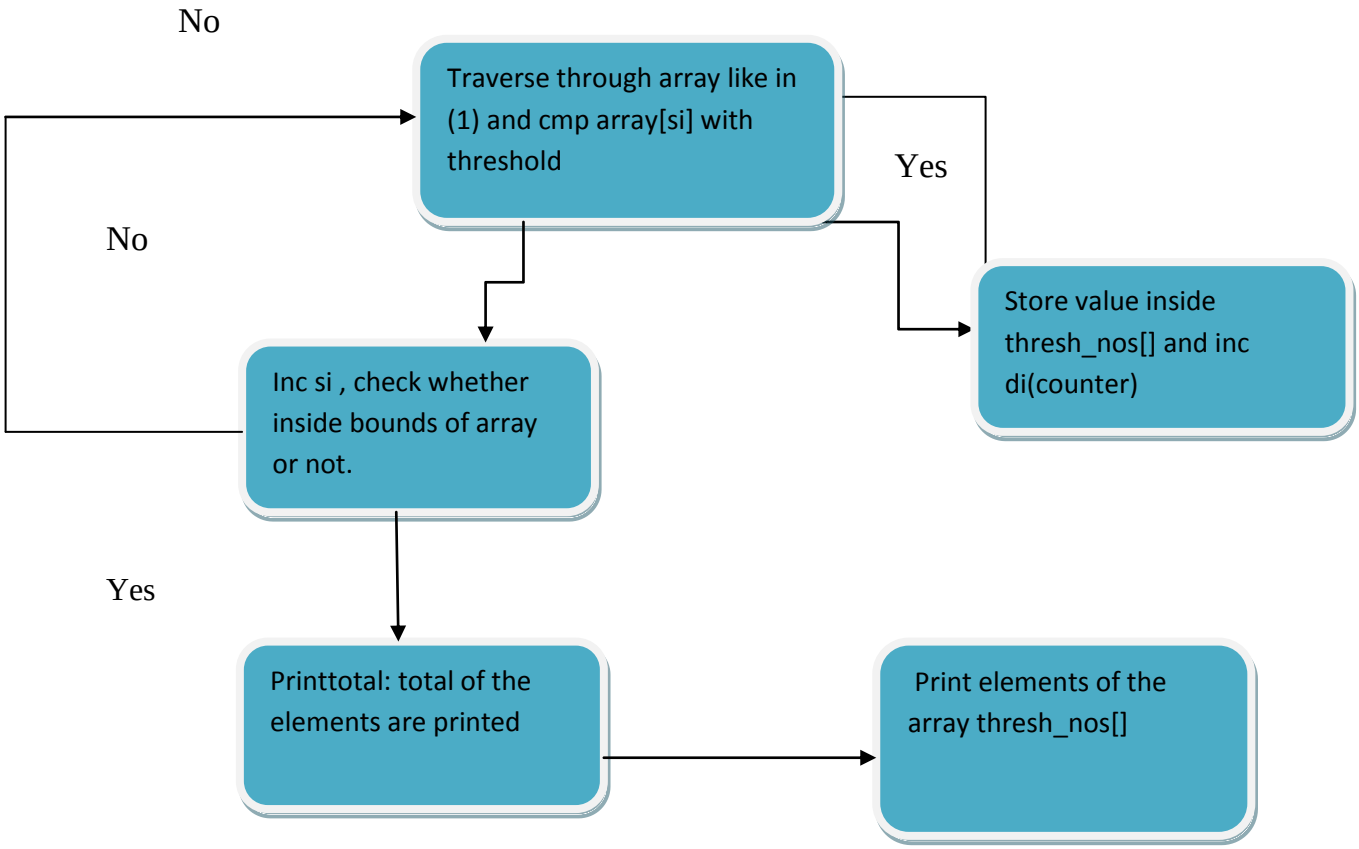
   $ indicates the current address being pointed at, therefore a simple $-array was sufficient to get the size of the array. The algorithm for printing is as follows:



2) Identify all numbers less than threshold and display them to the user. If there are none, display a message

Algorithm for printing elements:

Traverse array and compare elements against the threshold value, if it is less, than add it to another array, and increase counter. After traversing through entire array, check the count of numbers inside the new array. If it is zero, print out message otherwise display the entire array.



3) Identify ODD numbers count them and compute their average. Display your result.

Algorithm for printing odd numbers:

- Initialize the registers AX,BX,CX,DX to 0 using XOR
- Loop through the array storing array[si] in cl
- We check if its odd/even by performing bitwise AND with 01b
- If odd, we send it to code segment oddno.

  Oddno: store odd number in oddnos[], increment di

  Sum up the odd numbers so far, go to next element
- If even, we proceed to next element. This is continued till end of array is reached
- We check if oddtotal is zero, if so we display message to the user
- If not, we display the total number of odd elements (oddnos)
- We then loop through the array oddnos[] like Q1 and output them

Algorithm to print average of odd numbers

- The sum of the numbers are stored in odd_avg
- We move to it register ax, then perform a div operation with oddtotal
- The quotient is stored in al, remainder in ah
- We print the quotient using MACRO printnumbers al
- We multiply the remainder(ah) with 10
- Divide using oddtotal
- The value obtained is the first decimal place, it is printed after a decimal point

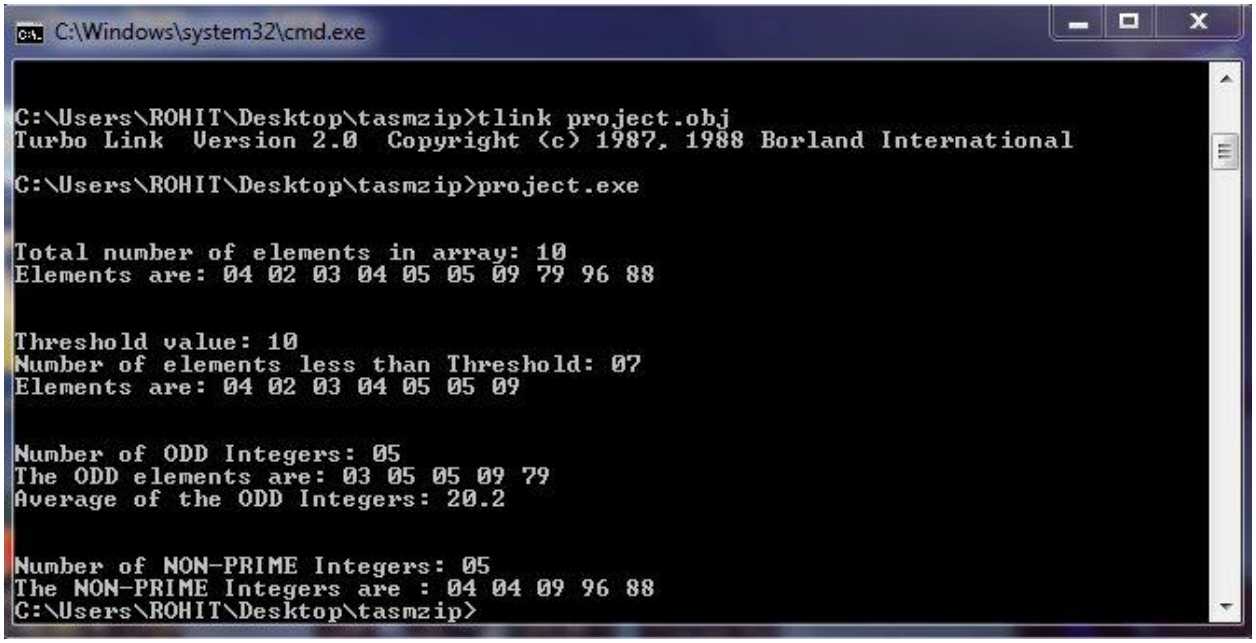
4) Identify all non-prime numbers in your array and display them

Algorithm used

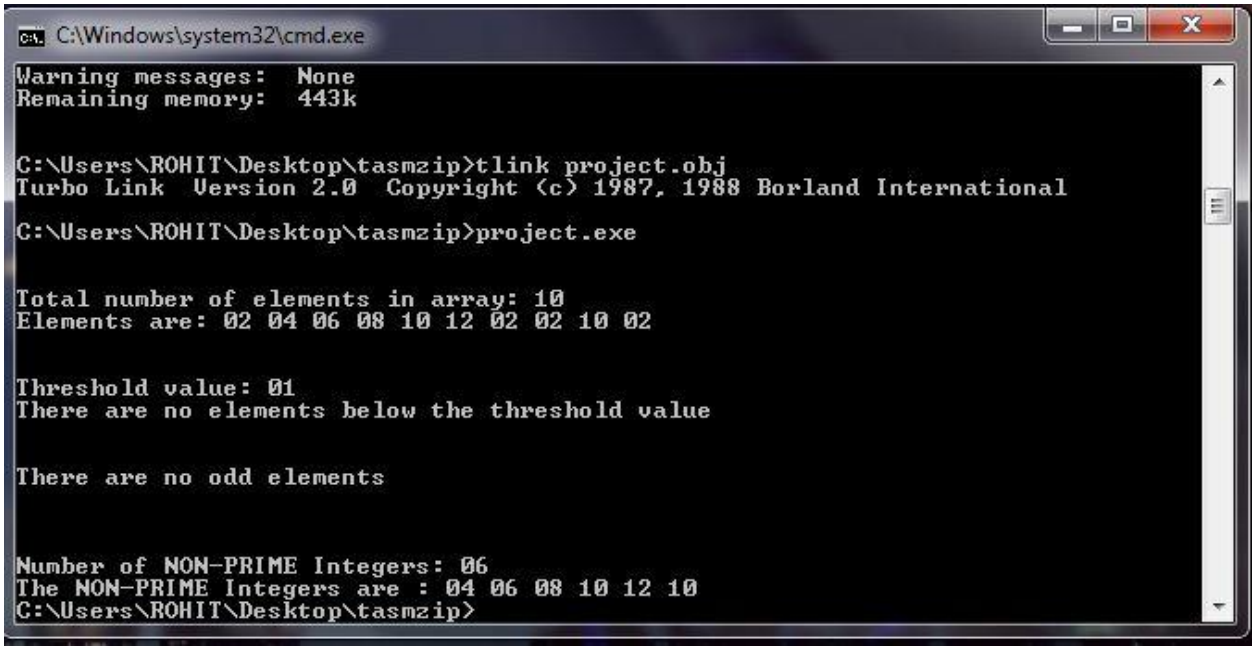
- Algorithm is based on mathematical lemma that when c = a * b then atleast a or b have to be less than or equal to the sqrt(c) if the other exceeds sqrt(c)
- Since the highest value that can be stored is 255, biggest prime number square in this limit is 169 (13 * 13). Therefore, the prime factors for integers in [0,255] are in the range primes={2, 3, 5, 7, 11, and 13}.
- Therefore the most efficient way in terms of time complexity in checking non-prime is to try to divide using the above set of prime factors. If divisible, then non-prime, otherwise it is a prime number.
- We check whether each element in array is divisible by primes [0], if not, we try primes[1] and so on. The array is traversed in reverse as stack operations occur in LIFO
- If indivisible, we send control to the prime segment which increases si(counter)
- If divisible, we push it into a stack for non-primes and increase our counter
- We check the total number of composite (di), if zero we print message
- Otherwise, we print all the non-prime numbers from the stack

Screenshots

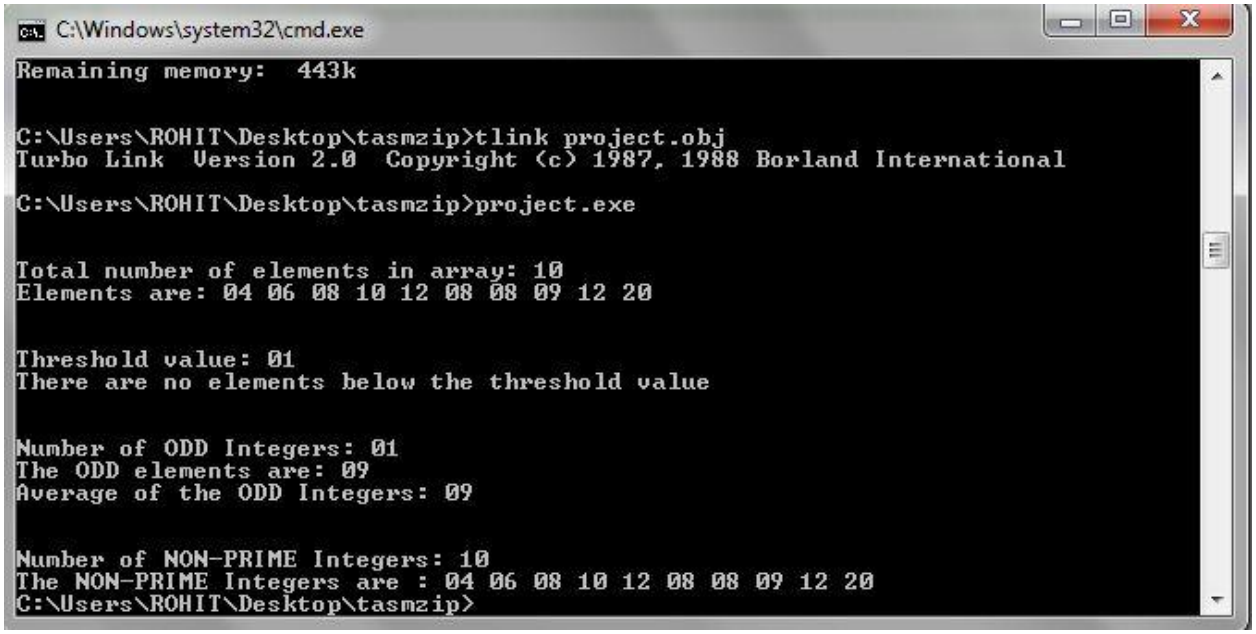
Main Screenshot



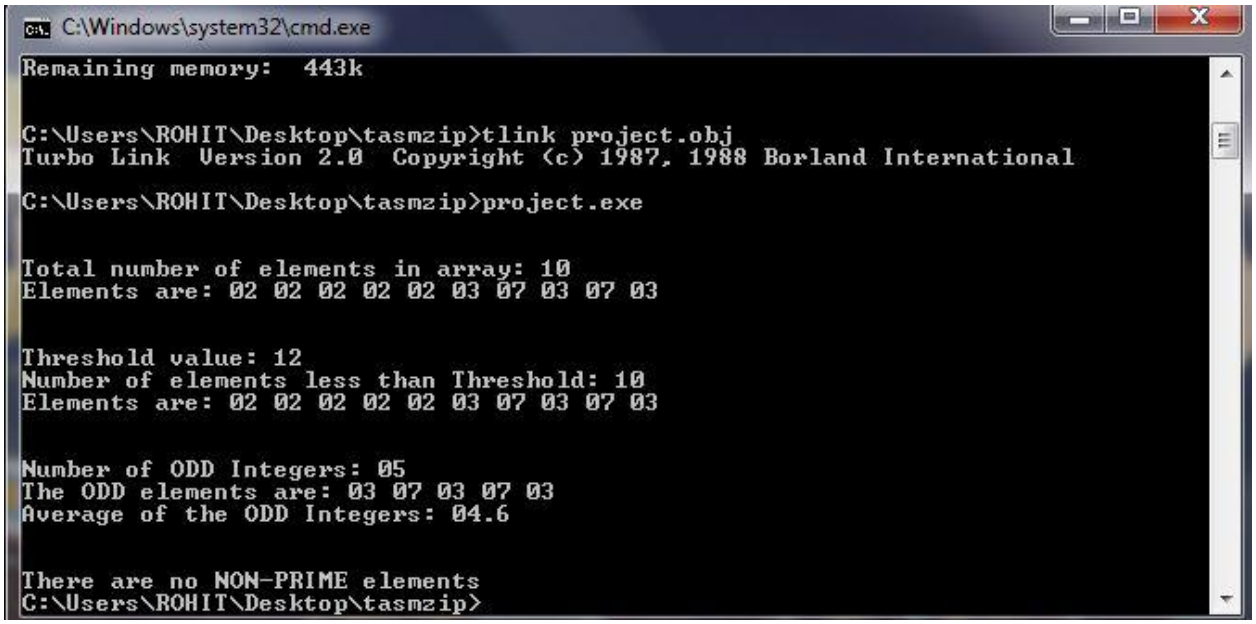
Array =[4,2,3,4,5,5,9,79,96,88] and threshold=10



Array=[2,4,6,8,10,12,2,2,10,2] and threshold =1



Array=[4,6,8,10,12,8,8,9,12,20] and threshold =1



Array=[2,2,2,2,2,3,7,3,7,3] and threshold=12

```
stk segment stack
            db 1024 DUP(?)
            tos label word
    stk ends

;---------------------------Constant/Variable
Declarations-----------------------------------------------------
    data segment
        array db  2,2,2,2,2,3,7,3,7,3
        arraylength =($-array)
        primes db 2,3,5,7,11,13,17,19
        primeslength=($-primes)
        thresh_nos db arraylength dup(?)
        oddnos db arraylength dup(?)
        npnos db arraylength dup(?)
        nptotal db 1 dup(?)
        oddtotal db 1 dup(?)
        array_less db 20 DUP(?)
        odd_avg dw 1 dup(?)
        odd_count db  1 dup(?)

        threshold db 12

        prompt1 db "Total number of elements in array: $"
        prompt2 db "Number of elements less than Threshold: $"
        prompt3 db "Elements are: $"
        prompt4 db "Number of ODD Integers: $"
        prompt5 db "Average of the ODD Integers: $"
        prompt6 db "Number of NON-PRIME Integers: $"
        prompt7 db "The NON-PRIME Integers are : $"
        prompt_threshold db "Threshold value: $"
        prompt_oddelements db "The ODD elements are: $"
        prompt_oddnone db "There are no odd elements $"
        prompt_nonps db "There are no NON-PRIME elements $"
        prompt8 db "There are no elements below the threshold value $"
        temp_variable db 1 dup(?)
        primeFlag db 1 dup(?)
        decimalpoint db ".$"
        colon db ":$"
        comma db  ",$"
        space db " $"
        newline db 0Dh,0Ah,"$"

    data ends;


    code segment
    assume cs:code, ss:stk, ds:data


        start: mov ax,stk       ;initializing stack
               mov ss,ax
               mov sp,offset tos
```

```
                mov ax,data      ;intiializing data segment
                mov ds,ax        ;loaded into data segment


;---------------------------------MACROS
DEFINITION----------------------------------------------------------------


    ;macro prints out single digit numbers, eg: 0-9
    printchar    MACRO char
                 push dx
                 push ax
                 mov ah, 2
                 mov dl,char
                 add dl, 30h;30h is the ascii for '0'
                 int 21h
                 pop dx
                 pop ax
                 ENDM
    ;macro prints out strings, eg: printstring newline will print a new line
printstring  MACRO string
                 push dx
                 lea dx,string
                 mov ah,9
                 int 21h
                 pop dx
                 ENDM

    ;macro prints multiple digit numbers, this is required to display the internal ASCII code
    in decimal
    printnumbers MACRO number
                push ax
                mov ax, number
                mov dx, 0
                mov bx, 10
                div bx
                mov cx, dx
                mov dl, 0
                cmp dl, al
                printchar al
                printchar cl
                pop ax
                ENDM
    ;macro definitions end
;-----------------------------------------Q1 : Printing out
elements------------------------------------------------------------------
        xor ax,ax
        xor bx,bx
        xor cx,cx
        printstring newline
        printstring newline
        ;printing out the array length
        printstring prompt1; prints out the prompt
```

```
        printnumbers arraylength;prints the length

    ;initializing registers si to 0 using XOR so it can be used as counter
        xor si,si
        mov cx,arraylength
        printstring newline
        printstring prompt3;printing out the prompt as per variables/constant section


    printLoop:  mov cl,array[si];moving each array element to cl
                printnumbers cx ;printnumbers macro called to print cx
                printstring space;sent space
                inc si
                cmp si,arraylength
                jnz printLoop
                je nextpart;moves on to the nexr question

    nextpart:  printstring newline
               ;printstring newline
               jmp Q2

;---------------------------------------Q2-------------------------------------------------------
--------------

    Q2:     ;initializing registers
            xor ax,ax
            xor bx,bx
            xor cx,cx
            xor si,si
            xor di,di
            ;threshold value is stored in cl
            mov cl,threshold
            ;odd_avg stores the total
            mov odd_avg,0
            printstring newline
            printstring newline

            printstring prompt_threshold
            printnumbers cx
            printstring newline
            ;printstring prompt3
            xor cx,cx

    thresholdLoop : mov cl,array[si];each array element is stored in cl
                    cmp cl,threshold;array element is checked against threshold
                    jl belowthreshold
                    jg continueloop

    belowthreshold: ;elements lower than the threshold value are stored in array thresh_nos[]
                    mov bl,array[si]
                    mov thresh_nos[di],bl
                    ;di is a counter for total number of values less than the threshold
                    inc di
```

```
                    push di
                    jmp continueloop

    continueloop:   ;continues loop to printloop
                    inc si
                    cmp si,arraylength
                    jl  thresholdLoop
                    jz  printtotal
                    jg  printtotal

    printtotal:     ;at the end of storing all  eligible elements in thresh_nos, it prints the
    total no of elements
                    cmp di,0
                    jz no_elements
                    printstring prompt2
                    pop di
                    mov cx,di
                    printnumbers cx
                    printstring newline
                    printstring prompt3
                    xor si,si
                    jmp print_elements

    no_elements:    ;when empty, print message
                    printstring prompt8
                    jmp Q3



    print_elements: ;print the elements in thresh_nos
                    mov al,thresh_nos[si]
                    xor ah,ah
                    printnumbers ax
                    printstring space
                    inc si
                    cmp si,di
                    jz  Q3
                    jmp print_elements


;---------------------------------------Q3-----------------------------------------------------
----------
    Q3:         ;initializing registers for Q3
                xor ax,ax
                xor bx,bx
                xor cx,cx
                xor dx,dx
                xor si,si
                xor di,di
                printstring newline
                printstring newline
                printstring newline
```

```
printLoop2:  mov cl,array[si]
             and cl,01b         ;bitwise AND with 01b gives 0 if even and 1 if odd
             jne oddno
             je gonext
oddno:  ;the odd elements are stored in array oddnos[]
        mov dl,array[si]
        mov oddnos[di],dl
        xor dh,dh
        inc di;number of odd elements

        add odd_avg,dx; contains the sum of the odd elements
        inc oddtotal;   tracks total number of odd numbers
        jmp gonext

gonext: inc si
        cmp si,arraylength ; check whether still in bounds of array
        jz  printoddno
        jnz printLoop2

printoddno:
            cmp oddtotal,0
            jg oddnonzero
            jz oddzero

oddzero:    ;there are no odd numbers, prints message and moves onto Q4
            printstring prompt_oddnone
            printstring newline
            jmp Q4

oddnonzero: ;odd elements are present, this prints the total number and moves to
print_odd_elements
            printstring prompt4
            mov cl,oddtotal
            mov dl,oddtotal
            mov di,dx
            xor ch,ch
            xor si,si
            printnumbers cx
            printstring newline
            printstring prompt_oddelements
            jmp print_odd_elements

print_odd_elements:
    ;prints all the elements in the array oddnos[]
    mov cl,oddnos[si]
    printnumbers cx
    printstring space
    inc si
    cmp si,di
    jl print_odd_elements
    je printavg
```

```
        printavg:
                    printstring newline
                    printstring prompt5
                    mov ax,odd_avg          ;odd_avg already contains the sum of odd numbers
                    div oddtotal
                    push ax
                    xor ah,ah
                    printnumbers ax
                    pop ax
                    cmp ah,0
                    push ax
                    jnz print_fractional
                    jz Q4

    ;program prints out the fractional part of the average of odd numbers
    print_fractional:printstring decimalpoint
                     pop ax             ;context restored
                     mov al,0Ah         ;decimal 10 (0Ah) is moved to register al
                     mul ah             ;it is multiplied with ah
                     push ax            ;pushed back into stack
                     div oddtotal       ;the quotient from this division is the first decimal
                     point
                     printchar al

;-------------------------------------------------------------Q4-----------------------------------
--------------------------
    Q4:  ;-----------------------------------Number of non-prime
    elements-----------------------------------------------------------

                printstring newline
                printstring newline
                printstring newline
                ;Initializing GPRs
                mov bx,arraylength; contains length of the array
                mov cx,arraylength; contains length of the array
                xor ax,ax
                xor si,si
                xor di,di; counts the total number of non-prime integers

                ;loops will compare the values in the array[] with values present in the primes
                array, if factors are present
                ;it means that the number is non-prime

                outerloop:
                xor bp,bp

                Primenext:
                xor ah,ah
                inc bp
                mov al,array[bx-1]
```

```
                ;1 and 0 are non-prime whole numbers, therefore will be included in our count
                of non-primes
                cmp al,0
                jz isComposite
                cmp al,1
                jz isComposite
                cmp al,primes[bp-1]
                jz Primenext


                ;algorithm used : c = a * b, either a or b must be <= than sqrt (c). In this
                case, the limit of our db array
                ;is 255. The nearest perfect square in range is 169(13 * 13). Here, we are
                dividing all elements by primes till 13
                ;if there is some divisior, it is composite else prime. 17 has been included in
                the primes array for safety.
                div primes[bp-1]
                cmp ah,0
                jz isComposite
                cmp bp,primeslength
                jnz primenext
                jz isPrime


                ;looping backwards because of LIFO nature of stack
                isComposite:
                xor ah,ah
                mov al,array[bx-1]
                push ax
                inc di

                isPrime:
                dec bx                      ;moving back to the previous element (for stack
                purposes)
                dec cx
                jnz outerloop

                ;printing prompt in case there are no prime elements
                cmp di,0
                jnz printtotal2
                printstring prompt_nonps
                jmp exit

                printtotal2:
                printstring prompt6     ;Printing prompt (see variable/constant declarations)
                printnumbers di
                printstring newline
                jmp Q4p2

;--------------------------Printing out non-prime elements (if any) using
stack-----------------------------------------------

      Q4p2 :printstring prompt7     ;Printing prompt (see variable/constant declarations)
```

```
    printelements:
                    pop ax              ;non-prime numbers stored in ax are now inside the stack
                    printnumbers ax     ;here the values are popped and then printed using
                    printnumbers MACRO
                    printstring space   ;space printed
                    dec di              ;counter with total np nos. is decermented
                    jnz printelements   ;loop statement
                    jz exit

;-----------------------------------ENDS-------------------------------------------------------
-------------------

            exit:   ;exit code using 4ch inside interrupt 21h, returns control to DOS
                    mov ah,4ch
                    int 21h
code ends
end start
```