BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Department of Electrical and Electronic Engineering

Course No. : EEE 416

Course Title: Microprocessor and Interfacing Laboratory

Procedures, Stacks, Arrays, Addressing Modes

Name: Mir Sayeed Mohammad

ID: 1606003

Level: 4

Term: 1

Section: A

Submission Deadline: 07 - 04 -2021

Exp 4 Report 1

Find the least common multiplier of four numbers

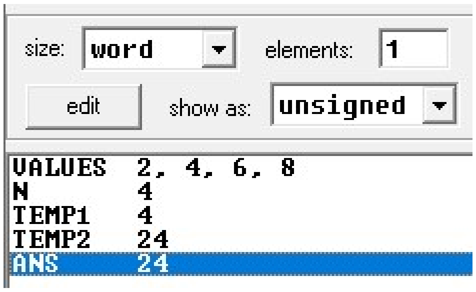
Assembly Code:

```
CODE    SEGMENT
   ASSUME CS:CODE, DS:CODE

   ; *************************** MAIN *************************** ;

   MOV     AX, values
   MOV     BX, values+2

   CALL    LCM                 ; call LCM procedure
   MOV     temp1, CX           ; store result

   MOV     AX, values+4
   MOV     BX, values+6

   CALL    LCM                 ; call LCM procedure
   MOV     temp2, CX           ; store result

   MOV     AX, temp1
   MOV     BX, temp2

   CALL    LCM                 ; call LCM procedure
   MOV     ANS, CX             ; store result

   HLT

   ; *************************** DATA *************************** ;

   values      DW 2, 4, 6, 8
   n           DW 4
   temp1       DW 0
   temp2       DW 0
   ANS         DW 0


   ; ************************ PROCEDURES ************************ ;

   LCM PROC           ; LCM inputs AX and BX, output in CX

      CALL GCD

      PUSH AX         ; Backup AX
      MUL BX          ; AX = AX * BX
      DIV CX          ; LCM = AX * BX / GCD

      MOV CX, AX     ; CX stores result
      POP AX          ; Restore AX

      RET

   LCM ENDP


   GCD PROC          ; GCD inputs AX and BX, output in CX

      PUSH AX        ; Backup AX
      PUSH BX        ; Backup BX

      LEV:

      XOR DX, DX     ; Set dividend to zero in start of each cycle
      DIV BX         ; Divide AX by BX, quotient saved to AX,
                     ; dividend to DX
      MOV AX, BX     ; AX = BX
      MOV BX, DX     ; BX = dividend

      CMP DX, 0H     ; check if dividend was 0

      JNZ LEV        ; keep jumping until dividend is zero
                      ; result will be in AX register

      MOV CX, AX     ; CX stores result
      POP BX         ; Restore BX
      POP AX         ; Restore AX

      RET

   GCD ENDP

CODE    ENDS
      END
```

Explanation:

The code has two procedures, an LCM and a GCD procedure. The LCM procedure calls the GCD procedure to calculate LCM between two numbers. In the main section of code 4 numbers are paired, and two LCMs are measured first. Then the LCM between the first two LCMs are evaluated for final result.

Result:



size: word elements: 1
show as: unsigned

```
VALUES  2, 4, 6, 8
N       4
TEMP1   4
TEMP2   24
ANS     24
```

Fig: LCM of 4 variables

Extra (LCM of n numbers)

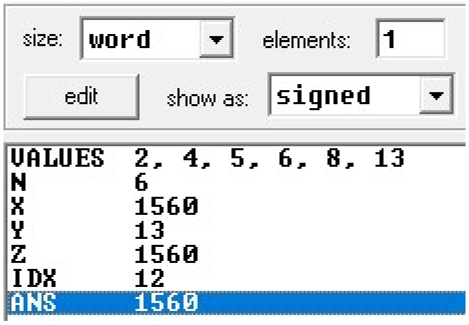
```
CODE    SEGMENT
    ASSUME CS:CODE, DS:CODE

    ; *************************** MAIN *************************** ;

    MOV     CX, n                       ; loop counter = array length

    PUSH    1
    POP     x                           ; x = 1 (first LCM input)

    ITERATOR:

        MOV     BX, idx                 ; array index

        PUSH    values[BX]
        POP     y                       ; y = values[idx]

        ADD     BX, 2
        MOV     idx, BX                 ; idx = idx+2

        CALL    LCM                     ; z = LCM(x,y)

        PUSH    z
        POP     x                       ; LCM output->input for next iter

        LOOP    ITERATOR

    PUSH    z
    POP     ANS                         ; final answer

    HLT


    ; *************************** DATA *************************** ;

    values  DW 2, 4, 5, 6, 8, 13
    n       DW 6      ; array length
```

```
x       DW 0      ; function input 1
y       DW 0      ; function input 2
z       DW 0      ; function output

idx     DW 0      ; array index

ANS     DW 0


; ************************ PROCEDURES ************************ ;

LCM PROC          ; z = LCM(x,y)

   CALL GCD       ; z = GCD(x,y)

   MOV  AX, x
   MOV  BX, y
   MUL  BX         ; AX = x*y

   MOV  BX, z
   DIV  BX         ; AX = x*y/GCD(x,y)

   MOV  z, AX    ; LCM result stored in z from AX

   RET

LCM ENDP


GCD PROC          ; z = GCD(x,y)

   MOV AX, x
   MOV BX, y

   LEV:

   XOR DX, DX    ; Set dividend to zero in start of each cycle
   DIV BX         ; Divide AX by BX, quotient saved to AX,
                  ; dividend to DX
   MOV AX, BX    ; AX = BX
   MOV BX, DX    ; BX = dividend

   CMP DX, 0H    ; check if dividend was 0

   JNZ LEV        ; keep jumping until dividend is zero
                  ; result will be in AX register

   MOV z, AX     ; GCD result stored in z from AX

   RET
   GCD ENDP

CODE    ENDS
      END
```

Explanation:

Same as before, only in the main module, a loop iterates where LCM is evaluated between the previous LCM and the next value in array. The first LCM value for first value in array is equal to that value.

Output:



size: word elements: 1

edit show as: signed

```
VALUES  2, 4, 5, 6, 8, 13
N       6
X       1560
Y       13
Z       1560
IDX     12
ANS     1560
```

Fig: LCM of n variables

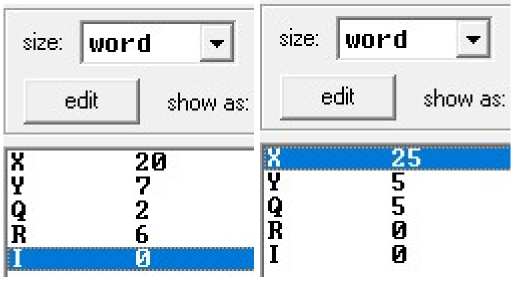
Exp 4 Report 2

Perform division operation by shifting

Assembly Code:

```
; Perform division operation by shifting

CODE    SEGMENT
   ASSUME CS:CODE, DS:CODE

   ; *************************** MAIN *************************** ;

   CALL    DIVIDE

   HLT

   ; *************************** DATA *************************** ;

   x       DW 20     ; dividend
   y       DW 7      ; divisor
   q       DW 0      ; quotient
   r       DW 0      ; remainder

   i       DW 8      ; loop counter

   ; ************************ PROCEDURES ************************ ;

   ; NOT YET IMPLEMENTED

   DIVIDE  PROC              ; q, r : x/y


      MOV CX, i              ; loop to be executed 8 times

      WHILE:

         PUSH CX
         DEC  CX
         MOV  i, CX

         MOV AX, x
         MOV BX, y
         MOV CX, q
         MOV DX, r

         SHL CX, 1            ; quotient <<= 1
         SHL DX, 1           ; remainder <<= 1

         MOV q, CX
         MOV r, DX
         MOV CX, i

         MOV DX, 1
         SHL DX, CL         ; 1<<i

         AND AX, DX          ; dividend & [1<<i]
         SHR AX, CL          ; [dividend & [1<<i]] >> i

         MOV DX, r
         OR  DX, AX         ; remainder |= [dividend & [1<<i]] >> i
         MOV r, DX

         CMP DX, BX          ; if remainder >= divisor
         JS continue

            SUB DX, BX
            MOV r, DX         ; remainder = remainder - divisor

            MOV CX, q
            OR  CX, 1
            MOV q, CX

         continue:

         POP CX
         LOOP WHILE

      RET

   DIVIDE  ENDP

CODE    ENDS
      END
```

Result:



```
X        20        X        25
Y        7         Y        5
Q        2         Q        5
R        6         R        0
I        0         I        0
```

Fig: Execution of Division by shifting

Dividend, divisor, quotient and remainder values are in the X, Y, Q, R variables respectively.

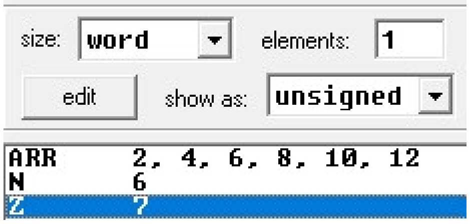
Exp 4 Report 3

Take an array and find its mean value

Assembly Code:

```
CODE    SEGMENT
   ASSUME CS:CODE, DS:CODE

   ; *************************** MAIN *************************** ;

   CALL    MEAN

   HLT

   ; *************************** DATA *************************** ;

   ARR          DW 2, 4, 6, 8, 10, 12
   n            DW 6
   z            DW 0

   ; ************************ PROCEDURES ************************ ;

   MEAN    PROC             ; z = MEAN(arr)

      XOR AX, AX            ; zero init
      XOR BX, BX            ; array index
      MOV CX, n             ; loop counter

      ADDER:

         ADD AX, ARR[BX] ; sum = sum + arr[i]
         ADD BX, 2          ; increment loop counter

         LOOP ADDER

      MOV CX, n
      DIV CX                ; avg = sum / n
      MOV z, AX

      RET

   MEAN    ENDP

CODE    ENDS
      END
```

Result:



size: word

elements: 1

show as: unsigned

```
ARR       2, 4, 6, 8, 10, 12
N         6
Z         7
```

Fig: Mean value of array = 7

Exp 5 Homework 1

Make a program that will sort an array content in both ascending and descending order and put in different arrays.

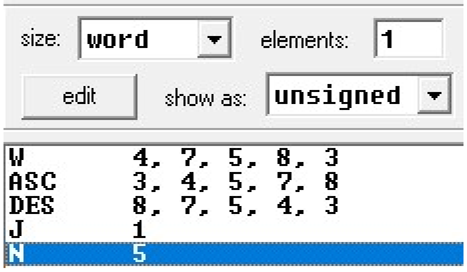
Algorithm:

- Declare 3 arrays, A (initialized with values), ASC (stores ascending order), DSC (stores descending order)
- Copy all elements of A into ASC with loop
- Implement bubble sort algorithm on ASC for ascending order
- Copy all elements of ASC into DSC into reverse order

Assembly Code:

```
CODE   SEGMENT

   ASSUME CS:CODE, DS:CODE

   MOV CX, n
   MOV BX, 0

   WHILE_0:                             ; copy elements of main array

      MOV AX, W[BX]
      MOV ASC[BX], AX
      ADD BX, 2
      LOOP WHILE_0

     MOV CX, n                          ; CX = n-1 at start of outer loop
   DEC CX

   WHILE_1:                             ; bubble sort in ascending order

       XOR AX, AX
       MOV j, AX                        ; j = 0 at start of inner loop

       WHILE_2:

          MOV AX, j
          MOV BX, 2
          MUL BX
          MOV BX, AX

          MOV AX, ASC[BX]
          ADD BX, 2
          CMP AX, ASC[BX]               ; comparing ASC[j] with ASC[j+1]

          JNG not_greater               ; ASC[j] !> ASC[j+1]
          XCHG AX, ASC[BX]
          SUB BX, 2
          MOV ASC[BX], AX            ; swap ASC[j], ASC[j+1]

          not_greater:

          MOV AX, j
          INC AX
          MOV j, AX                  ; j = j+1

          CMP AX, CX
          JNZ WHILE_2                ; loop if AX != CX

        LOOP WHILE_1

   MOV CX, n
   XOR BX, BX
   MOV AX, n
   MOV DX, 2
   MUL DX
   SUB AX, 2

   WHILE_3:                        ; store reverse of first array

      PUSH ASC[BX]
      XCHG AX, BX
      POP  DES[BX]
      XCHG AX, BX

      ADD  BX, 2
      SUB  AX, 2

      LOOP WHILE_3

   HLT

   W   DW 4, 7, 5, 8, 3
   ASC DW 5 DUP(0)
   DES DW 5 DUP(0)

   j   DW 0                        ; loop counter
   n   DW 5                         ; array length

CODE    ENDS
      END
```

Result:



size: word elements: 1

show as: unsigned

```
W        4, 7, 5, 8, 3
ASC      3, 4, 5, 7, 8
DES      8, 7, 5, 4, 3
J        1
N        5
```

Fig: Sorting into ascending and descending order

Exp 5 Homework 2

Write an algorithm to convert a binary number into decimal and implement in assembly.

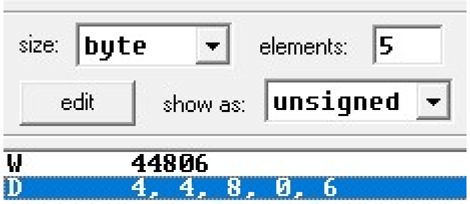
Algorithm:

- Initialize a binary sequence in a variable W
- 16 bits binary has maximum 5 digits in decimal, so initialize result array of length 5
- Loop 5 times → divide W by 10, store remainder in last free slot of result array

Assembly Code:

```
CODE    SEGMENT

   ASSUME CS:CODE, DS:CODE

   MOV CX, 5
   MOV AX, W

   WHILE:


      XOR DX, DX            ; remainder to 0
      MOV BX, 10
      DIV BX                ; Divide by 10

      MOV BX, CX
      DEC BX                ; array index

      MOV D[BX], DL

      LOOP WHILE            ; Move remainder to array

   HLT

   W  DW   1010111100000110B
   D  DB 5 DUP(0)

CODE    ENDS
      END
```

Result:



size: byte elements: 5

show as: unsigned

```
W       44806
D       4, 4, 8, 0, 6
```

Fig: Binary to decimal

In the W variable, the value is being showed as a decimal number, and in the D array, we can see the corresponding digits.

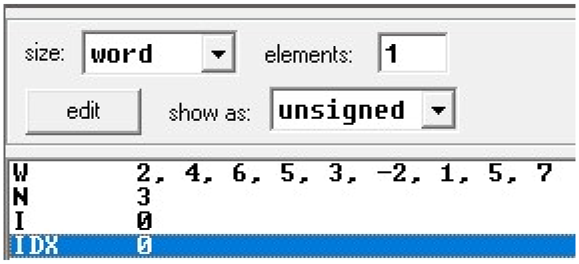
Exp 5 Labtask

Write a code to convert a square matrix to diagonally dominant form. Generalize the code to work with square matrix of any size.

Assembly Code:

```
CODE    SEGMENT

    ASSUME CS:CODE, DS:CODE

    ORG 100H

    MOV CX, n
    XOR BX, BX

    MOV i, BX

    WHILE_0:                            ; loop through all the rows

        PUSH CX
        MOV CX, n

        XOR SI, SI
        XOR AL, AL
        MOV idx, SI

        WHILE_1:                        ; find index to greatest value in row

            CMP W[BX+SI], AL
            JNG continue_1
            MOV AL, W[BX+SI]
            MOV idx, SI
            continue_1:
            INC SI
            LOOP WHILE_1

        POP CX

        MOV SI, idx                     ; swap greates value with diagonal element
        XCHG AL, W[BX+SI]
        MOV SI, i
        XCHG AL, W[BX+SI]
        MOV SI, idx
        XCHG AL, W[BX+SI]

        MOV AX, i
        INC AX
        MOV i, AX
        ADD BX, n
        LOOP WHILE_0

    HLT

    W    DB  2,  4,  6             ; input square matrix
         DB  5,  3,  -2
         DB  1,  5,  7

    n   DW  3                      ; square matrix dimension
    i   DW  0                      ; outer loop index

    idx DW  0                      ; maximum value index

CODE    ENDS
        END
```

Result:

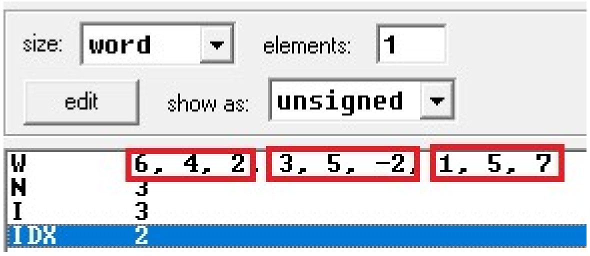


size: word elements: 1

edit show as: unsigned

```
W          2, 4, 6, 5, 3, -2, 1, 5, 7
N          3
I          0
IDX        0
```

(a)



size: word elements: 1

edit show as: unsigned

```
W          6, 4, 2, 3, 5, -2, 1, 5, 7
N          3
I          3
IDX        2
```

(b)

Fig: (a) Input matrix (b) Matrix in diagonally dominant form