

Fall Semester 2021-2022
Microprocessor and Interfacing
Lab Report

Digital Assignment-4

Experiment No: 5

Task No: 4

Course Code: CSE2006

Slot: L7+L8



Submitted By: Alokam Nikhitha

Reg. Numb: 19BCE2555

Submitted To: Dr. Abdul Majed KK

EXPERIMENT 5:

Aim:

1. Write an ALP Program to find LCM of a given numbers
2. Write an ALP program to find the average of N numbers.
3. Write an ALP to find the greatest among two numbers.

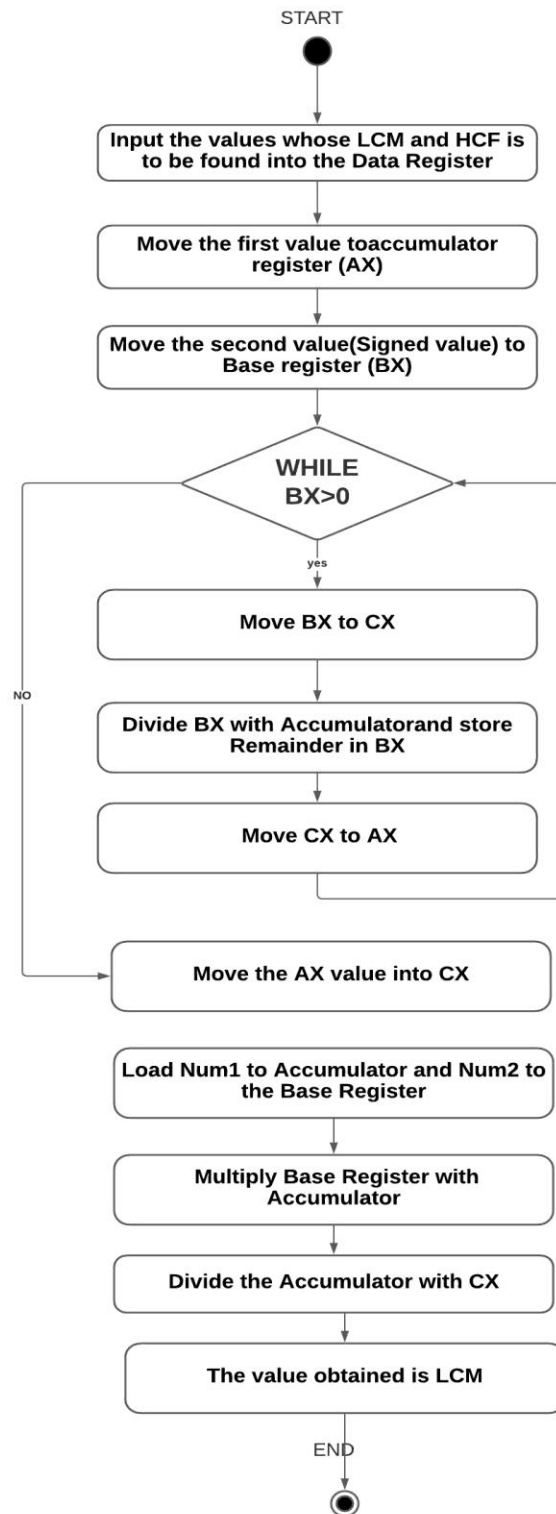
Tool Used: emu8086 simulator

1. Write an ALP Program to find LCM of a given numbers

Algorithm:

- Input the values whose LCM and HCF is to be found into the Data Register
- Move the first value to _accumulator register (AX).
- Move the second value(Signed value) to Base register (BX).
- Create a while loop until until BX becomes 0
- Move BX to CX.
- Divide BX with Accumulator and store Remainder in BX
- Move CX to AX and repeat the While loop
- After the While loop ends the value in AX is HCF
- Move the AX value into CX
- Load Num1 to Accumulator and Num2 to the Base Register
- Multiply Base Register with Accumulator
- Divide the Accumulator with CX(HCF)
- The value obtained is LCM(stored in AX)
- Halt the overall process.

Flow Chart:



Design and Calculations:

Input the values whose LCM and HCF is to be found into the Data Register. Move the first value to accumulator register (AX). Move the second value (Signed value) to Base register (BX). Create a while loop until until BX becomes 0. Move BX to CX. Divide BX with Accumulator and store Remainder in BX. Move CX to AX and repeat the While loop. After the While loop ends the value in AX is HCF. Move the AX value into CX. Load Num1 to Accumulator and Num2 to the Base Register. Multiply Base Register with Accumulator. Divide the Accumulator with CX (HCF). The value obtained is LCM (stored in AX)

Number 1: $24H = (36)$ Decimal
Number 2: $0FH = (15)$ Decimal

HCF

$$\begin{array}{r} 3 \overline{) 36, 15} \\ \underline{12, 15} \\ 12, 15 \\ \underline{12, 15} \\ 0 \end{array}$$

$\therefore \text{HCF} = 3$ Decimal
 $= (3)H$

LCM = $\frac{\text{Number 1} \times \text{Number 2}}{\text{HCF}}$

$$= \frac{36 \times 15}{3} = 180$$

$(B4)_{\text{Hexa}}$

Program Code:

ASSUME CS:CODE DS:DATA

DATA SEGMENT

NUM1 DW 24H

NUM2 DW 0FH

HCF DW ?

LCM DW ?

ENDS

CODE SEGMENT

ASSUME DS:DATA CS:CODE

START:

MOV AX,DATA

MOV DS,AX

MOV AX,NUM1

MOV BX,NUM2

WHILE:MOV DX,0

MOV CX,BX

DIV BX

MOV BX,DX

MOV AX,CX

CMP BX,0

JNE WHILE

MOV HCF,AX
MOV CX,AX
MOV AX,NUM1
MOV BX,NUM2
MUL BX
DIV CX
MOV LCM,AX
MOV BX,0
INT 21H
ENDS
END START

```
01 ASSUME CS:CODE DS:DATA
02 DATA SEGMENT
03 NUM1 DW 24H
04 NUM2 DW 0FH
05 HCF DW ?
06 LCM DW ?
07 ENDS
08 CODE SEGMENT
09 ASSUME DS:DATA CS:CODE
10 START:
11 MOV AX,DATA
12 MOV DS,AX
13 MOV AX,NUM1
14 MOV BX,NUM2
15 WHILE:MOV DX,0
16 MOV CX,BX
17 DIV BX
18 MOV BX,DX
19 MOV AX,CX
20 CMP BX,0
21 JNE WHILE
22 MOV HCF,AX
23 MOV CX,AX
24 MOV AX,NUM1
25 MOV BX,NUM2
26 MUL BX
27 DIV CX
28 MOV LCM,AX
29 MOV BX,0
30 INT 21H
31 ENDS
32 END START
```

Output:

The screenshot displays an x86 emulator interface with three main windows:

- variables**: A window showing memory variables. The 'size' is set to 'word' and 'elements' to '1'. The 'show as' is set to 'hex'. The variables listed are:
 - NUM1: 0024h
 - NUM2: 000Fh
 - HCF: 0003h
 - LCM: 00B4h
- original source co...**: A window showing assembly code. The code is as follows:

```
13 MOV AX, NUM1
14 MOV BX, NUM2
15 WHILE: MOV DX, 0
16 MOV CX, BX
17 DIV BX
18 MOV BX, DX
19 MOV AX, CX
20 CMP BX, 0
21 JNE WHILE
22 MOV HCF, AX
23 MOV CX, AX
24 MOV AX, NUM1
25 MOV BX, NUM2
26 MUL BX
27 DIV CX
28 MOV LCM, AX
29 MOV BX, 0
30 INT 21h
31 END
```
- emulator: Exp5 Q1.exe**: The main emulator window. It shows the CPU registers on the left and memory on the right. The registers window shows:
 - AX: 00 B4
 - BX: 00 00
 - CX: 00 03
 - DX: 00 00
 - CS: F400
 - IP: 0204
 - SS: 0710
 - SP: FFFA
 - BP: 0000
 - SI: 0000
 - DI: 0000
 - DS: 0710
 - ES: 0700The memory window shows the instruction stream. The instruction at address F4204 is highlighted: `F4204: CF 207 ?`. The BIOS DI register is also visible, showing `INT 021h`.

Result and Inference:

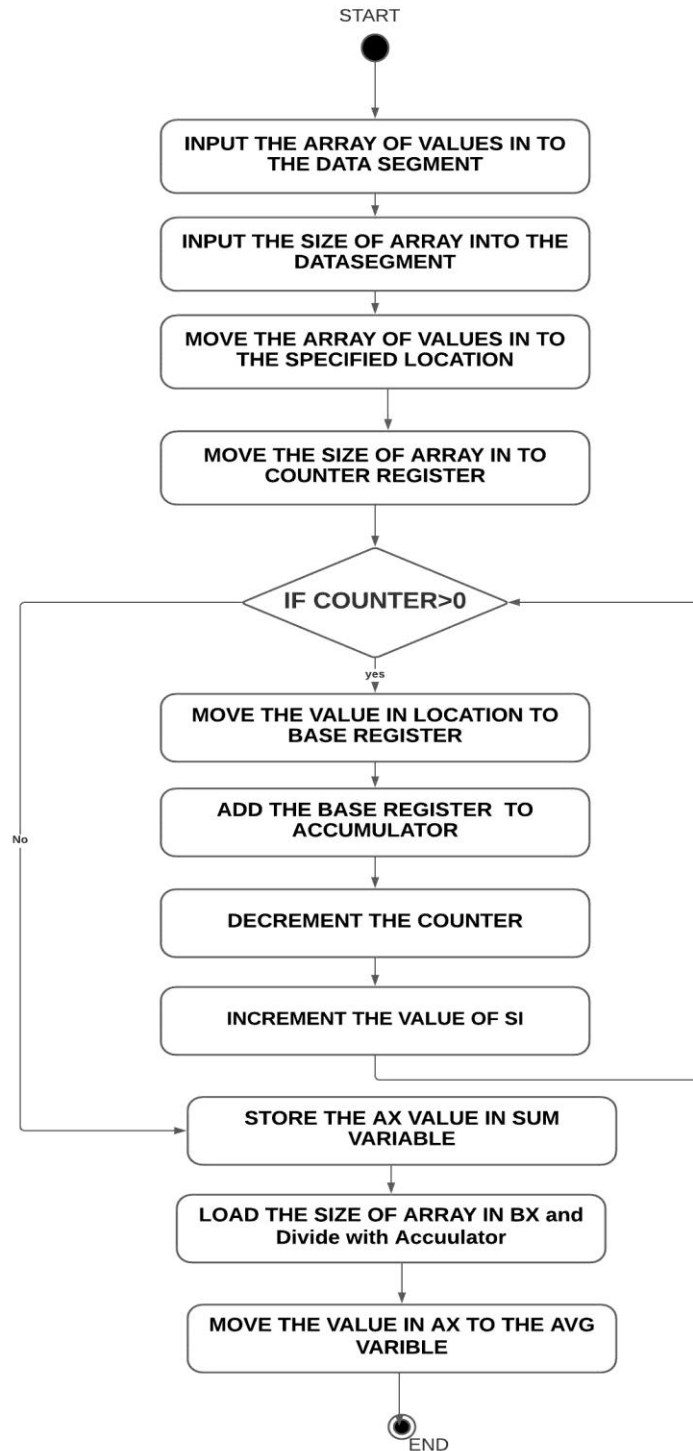
-The HCF of the 2 input values is Stored in CX and LCM is stored in AX and Variable we can see that HCF is 0003H and LCM is 00B4H

2) Write an ALP program to find the average of N numbers.

Algorithm

- **Input the Array of Values in Data Segment**
- **Input the Size of the Array in Data segment**
- **Take SUM and AVG in DataSegment**
- **Move all the values of the Array to a specified Memory Location(SI) in Code Segment**
- **Move the size of Array from Data segment in to Counter Register**
- **Now Run a loop and Add move the values in SI location of to Base Register and add that to Accumulator .Increment the SI value and**
- **Decrement the Counter Register**
- **Repeat the Process until the Counter becomes 0**
- **The Sum is stored in Accumulator.**
- **Move Size of Array in to BX**
- **-Divide the Sum with BX**
- **-Move the value to AVG**

Flow chart:



Design and Calculations:

Input the Array of Values(001H, 023H, 045H, 067H, 042H, 04BH, 012H, 0EFH) in Data Segment .Input the Size of the Array(8) in Data segment.Move all the values of the Array to a specified Memory Location(SI) in Code Segment.Move the size of Array from Data segment in to Counter Register.Now Run a loop and Add move the values in SI location of to Base Register and add that to Accumulator .Increment the SI value and Decrement the Counter Register.Repeat the Process until the Counter becomes 0 .The Sum is stored in Accumulator. Move Size of Array in to BX.Divide the Sum with BX.Move the value to AVG

Sum of the values in the array with n numbers
Average

Values:
01H, 23H, 45H, 67H, 42H, 4BH, 12H, EFH

01H + 23H = 24H
24H + 45H = 69H
69H + 67H = D0H
D0H + 42H = 112H
112H + 4BH = 15DH
15DH + 12H = 16FH
16FH + EFH = 25EH

∴ Sum of Values = 25EH

Average
= $\frac{25EH}{8H}$
= 4BH

Program Code:

ASSUME CS: CODE ,DS: DATA

DATA SEGMENT

**ARR DB 001H, 023H, 045H, 067H, 042H, 04BH,
012H, 0EFH**

N DW 08H

SUM DW 01 DUP (?)

AVG DW 01 DUP (?)

DATA ENDS

CODE SEGMENT

START:

MOV AX, @DATA

MOV DS, AX

MOV CX, N

MOV AX, 0000H

MOV SI, OFFSET ARR

ABC:

MOV BL, [SI]

INC SI

ADD AX, BX

DEC CX

JNZ ABC

MOV SUM, AX
MOV CX, AX
MOV BX, N
DIV BX
MOV AVG, AX
INT 21H
CODE ENDS
END START

```
01  ASSUME CS: CODE ,DS: DATA
02
03
04
05  DATA SEGMENT
06      ARR DB 001H, 023H, 045H, 067H, 042H, 04BH, 012H, 0EFH
07      N DW 08H
08      SUM DW 01 DUP (?)
09      AVG DW 01 DUP (?)
10      DATA ENDS
11
12
13
14  CODE SEGMENT
15      START:
16          MOV AX, @DATA
17          MOV DS, AX
18          MOV CX, N
19          MOV AX, 0000H
20          MOV SI, OFFSET ARR
21      ABC:
22          MOV BL, [SI]
23          INC SI
24          ADD AX, BX
25          DEC CX
26          JNZ ABC
27          MOV SUM, AX
28          MOV CX, AX
29          MOV BX, N
30          DIV BX
31          MOV AVG, AX
32          INT 21H
33      CODE ENDS
34
35
36
37  END START
```

Output:

The screenshot displays a debugger interface with three main windows:

- variables**: Shows a list of variables. The 'size' is set to 'word' and 'elements' to '1'. The 'show as' is set to 'hex'. The variables are: ARR (01h, 23h, 45h, 67h, 42h, 4Bh, 12h, 0EFh), N (0008h), SUM (025Eh), and AVG (004Bh). The AVG variable is highlighted in blue.
- original source code**: Shows assembly code. Line 32, `INT 21h`, is highlighted in yellow.
- emulator: Exp5 Q2.exe**: Shows the CPU registers and memory. The registers window shows: AX (00 4B), BX (00 08), CX (02 5E), DX (00 06), CS (F400), IP (0204), SS (0710), SP (FFFA), BP (0000), SI (0008), DI (0000), DS (0710), and ES (0700). The memory window shows the instruction at address F42004: `CF 207`.

Result and Inference:

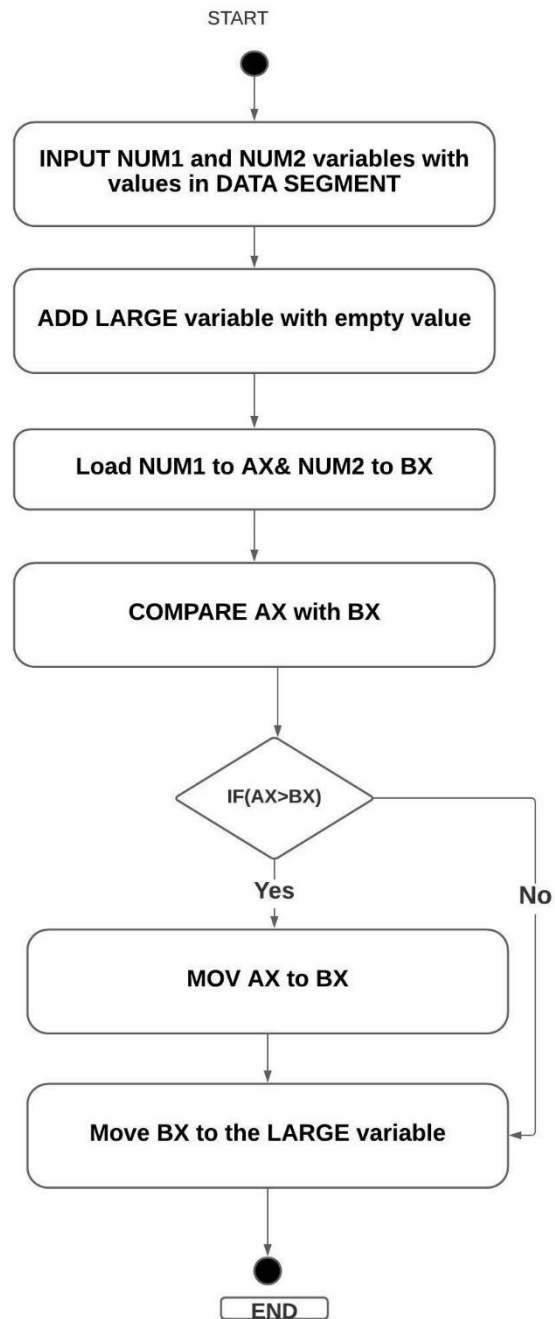
-The sum of the Values is stored in CX, and Average is Stored in Accumulator. In the Variables We can see the value of Sum(25EH) and Average(4BH)

3) Write an ALP to find the greatest among two numbers.

ALGORITHM

- **Input the 2 numbers NUM1 and NUM2 in the Data segment**
- **Move NUM1 into AX and NUM2 into BX**
- **Compare AX with BX**
- **If carry is obtained (ie if AX is smaller than BX) .Jump to Next**
- **Move AX to BX (if BX greater than AX)**
- **Next Loop: Move the BX value to LARGE variable**

FLOWCHART



Design and Calculations:

Input the 2 numbers NUM1 and NUM2 in the Data segment. Then move NUM1 into AX and NUM2 into BX. Compare AX with BX. If carry is obtained (ie if AX is smaller than BX) .Jump to Next. Move AX to BX (if BX greater than AX). In Next Loop Move the BX value to LARGE variable

Program Code:

DATA SEGMENT

NUM1 DW 1529H

NUM2 DW 1231H

LARGE DW ?

DATA ENDS

CODE SEGMENT

ASSUME DS:DATA,CS:CODE

START:

MOV AX,DATA

MOV DS,AX

MOV CX, 01h

MOV BX, NUM2

UP:

MOV AX, NUM1

CMP AX, BX

JL NXT

MOV BX,AX


NXT:

MOV LARGE,BX

INT 21H

CODE ENDS

END START

 edit: C:\emu8086\MySource\Exp5 Q3.asm

file edit bookmarks assembler emulator math ascii codes help



new



open



examples



save



compile



emulate



calculator



converter



options



help



about

```
01 DATA SEGMENT
02
03 NUM1 DW 1529H
04 NUM2 DW 1231H
05 LARGE DW ?
06
07 DATA ENDS
08
09 CODE SEGMENT
10     ASSUME DS:DATA,CS:CODE
11 START:
12     MOV AX,DATA
13     MOV DS,AX
14     MOV CX, 01h
15     MOV BX, NUM2
16 UP:
17     MOV AX, NUM1
18     CMP AX, BX
19     JL NXT
20     MOV BX,AX
21
22 NXT:
23     MOV LARGE,BX
24     INT 21H
25 CODE ENDS
26 END START
```

Output:

The screenshot displays three windows from a debugger interface:

- variables**: A window showing the state of variables. It has a 'size' dropdown set to 'word', 'elements' set to '1', and a 'show as' dropdown set to 'hex'. The variables listed are:

Variable	Value
NUM1	1529h
NUM2	1231h
LARGE	1529h
- original source code**: A window showing the assembly code. The current line of execution is highlighted in yellow:

```
10 ASSUME DS:DATA,CS:CODE
11 START:
12 MOV AX,DATA
13 MOV DS,AX
14 MOV CX,01h
15 MOV BX,NUM2
16 UP:
17 MOV AX,NUM1
18 CMP AX,BX
19 JC NXT
20 MOV BX,AX
21
22 NXT:
23 MOV LARGE,BX
24 INT 21H
```
- emulator: Exp5 Q3.exe**: The main debugger window. It shows the CPU registers on the left and the instruction stream on the right. The instruction stream is divided into two columns, both showing the current instruction being executed: `MOV BX, NUM1` at address `0713F: F4 244`. The registers window shows the following values:

Register	H	L
AX	15	29
BX	15	29
CX	00	01
DX	00	00
CS	0711	
IP	002F	
SS	0710	
SP	0000	
BP	0000	
SI	0000	
DI	0000	
DS	0710	
ES	0700	

This is a close-up view of the 'variables' window. It shows the 'size' dropdown set to 'word', 'elements' set to '1', and 'show as' set to 'hex'. The variables are listed as follows:

Variable	Value
NUM1	1529h
NUM2	1231h
LARGE	1529h

Result and Inference:

Since AX is greater than BX. AX is moved to BX and BX value is moved to LARGE variable(1529H)