MICROPROCESSOR & INTERFACING LAB NOTES

AKSHANSH CHAUDHARY

Microprocessor and Interfacing Lab Notes, First Edition

Presented by: Akshansh Chaudhary
Graduate of BITS Pilani, Dubai Campus
Batch of 2011

Course content by: Ms. Susanna S. Henry
Then Faculty, BITS Pilani, Dubai Campus

Layout design by: AC Creations © 2013

The course content was prepared during Spring, 2013.
More content available at: www.Akshansh.weebly.com

18.2.2013

Lab - 1

Introduction to debug

```
-r
AX=0000 BX=0000 CX=0000 DX=C000 SP= FFEE BP=0000
SI=0000 DI=0000 DS=1378 ES=1378 SS=1378 CS=1378
IP=0100   NV UP EI PL NZ NA PO NC
1378:0100 0000    ADD [BX+SI], AL
                      DS:0000 = CD
```

```
-?
assemble   A [address]
compare    C range address
dump       D [range]
enter      E address [list]
fill       F range list
go         G [=address][addresses]
hex        H value1 value2
input      I port
load       L [address][drive][first sector][number]
move       M range address
name       N [pathname][arglist]
output     O port byte
```

proceed P [= address] [number]
quit Q
register R [register]
search S range list
trace T [= address] [value]
unassemble U [range]
write W [address] [drive] [first sector] [number]
allocate expanded memory XA [# pages]
deallocate expanded memory XD [handle]
map expanded memory pages XM [L page] [P page] [handle]
display expanded memory status XS
_

3. Move 9c to the lower bit of AX registes and move 64 to highes bit of DX registes. Add AX and DX registes. Note the flag changes and explain in your record.

```
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE
BP=0000 SI=0000 DI=0000 DS=1378 ES=1378
SS=1378 IP=0100 NV UP EI PL NZ NA PO NC
1378:0100 0000        ADD [BX+SI],AL  DS:0000=CD

-a cs:100
1378:0100 mov ax,9c
1378:0103 mov dh,64
1378:0105 add ax,dx
1378:0107
-t = 100 3
AX=009C BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000
SI=0000 DI=0000 DS=1378 ES=1378 SS=1378 CS=1378
IP=0103 NV UP EI PL NZ NA PO NC
1378:0103 B664        MOV DH,64
```

AX=009C BX=0000 CX=0000 DX=6400 SP=FFEE BP=0000
SI=0000 DI=0000 DS=1378 ES=1378 SS=1378 CS=1378
IP=0105 NV UP EI PL NZ NA PO NC
1378:0105 01D0 ADD AX, DX

AX=649C BX=0000 CX=0000 DX=6400 SP=FFEE BP=0000
SI=0000 DI=0000 DS=1378 ES=1378 SS=1378 CS=1378
IP=0107 NV UP EI PL NZ NA PE NC
1378:0107 0000 ADD [BX+SI], AL.
DS:0000=CD

- Seeing the changes in the flag, we find, only the Parity flag changing from PO (odd) to PE (even). This occurs due to the presence of even number of 1's in the final (last) place of data storage i.e AX.
As shown above, AX = 649C.
Converting to binary, 649C = 0110 0100 1001 1100

Seeing the last 8 bits, the number of 1's are even. Hence, the parity.

Move 5 to higher bit of BX register and move 6 to lower bit of CX register. Add both registers and show result in AX register.

```
-a CS:100
1378:0100 mov bh,5
1378:0102 mov cl,6
1378:0105 add bx,cx
1378:0107 mov ax,bx
1378:0109
-t=100 4

AX=0000 BX=0500 CX=0000 DX=0000 SP=FFEE BP=0000
SI=0000 DI=0000 DS=1378 ES=1378 SS=1378 CS=1378
IP=0102 NV UP EI PL NZ NA PO NC
1378:0102 B90600 MOV CX,0006

AX=0000 BX=0500 CX=0006 DX=0000 SP=FFEE BP=0000
SI=0000 DI=0000 DS=1378 ES=1378 SS=1378 CS=1378
IP=0105 NV UP EI PL NZ NA PO NC
1378:0105 01CB ADD BX,CX

AX=0000 BX=0506 CX=0006 DX=0000 SP=FFEE BP=0000
SI=0000 DI=0000 DS=1378 ES=1378 SS=1378 CS=1378
```

IP=0107 NV UP EI PL NZ NA PE NC
1378:0107 89D8 MOV AX, BX

AX=0506 BX=0506 CX=0006 DX=0000 SP=FFEE BP=0000
SI=0000 DI=0000 DS=1378 ES=1378 SS=1378 CS=1378
IP=0109 NV UP EI PL NZ NA PE NC
1378:0109 0000 ADD [BX+SI], AL
DS:0506=00

—x—

2. Move 5 to the higher bit of BX register and move 6 to the lower bit of CX register. SUB both registers and show result in AX register. Note flag changes and explain in your record.

-a 100
1378:0100 mov bh, 5
1378:0102 mov cx, 6
1378:0105 sub bx, cx
1378:0107 mov ax, bx
1378:0109
-t=100 4

AX=0000 BX=0500 CX=0006 DX=0000 SP=FFEE BP=0000
SI=0000 DI=0000 DS=1378 ES=1378 SS=1378 CS=1378

IP = 0102 NV UP EI PL NZ NA PO NC
1378:0102 B90600 MOV CX, 0006

4/3/2013

AX=0000 BX=0500 CX=0006 DX=0000 SP=FFEE BP=0000
SI=0000 DI=0000 DS=1378 ES=1378 SS=1378 CS=1378
IP=0105 NV UP EI PL NZ NA PO NC
1378:0105 29CB SUB BX, CX

AX=0000 BX=04FA CX=0006 DX=0000 SP=FFEE BP=0000
SI=0000 DI=0000 DS=1378 ES=1378 SS=1378 CS=1378
IP=0107 NV UP EI PL NZ AC PE NC
1378:0107 89D8 MOV AX, BX

AX=04FA BX=04FA CX=0006 DX=0000 SP=FFEE BP=0000
SI=0000 DI=0000 DS=1378 ES=1378 SS=1378 CS=1378
IP=0109 NV UP EI PL NZ AC PE NC
1378:0109 0000 ADD [BX+SI], AL DS:04FA=00

—

Observing the flag changes, we find 2 flags - parity and auxiliary carry to be changing.
Now, $AX = (04FA)_{16} = (0000\ 0100\ 1111\ 1010)_2$
~~No. of 1's in AX are 7 And, there is one more 1 for AC. So, total number of 1's are 7+1 = 8.~~
~~Hence,~~ The flag changes from PO (odd parity) to PE (even parity) ∵ No. of 1's in last 8 bits are even.
Hence, the parity.

25.2.2013

Lab - 2

Addressing Modes

Q.1. Write an assembly language program to move 9f2 to AX and 89 to BX respectively, add and store the result in AX.

```
-a cs:300
137F:0300 mov ax, 9f2
137F:0303 mov bx, 89
137F:0306 add ax, bx
137F:0308
-t=300 3
```

```
AX=09F2 BX=00F9 CX=0012 DX=0000 SP=FFEE BP=0000
SI=0000 DI=0000 DS=137F ES=137F SS=137F CS=137F
IP=0303 NV UP EI PL NZ NA PO NC
137F:0303 BB8900 MOV BX,0089
```

```
AX=09F2 BX=0089 CX=0012 DX=0000 SP=FFEE BP=0000
SI=0000 DI=0000 DS=137F ES=137F SS=137F CS=137F
IP=0306 NV UP EI PL NZ NA PO NC
137F:0306 01D8 ADD AX,BX
```

AX=0A7B BX=0089 CX=0012 DX=0000 SP=FFEE BP=0000
SI=0000 DI=0000 DS=137F ES=137F SS=137F CS=137F
IP=0308 NV UP EI PL NZ NA PE NC
137F:0308 0000 ADD [BX+SI], AL
DS:0089=42

*

On adding AX and BX, the parity changes from odd (PO) to even (PE).
Seeing the last 8 bits of result stored in AX, 0A7B,
7B ≡ 0111 1011
Total no. of 1's are even. Hence, the change.

Q.2. Write an assembly language program to move 7812 to CX.

- a cs:400
137F:0400 mov CX, 7812
137F:0403
- t=400

DX=0000

AX=0A7B BX=0089 CX=7812 SP=FFEE BP=0000
SI=0000 DI=0000 DS=137F ES=137F SS=137F
CS=137F IP=0403 NV UP EI PL NZ NA PO CY
137F:0403:0403 0000 ADD [BX+SI], AL
DS:0089=4C

Q.3. Write an assembly language program to move the data stored in address location 4050 to bh register.

```
-e4050
137F:04050 AA:bb
-a cs:300
137F:0300 mov bh, [4050]
137F:0304
-t=300

AX=0000 BX=BB00 CX=0000 DX=0000 SP=FFEE BP=0000
SI=0000 DI=0000 DS=137F ES=137F SS=137F CS=137F
IP=0304 NV UP EI PL NZ NA PO CY
137F:0304 0000 ADD [BX+SI],AL
                    DS:BB00=00
```

* Different addressing modes used in this lab:
  1. Register addressing
  2. Immediate addressing
  3. Direct addressing

1. Register addressing
   Most 8086 instructions can operate on the 8086's general purpose register set.

By specifying the name of the register as an operand to instruction, you may access the contents of that register.

mov AX, BX : copies value from BX into AX
mov DL, AL : copies value from AL into DL
mov SI, DX : copies value from DX into SI
mov AX, AX : yes, this is legal.

2. Immediate addressing

Source operand can be immediate data (8 bit or 16 bit data).

mov AH, 12H
mov CX, 1234H.

(note that changes done to 8 bits of the 16 bit register does not affect the other 8 bits of the register)

3. Direct addressing

The direct addressing, directly specifies the address as of one of the operand

mov AL, [2088H]

loads the AL register with a copy of the byte at memory location.

4.3.2013. LAB-2 Continued

Q.4. Write an assembly language program to mov 35 to AX and 25 to BX respectively. Subtract and store the result in AX.

```
-a cs:100
13A4:0100 mov ax,35
13A4:0103 mov bx,25
13A4:0106 sub ax,bx
13A4:0100
-t=100 3

AX=0035 BX=0000 CX=0000 DX=0000 .....
IP=0103 NV UP EI PL NZ NA PO NC
13A4:0103 BB2500 MOV BX,0025

AX=0035 BX=0025 - - - - - - - -
IP=0106 NV UP EI PL NZ NA PO NC
13A4:0106 29D8 SUB AX,BX

AX=0010 BX=0025 - - - - - - - -
IP=0100 NV UP EI PL NZ NA PO NC
13A4:0100 0000 ADD [BX+SI],AL
                    DS:0025=FF
```

Here, there are no flag changes. The parity (PO) is odd finally as well.

This is because, AX = 0035. = 0011 0101, for last 8 bits, number of 1's is odd.
Similarly, for AX = 0010 = 0001 0000.
So, no. of 1's is again odd.
So, no change in parity.

Final result is got as: AX = 0035
BX = 0025
Sub
finally AX = 0010

Q.5. Write an assembly language program to move the data stored in address location 3050 to bh register and add with AL register.
Where AL = 20 and memory location 3050 = 30.

```
-e3050
13A4:3050 00.30
-a cs:200
13A4:0200 mov al,20
13A4:0202 mov bh, [3050]
13A4:0206 add al, bh
13A4:0208
-t =200 3
```

AX=0200 BX=0025 - - - - - - - -
- - IP=0202 NV UP EI PL NZ NA PO NC
13A4:0202 8A3E5030 MOV BH, [3050]
DS:3050 = 30.

AX=0020 BX=3025 - - - - - - - -
- - IP=0206 NV UP EI PL NZ NA PO NC
13A4:0206 00F8 ADD AL, BH.

AX=0050 BX=3025 - - - - - - - -
- - IP=0200 NV UP EI PL NZ NA PE NC
13A4:0200 0000 ADD [BX+SI], AL
DS:3025 = 00

Here, as asked, the value 30 is stored in the address 3050.
Performing the addition operation, we find flag changes from PO to PE.
We had BX = 3025 = 0010 0101
The number of 1's for last 8 bits is even.
Hence, the parity.

Q.6. From the 2nd program data, write an assembly language program to move the data stored in [BX] to AL register.

-e 3025
13A4:3025 50.30
-a cs:200
13A4:0200 mov al, [bx]
13A4:0202
-t = 200 1

AX=0050 BX=3025 _ _ _ _ _
_ _ _ _ IP=0202 NV UP EI PL NZ NA PE NC
13A4:0202 8A3E5030 MOV BH, [3050]

AX=0030 BX=3025 _ _ _ _ _ _
_ _ _ IP=0206 NV UP EI PL NZ NA PE NC
13A4:0206 00F8 mov AL, [BX]
DS:3050 = 30.

In this question, the value stored in the address 3025 (=30) gets stored or moved to AL. So, parity doesn't change from even ~~to odd~~.
we had ~~3025~~ BX=3025 = 0010 0101
Seeing no. of 1's, its even.
Now, AX=0030 = 0011 0000.
So, no. of 1's is again even.

* Addressing mode used in Q.6 :

- Register Indirect Addressing

The 80X86 CPUs let you access memory indirectly through a register using the register indirect addressing modes. Register indirect addressing allows data to be addressed at any memory location through an offset address held in any following register:
BX, BP, SI, DI.

```
mov AL, [BX]
mov AL, [BP]
mov AL, [SI]
mov AL, [DI]
```

The [BX], [SI] and [DI] modes, use data segment by default. The [BP] addressing mode uses the stack segment (SS) by default. Segment Overriding can also be done.

4/3/2013

Sample Program 1 : output

AX = 1000 ... DS = 1000 .... ES = 1000 IP = 001E
NV UP EI NG NZ NA PO NC

The output shows how the mov instruction transfers the data between registers.

The value of the memory, data 2 i.e, 1000, goes to AX, which is moved to ES. This is confirmed from the got output.

The memory locations - data 1, data w, data b, data 3, data 4 were not used in program, output.

18.3.2013

Lab-3

MASM: An Overview

Example: Sample Program 1.

```
.MODEL SMALL
.386 data
data 1 db 23h
data 2 dw 01000h
data w dw 50 dup(?)
datab db 10 dup(?)
data 3 db 'a'
data 4 db 11000100b

.CODE
.STARTUP
mov ax, data 2
mov ds, ax
mov es, ax
.exit
end
```

Sample Program 2:

Initial:

AX=0000 BX=FF90 CX=0000 DX=12A3 SP=FF90 BP=0000
SI=0000 DI=0000 DS=12A3 ES=128G SS=12A3 CS=129C
IP=0016 FL=3086 NV UP DI NG NZ NA PE NC

Instruction number:	Output change:	
1)	AX=0C00	IP=0091
	↳ hexadecimal value of 0012	
2)	CX=0022	IP=001B
	↳ 0022: hexadecimal of 34	
3)	AX=0C22	IP=001D
4)	AX=0004	IP=0020
5)	BX=9999	IP=0024
6)	CX=270F	IP=0028
7)		IP=002C
8)	DX=1299	IP=0030
9)	CX=278B	IP=0034
10)	BX=1E8B	IP=0038
11)		IP=003B
12)		IP=003F
~~13)~~	~~BX=000F~~	~~IP=0042~~
~~14)~~	~~AX=0001~~	~~IP=0044~~
~~15)~~		~~IP=0047~~

Example : Sample Program 2

```
.model small
.data
data1 db 23
data2 dw 9999h
data3 dw 9999
array dw 01,02,03,04,05,06,07,08

.code
.startup
1) mov ah, 12
2) mov cl, 34
3) mov al, cl
4) mov ax, 04
5) mov bx, data2
6) mov cx, data3
7) mov data1, bl
8) mov dl, data1
9) mov cl, ds:[0520]
10) mov bx, ds:[0520]
11) mov ds:[0520], al
12) mov ds:[0520], cx
13) mov bx, offset array
14) mov ds:[0520], al
15) mov ds:[0520], cx
```

Instruction number:	Output Change:	
16)	BX = 000F	IP = 0042
17)	AX = 0001	IP = 0044
18)		IP = 0047
19)	CX = 0099	IP = 004A
20)		IP = 004C
21)	BX = 1000	IP = 004F
22)	SI = 3000	IP = 0052
23)		IP = 0056
24)	AX = 0099	IP = 005A
25)	AX = 0200	IP = 005E
26)	AX = 0001	IP = 0062
27)		IP = 0065
28)	BX = 000F	IP = 0068
29)	DX = 0001	IP = 006A
30)	AX = 0099	IP = 006C
31)	AX = 0300	IP = 006F
32)	BX = 0099	IP = 0072
33)	DX = 00A8	IP = 0075

```
16) mov bx, offset array
17) mov al, [bx]
18) mov [bp], dl
19) mov cx, [bp]
20) mov ax, [bx]
21) mov bx, 1000H
22) mov si, 3000.H
23) mov [bx+1000h], cx
24) mov ax, [si-1000h]
25) mov ax, array[di+1]
26) mov ax, array[di]
27) mov di, 0
28) mov bx, offset array
29) mov dx, [bx+di]
30) mov ax, [bp+di]
31) mov ax, [bx+di+3h]
32) mov bx, 3
33) mov dx, array[bx+di]
    • exit
    end
```

Q.1) (a) Output

AX = 000A BX=0002 CX=0003 DX=0004 SP=0000
BP=0000 SI=0000 DI=0000 DS=128C ES=128C
SS=129C CS=129C IP=0012 FL=3206
NV UP EI PL NZ NA PE NC.

The sum of AX, BX, CX, DX registers' value gets stored in AX.

(b) Output

AX=0018 BX=0005 CX=0007 DX=0009 SP=0000
BP=0000 SI=0000 DI=0000 DS=128C ES=128C
SS=129C CS=129C IP=010E FL=3216
NV UP EI PL NZ AC PE NC

The sum of the hexadecimal values stored in ~~AX~~, BX, CX and DX register gets added to AX as:

AX = AX + BX = 03 + 05 = 08
AX = AX + CX = 08 + 07 = 0F
AX = AX + DX = 0F + 09 = 18

Here, for bytes of data final value of AX, having 1 will be an auxiliary carry. Its corresponding flag is also high.

25.3.2013

Lab - 3
continued.

Q.1. (a) WAP to add the values stored in the 4 registers AX, BX, CX, DX.

(b) Perform the same operation using bytes of data using registers AL, BL, CL, DL.

(a)
```
.model tiny
.code
.startup
mov cx, 1
mov bx, 2
mov cx, 3
mov dx, 4
add ax, bx
add ax, cx
add ax, dx
.exit
end
```

(b)
```
.model tiny
.code
.startup
mov bl, 5
mov cl, 7
mov al, 3
mov dl, 9
add ax, bx
add ax, cx
add ax, dx
.exit
end
```

Q.2) (a) Output

AX = 0017 - - - - - - - - -
DS = 129E ES = 128C SS = 129E CS = 129C IP = 001E
FL = 3216 NV UP EI PL NZ AC PE NC

Here, AL being a register of 8 bits, will show auxiliary carry as high when there is a carry after 1st byte. So, as per output, $10 + 13 = (17)_{16}$ which gives AC.
Also, $(17)_{16} = (10001)_2$. So, no. of 1's are even.
Hence, PE flag is high.

(b) Output
AX = 08FC - - - - - - - - - -
DS = 129E ES = 128C SS = 129E CS = 129C IP = 001E
FL = 3206 NV UP EI PL NZ NA PE NC.

AX = 08FC is got by adding hexadecimal values of 1000 and 1300.
$(1000)_{10} = (03E8)_{16}$ and
$(1300)_{10} = (0514)_{16}$.

$$\begin{array}{r} 03E8 \\ +\ 0514 \\ \hline 08FC \end{array}$$

Also, $(08FC)_{16} = (1000\ 1111\ 1100)_2$. Seeing the last 8 bits, no. of 1's are 6 (= even). So, PE flag is high.

Q.2. WAP to add the contents of 2 memory locations num1, num2.
Consider the numbers to be (a) bytes (b) words

(a)
```
.model small
.data
num1 db 10
num2 db 13
.code
.startup
mov al, num1
mov
add al, num2
.exit
end.
```

(b)
```
.model small
.data
num1 dw 1000
num2 dw 1300
.code
.startup
mov ax, num1
add ax, num2
.exit
end
```

Q. 3) (a) Output.

AX = 000A .... DS = 129E ES = 128C SS = 129E CS = 129C
IP = 0029 FL = 3206 NV UP EI PL NZ NA PE NC

Here, AX holds the sum of data 1, data 2, data 3 and data 4. (= 1 + 2 + 3 + 4 = 0A)
This sum is stored in the memory location, SUM.
To access the location of sum, from the output window, note the value of DS and address associated with the last instruction where value of AL is going (i.e BYTE PTR [0012], AL)
Here, DS = 129E. So, in the memory tab, editing the location as, 129E : 0812 gives location of SUM. The value of sum gets shown after the address as:
129E : 0012 00 0A .....

(b) Output.

AX = 2710 .... DS = 129E ES = 128C SS = 129E CS = 129C
IP = 0029 FL = 3202 NV UP EI PL NZ NA PO NC
The last instruction on output window:
MOV WORD PTR [0016], AX.
So, going to location 129E : 0016, we get output of SUM as: 129E : 0016 27 10 .....

Q. 3) WAP to add the contents of 4 memory locations data 1, data 2, data 3, data 4.
Consider the numbers to be (a) Bytes (b) words

(a)
```
.model small
.data
data1 db 1
data2 db 2
data3 db 3
data4 db 4
sum db 0
.code
.startup
mov al, data1
add al, data2
add al, data3
add al, data4
mov sum, al
.exit
end
```

(b)
```
.model small
.data
data1 dw 1000
data2 dw 2000
data3 dw 3000
data4 dw 4000
Sum dw 0
.code
.startup
mov ax, data1
add ax, data2
add ax, data3
add ax, data4
mov sum, ax
.exit
end
```

Q. 1) (a) Output, finally:

AX = 0028 BX = 0013 CX = 0000 .... DS = 129E ES = 128C
SS = 129E CS = 129C IP = 0027 FL = 3202
NV UP EI PL NZ NA PO NC

Seeing DS (= 129E) and the offset address of data1 (from the instruction mov BX, 000C), we locate the memory location values as:

129E:000C 05 0A 0F 14 19 1E 23

Each of the array elements are the hexadecimal values of multiples of 5.

(b) Output, finally:

The output for a word data remains same as that of a byte, with the difference of the way, the data is added to the array, data1 (2 + 2 bytes = 16 bits of data updated everytime with every operation).

Hence, for the last instruction, data will be stored as

129E:000C ~~05~~ 05 0A 0F 14 19 1E 23 00

23 00 : It is the last 16 bit data corresponding to $(0023)_{16} = (35)_{10}$, which is what is expected.

1.4.2013

Lab - 3

- continued 2.

Q.1. WAP to create an array of size 7 and load this array with data elements which are multiple of 5. Consider (a) bytes (b) words.

(a)
```
.model small
.data
data1 db 7 dup(?)
.code
.startup
mov ax, 5h
mov d, 7
mov bx, offset data1
l1:
mov [bx], ax
add ax, 5h
inc bx
loop l1
.exit
end
```

(b)
```
.model small
.data
data1 dw 7 dup(?)
.code
.startup
mov ax, 5h
mov cx, 7
mov bx, offset data1
l1:
mov [bx], ax
add ax, 5h
inc bx
loop l1
.exit
end
```

Q.2) (a) Output

AX=0009 BX=0010 CX=0000 DX=129E .... DS=129E CS=129C
IP=0025 FL=3212 NV UP EI PL NZ AC PO NC

AX: holds final result of alternate elements addition
= 1+3+5 = 0009.

BX: holds final address of last element of data1.
Address of memory location is got as:
DS: Offset address of data1 = 129E:000A.

Memory location output: 129E:000A 01 02 03 04 05 ....

CX: A counter which runs 3 times, finally coming to 0.

(b) Output

AX=0009 BX=0018 CX=0000 ....DS=129E CS=129C
IP=0028 FL=3206 NV UP EI PL NZ NA PE NC

In the output for word data input, the function/role and/or the value stored in the registers AX, BX and CX remains the same. The difference comes in the way, the word data is present in memory location, and, its used in the program:

129E:000C 01 00 02 00 0300 0400 0500 ..

The data has been stored as 0001 → 01 00.

000C is the offset address of data1 giving rise to show of memory location contents of data1.

Q.2) WAP to create an array of size 5 and find the sum of alternate elements of this array. Consider (a) Bytes (b) Word.

(a)
```
.model small
.data
data1 db 1,2,3,4,5
.code
.startup
mov bx, offset data1
mov ax, 0H
mov cl, 3
l1:
add al, [bx]
inc bx
inc bx
loop l1
.exit
end
```

(b)
```
.model small
.data
data1 dw 1,2,3,4,5
.code
.startup
mov bx, offset data1
mov ax, 0H
mov cx, 3
l1:
add ax, [bx]
inc bx
inc bx
loop l1
.exit
end
```

Q.1 : Output :

AX = 0105 BX = FF02 DS = 129E ES = 128C
CS = 129C IP = 0027 NV UP EI NG NZ NA PO NC.

BL gets stored with 02 as shown in output. The quotient of division gets stored in AL and remainder gets stored in AH. hence, AX has the value of 0105 with 01 as remainder and 05 as quotient.

The values moved to ANSQ and ANSR memory locations can be seen by seeing the DS: offset address.

From code view,
for ANSQ : 129E :000F
ANSR : 129E : 0010

15.4.2013

Lab - 3.
- continued 3.

Q.1. WAP to divide unsigned byte contents of memory location NUMB by NUMB1. Store the quotient and remainder in locations ANSQ and ANSR respectively.

```
.model small
.data
numb  dw  11
numb1 db  2
ansq  db  0
ansr  db  0
      .code
      .startup
      mov ax, numb
      mov bl, numb1
      div bl
      mov ansq, al
      mov ansr, ah

.exit
end.
```

Q.2 : Output :

$(-100)_{10} = (FF9C)_{16}$

$-100/9 = (-11.11)_{10} = (FFF5)_{16}$.

AX = FFF5 BX = FF09 ... DS = 129E ES = 128C -- CS = 1296
IP = 001E NV UP EI NG NZ NA PO NC.

Clearly, AX has the final result FFF5. Value FF9C, stored in dI can be got from the address 129E : 0002.

Q.3 : Output :

$1+2+3+4+5+6+7+8+9+10 = (55)_{10} = (37)_{16}$.

AX = 0037 BX = 001A CX = 0000 - - - - DS = 129E
CS = 129C IP = 0022 NV UP EI PL NZ NA PO NC.

The final value of sum gets stored in AL. Value of count decreased from 20 to 0 at the end.

Q.2. WAP to divide -100/+9.

```
.model small
.data
d1 dw -100
d2 db 9
.code
.startup
mov ax, d1
mov bl, 9
idiv bl
.exit
end
```

Q.3. WAP to find the sum of an array of size 10 words.

```
.model small
.data
d1 dw 1,2,3,4,5,6,7,8,9,10
.code
.startup
mov cx,20
mov bx, offset d1
l1:
add al,[bx]
inc bx
loop l1
.exit
end
```

Q.4 : Output :

AX = 0505 BX = 0020 CX = 0000 DX = 120A .. DS = 129E
CS = 129C IP = 0028 NV UP EI PL NZ ~~NA~~ AC PO NC.

The sum of elements of array =
$1+2+3+4+5+6+7+8+9+10 = (55)_{10}$

To find average, $\frac{\text{Sum}}{\text{no. of elements}} = \frac{55}{10}$

$= 5.5$

or 5 quotient
5 remainder.

AH : stores remainder } $\Rightarrow$ AX = 0505, as shown.
AL : stores quotient }
The count decreases from 20 to 0.

The auxiliary carry bit became high during the operation $\because$ addition was going on in 8 bits (AL).

Q.4. WAP to find average of 10 elements stored in an array.

```
.model small
.data
d1  dw  1,2,3,4,5,6,7,8,9,10
.code
.startup
    mov  cx, 20
    mov  bx, offset d1
    mov  ax, 0
l1:
add  al, [bx]
inc  bx
loop l1

mov  dl, 10
div  dl

.exit
end
```

Output : Q.1) (a) Multiply 5 by 17

AX = 0055 BX = 0005 .... DS = 1299 CS = 1298
IP = 001A NV UP EI PL NZ NA PE NC

$17 \times 5 = (85)_{10} = (55)_{16}$.

It is stored in AX, as shown.

• 386 is required while using SHIFT instruction more than once.

(b) Multiply 5 by 30

AX = 0096 BX = 001E .... DS = 1288 CS = 1298
IP = 010A FL = 3216 NV UP EI PL NZ AC PE NC

$30 \times 5 = (150)_{10} = (96)_{16}$.

It is stored in AX.

While multiplication, there is a carry that comes after 8 bits. So, AC bit becomes high.

The logic is: 30 is stored in AX.
Then, with AX left shifted by 2, we get 30 X 4.
Finally, (30 X 4) + 30 happens.
So, 30 X 5 is the result.

22.4.2013

Lab - 3
- continued 4.

Q.1). Write an assembly language program to multiply two numbers without using MUL instruction
(Multiply 5 by 17 and 30)

(a)

```
.model small
.code
.386
.startup

mov ax, 5
mov bx, ax
shl ax, 4
add ax, bx

.exit
end
```

(b)

```
.model tiny
.code
.386
.startup

mov ax, 30
mov bx, ax
shl ax, 2
add ax, bx

.exit
end
```

Output : Q.2)

AX = 0083 BX = 0048 ... DS = 1288 CS = 1298
IP = 0108 NV UP EI PL NZ NA PO NC

EF - 6C = 83.

This result is stored in AX, as shown.

Output : Q.3)

AX = 0070 BX = 64E0 DS = 129A CS = 1298 IP = 0025
FL = 3212 NV UP EI PL NZ AC PO NC

$100 + 12 = (112)_{10} = (70)_{16}$.

This value is stored in AX.
To check the value stored in 3050 location, go to :

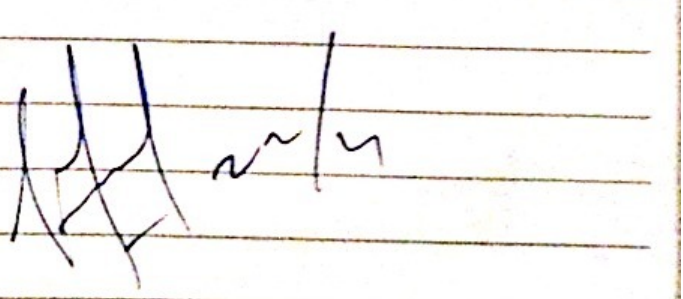
129A : 0BEA ,
which is
DS : offset address.

Q.2) Write an assembly language program to mov AX EF and mov BX 6C. SUB BX from AX and store result in AX register.

```
.model tiny
.code
.startup

mov ax, 0EFh
mov bx, 06Ch
sub ax, bx

.exit
end
```

Q.3) WAP to move data stored in address location 3050 to bh register and add with AL register. ([3050] store 100 and AL store 12)

```
.model small
.code
.startup
mov ds:[3050], 100
mov al, 12
mov bh, ds:[3050]
add al, bh
.exit
end.
```



Output : Q.1)

AX=0090 BX=000C SP=0000 IP=0103 DS=128C
NV UP EI PL NZ NA PE NC

Final ~~result~~ result on multiplication should be
$12 \times 12 = (144)_{10} = (90)_{16}$.

That is stored in AX, as shown.

1.5.2013

Lab - 3

- continued 5

Q.1) WAP to perform 16 bit multiplication using procedure.

```
.model tiny
.code
.startup
call mult
.exit

mult proc near
mov ax, 12
mov bx, 12
mul bx
ret
mult endp
end
```

Output : Q2)

AX = 2D BX = 0005 CX = 0003 DX = 0000 SP = FFE0
DS = 129E IP = 001A
FL = 3206 NV UP EI PL NZ NA PE NC

BX = BX + CX = 2 + 3 = 5
AX = BX * (AX - CX)

= BX * (12 - 3)
= 5 * 9
⇒ AX = $(45)_{10}$ = $(2D)_{16}$

This gets stored in AX, as shown.

Q.2) WAP to perform ADD, SUB and MUL using procedure.

```
.model small
.code
.startup
call logic
.exit

logic proc near
mov ax, 12
mov bx, 2
mov cx, 3
add bx, cx
sub ax, cx
mul bx
ret
logic endp
end
```

(Q.3) Output :

AX = 0037 BX = FFE0 CX = 0000 SI = 0012 DS = 13B0
IP = 0024 NV UP EI PL NZ NA PE NC

The sum of contents of table = 1+2+3+4+5+6+7+8+9+10

$= (55)_{10}$

$= (37)_{16}$

It gets stored in AX, as shown. The counter register, CX gets decremented from $(10)_{10}$ [$(A)_{16}$] to 0.

(Q3) WAP to add 10 bytes of memory array called TABLE using MACRO.

```
.model small
.data
table db 01,02,03,04,05,06,07,08,09,10
.code
Sums macro NUM, count
  local , addn
  mov SI, offset num
  mov cx, count
  mov AL, 0
  addn :
  add AL, [SI]
  inc SI
   loop addn
  end m
.startup
 Sums table ,10
.exit
end
```

Q.1) Output

Procedure

AX = 000A BX = 0019 SP = FFEO IP = 001A FL = 3216
NV UP EI PL NZ AC PE NC.

$AX = (35)_{10} = (23)_{16}$.
$BX = (25)_{10} = (19)_{16}$: gets stored in BX

Now,
$AX - BX = 35 - 25 = (10)_{10} = (000A)_{16}$

000A is stored in AX, as shown.

Macro

AX = 000A BX = 0019 IP = 001F FL = 3216
NV UP EI PL NZ AC PE NC.

The format/syntax is different as of procedure, but output ~~is~~ is same.

6.5.2013

Lab - 3
- continued 6

Q.1. Write a procedure and macro for the following, to mov 35 to AX and 25 to BX respectively, subtract and store result in AX.

Procedure

```
.model small
.code
.startup
call move
.exit

move proc near
mov ax, 35
mov bx, 25
Sub ax, bx
ret
move endp
end
```

Macro

```
.model small
.code
Sums macro a, b
mov ax, a
mov bx, b
Sub ax, bx
end m
.startup

Sums 35, 25
.exit
end
```

Q.2) Output

Procedure

AX = 0032 BX = 1EE0 SP = FFE0 IP = 001A DS = 129E
FL = 3212 NV UP EI PL NZ AC PO NC

Value of 30+20 = $(50)_{10}$ = $(32)_{16}$.

This gets stored in AX.
BH has value of $(1E)_{16}$ = $(30)_{10}$.

Also, to access the memory location of 3050, we go to 129E : 0BEA.

Macro

AX=0032 BX=1EE0 DS=129E IP=0025 FL=3212
NV UP EI PL NZ AC PO NC

The same procedure is applied here also, with the registers having same values.
Only the sequence has changed.

Q2. Write a procedure and macro to move the data stored in address location to bh register and add with AL register.
(where AL = 20 & memory location 3050 = 30)

Procedure

```
.model small
.code
.startup
call logic
.exit
logic proc near
mov al, 20
mov ds:[3050], 30
mov bh, ds:[3050]
add al, bh
ret
logic endp
end
```

6/5/2013

Macro

```
.model small
.code
sums macro b
add al, b
endm
.startup
mov al, 20
mov ds:[3050], 30
mov bh, ds:[3050]
sums bh
.exit
end
```

Q1 : Output

AX = 0161 ... SP = 0000 ... IP = 0103 FL = 3202
NV UP EI PL NZ NA PO NC

Input given : a

So, $a = (0097)_{ASCII} = (0061)_{16}$.

First 01 goes to AH
next 61 goes to AL (value from INT 21h)
So, AX = 0161, as shown.

The output is displayed on the screen.
('a' with echo)

13.5.13

Lab - 3
- continued 7

Q.1: WAP to enter character using DOS interrupts with echo.

Ans:

```
.model tiny
.code
.startup
        call sums
.exit

sums proc near
mov ah, 1
int 21h
or al, al
jnz l1
int 21h
stc
l1 : ret
sums endp
end
```

Q.2) Output:

AX = 0761 .... SP = 0000 --- IP = 0103 FL = 3202
NV UP EI PL NZ NA PO NC

We give 07, stored in AH.
& value entered from keyboard was the character → a = $(97)_{ASCII} = (61)_{16}$.

This gets stored in AL.
So, AX = AH AL = 0761., as shown.

The value entered (a) isn't shown on the screen.

Q.2) WAP to enter a character using DOS language interrupts without echo.

```
.model tiny
.code
.startup
        call sums
.exit
        sums proc near
        MOV AH, 07
        MOV DL, 0FFH
        int 21h
        JE SUMS
        or al, al
        jnz L1
        int 21h
        stc
        L1 : ret
        sums endp
end
```

Q.3) Output :

Output is got by directly typing the name of program (instead of going into Code View).

```
masm  p1.asm
link  p1
p1
```

; p1 : name of program.

Output : where r u?

Note : $ sign after program is necessary.

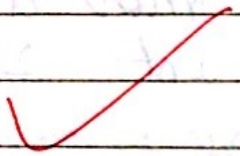
✓

Q.3) WAP to input line from standard input.

```
.model small
.data
mes db 13,10,'which year r u?$!'
mes1 db 10,10
.code
.startup
        mov ah,9
        mov dx, offset mes
        int 21h

        mov ah,0ah
        mov dx, offset mes1
        int 21h
.exit
end
```

Q.4) Output :-

On typing the name of the program, the screen gets cleared.
The size of clearance is 25 * 80.



Q.4) WAP to clear monitor using BIOS interrupt.

```
.model small
.code
.startup
cld
  mov ax, 0b800h
  mov es, ax
  mov di, 0h
  mov cx, 25*80
   mov ax, 0720h
   rep stosw
.exit
end
```

Q.5) Output:

Output is got by directly writing name of program. (after link).

p5

Output: Akshansh
↵ (enter key)
Welcome Akshansh!

(Akshansh & Welcome Akshansh are typed on the screen.
They become visible side by side)

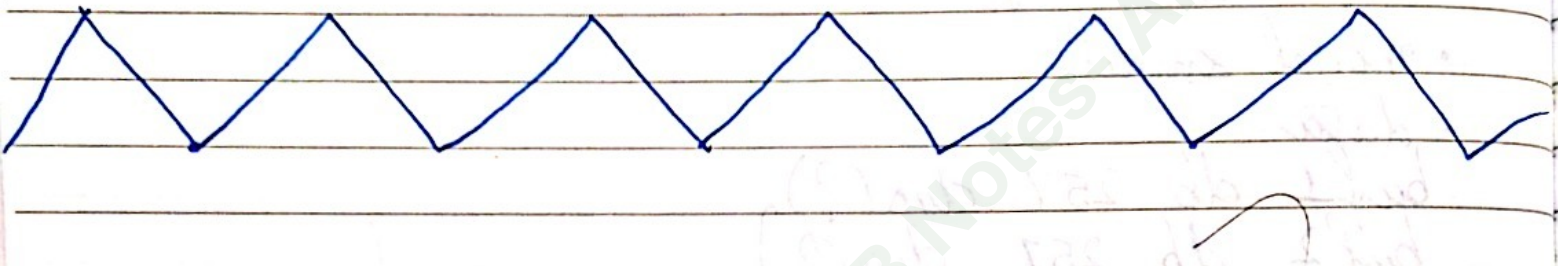
Note :- Akshansh gets over writted by Welcome Akshansh (∵ screen spacing 25*80 isn't given)

Q5) WAP to enter a name and then print welcome along with the name.

```
.model small
.data
buf1 db 257 dup(?)
buf2 db 257 dup(?)
.code
.startup
        mov buf1, 255
        mov dx, offset buf1
        call line
mov buf2, 255
mov dx, offset buf2
call line
.exit

line proc near
mov ah, 0ah
int 21h
ret
line endp
end
```

13/5/13

Output on the OHP:-



✡ MPT 85 Program for Triangle Wave:

ADDRESS	CODES	MNEMONICS
8000	31 C2 FF	LXI SP, FFC2
8003	3E 80	MVI A, 80H
8005	D3 2B	OUT 2BH
8007	3E 00	MVI A, 00H
8009	D3 28	OUT 28H
800B	3C	INR A
800C	FE FF	CPI FFH
800E	C2 09 80	JNZ 8009
8011	3D	DCR A
8012	D3 28	OUT 28H
8014	FE 00	CPI 00H
8016	C2 11 80	JNZ 8011
8019	C3 07 80	JMP 8007

Output on the CRO :-

8085 Program for Square Wave

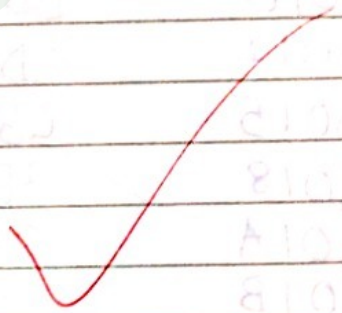
ADDRESS	CODES	MNEMONICS
8000	31 C2 FF	LXI SP, FFC2
8003	3E 80	MVI A, 80
8005	D3 2B	OUT 2B
8007	3E 00	MVI A, 00H
8009	D3 28	OUT 28
800B	CD 18 80	CALL DELAY
800E	3E FF	MVI A, FFH
8010	D3 28	OUT 28H
8012	CD 18 80	CALL DELAY
8015	C3 07 80	JMP 8007
8018	3E FF	MVI A, FFH
801A	0D	DCR C
801B	C2 1A 80	JNZ 801A
801E	C9	RET

Output :-

CP Lab

CP Lab

CP Lab



CP Lab

CP Lab

20.5.2013

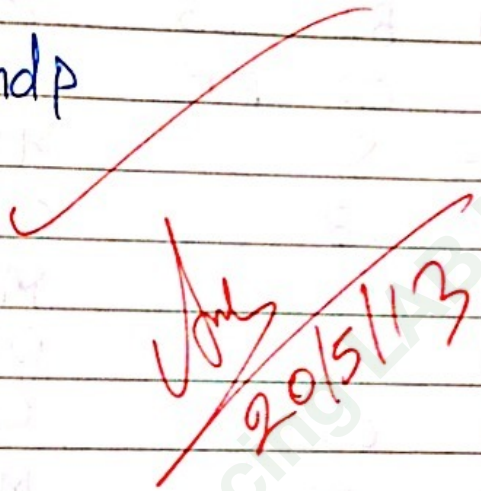
Lab - 8.9
- ~~continued &~~

Q. WAP using procedure to display CP LAB at
Top Right
Top Left
Center
Bottom Right
Bottom Left of screen.

```
.model small
.data
str db 'CP Lab $'
.code
.startup

mov dh, 21
mov dl, 38
call position
mov dx, offset str
call display

mov dh, 45
mov dl, 72
call position
mov dx, offset str
call display
```

```
mov dh, 0
mov dl, 0
call position
mov dx, offset str
call display

mov dh, 45
mov dl, 0
call position
mov dx, offset str
call display

mov dh, 0
mov dl, 72
call position
mov dx, offset str
call display

exit

position proc
mov ah, 2
mov bh, 0
int 10h
ret
position endp
```

```
display pro
mov ah, 9
int 21 h
ret
display endp
end z
```



* Program to rotate the stepper motor in clockwise direction

Label	Address	B1	B2	B3	Mnemonic
	8000	31	C2	FF	LXI SP, FFC2H
	8003	3E	80		MVI A, 80H
	8005	D3	2B		OUT 2BH
Loop	8007	3E	50		MVI A, 50H
	8009	D3	28		OUT 28H
	800B	CD	00	81	CALL Delay
	800E	3E	90		MVI A, 90H
	8010	D3	28		OUT 28H
	8012	CD	00	81	CALL Delay
	8015	3E	A0		MVI A, A0H
	8017	D3	28		OUT 28H
	8019	CD	00	81	CALL Delay
	801C	3E	60		MVI A, 60H
	801E	D3	28		OUT 28H
	8020	CD	00	81	CALL Delay
	8023	C3	07	80	JMP Loop

Note :- The delay routine (the speed of output) will also be required to be ~~executed~~ entered before executing the above program.

Output : Stepper motor rotates in clockwise direction.

✡ Program to rotate the stepper motor in anti-clockwise direction.

Label	Address	B1	B2	B3	Mnemonic
	8000	31	C2		LXI SP, FFC2H
	8003	3E	80		MVI A, 80H
	8005	D3	2B		OUT 2BH
Loop	8007	3E	60		MVI A, 60H
	8009	D3	28		OUT 28H
	800B	CD	00	81	CALL Delay
	800E	3E	A0		MVI A, A0H
	8010	D3	28		OUT 28H
	8012	CD	00	81	CALL Delay
	8015	3E	90		MVI A, 90H
	8017	D3	28		OUT 28H
	8019	CD	00	81	CALL Delay
	801C	3E	50		MVI A, 50H
	801E	D3	28		OUT 28H
	8020	CD	00	81	CALL Delay
	8023	C3	07	80	JMP Loop.

The circles show the change done in previous program to reverse the direction from clockwise to anti-clockwise.

To enter value :- Set Address > 8000 >> Next >> Reset

To get o/p or execute :- Reset > Go Previous > 8000 > Exe program

Output : Stepper motor rotates in anti-clockwise direction.

* Program to be written before executing the above programs :-

To set the delay of the stepper motor, i.e, the speed at which it'll rotate.

Label	Address	B1	B2	B3	Mnemonics
Delay	8100	1E	FF		MVI E, FF
	8102	CD	0A	81	CALL DLY
	8105	1D			DCR E
	8106	C2	0A	81	JNZ DLY
	8109	C9			RET
DLY	810A	00			NOP
	810B	00			NOP
	810C	0E	00		MVI C, 00H
Loop 2	810E	3A	50	80	LDA 8050H
	8111	3D			DCR A
	8112	C2	11	81	JNZ loop2
	8115	0D			DCR C
	8116	C2	0E	81	JNZ loop2
	8119	C9			RET

Note : For changing the speed of rotation, go to the address : 8050 :-

Reset > Go Prev > 8050 > ( ) > Next > Reset

↳ It can have the value from 02 (max. speed) to 20 (minimum speed).