Assignment #01

Name: Muhammad Imran Butt
Reg no: L1F24BSCS0546
Section # C8
Course: COAL

Question # 01

Values of register before execution:

DS: 0xA1B9, DI: 0x0032, BP: 0x110

SI: 0x0024, SS: 0x3C80

Ax: 0xABCD, Bx: 0x12C2

Cx: 0xFEED, Dx: 0x2032, ES: 0x1AB0

(a) Execute instructions and update registers, memory and calculate physical address.

i) Instructions:

	Physical Address
MOV [BX+SI], CX	A2E76
MOV [BP+DI+0x0016], DX	3C958
MOV AX, [BX+DI]	A2E84
MOV [BX+SI+9D], CX	A2E7F
MOV DX, DS:[BP+SI+11BCH]	A2E80

Registers after execution:

~~AX = [illegible]~~
~~DX = [illegible]~~

AX = 2312
DX = F0FE

Memory after execution:

Addresses	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0x1AB0:0x0120																
0x1AB0:0x0130				50	56	78	AB									
0x1AB0:0x0140																
0x3C80:0x0150								20	32	20					AB	CD
0x3C80:0x0160																
0xA189:0x12E0							ED	FE	67						40	ED
0xA189:0x12F0	FE	F0	50	30	12	23	45									
0xA189:0x1300															EE	89

b) Status of flag after execution:

MOV AL, 0x80

Add, AL, AL

Performing Binary addition

$(80)_{16} = (1000\ 0000)_2$

AL = AL + AL

1000 000 (or 80)

+ 1000 000 (or 80)

$(1\ 0000\ 000)_2 = (100)_{16}$

result $(0x100)_{16}$ =

→ Since there's a carry in binary answer so carry flag (CF) will be set 1 and only lower 8-bits will be store in AL and the carry will go to carry Flag (CF).

→ Zero Flag (ZF) will be set to set since all the 8 bits are 0.

→ As both operand's sign was 1 (same) but the resultant (MSB) is 0 not the same then it means there's overflow so Overflow Flag is set to 1

OF = 1

→ As the number of 1's in lower 8-bit is 0 zero which is considered as even number so count of ones is even and when the count of 1's is even the "parity flag" PF is set to 1.

PF = 1

→ As there's no carry on the nibble's (4th) bit so Auxiliary Flag AF is set to zero : AF = 0

Final Flags:

MOV AL, 0x80	CF	ZF	SF	OF	PF	AF
ADD AL, AL	1	1	0	1	1	0

Question # 2

(a) Direct Addressing mode

```
.model small
.stack 100h
.data
.code
mov ax, 07DEH
mov ds, ax
mov [0001H], 0X0BAD
```

(b) Register indirect addressing mode

```
.model small
.stack 100h
.data
.code
mov ax, 07DEh
mov ds, ax
mov bx, 0001h
mov [bx], 0X0BAD
```

(C) Register relative base plus index addressing mode

```
.model small
.stack 100h
.data
.code
mov ax, 07DEh
mov ds, ax
mov bx, 0000h
mov SI, 0000h
mov [bx + SI + 1h], 0x0BAD
```

Question # 03

1) Mov 0x1234, AX , **Invalid**

Reason: Because we can't move the value of ax into a constant literal. so in order to move data from an register to a memory address we should round it with brackets like: Mov [ox1234], ax

2) Mov, ax, [cx] , **Invalid**

Reason: Because assembler only allows base and index register as memory address inside brackets but cx is not one of them.

3) Mov al, [bl + 4] , **Invalid**

Reason: Because the BL is of 8-bit and only 16-bit (base/Index) registers can hold memory address. so BL isn't allowed.

4) Add Dx, [BX + Dx + 2] , **Invalid**

Reason: Dx register is not allowed in memory addressing inside brackets only base/Index registers are allowed.

5) Mov, CS, AX , Invalid

Reason: According to rules CS register can't be in a destination operand.

6) Add num1, num2 , invalid

Reason: both operands can't be memory locations at the same time. At least one operand must be a register or an immediate value. CPU (8086 processor) can't perform memory to memory arithmetic directly

7) Mov DS, 0X720, Invalid

Reason: Because we can't assign immidiate value directly to a segment register.

8) Add cx, BL , Invalid

Reason: Both operand cx, bL have size difference so it's not allowed.

9) mov BL, SI , Invalid

Reason: Both operand BL, SI have size difference so it's not allowed.

10) MOV AX, [BP+BX] , Invalid

Reason: Two Base registers can't be used in the [] at the same time.

11) Mov Dx, [SI+DI], Invalid

Reason: Two index registers can't be used in the [] at the same time.

12) Add bx, [number + cx] , **Invalid**

Reason: Cx register is not allowed here in [] only base/Index registers.

13) Mov [0x12345], BL , **valid**

Reason: If we write [0x12345] without specifying the size, the assembler infers the size from the register we are using.

14) Mov Ax, [Bx + SI + 4] , **valid**

Reason: It uses Base + Index + displacement addressing which is allowed.

→ END ←

Q.S 1 calculations working:

i) Bx + SI DS: A1B9

```
  12C2
+ 0024
------
  12E6
```

DSx10h = A1B90

```
  A1B90
   12E6
--------
  A2E76
```

23-16 = 7

Physical address: A2E76

ii) BP + DI + 0016

```
  0110
  0032
------
  0142
  0016
------
  0158
```

As SS = 3C80

SSx10h = 3C800

So

```
  3C800
   0158
--------
  3C958
```

Physical address: 3C958

iii) BX + DI =

```
  12C2
  0032
------
  12F4
```

as DS x 10h = A1B90

so

```
  A1B90
    12F4
--------
  A2E84
```

15+9 = 24

24 - 16 = 8

Physicall address = A2E84

iv) BX + SI + 9D as 9D = 09h

as we calculate BX + SI in 2ird instruction so

BX + SI = 12E6

so

```
  12E6
  0009
------
  12EF
```

DS x 10h = 12B90

so

```
  A1B90
    12EF
--------
  A3E7F
```

14+9 = 23

23 - 16 = 7

Physical address - ~~12EF~~ A2E7F

(v) BP + SI + 11BCH

BP + SI = 0110

```
  0024
------
  0134
  11BC
------
  12F0
```

16 - 16 = 0

as DS x 10h = A1B90 so,

```
  A1B90
    12F0
--------
  A2E80
```

15+9 = 24

24 - 16 = 8

Physical address: A2E80