1.1. Perform Addition, Subtraction, Multiplication, and Division of two 16-bit numbers using immediate addressing mode and store the results using direct addressing mode.

```
MOV AX,0028H
MOV CX,AX
MOV BX,0015H
ADD AX,BX
MOV [0300H],AX
MOV AX,CX
SUB  AX,BX
MOV [0302H],AX
MOV AX,CX
MUL BX
MOV [0304H],AX
MOV AX,CX
DIV BX
MOV [0306H],AX
MOV [0308H],DX
HLT
```

1.2. Perform the following operations on two 8-bit data (data1, data2) given in memory locations and store the result in another memory location using indirect addressing mode.
i. Swapping of nibble of data1

```
MOV S,0300H
MOV AX,[SI]
MOV BX,AX
SHR AX,08H
SHL BX,08H
XOR AX,BX
INC SI
INC SI
MOV [SI],AX
HLT
```

1.2. Perform the following operations on two 8-bit data (data1, data2) given in memory locations and store the result in another memory location using indirect addressing mode.

ii. = (**data1** and **data2**) or (**data1** xor **data2**)

```
MOV SI,0300H
MOV AL,[SI]
MOV BL,AL
INC SI
MOV DL,[SI]
MOV CL,DL
AND AL,CL
XOR DL,BL
OR AL,DL
INC S1
MOV [SI],AL
HLT
```

1.3. Find the Gray code of an 8-bit binary number.

```
MOV  SI,0200H
MOV AL,[SI]
MOV BL,AL
SHR AL,01H
XOR AL,BL
INC SI
MOV [SI],AL
HLT
```

1.4. Find the 2's complement of an 8-bit number.

```
MOV SI,0200H
MOV AL,[SI]
NOT AL
ADD AL,01H
INC SI
MOV [SI],AL
HLT
```

2.1.Find the sum and average of N 16-bit numbers.

```
MOV SI,0200H
MOV DI,0300H
MOV CL,05H
MOV AX,2000H
MOV DS,AX
L1:MOV BX,[SI]
MOV [DI],BX
ADD SI,02H
ADD DI,02H
DEC CL
JNZ L1
HLT
```

2.2.Count no. of 0's in an 8-bit number.

```
MOV SI,2E00H
MOV CL,SI
MOV CH,00H
MOV AX,0000H
LOOP2:INC SI
INC SI
ADD [AX],SI
JNC LOOP1
INC CH
LOOP1:DEC CL
JNZ LOOP2
INC SI
INC SI
MOV [SI],AX
HLT
```

2.3.Move a block of 16-bit data from one location to other.

```
MOV BX,0200H
MOV AL,[BX]
MOV CL,08H
MOV CH,00H
L2:SHR AL,01H
JC L1
INC CH
L1:DEC CL
JNZ L2
INC BX
MOV [BX],CH
HLT
```

2.4.Multiplication of two 16-bit numbers without using MUL instruction in direct addressing mode.

```
MOV AX,2000H
MOV DS,AX
MOV BX,[0200H]
MOV CX,[0202H]
MOV AX,0000H
MOV DX,0000H
L2:ADD AX,BX
JNC L1
INC DX
L1:DEC CX
JNZ L2
MOV [0204H],AX
MOV [0206H],DX
HLT
```

3.1.Find the largest/smallest number (8-bit number) from a given array of size N.

```
.DATA
COUNT DB 04H
VALUE DB 09H,10H,05H,03H
RES DB 0
.CODE
MAIN PROC
MOV AX,DATA
MOV DS,AX
MOV CL,COUNT
LEA SI,VALUE
MOV AL,[SI]
L2:DEC CL
JZ L1
INC SI
CMP AL,[SI]
JNL L2
MOV AL,[SI]
JMP L2
L1: LEA DI,RES
MOV [DI],AL
END MAIN
```

3.2.Arrange the elements (8-bit number) of a given array of size N in ascending/descending order

```
.DATA
COUNT DB 06H
VALUE DB 09H,0FH,14H,45H,24H,3FH
.CODE
MAIN PROC
MOV AX,DATA
MOV DS,AX
MOV CH,COUNT
DEC CH
UP2:MOV CL,CH
LEA SI,VALUE
UP1:MOV AL,[SI]
CMP AL,[SI+1]
JC DOWN
MOV DL,[SI+1]
XCHG[SI],DL
MOV [SI+1],DL
DOWN: INC SI
      DEC CL
      JNZ UP1
      DEC CH
      JNZ UP2
      END MAIN
```

4.1.Perform Addition and Subtraction of two 32-bit numbers using data processing addressing mode (with immediate data

Program:

```
.global _start
_start:
        mov r0, #0x40
        mov r1, #0x50
        adds r2,r0,#0x50
        subs r3,r0,#0x50
        mul r4,r0,r1
my_exit: b my_exit
```

4.2.Perform Addition, Subtraction, and Multiplication of two 32-bit numbers using load/store addressing mode.

```
.global _start
 _start:
        LDR R0,=0X10100000
        LDR R1,[R0],#4
        LDR R2,[R0],#4
        ADDS R3,R1,R2
        STR R3,[R0],#4
        SUBS R4,R1,R2
        STR R4,[R0],#4
        MUL R5,R1,R2
        STR R5,[R0]
my_exit: b my_exit
```

4.3.Perform the logical operations (AND, OR, XOR, and NOT) on two 32-bit numbers using load/store addressing mode.

```
.global _start
 _start:
        LDR R0,=0X10100000
        LDR R1,[R0],#4
        LDR R2,[R0],#4
        ANDS R3,R2,R1
        STR R3,[R0],#4
        ORR R4,R2,R1
        STR R4,[R0],#4
        EOR R5,R2,R1
        STR R5,[R0],#4
        MVN R6, R1
        STR R6,[R0]
my_exit: b my_exit
```

5.1. Find the largest/smallest number in an array of size N.

```
.global _start
_start:
        ldr r0,=count
        ldr r1,[r0]
        mov r4,#0x00
        ldr r2,=array
   back: ldr r3, [r2],#4
        cmp r4,r3
        bgt fwd / blt fwd
        mov r4,r3
    fwd: subs r1,r1,#01
         bne back
        str r4,[r2]
exit: b exit
```

5.2. Separate Even numbers and odds numbers in an array of size N

```
.global _start
_start:
        ldr r0,=count
        ldr r1,[r0]
        ldr r3,=array
        ldr r4,=even
        ldr r5,=odd
    back: ldr r6, [r3],#4
        ands r7,r6,#1
        beq fwd
        str r6,[r5],#4
        b fwd1
     fwd: str r6,[r4],#4
    fwd1: subs r1,r1,#01
        bne back
exit: b exit
```

CORRECTED PROGRAMS

1.2 (i) Swapping Nibbles of Data1

```
MOV SI,0300H
MOV AL,[SI]
ROL AL,4
INC SI
MOV [SI],AL
HLT
```

2.1 Sum and Average of N 16-bit Numbers

```
MOV SI,0200H
MOV CX,0005H
MOV AX,0000H
MOV DX,0000H

L1: ADD AX,[SI]
    ADD SI,02H
    DEC CX
    JNZ L1

MOV BX,0005H
MOV DX,0000H
DIV BX

MOV [0300H],AX
MOV [0302H],AL
HLT
```

2.2 Count Number of 0’s in an 8-bit Number

```
MOV SI,0200H
MOV AL,[SI]
MOV CL,08H
MOV CH,00H

NEXT_BIT: ROL AL,1
          JC SKIP_ZERO
          INC CH

SKIP_ZERO: DEC CL
           JNZ NEXT_BIT

INC SI
MOV [SI],CH
HLT
```

2.3 Move a Block of 16-bit Data

```
MOV SI,0200H
MOV DI,0300H
```