Chapter-5) 8086 Assembly Language Programming

Model of Assembly Language Programming:

Model 1: Using SEGMENT, ASSUME and ENDS directives:

```
DATA SEGMENT
    :
    :
    :
    Program data declaration
    [Data segment of the program]
    :
    :
    :
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE, DS:DATA
MOV AX, DATA
MOV DS, AX
    :
    :
    :
    Program codes
    [Code segment of the program]
    :
    :
    :
CODE ENDS
END
```

In above model, DATA is the name of data segment used to declare data of the program along with its data type i.e DB, DW, etc.

CODE is the name of the code segment used to write code of the program or task to perform.

The END indicates termination of a program.

Model 2: Using .DATA and .CODE directives :

```
.MODEL SMALL
.STACK 100
.DATA
    :
    :
    Program data declaration
    [Data segment of the program]
    :
    :
.CODE
    MOV AX, @DATA
    MOV DS, AX
    :
    :
    Program codes
    [Code segment of the program]
    :
    :
ENDS
END
```

In above model, .model small indicate memory model to be used in the program.
.STACK 100 indicate 100 word memory location are reserved for the stack.
.DATA indicates start of data segment where data declaration of program is made.
.CODE indicates start of the code segment by actual code of the program.
The ENDS directive indicate end of data and code segments and END indicate termination of program.

Symbols, variables and constants :

Variables are symbols whose value can be dynamically varied during the run time. The assembler assigns a memory location to a variable, which is not visible for the user directly and translate the assembly language statements. The Assembler symbols consists of the following characters
Uppercase alphabets A-Z lowercase alphabets a-z
Digits 0-9 Special Characters _, @, $, ?
Rules for variable names :
1) Variable name can have any of the following characters
A-Z, a-z, 0-9, _ (underscore), $, @
2) A variable name must start with a letter (alphabets) or an underscore. so a digit cannot be used as first character.
3) The length of variable name depends on the assembler, normally, the maximum length is 32 characters.
4) There is no distinction between upper and lower case letters. Hence, variable names are case insensetive in 80x86 assembly language.

Numeric constants :- The numeric constants must end with can be represented as binary, decimal or hexadecimal.
The binary number must end with B, decimal number with D and hexadecimal numbers with H. However, if the the number does not end with either 'D', 'B' or 'H' then it is treated as by default decimal numbers.

A valid binary constant must have only '0' and '1', a valid decimal constant must have only numbers '0' to '9' whereas a hexadecimal constants must have both 0 to 9 and A to F.

A hexadecimal digit should always start with 0, as numbers like BA, AD, CD are treated as symbols. They can be written as 0BA, 0AD, 0CD.

138

Programming using assembler :

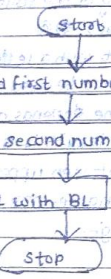
1 Addition of two numbers :

Program 1 : Addition of two 8 bit numbers

Algorithm:

1) Start
2) Load first number in AL
3) Load second number in BL
4) Add AL with BL
5) Stop

Flowchart:



Program 1(a):

```
• MODEL SMALL
• CODE
    MOV AL, 80H
    MOV BL, 70H
    ADD AL, BL
    ENDS
    END
```

139

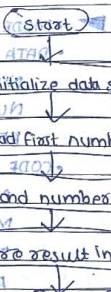
Program 2 : Two 8 bit numbers are stored in memory. Store result in memory.

1(b)

Algorithm:

1) Start
2) Initialize data segment
3) Load first number in AL
4) Add second number with first number
5) Store result in memory location.
6) Stop

Flowchart:



Program 1(b):

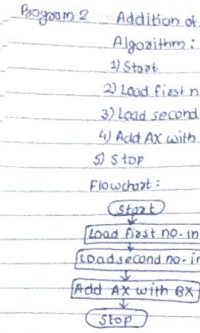
```
• MODEL SMALL
• DATA
    NUM1 DB 80H
    NUM2 DB 08H
    RES DB ?
• CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AL, NUM1
    ADD AL, NUM2
    MOV RES, AL
    ENDS
    END
```

140

Program 2: Addition of two 16 bit numbers

Algorithm:
1) Start
2) Load first number in AX
3) Load second number in BX
4) Add AX with BX
5) Stop

Flowchart:

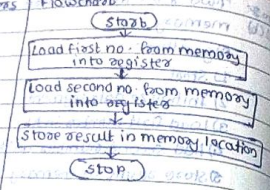



```
•MODEL SMALL
•CODE
MOV AX, 8960H
MOV BX, 7C60H
ADD AX, BX
ENDS
END
```

Program 2(b): In case of Numbers stored in memory,

Algorithm:
1) Start
2) Load first no. from memory in register
3) Load second no. from memory in register
4) Store result in memory location
5) Stop

Flowchart:



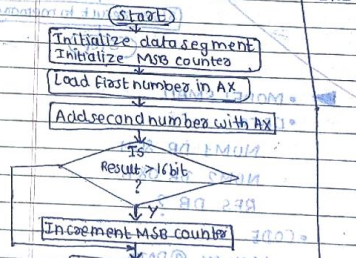

```
•MODEL SMALL
•DATA
    NUM1 DW 8A64H
    NUM2 DW 5F98H
    RES DW ?
•CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, NUM1
    ADD AX, NUM2
    MOV RES, AX
    ENDS
    END
```

141

Program 3: Addition of two 16 bit numbers where result may be more than 16-bit

Algorithm:
1) Start
2) Initialize data segment
3) Load 1st no. in register
4) Add second no. with 1st no.
5) Check result > 16 bit if yes, then goto step 6 else step 7
6) Increment MSB counter
7) Store result
8) Stop

Flowchart:




```
•MODEL SMALL
•DATA
    NUM1 DW OFFFFH
    NUM2 DW OFFFFH
    RES_LSW DW 0
    RES_MSW DW 0
•CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, NUM1
    ADD AX, NUM2
    JNC EXIT
    INC RES_MSW
EXIT:
    MOV RES_LSW, AX
    END
```

142

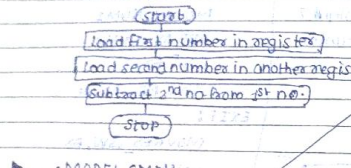
2 Subtraction:

Program 4 Subtraction of two 8 bit numbers

Algorithm:

1) Start
2) Load first number in register
3) Load second number in another register.
4) Subtract second number from first number
5) Stop

Flowchart:



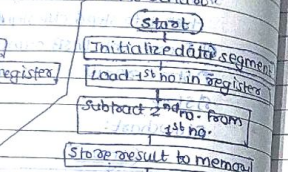

```
• MODEL SMALL
• CODE
    MOV AL, 80H
    MOV BL, 70H
    SUB AL, BL
    ENDS
    END
```

(b) Similarly ...

Algorithm:

1) Start
2) Initialize data segment
3) Load 1st no. from memory in register.
4) Subtract 2nd no from 1st no.
5) Store result in memory location
6) Exit

Flowchart:




```
• MODEL SMALL
• DATA
    NUM1 DB 80H
    NUM2 DB 08H
    RES DB ?
• CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AL, NUM1
    SUB AL, NUM2
    MOV RES, AL
    ENDS
    END
```

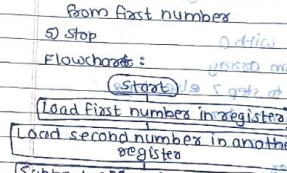
143

Program 5 Subtraction of two 16 bit numbers

Algorithm:

1) Start
2) Load first number in register
3) Load second number in another register.
4) Subtract second number from first number
5) Stop

Flowchart:



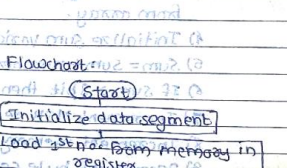

```
• MODEL SMALL
• CODE
    MOV AX, 8960H
    MOV BX, 7C60H
    SUB AX, BX
    ENDS
    END
```

(b) Similarly ...

Algorithm:

1) Start
2) Initialize data segment
3) Load 1st no. from memory in register
4) Subtract 2nd no. from 1st no.
5) Store result in memory location
6) Stop.

Flowchart:




```
• MODEL SMALL
• DATA
    NUM1 DW 8A64H
    NUM2 DW 5F98H
    RES DW ?
• CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, NUM1
    SUB AX, NUM2
    MOV RES, AX
    ENDS
    END
```

144

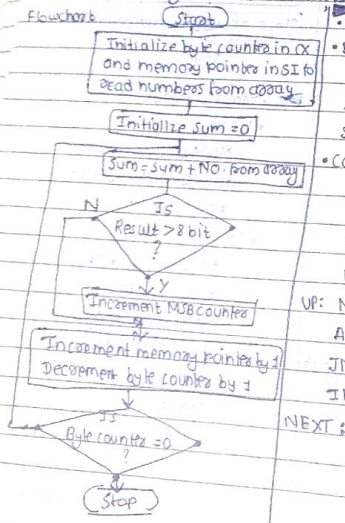
5 Sum numbers in the Array [SUM of series]:

Program 5(a): Addition of five 8-bit numbers in series, result may be greater than 16 bit

Algorithm:

1) Start
2) Initialize data segment
3) Initialize byte counter and memory pointer to read numbers from array.
4) Initialize sum variable with 0
5) Sum = sum + number from array
6) If sum > 8 bit then go to step 7 else step 8
7) Increment MSB result counter
8) Increment memory pointer
9) Decrement byte counter
10) If byte counter = 0 then step 11 else step 5

Flowchart:

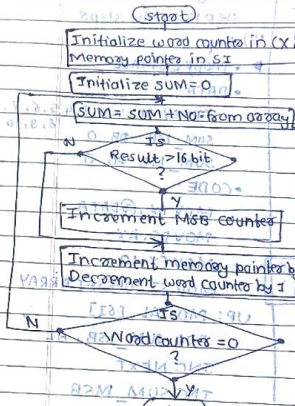



```
• MODEL SMALL
• DATA
  ARRAY DB 0FFH, 0FFH, 0FFH, 0FFH, 0FFH
  SUM_LSB DB 0
  SUM_MSB DB 0
• CODE
  MOV AX, @DATA
  MOV DS, AX
  MOV CX, 5
  MOV SI, OFFSET ARRAY
UP: MOV AL, [SI]
  ADD SUM_LSB, AL
  JNC NEXT
  INC SUM_MSB
NEXT: INC SI
  DEC CX
  JNZ UP
  ENDS
  END
```

145

Program 6(b): Add five 16 bit numbers in series result may be greater than 32 bit

Flowchart:



Algorithm:

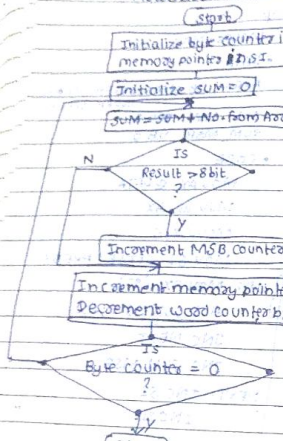
1) Start
2) Initialize data segment
3) Initialize word counter and memory pointer to read numbers from array.
4) Initialize sum variable with 0
5) Sum = sum + no. from array
6) If sum > 16 bit then goto step 7 else step 8.
7) Increment MSB result counter
8) Increment memory pointer
9) Decrement word counter
10) If word counter = 0, then step 11 else step 5
11) Stop.

```
• MODEL SMALL
• DATA
  ARRAY DB 0FFH, 0FFFH, 0FFFFH, 0FFFFH, 0FFFFH
  SUM_LSW DW 0
  SUM_MSB DB 0
• CODE
  MOV AX, @DATA
  MOV DS, AX
  MOV CX, 5
  MOV SI, OFFSET ARRAY
UP: MOV AX, [SI]
  ADD SUM_LSW, AX
  JNC NEXT
  INC SUM_MSB
NEXT: INC SI
  INC SI
  LOOP UP
  ENDS
  END
```

146

Program 6(a) Addition of 10 BCD Numbers in series

Flowchart:



Algorithm:

1) Start
2) Initialize data segment
3) Initialize byte counter and memory pointer to read number from array.
4) Initialize sum variable with 0
5) Sum = sum + no. from array
6) Adjust result to BCD
7) If sum > 8 bit then go to step 8 else step 9
8) Increment MSB result counter
9) ~~Become~~ Increment memory pointer
10) Decrement byte counter
11) If byte counter = 0, then step 12 or else step 5
12) Stop.

```
• MODEL SMALL
• DATA
    ARRAY DB 1,2,3,4,5,6,7,8,9,10
    SUM_LSB DB 0
    SUM_LSB DB 0
•CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CX, 10
    MOV SI, OFFSET ARRAY
UP: MOV AL, [SI]
    ADD SUM_LSB, AL
    INC NEXT
    INC SUM_MSB
NEXT: INC SI
    LOOP UP
    ENDS
    END
```

147

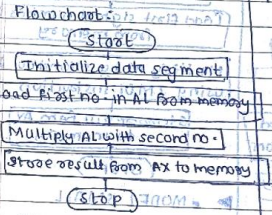
4) Multiplication of unsigned and signed numbers:

Program 7(a) Program to multiply two 8 bit unsigned numbers result is max. 16 bit.

Algorithm:

1) Start
2) Initialize data segment
3) Load first number
4) Multiply first no. with second no.
5) Store result
6) Stop

Flowchart:



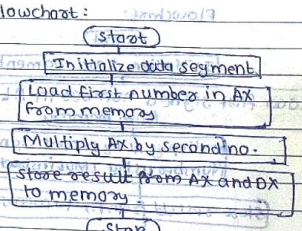

```
• MODEL SMALL
• DATA
    NUM1 DB OFFH
    NUM2 DB OFFH
    RESULT DW 0
• CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AL, NUM1
    MUL NUM2
    MOV RESULT, AX
    ENDS
    END
```

Program 7(b): Program to multiply two 16 bit unsigned numbers result is max 32 bit.

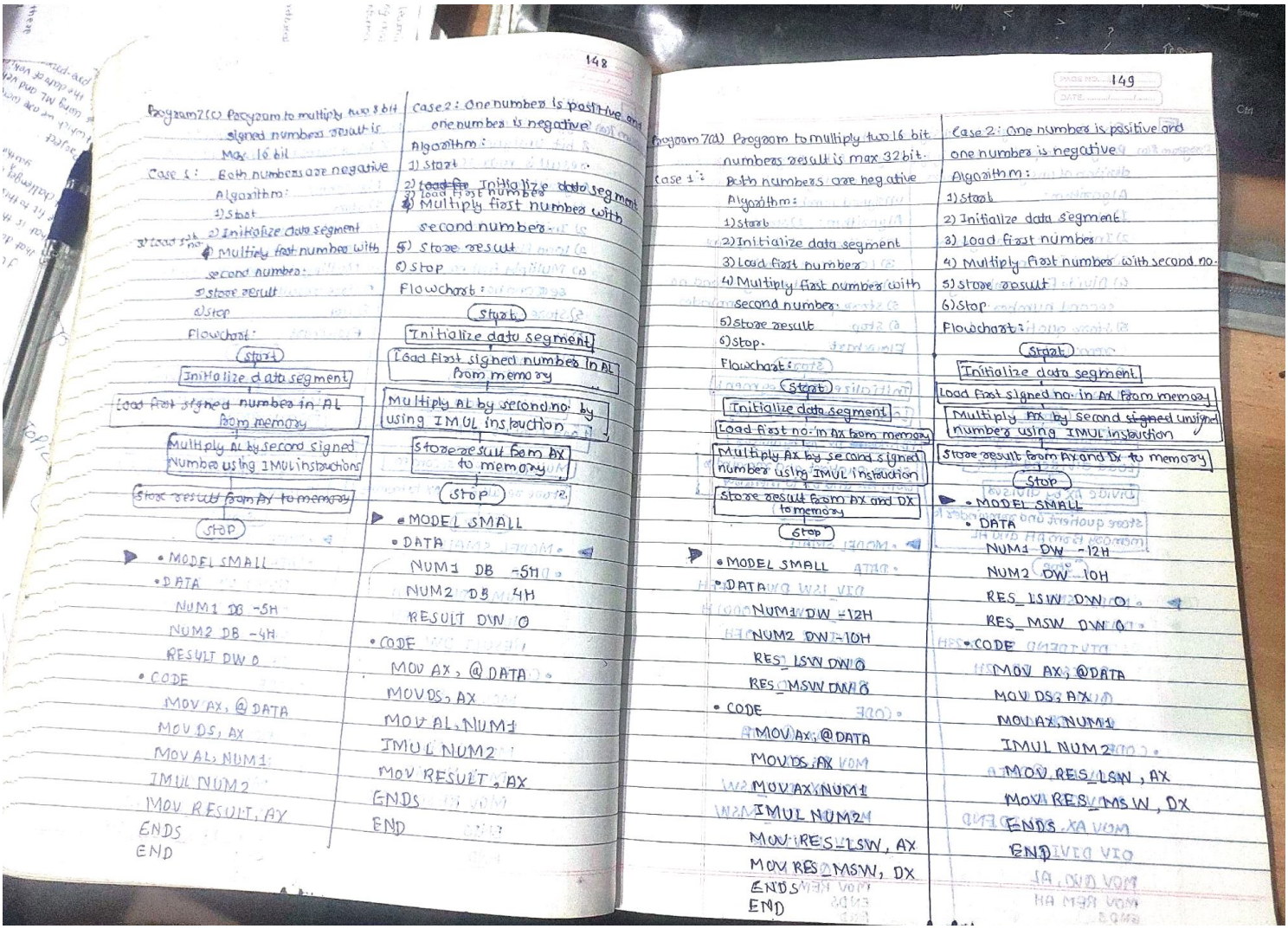
Algorithm:

1) Start
2) Initialize data segment
3) Load first number
4) Multiply first no. with second no.
5) Store result
6) Stop

Flowchart:




```
• MODEL SMALL
• DATA
    NUM1 DW OFFFH
    NUM2 DW OFFFH
    RES_LSB DW 0
    RES_MSB DW 0
• CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, NUM1
    MUL NUM2
    MOV RES_LSB, AX
    MOV RES_MSB, DX
    ENDS
    END
```



Program 7(c) Program to multiply two 8 bit signed numbers result is Max 16 bit

Case 1: Both numbers are negative

Algorithm:
1) Start
2) Initialize data segment
3) Load first numbers
4) Multiply first numbers with second numbers.
5) Store result
6) Stop

Flowchart:
Start → Initialize data segment → Load first signed numbers in AL from memory → Multiply AL by second signed number using IMUL instructions → Store result from AX to memory → Stop

```
.MODEL SMALL
.DATA
NUM1 DB -5H
NUM2 DB -4H
RESULT DW 0
.CODE
MOV AX, @DATA
MOV DS, AX
MOV AL, NUM1
IMUL NUM2
MOV RESULT, AX
ENDS
END
```

Case 2: One numbers is positive and one numbers is negative

Algorithm:
1) Start
2) Initialize data segment
3) Load first numbers
4) Multiply first numbers with second numbers.
5) Store result
6) Stop

Flowchart:
Start → Initialize data segment → Load first signed numbers in AL from memory → Multiply AL by second no. by using IMUL instruction → Store result from AX to memory → Stop

```
.MODEL SMALL
.DATA
NUM1 DB -5H
NUM2 DB 4H
RESULT DW 0
.CODE
MOV AX, @DATA
MOV DS, AX
MOV AL, NUM1
IMUL NUM2
MOV RESULT, AX
ENDS
END
```

Program 7(d) Program to multiply two 16 bit numbers result is max 32 bit.

Case 1: Both numbers are negative

Algorithm:
1) Start
2) Initialize data segment
3) Load first number
4) Multiply first number with second numbers.
5) Store result
6) Stop.

Flowchart:
Start → Initialize data segment → Load first no. in AX from memory → Multiply AX by second signed numbers using IMUL instruction → Store result from AX and DX to memory → Stop

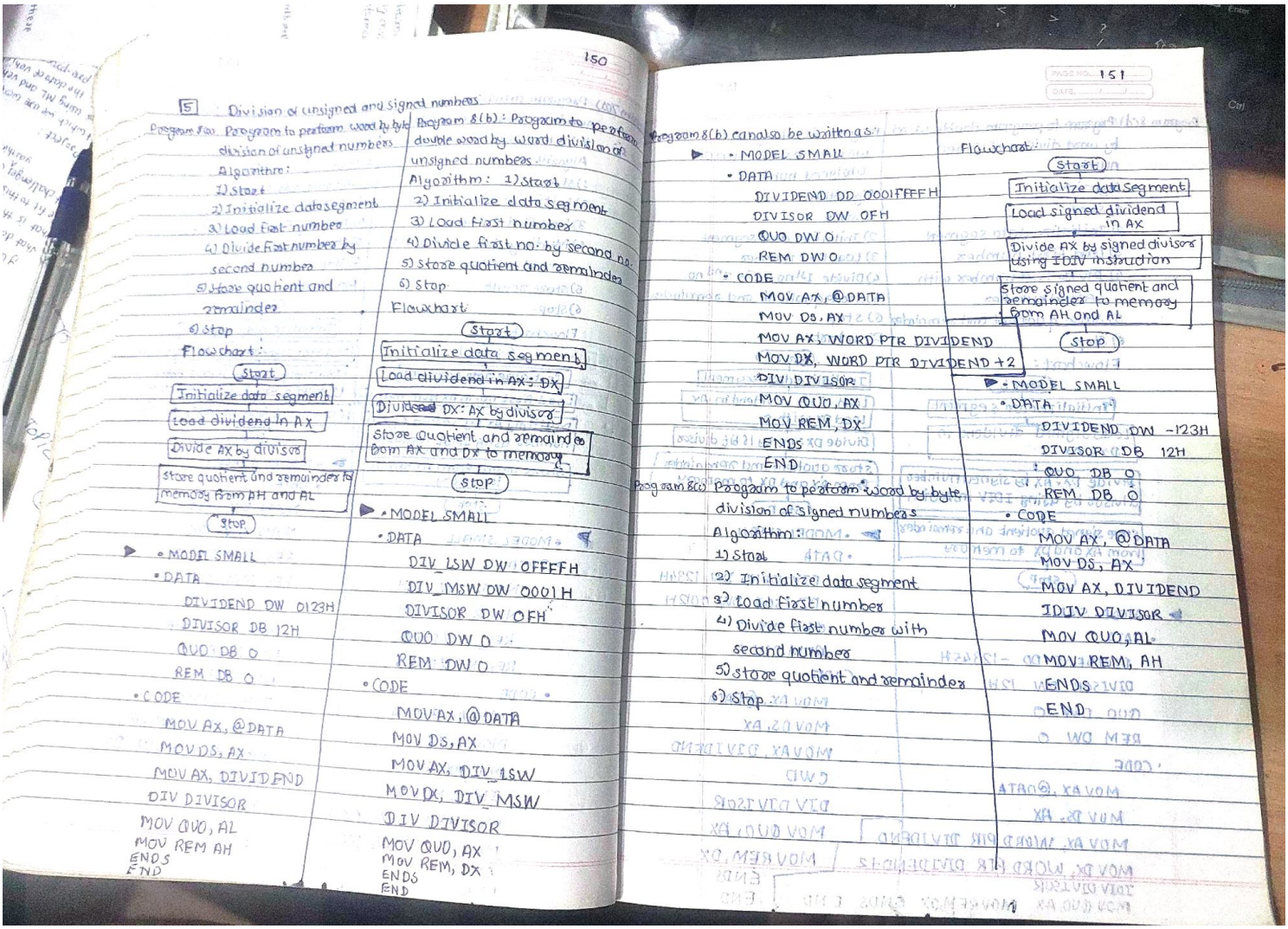
```
.MODEL SMALL
.DATA
NUM1 DW -12H
NUM2 DW -10H
RES_LSW DW 0
RES_MSW DW 0
.CODE
MOV AX, @DATA
MOV DS, AX
MOV AX, NUM1
IMUL NUM2
MOV RES_LSW, AX
MOV RES_MSW, DX
ENDS
END
```

Case 2: One numbers is positive and one numbers is negative

Algorithm:
1) Start
2) Initialize data segment
3) Load first numbers
4) Multiply first number with second no.
5) Store result
6) Stop

Flowchart:
Start → Initialize data segment → Load first signed no. in AX from memory → Multiply AX by second signed unsigned numbers using IMUL instruction → Store result from AX and DX to memory → Stop

```
.MODEL SMALL
.DATA
NUM1 DW -12H
NUM2 DW 10H
RES_LSW DW 0
RES_MSW DW 0
.CODE
MOV AX, @DATA
MOV DS, AX
MOV AX, NUM1
IMUL NUM2
MOV RES_LSW, AX
MOV RES_MSW, DX
ENDS
END
```



150

5 Division of unsigned and signed numbers

Program 8(a): Program to perform word by byte division of unsigned numbers

Algorithm:
1) Start
2) Initialize data segment
3) Load first number
4) Divide first number by second number
5) Store quotient and remainder
6) Stop

Flowchart:

Start
Initialize data segment
Load dividend in AX
Divide AX by divisor
Store quotient and remainder from AH and AL
Stop

```
.MODEL SMALL
.DATA
    DIVIDEND DW 0123H
    DIVISOR DB 12H
    QUO DB 0
    REM DB 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, DIVIDEND
    DIV DIVISOR
    MOV QUO, AL
    MOV REM AH
    ENDS
END
```

Program 8(b): Program to perform double word by word division of unsigned numbers

Algorithm: 1) Start
2) Initialize data segment
3) Load first number
4) Divide first no. by second no.
5) Store quotient and remainder
6) Stop

Flowchart:

Start
Initialize data segment
Load dividend in AX: DX
Divide DX: AX by divisor
Store quotient and remainder from AX and DX to memory
Stop

```
.MODEL SMALL
.DATA
    DIV_LSW DW 0FFEFH
    DIV_MSW DW 0001H
    DIVISOR DW 0FH
    QUO DW 0
    REM DW 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, DIV_LSW
    MOV DX, DIV_MSW
    DIV DIVISOR
    MOV QUO, AX
    MOV REM, DX
    ENDS
END
```

151

Program 8(b) can also be written as:

```
.MODEL SMALL
.DATA
    DIVIDEND DD 0001FFEFH
    DIVISOR DW 0FH
    QUO DW 0
    REM DW 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, WORD PTR DIVIDEND
    MOV DX, WORD PTR DIVIDEND+2
    DIV DIVISOR
    MOV QUO, AX
    MOV REM, DX
    ENDS
    END
```

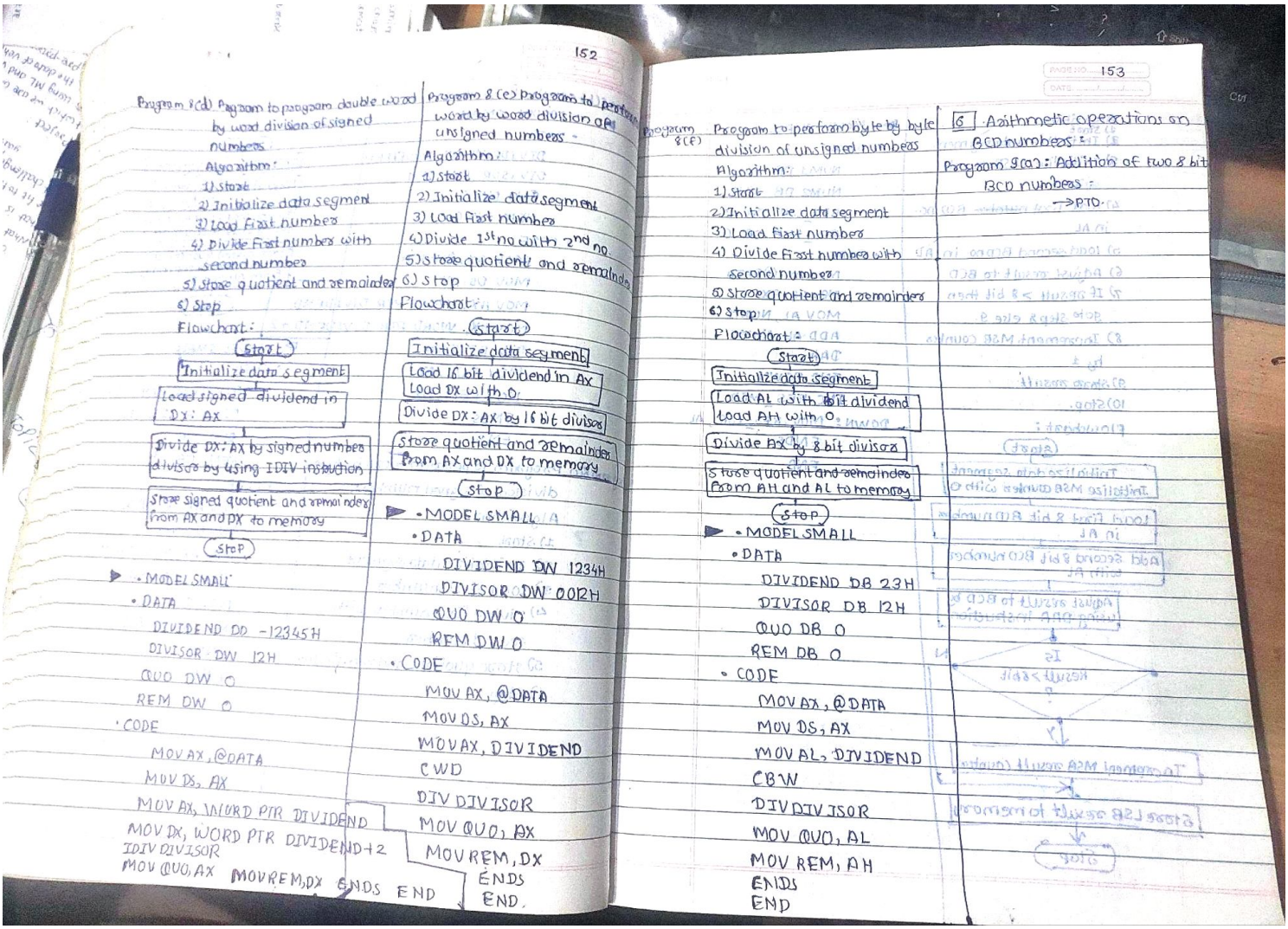
Program 8(c): Program to perform word by byte division of signed numbers

Algorithm:
1) Start
2) Initialize data segment
3) Load first number
4) Divide first number with second number
5) Store quotient and remainder
6) Stop

Flowchart:

Start
Initialize data segment
Load signed dividend in AX
Divide AX by signed divisor using IDIV instruction
Store signed quotient and remainder to memory from AH and AL
Stop

```
.MODEL SMALL
.DATA
    DIVIDEND DW -123H
    DIVISOR DB 12H
    QUO DB 0
    REM DB 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, DIVIDEND
    IDIV DIVISOR
    MOV QUO, AL
    MOV REM, AH
    ENDS
    END
```



Program 8(d) Program to program double word by word division of signed numbers

Algorithm:
1) Start
2) Initialize data segment
3) Load first number
4) Divide first numbers with second number
5) Store quotient and remainder
6) Stop

Flowchart:

Start → Initialize data segment → Load signed dividend in DX : AX → Divide DX : AX by signed numbers divisor by using IDIV instruction → Store signed quotient and remainder from AX and DX to memory → Stop

```
.MODEL SMALL
.DATA
    DIVIDEND DD -12345H
    DIVISOR DW 12H
    QUO DW 0
    REM DW 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, WORD PTR DIVIDEND
    MOV DX, WORD PTR DIVIDEND+2
    IDIV DIVISOR
    MOV QUO, AX
    MOV REM, DX
    ENDS
    END
```

Program 8(e) Program to perform word by word division of unsigned numbers

Algorithm:
1) Start
2) Initialize data segment
3) Load first number
4) Divide 1st no with 2nd no.
5) Store quotient and remainder
6) Stop

Flowchart:

Start → Initialize data segment → Load 16 bit dividend in AX, Load DX with 0 → Divide DX : AX by 16 bit divisor → Store quotient and remainder from AX and DX to memory → Stop

```
.MODEL SMALL
.DATA
    DIVIDEND DW 1234H
    DIVISOR DW 0012H
    QUO DW 0
    REM DW 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, DIVIDEND
    CWD
    DIV DIVISOR
    MOV QUO, AX
    MOV REM, DX
    ENDS
    END
```

Program 8(f) Program to perform byte by byte division of unsigned numbers

Algorithm:
1) Start
2) Initialize data segment
3) Load first number
4) Divide first numbers with second number
5) Store quotient and remainders
6) Stop

Flowchart:

Start → Initialize data segment → Load AL with 8 bit dividend, Load AH with 0 → Divide AX by 8 bit divisor → Store quotient and remainder from AH and AL to memory → Stop

```
.MODEL SMALL
.DATA
    DIVIDEND DB 23H
    DIVISOR DB 12H
    QUO DB 0
    REM DB 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AL, DIVIDEND
    CBW
    DIV DIVISOR
    MOV QUO, AL
    MOV REM, AH
    ENDS
    END
```

6. Arithmetic operations on BCD numbers:

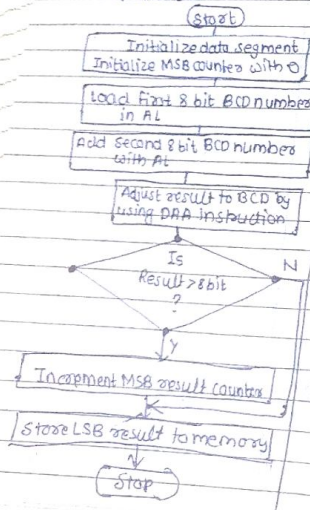
Program 9(a): Addition of two 8 bit BCD numbers:

→ PTO.

154

Algorithm:
1) Start
2) Initialize data segment
3) Initialize MSB counter with 0
4) Load first ~~number~~ BCD no. in AL
5) Load second BCD no. in AL
6) Adjust result to BCD
7) If result > 8 bit then goto step 8 else 9
8) Increment MSB counter by 1
9) Store result
10) Stop.

Flowchart:



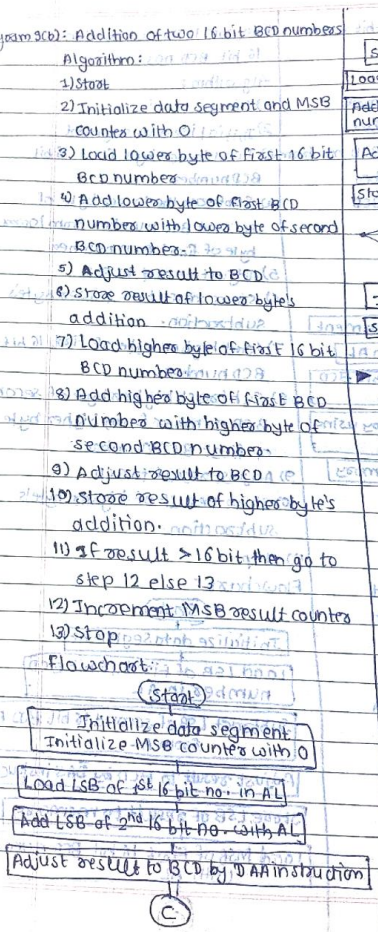

```
• MODEL SMALL
• DATA
    NUM1 DB 84H
    NUM2 DB 28H
    RES_LSB DB 0
    RES_MSB DB 0
• CODE
    MOV AX,@DATA
    MOV DS,AX
    MOV AL, NUM1
    ADD AL, NUM2
    DAA
    JNC DOWN
    INC RES_MSB
DOWN: MOV RES_LSB, AL
    ENDS
    END
```

155

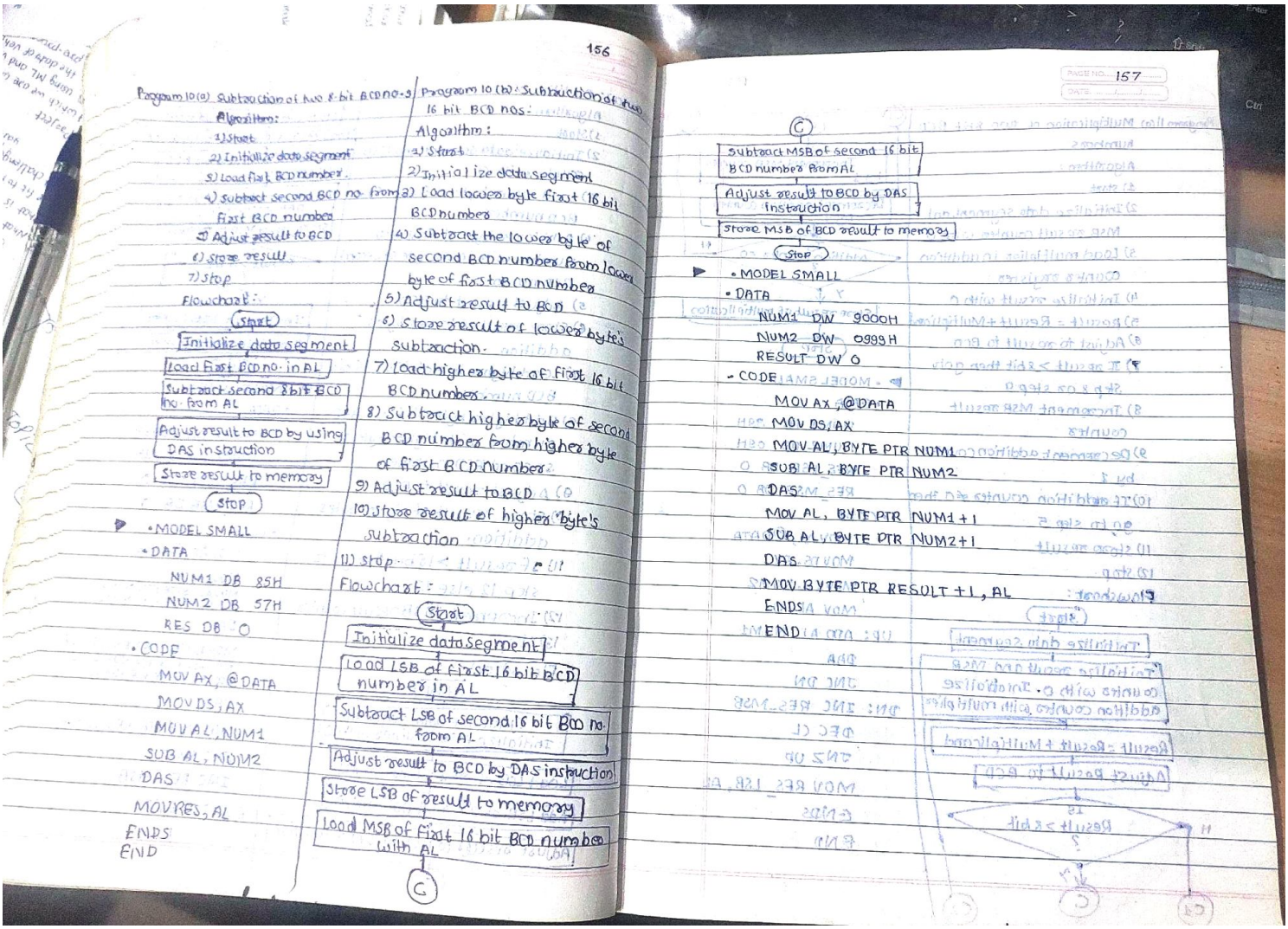
Algorithm(CD): Addition of two 16 bit BCD numbers

Algorithm:
1) Start
2) Initialize data segment and MSB counter with 0
3) Load lower byte of first 16 bit BCD number
4) Add lower byte of first BCD numbers with lower byte of second BCD numbers.
5) Adjust result to BCD
6) Store result of lower byte's addition
7) Load higher byte of first 16 bit BCD numbers
8) Add higher byte of first BCD numbers with higher byte of second BCD numbers.
9) Adjust result to BCD
10) Store result of higher byte's addition.
11) If result > 16 bit then go to step 12 else 13
12) Increment MSB result counter
13) Stop

Flowchart:




```
• MODEL SMALL
• DATA
    NUM1 DW 9999H
    NUM2 DW 9999H
    RES_LSW DW 0
    RES_MSB DB 0
• CODE
    MOV AX,@DATA
    MOV DS,AX
    MOV AL, BYTE PTR NUM1
    ADD AL, BYTE PTR NUM2
    DAA
    MOV BYTE PTR RES_LSW, AL
    MOV AL, BYTE PTR NUM1+1
    ADD AL, BYTE PTR NUM2+1
    DAA
    MOV BYTE PTR RES_LSW+1, AL
    JNC EXIT
    INC RES_MSB
EXIT: ENDS
    END
```



156

Program 10(a) Subtraction of two 8 bit BCD no.s

Algorithm:

1) Start
2) Initialize data segment
3) Load first BCD number
4) Subtract second BCD no. from first BCD number
5) Adjust result to BCD
6) Store result
7) Stop

Flowchart:

Start
Initialize data segment
Load first BCD no. in AL
Subtract second 8 bit BCD no. from AL
Adjust result to BCD by using DAS instruction
Store result to memory
Stop

```
.MODEL SMALL
.DATA
    NUM1 DB 85H
    NUM2 DB 57H
    RES DB ?
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AL, NUM1
    SUB AL, NUM2
    DAS
    MOV RES, AL
    ENDS
END
```

Program 10 (b): Subtraction of two 16 bit BCD nos:

Algorithm:

1) Start
2) Initialize data segment
3) Load lowers byte first 16 bit BCD numbers
4) Subtract the lower byte of second BCD number from lower byte of first BCD number
5) Adjust result to BCD
6) Store result of lowers byte's subtraction.
7) Load higher byte of first 16 bit BCD number.
8) Subtract higher byte of second BCD number from higher byte of first BCD number
9) Adjust result to BCD
10) Store result of higher byte's subtraction
11) Stop

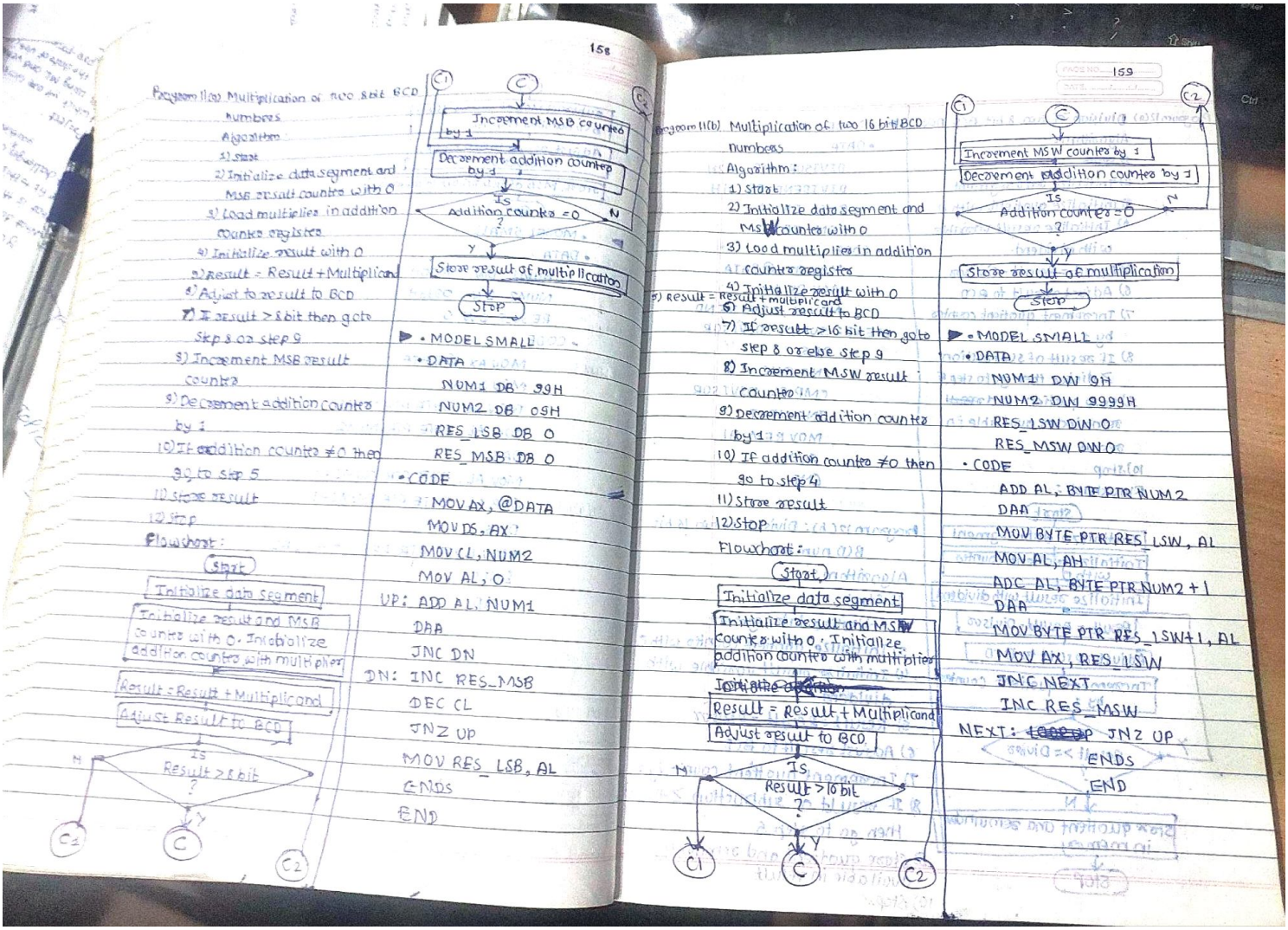
Flowchart:

Start
Initialize data segment
Load LSB of first 16 bit BCD number in AL
Subtract LSB of second 16 bit BCD no. from AL
Adjust result to BCD by DAS instruction
Store LSB of result to memory
Load MSB of first 16 bit BCD number with AL
(C)

157

(C)
Subtract MSB of second 16 bit BCD number from AL
Adjust result to BCD by DAS instruction
Store MSB of BCD result to memory
Stop

```
.MODEL SMALL
.DATA
    NUM1 DW 9000H
    NUM2 DW 0999H
    RESULT DW 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AL, BYTE PTR NUM1
    SUB AL, BYTE PTR NUM2
    DAS
    MOV AL, BYTE PTR NUM1+1
    SUB AL, BYTE PTR NUM2+1
    DAS
    MOV BYTE PTR RESULT+1, AL
    ENDS
END
```



Program 11(a) Multiplication of two 8 bit BCD numbers

Algorithm

1) Start
2) Initialize data segment and MSB result counter with 0
3) Load multiplies in addition counter register
4) Initialize result with 0
5) Result = Result + Multiplicand
6) Adjust to result to BCD
7) If result > 8 bit then goto step 8 or step 9
8) Increment MSB result counter
9) Decrement addition counter by 1
10) If addition counter ≠ 0 then go to step 5
11) Store result
12) Stop

Flowchart:


```
• MODEL SMALL
• DATA
    NUM1 DB 99H
    NUM2 DB 05H
    RES_LSB DB 0
    RES_MSB DB 0
• CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CL, NUM2
    MOV AL, 0
UP: ADD AL, NUM1
    DAA
    JNC DN
DN: INC RES_MSB
    DEC CL
    JNZ UP
    MOV RES_LSB, AL
    ENDS
    END
```

Program 11(b) Multiplication of two 16 bit BCD numbers

Algorithm:

1) Start
2) Initialize data segment and MSW counter with 0
3) Load multiplies in addition counter register
4) Initialize result with 0
5) Result = Result + multiplicand
6) Adjust result to BCD
7) If result > 16 bit then go to step 8 or else step 9
8) Increment MSW result counter
9) Decrement addition counter by 1
10) If addition counter ≠ 0 then go to step 4
11) Store result
12) Stop

Flowchart:


```
• MODEL SMALL
• DATA
    NUM1 DW 9H
    NUM2 DW 9999H
    RES_LSW DW 0
    RES_MSW DW 0
• CODE
    ADD AL, BYTE PTR NUM2
    DAA
    MOV BYTE PTR RES_LSW, AL
    MOV AL, AH
    ADC AL, BYTE PTR NUM2+1
    DAA
    MOV BYTE PTR RES_LSW+1, AL
    MOV AX, RES_LSW
    JNC NEXT
    INC RES_MSW
NEXT: JNZ UP
    ENDS
    END
```

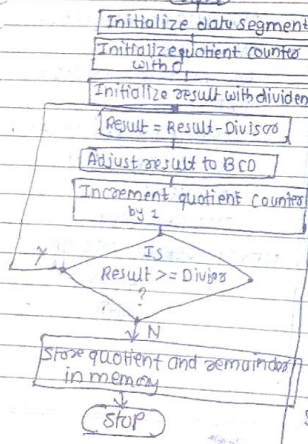
160

Program 12(a) Division of two 8 bit BCD nos.

Algorithm:

1) Start
2) Initialize data segment
3) Initialize quotient with 0
4) Initialize result variable with dividend
5) Result = Result - Divisor
6) Adjust result to BCD
7) Increment quotient counter by 1
8) If result of subtraction ≥ divisor then go to step 5
9) Store quotient and remainder available in result
10) Stop

Flowchart:



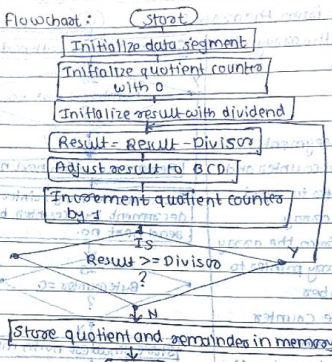

```
▶ .MODEL SMALL
  .DATA
     DIVISOR  DB  02H
     DIVIDEND DB  11H
     QUO      DB  0
     REM      DB  0
  .CODE
     MOV AX, @DATA
     MOV DS, AX
     MOV AL, DIVIDEND
NEXT: SUB AL, DIVISOR
     DAS
     INC QUO
     CMP AL, DIVISOR
     JNC NEXT
     MOV REM, AL
     ENDS
     END
```

Program 12(b): Division of two 16 bit BCD numbers

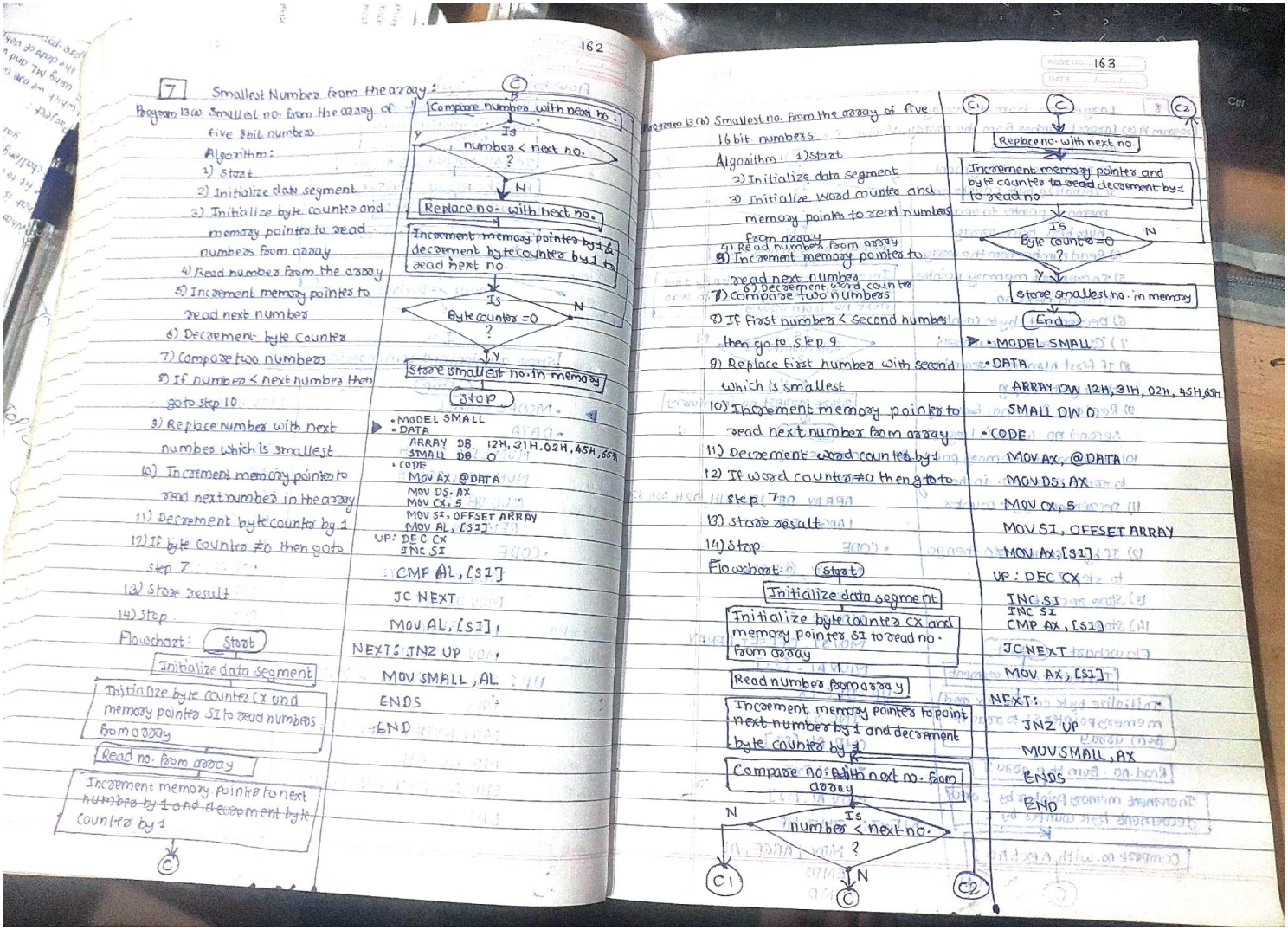
Algorithm:

1) Start
2) Initialize data segment
3) Initialize quotient counter with 0
4) Initialize result variable with dividend
5) Result = Result - Divisor
6) Adjust result to BCD
7) Increment quotient counter by 1
8) If result of subtraction ≥ divisor then go to step 5
9) Store quotient and remainder available in result
10) Stop.

Flowchart:




```
▶ .MODEL SMALL
  .DATA
     NUM1 DW 0009H
     NUM2 DW 0099H
     QUO DB 0
     REM DW 0
  .CODE
     MOV AX, @DATA
     MOV DS, AX
     MOV BH, 0
     MOV AX, NUM2
UP:  SUB AL, BYTE PTR NUM1
     DAS
     MOV BYTE PTR REM, AL
     MOV AL, AH
     SBB AL, BYTE PTR NUM1+1
     DAS
     MOV BYTE PTR REM+1, AL
     MOV AL, BL
     ADD AL, 1
     DAA
     MOV BH, AL
     MOV QUO, AL
     MOV AX, REM
     CMP AX, NUM1
     JGE UP
     ENDS
     END
```



7 Smallest Numbers from the array :

Program 13(a) Smallest no. from the array of five 8bit numbers

Algorithm:
1) Start
2) Initialize data segment
3) Initialize byte counter and memory pointer to read numbers from array
4) Read numbers from the array
5) Increment memory pointer to read next numbers
6) Decrement byte counter
7) Compare two numbers
8) If numbers < next numbers then go to step 10
9) Replace number with next numbers which is smallest
10) Increment memory pointer to read next numbers in the array
11) Decrement byte counter by 1
12) If byte counter ≠0 then go to step 7
13) Store result
14) Stop

Flowchart: Start → Initialize data segment → Initialize byte counter CX and memory pointer SI to read numbers from array → Read no. from array → Increment memory pointer to next number by 1 and decrement byte counter by 1 → C

C → Compare numbers with next no. → Is numbers < next no. ? Y → Replace no. with next no. / N → Increment memory pointer by 1 & decrement byte counter by 1 to read next no. → Is Byte counter = 0 ? Y → Store smallest no. in memory → Stop

```
.MODEL SMALL
.DATA
    ARRAY DB 12H,31H,02H,45H,65H
    SMALL DB 0
.CODE
    MOV AX,@DATA
    MOV DS,AX
    MOV CX,5
    MOV SI,OFFSET ARRAY
    MOV AL,[SI]
UP: DEC CX
    INC SI
    CMP AL,[SI]
    JC NEXT
    MOV AL,[SI]
NEXT: JNZ UP
    MOV SMALL,AL
    ENDS
    END
```

Program 13(b) Smallest no. from the array of five 16 bit numbers

Algorithm: 1) Start
2) Initialize data segment
3) Initialize word counter and memory pointer to read numbers from array
4) Read numbers from array
5) Increment memory pointer to read next numbers
6) Decrement word counter
7) Compare two numbers
8) If first number < second number then go to step 9
9) Replace first number with second which is smallest
10) Increment memory pointer to read next number from array
11) Decrement word counter by 1
12) If word counter ≠0 then go to step 7
13) Store result
14) Stop

Flowchart: Start → Initialize data segment → Initialize byte counter CX and memory pointer SI to read no. from array → Read number from array → Increment memory pointer to point next number by 1 and decrement byte counter by 1 → Compare no. with next no. from array → Is number < next no. ? N → C1 / N → C

C1 → Replace no. with next no. → Increment memory pointer and byte counter to read decrement by 1 to read no. → Is Byte counter = 0 ? N → C2 / Y → Store smallest no. in memory → End

```
.MODEL SMALL
.DATA
    ARRAY DW 12H,31H,02H,45H,65H
    SMALL DW 0
.CODE
    MOV AX,@DATA
    MOV DS,AX
    MOV CX,5
    MOV SI,OFFSET ARRAY
    MOV AX,[SI]
UP: DEC CX
    INC SI
    INC SI
    CMP AX,[SI]
    JC NEXT
    MOV AX,[SI]
NEXT:
    JNZ UP
    MOV SMALL,AX
    ENDS
    END
```

8 Largest no. from the array

Program 14(a) Largest number from the array of five 8 bit numbers

Algorithm : 1) Start
2) Initialize data segment
3) Initialize byte counts and memory pointer to read numbers from array
4) Read number from the array
5) Increment memory pointer to read next no
6) Decrement byte counter
7) Compare two numbers
8) If first numbers > second no. then go to step 9
9) Replace first no. from by second no. which is largest
10) Increment memory pointer to read next no. in the array
11) Decrement byte counter by 1
12) If byte counter ≠ 0 then go to step 7
13) Store result
14) Stop

Flowchart: Start → Initialize data segment → Initialize byte counter CX and memory pointer SI to read no. from array → Read no. from the array → Increment memory pointer by 1 and decrement byte counter by 1 → Compare no. with next no. (C)

(C) → Is no. < next no. ? Y → Replace no. with next no. → Increment memory pointer by 1 and decrement byte counter by 1 to read next no. from array → Is Byte counter = 0 ? Y → Store largest no. in memory → STOP; N → (C1)

```
.MODEL SMALL
.DATA
    ARRAY DB 12H, 31H, 02H, 45H,
    LARGE DB 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CX, 5
    MOV SI, OFFSET ARRAY
    MOV AL, [SI]
UP: DEC CX
    INC SI
    CMP AL, [SI]
    JNC NEXT
    MOV AL, [SI]
NEXT: JNZ UP
    MOV LARGE, AL
    ENDS
    END
```

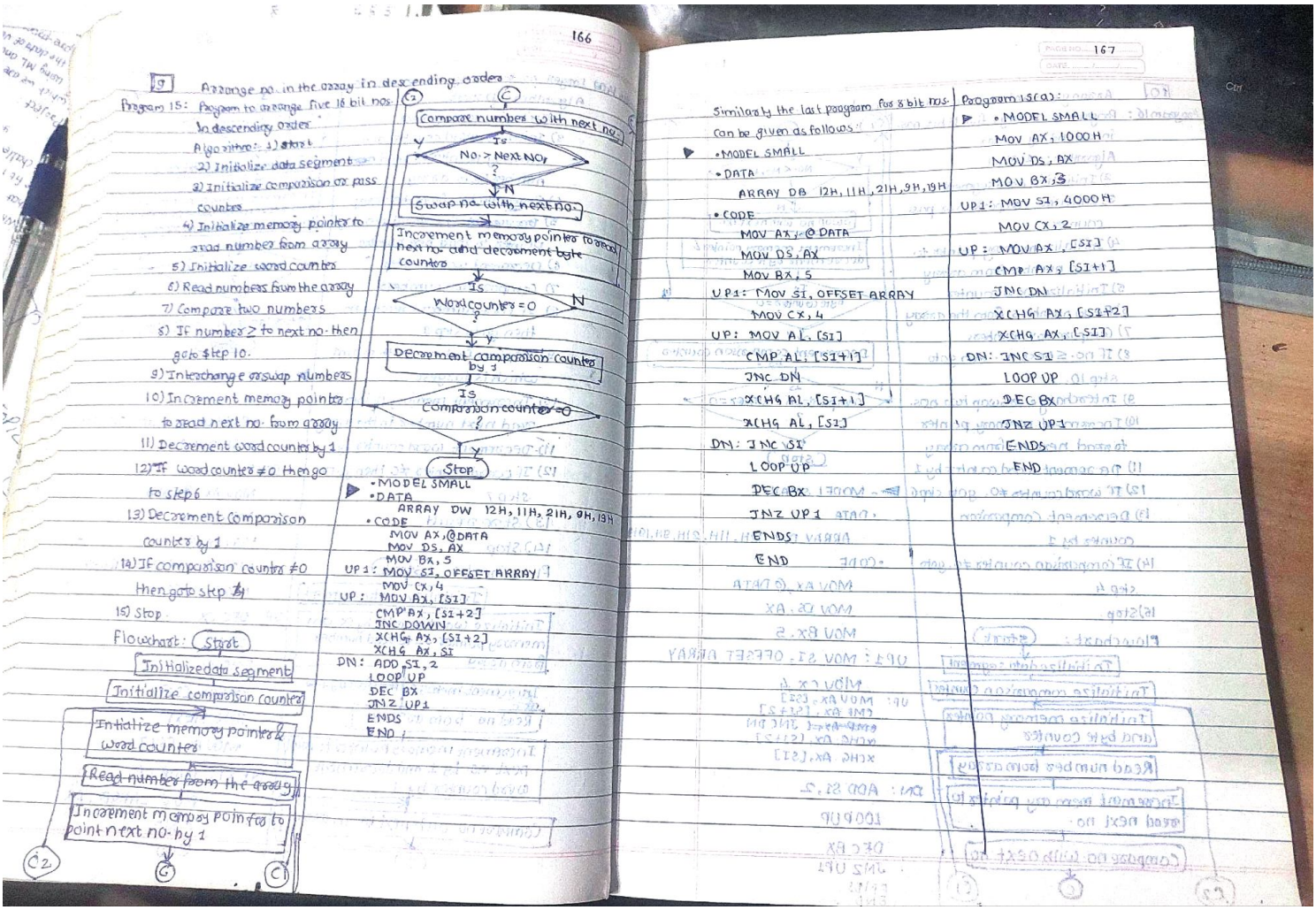

Program 14(b) Largest no. from array of five 16 bit nos.

Algorithm : 1) Start
2) Initialize data segment
3) Initialize word counter and memory pointer to read numbers from array
4) Read number from the array
5) Increment memory pointer to read next number
6) Decrement word counter
7) Compare two numbers
8) If first number ≥ second no. then go to step 9
9) Replace first no. with second which is largest
10) Increment memory pointer to read next number in the array
11) Decrement word counter by 1
12) If word counter ≠ 0 then go to step 7
13) Store result
14) Stop

Flowchart: Start → Initialize data segment → Initialize word counter CX and memory pointer SI to read number from array → Read no. from array → Increment memory pointer to point next no. by 1 and decrement word counter by 1 → Compare no. with next no. in array (C)

(C) → Is no. < next no. ? Y → Replace no. with next no. → Increment memory pointer by 1 and decrement word counter by 1 to read next no. from array → Is Word counter = 0 ? Y → Store largest no. in memory → STOP; N → (C1)

```
.MODEL SMALL
.DATA
    ARRAY DW 12H, 31H, 02H, 45H,
    LARGE DW 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CX, 5
    MOV SI, OFFSET ARRAY
    MOV AX, [SI]
UP: DEC CX
    INC SI
    INC SI
    CMP AX, [SI]
    JNC NEXT
    MOV AX, [SI]
NEXT: LOOP UP
    MOV LARGE, AX
    ENDS
    END
```



166

9 Arrange no. in the array in descending order

Program 15: Program to arrange five 16 bit nos in descending order

Algorithm: 1) start
2) Initialize data segment
3) Initialize comparison or pass counter
4) Initialize memory pointer to read number from array
5) Initialize word counter
6) Read numbers from the array
7) Compare two numbers
8) If number ≥ to next no. then goto step 10.
9) Interchange or swap numbers
10) Increment memory pointer to read next no. from array
11) Decrement word counter by 1
12) If word counter ≠ 0 then go to step 6
13) Decrement comparison counter by 1
14) If comparison counter ≠ 0 then goto step 8
15) Stop

Flowchart: Start → Initialize data segment → Initialize comparison counter → Initialize memory pointer & word counter → Read number from the array → Increment memory pointer to point next no. by 1 → C2 → 6 → C1

C → Compare numbers with next no. → Is No. > Next No.? Y → Swap no. with next no. → N → Increment memory pointer to read next no. and decrement byte counter → Is Word counter = 0? N → Y → Decrement comparison counter by 1 → Is Comparison counter = 0? Y → Stop

```
•MODEL SMALL
•DATA
    ARRAY DW 12H, 11H, 21H, 9H, 19H
•CODE
    MOV AX,@DATA
    MOV DS, AX
    MOV Bx, 5
UP1: MOV SI, OFFSET ARRAY
    MOV CX,4
UP: MOV AX, [SI]
    CMP AX, [SI+2]
    JNC DOWN
    XCHG AX, [SI+2]
    XCHG AX, SI
DN: ADD SI,2
    LOOP UP
    DEC BX
    JNZ UP1
    ENDS
    END
```

167

Similarly the last program for 8 bit nos can be given as follows:

```
•MODEL SMALL
•DATA
    ARRAY DB 12H, 11H, 21H, 9H, 19H
•CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV BX, 5
UP1: MOV SI, OFFSET ARRAY
    MOV CX, 4
UP: MOV AL, [SI]
    CMP AL, [SI+1]
    JNC DN
    XCHG AL, [SI+1]
    XCHG AL, [SI]
DN: JNC SI
    LOOP UP
    DEC BX
    JNZ UP1
    ENDS
    END
```

Program 15(a):

```
•MODEL SMALL
    MOV AX, 1000H
    MOV DS, AX
    MOV BX, 3
UP1: MOV SI, 4000H
    MOV CX, 2
UP: MOV AX, [SI]
    CMP AX, [SI+1]
    JNC DN
    XCHG AX, [SI+2]
    XCHG AX, [SI]
DN: INC SI
    LOOP UP
    DEC BX
    JNZ UP1
    ENDS
    END
```

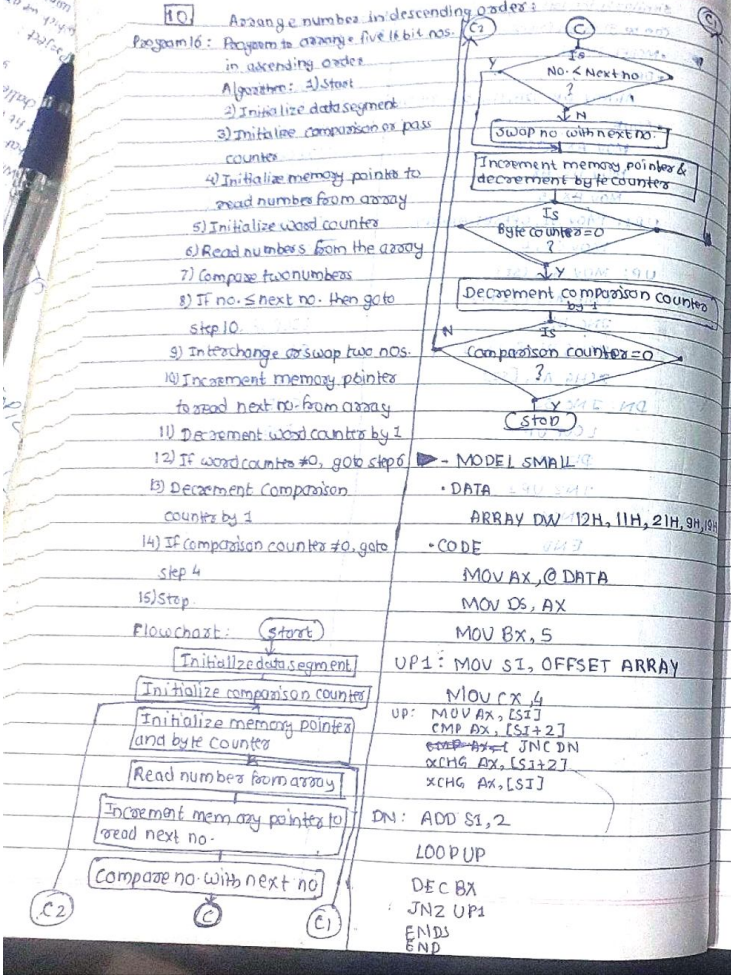
168

10 Arrange numbers in descending orders:

Program 16: Program to arrange five 16 bit nos. in ascending order

Algorithm: 1) Start
2) Initialize data segment
3) Initialize comparison or pass counter
4) Initialize memory pointer to read numbers from array
5) Initialize word counter
6) Read numbers from the array
7) Compare two numbers
8) If no. ≤ next no. then go to step 10
9) Interchange or swap two nos.
10) Increment memory pointer to read next no. from array
11) Decrement word counter by 1
12) If word counter ≠0, goto step 6
13) Decrement comparison counter by 1
14) If comparison counter ≠0, goto step 4
15) Stop

Flowchart:




```
- MODEL SMALL
- DATA
    ARRAY DW 12H, 11H, 21H, 9H, 19H
- CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV BX, 5
UP1: MOV SI, OFFSET ARRAY
    MOV CX, 4
UP: MOV AX, [SI]
    CMP AX, [SI+2]
    JNC DN
    XCHG AX, [SI+2]
    XCHG AX, [SI]
DN: ADD SI, 2
    LOOP UP
    DEC BX
    JNZ UP1
    ENDS
    END
```

169

Similarly last program for five 8 bit nos. can be given as follows:

```
- MODEL SMALL
- DATA
    ARRAY DB 12H, 11H, 21H, 9H, 19H
- CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV BX, 5
UP1: MOV SI, OFFSET ARRAY
    MOV CX, 4
UP: MOV AL, [SI]
    CMP AL, [SI+1]
    JNC DN
    XCHG AL, [SI+1]
    XCHG AL, [SI]
DN: INC SI
    LOOP UP
    DEC BX
    JNZ UP1
    ENDS
    END
```

Program 16 (a):

```
- MODEL SMALL
- DATA
    ARRAY DW 9,6,8,2,6,
             7,3,4,2,4
- CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV BX, 10
UP1: MOV SI, OFFSET ARRAY
    MOV CX, 9
UP: MOV AX, [SI]
    CMP AX, [SI+2]
    JNC DN
    XCHG AX, [SI+2]
    XCHG AX, [SI]
DN: ADD SI, 2
    LOOP UP
    DEC BX
    JNZ UP1
    ENDS
    END
```

11) Finding odd and even numbers in the array:

Program 17: Test the 8 bit no. for odd or even

Algorithm: 1) Start
2) Initialize data segment
3) Load number in register
4) Check number is odd or even
5) If number is odd store result to odd
6) Store result to even
7) Stop

Flowchart: Start → Initialize data segment → Load no. in AL → Rotate AL towards right by 1 bit to check no. → If CF = 1 ? (N / Y) → Store EVEN no. to the memory / Store ODD no. to the memory → Stop

```
.MODEL SMALL
.DATA
    NUM  DB  29H
    ODD  DB  0
    EVEN DB  0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AL, NUM1
    ROR AL, 1
    JNC DN
    ROL AL, 1
    MOV ODD, AL
    JMP EXIT
DN: ROL AL, 1
    MOV EVEN, AL
EXIT: ENDS
    END
```

Program 18: Test the 16 bit no. for odd or even

Algorithm: 1) Start
2) Initialize data segment
3) Load number in register
4) Check no. is odd or even
5) If no. is odd then store result to odd
6) Store result to even
7) Stop

Flowchart: Start → Initialize data segments → Load number in AX → Rotate AX to right by 1 to check number → If CF = 1 ? (N / Y) → Store EVEN no. to memory / Store ODD no. to memory → Stop

```
.MODEL SMALL
.DATA
    NUM  DW  898BH
    ODD  DW  0
    EVEN DW  0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, NUM
    ROR AX, 1
    JNC DN
    ROL AX, 1
    MOV ODD, AX
    JNC EXIT
DN: ROL AX, 1
    MOV EVEN, AX
EXIT: ENDS
    END
```


Program 19: Count odd numbers in the array of 16 bit numbers:

Algorithm: 1) Start
2) Initialize data segment
3) Initialize ODD_counter to 0
4) Initialize word counter
5) Initialize memory pointer to read numbers.
6) Read number
7) Check number for ODD
8) If number ≠ ODD then goto step 10
9) Increment ODD_counter by 1
10) Increment memory pointer to read next number
11) Decrement word counter by 1
12) If word counter ≠ 0 then go to step 6.
13) Store result
14) Stop

Flowchart: Start → Initialize data segment → Initialize ODD_counter to 0 / count odd numbers → Initialize word counter CX and memory pointer SI → Read number from array to AX → Rotate AX by 1 bit to right → If CF = 1 ? (N / Y) → Increment ODD counter by 1 → Increment memory pointer by 2 and decrement word counter by 1 → C → If Word counter = 0 ? (N / Y) → Stop → C1

```
.MODEL SMALL
.DATA
    ARRAY DW 134H, 65H, 876H, 976H, 23H
    ODD_NO DW 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CX, 5
    MOV SI, OFFSET ARRAY
NEXT: MOV AX, [SI]
    ROR AX, 1
    JNC DN
    INC ODD_NO
DN: ADD SI, 2
    LOOP NEXT
    ENDS
    END
```

172

(6)

Program: To transfer block of 10 numbers from one location to another location.

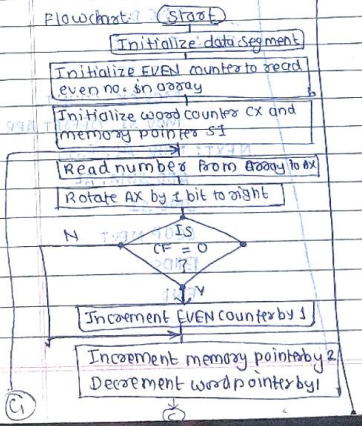
```
.MODEL SMALL
.DATA
    SRC_ARR  DW  1,2,3,4,5,6,7,8,9,10
    DST_ARR  DW  5 DUP(0)
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CX, 10
    MOV SI, OFFSET SRC_ARR
    MOV DI, OFFSET DST_ARR
UP: MOV AX, [SI]
    MOV [DI], AX
    ADD SI, 2
    ADD DI, 2
    LOOP NEXT
    ENDS
    END
```

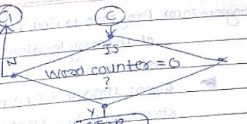
173

Program 20: Count even nos. in array of 16 bit numbers:

Algorithm: 1) Start
2) Initialize data segment
3) Initialize ODD_counter to 0
4) Initialize word counter
5) Initialize memory pointer to read numbers.
6) Read number
7) Check number for EVEN
8) If no. ≠ EVEN then goto step 10.
9) Increment EVEN_ counter by 1
10) Increment memory pointer to read next number
11) Decrement word counter by 1
12) If word counter ≠ 0 then goto step 6
13) Store result
14) Stop

Flowchart:






```
.MODEL SMALL
.DATA
    ARRAY DW 134H, 65H, 876H, 976H, 23H
    EVEN_NO DW 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CX, 5
    MOV SI, OFFSET ARRAY
NEXT: MOV AX, [SI]
    ROR AX, 1
    JC DN
    INC EVEN_NO
DN: ADD SI, 2
    DEC CX
    LOOP NEXT
    ENDS
    END
```

174

Program 20(b): Program to find number stored at memory location 1D005H is odd or even

Given: 1D005H = 8 bit number
store result in: 1D006 = if no. is even
1D007 = if no. is odd

```
MOV AX, 1D00SH
MOV DS, AX
MOV AL, [0005H]
MOV BL, AL
ROR AL, 1
JC DN
MOV [0006H], BL
JMP EXIT
DN : MOV [0007H], BL
EXIT: ENDS
      END
```

Program 20(c): Addition of only ODD numbers in the array

```
• MODEL SMALL
• DATA
    ARR DB 6,5,21,3,8,..
    ARR_ODD DB 6 DUP(0)
    COUNT DW 0
    SUM DB 0
• CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CX, 5
    MOV SI, OFFSET ARR
    MOV DI, OFFSET ARR_ODD
UP: MOV AL, [SI]
    ROR AL, 1
    JNC DN
    ROL AL, 1
    INC COUNT
    MOV [DI], AL
    INC DI
DN: INC SI
    LOOP UP
    MOV CX, COUNT
    MOV SI, OFFSET ARR_ODD
NEXT: MOV AL, [SI]
    ADD SUM, AL
    INC SI
    LOOP NEXT
    ENDS
    END
```

175

Program 20(c): Addition of only EVEN numbers in the array

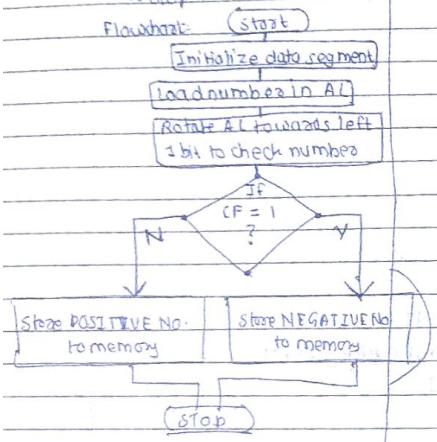
```
• MODEL SMALL
• DATA
    ARR DB 6,5,21,3,8,9
    ARR_EVEN DB 6 DUP(0)
    COUNT DW 0
    SUM DB 0
• CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CX, 6
    MOV SI, OFFSET ARR
    MOV DI, OFFSET ARR_EVEN
UP: MOV AL, [SI]
    ROR AL, 1
    JC DN
    ROL AL, 1
    INC COUNT
    MOV [DI], AL
    INC DI
DN: INC SI
    LOOP UP
    MOV CX, COUNT
    MOV SI, OFFSET ARR_EVEN
NEXT: MOV AL, [SI]
    ADD SUM, AL
    INC SI
    LOOP NEXT
    ENDS
    END
```

12 Finding positive and negative numbers from the array

Program 21: Test the 8 bit no for positive or negative

Algorithm: 1) Start
2) Initialize data segment
3) Load number in register
4) Check number is positive or negative
5) If number is positive then then store result to positive and goto to step 7
6) Store result to negative
7) Stop

Flowchart:

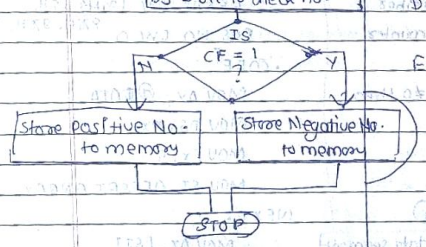



```
▶ •MODEL SMALL
  •DATA
     NUM DB 89H
     POS DB 0
     NEG DB 0
  •CODE
     MOV AX, @DATA
     MOV DS, AX
     MOV AL, NUM1
     ROL AL, 1
     JNC DN
     ROR AL, 1
     MOV NEG, AL
     JMP EXIT
DN:  ROR AL, 1
     MOV POS, AL
EXIT: ENDS
     END
```

Program 22 Test the 16 bit no. for positive or negative.

Algorithm: 1) Start
2) Initialize data segment
3) Load number in register
4) Check number is positive or negative.
5) If number is positive then store result to positive and goto step 7
6) Store result to negative
7) Stop

Flowchart:




```
▶ •MODEL SMALL
  •DATA
     NUM DW -99H
     POS DW 0
     NEG DW 0
  •CODE
     MOV AX, @DATA
     MOV DS, AX
     MOV AX, NUM1
     ROL AX, 1
     JNC DN
     ROR AX, 1
     MOV NEG, AX
     JMP EXIT
DN:  ROR AX, 1
     MOV POS, AX
EXIT: ENDS
     END
```

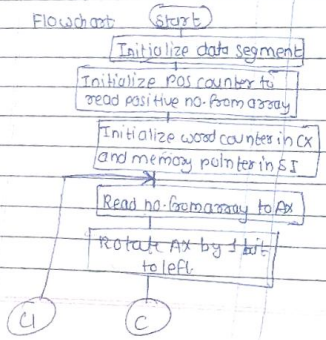
178

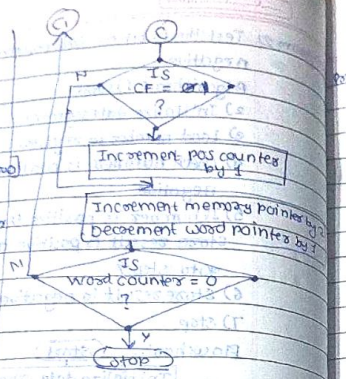
Program 23 Count Positive Numbers in the array of 16 bit numbers

Algorithm

1) Start
2) Initialize data segment
3) Initialize POSITIVE_count to 0
4) Initialize word counter
5) Initialize memory pointer to read number
6) Read number
7) Check number for POSITIVE
8) If number ≠ POSITIVE then goto step 10
9) Increment POSITIVE_ counter by 1
10) Increment memory pointer to read next number
11) Decrement word counter by one
12) If word counter ≠ 0 then goto step 6
13) Store result
14) Stop

Flowchart






```
▶ .MODEL SMALL
.DATA
    ARRAY DW 134H, 65H, 876H, 976H, 23H
    POS_NO DW 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CX, 5
    MOV SI, OFFSET ARRAY
NEXT:
    MOV AX, [SI]
    ROL AX, 1
    JC DN
    INC POS_NO
DN:
    ADD SI, 2
    DEC CX
    LOOP NEXT
    ENDS
    END
```

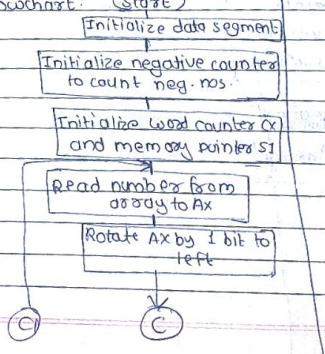
179

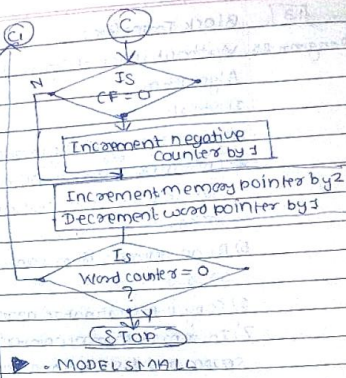
Program 24 Count negative numbers in the array of 16 bit numbers

Algorithm:

1) start
2) Initialize data segment
3) Initialize NEGATIVE_counter to 0
4) Initialize word counter
5) Initialize memory pointer to read next number
6) Read number
7) Check number for NEGATIVE
8) IF NUMBER ≠ NEGATIVE then goto step 10
9) Increment NEGATIVE_COUNT by 1
10) Increment memory pointer to read next number
11) Decrement word counter by one
12) If word counter ≠ 0 then goto step 6
13) Store result
14) stop

Flowchart:






```
▶ .MODEL SMALL
.DATA
    ARRAY DW 134H, 65H, 876H, 976H, 23H
    NEG_NO DW 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CX, 5
    MOV SI, OFFSET ARRAY
NEXT:
    MOV AX, [SI]
    ROL AX, 1
    JNC DN
    INC NEG_NO
DN:
    ADD SI, 2
    DEC CX
    LOOPNEXT
    ENDS
    END
```

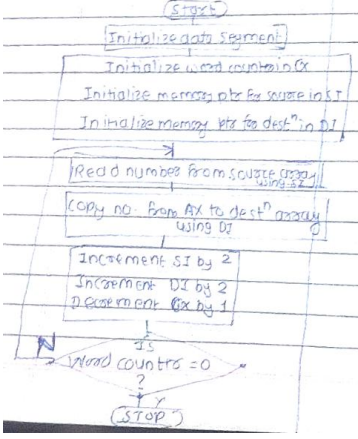
13 Block Transfer:

Program 25: Without using string instruction

Algorithm:

1) Start
2) Initialize data segment
3) Initialize word counter
4) Initialize memory pointer for source and destination array.
5) Read number from source array
6) Copy it to destination array
7) Increment memory pointer for source and destination array for next number
8) Decrement word counter by 1
9) If word counter ≠ 0 then go to step 5
10) Stop

Flowchart:




```
.MODEL SMALL
.DATA
SRC_ARR DW 1234H, 4321H, 7894H, 3658H, 45ABH
DST_ARR DW 5 DUP(0)
.CODE
MOV AX, @DATA
MOV DS, AX
MOV CX, 5
MOV SI, OFFSET SRC_ARR
MOV DI, OFFSET DST_ARR
UP:
MOV AX, [SI]
MOV [DI], AX
ADD SI, 2
ADD DI, 2
LOOP NEXT
ENDS
END
```

Program 26: Using string instruction

Algorithm:

1) Start
2) Initialize data and extra segment i.e DS and ES
3) Initialize word counter
4) Initialize memory pointer for source and destination array
5) Read number from source array
6) Copy it to destination array.
7) Increment memory pointer for source and destination array for next number
8) Decrement word counter by 1
9) If word counter ≠ 0 then go to step 5
10) Stop

```
.MODEL SMALL
.DATA
SRC_ARR DW 1234H, 4321H, 7894H, 3658H, 45ABH
DST_ARR DW 5 DUP(0)
.CODE
MOV AX, @DATA
MOV DS, AX
MOV ES, AX
MOV CX, 5
MOV SI, OFFSET SRC_ARR
MOV DI, OFFSET DST_ARR
UP: MOVSW
LOOP UP
ENDS
END
```

Program 26(c) Program to transfer 10 byte data from base of data segment to base of extra segment

DS base address = 2C000H

ES base address = DE0C0H

```
.MODEL SMALL
.CODE
MOV AX, 2C00H
MOV DS, AX
MOV AX, DE00H
MOV ES, AX
MOV SI, 0000H
MOV DI, 0000H
MOV CX, 000AH
UP: MOV AL, [SI]
MOV ES:[DI], AL
INC SI
INC DI
LOOP UP
ENDS
END
```

PAGE NO. 182

Program 26(b) Write an ALP to transfer a block of 1 kb of data stored at location 1000H onward to location 4000H onwards in the data segment

```
.MODEL SMALL
.CODE
MOV AX, 1000H
MOV DS, AX
MOV AX, 4000H
MOV ES, AX
MOV SI, 0000H
MOV DI, 0000H
MOV CX, 03FFH
UP: MOVSB
LOOP UP
ENDS
END
```

```
.MODEL SMALL
.DATA
SRC_ARR DW 1234H, 4321H, 7894H, 9658H, 4567H
DST_ARR DW 5 DUP(0)
.CODE
MOV AX, @DATA
MOV DS, AX
MOV CX, 5
MOV SI, OFFSET SRC_ARR
MOV DI, OFFSET DST_ARR
ADD SI, 4
ADD DI, 4
UP: MOV AX, [SI]
MOV [DI], AX
SUB SI, 2
SUB DI, 2
LOOP NEXT
ENDS
END
```

Program 26(c) Overlapping block transfer

Algorithm 1) Start
2) Initialize data segment
3) Initialize word counter
4) Initialize memory pointers for source and destination array.
5) Read number from array
6) Copy it to destination array.
7) Decrement memory pointers for source and destination array for next number.
8) Decrement word counter by 1
9) If word counter ≠ 0, go to step 5

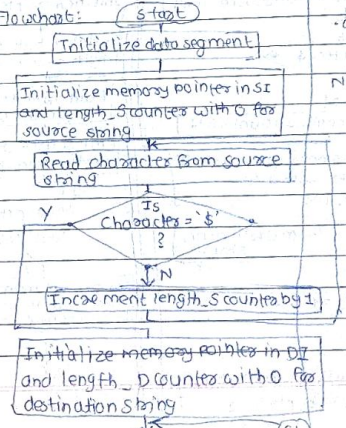
PAGE NO. 183

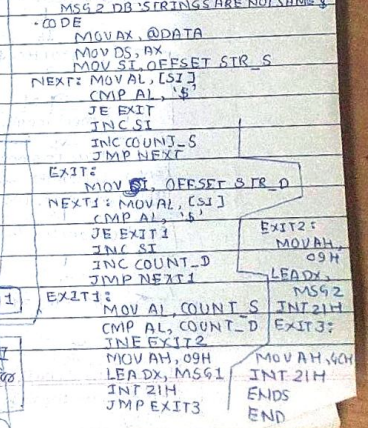
14 Comparison of two strings without using string instructions

Algorithm
1) Comp. by checking length of both the strings

Program 27 Algorithm: 1) Start
2) Initialize data segment
3) Find the length of source string.
4) Find the length of destination string.
5) Compare length of both the strings
6) If length of both string are not same then go to step 9
7) Display message 'Strings are same'
8) Stop
9) Display message 'strings are not same
10) stop

Flowchart:





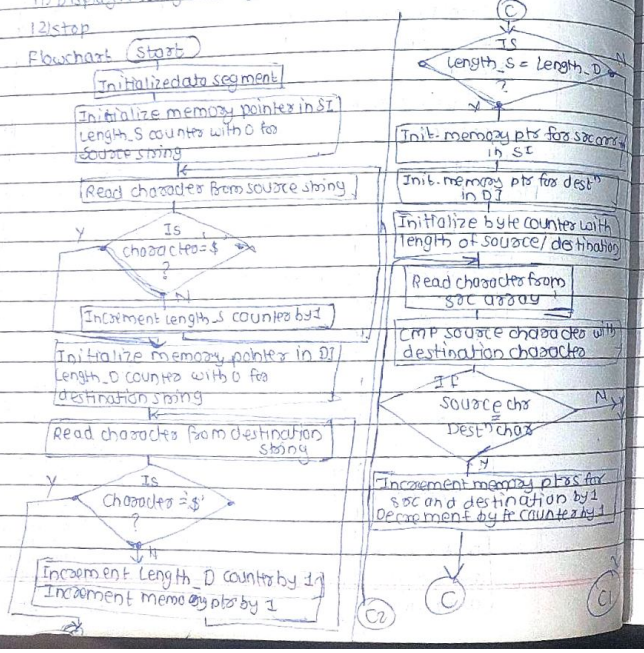

```
.MODEL SMALL
.DATA
STR_S DB 'COMPUTER$'
STR_D DB 'computer$'
COUNT_S DB 0
COUNT_D DB 0
MSG1 DB 'STRINGS ARE SAME$'
MSG2 DB 'STRINGS ARE NOT SAME$'
.CODE
MOV AX, @DATA
MOV DS, AX
MOV SI, OFFSET STR_S
NEXT: MOV AL, [SI]
CMP AL, '$'
JE EXIT
INC SI
INC COUNT_S
JMP NEXT
EXIT: MOV SI, OFFSET STR_D
NEXT1: MOV AL, [SI]
CMP AL, '$'
JE EXIT1
INC SI
INC COUNT_D
JMP NEXT1
EXIT1: MOV AL, COUNT_S
CMP AL, COUNT_D
JNE EXIT2
MOV AH, 09H
LEA DX, MSG1
INT 21H
JMP EXIT3
EXIT2: MOV AH, 09H
LEA DX, MSG2
INT 21H
EXIT3: MOV AH, 4CH
INT 21H
ENDS
END
```

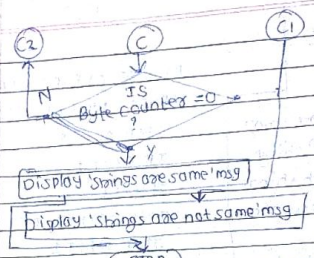
2) Comparison of string by checking length and character with case

Algorithm:

1) Start
2) Initialize data segment
3) Find length of source string
4) Find length of destination string
5) Compare length of both strings
6) If length of both string are not same then goto step 11
7) Compare string character by character
8) If characters of both the strings are not same then goto step 11
9) Display message 'Strings are same'
10) stop
11) Display message 'strings are not same'.
12) stop

Flowchart






```
.MODEL SMALL
.DATA
    STR_S DB 'COMPUTER$'
    STR_D DB 'computer$'
    COUNT_S DB 0
    COUNT_D DB 0
    MSG1 DB 'Strings are same$'
    MSG2 DB 'Strings are not same$'
.CODE
    MOV AX,@DATA
    MOV DS,AX
    MOV SI,OFFSET STR_S
NEXT: MOV AL,[SI]
    CMP AL,'$'
    JE EXIT
    INC SI
    INC COUNT_S
    JMP NEXT
EXIT:
    MOV SI,OFFSET STR_D
NEXT1: MOV AL,[SI]
    CMP AL,'$'
    JE EXIT1
    INC SI
    INC COUNT_D
    JMP NEXT1
EXIT1:
    MOV AL,COUNT_S
    CMP AL,COUNT_D
    JNE EXIT2
    MOV SI,OFFSET STR_S
    MOV DI,OFFSET STR_D
UP:
    MOV AL,[SI]
    CMP AL,[DI]
    JNE EXIT2
    INC SI
    INC DI
    DEC COUNT_S
    JNZ UP
    MOV AH,09H
    LEA DX,MSG1
    INT 21H
    JMP EXIT3
EXIT2: MOV AH,09H
    LEA DX,MSG2
    INT 21H
EXIT3: MOV AH,4CH
    INT 21H
    END DS
    END
```

PAGE NO. 186

Program 29 Comp. of two strings using string instructions

1) Start
2) Initialize DS and ES
3) Find the length of source string
4) Find the length of dest^n string
5) Compare length of both strings
6) If length not same then goto step 11
7) Compare string character by character
8) If characters of both strings not same then goto step 11
9) Display message 'strings are same'$
10) Stop
11) Display message 'strings are not same'$

```
.MODEL SMALL
.DATA
STR_S DB 'COMPUTER$'
STR_D DB 'computer$'
COUNT_S DB 0
COUNT_D DB 0
MSG1 DB 'strings are same$'
MSG2 DB 'Strings are not same$'
.CODE
MOV AX, @DATA
MOV DS, AX
MOV ES, AX
MOV SI, OFFSET STR_S
NEXT: MOV AL, [SI]
CMP AL, '$'
JE EXIT
INC SI
INC COUNT_S
JMP NEXT
EXIT:
MOV SI, OFFSET STR_D
NEXT1: MOV AL, [SI]
CMP AL, '$'
JE EXIT1
INC SI
INC COUNT_D
JMP NEXT1
EXIT1:
MOV AL, COUNT_S
CMP AL, COUNT_D
JNE EXIT2
CLD
MOV CH, 0
MOV CL, COUNT_S
MOV SI, OFFSET STR_S
MOV DI, OFFSET STR_D
UP:
CMPSB
JNZ EXIT2
LOOP UP
MOV AH, 09H
LEA DX, MSG1
INT 21H
JMP EXIT3
EXIT2:
MOV AH, 09H
LEA DX, MSG2
INT 21H
EXIT3:
MOV AH, 4CH
INT 21H
ENDS
END
```

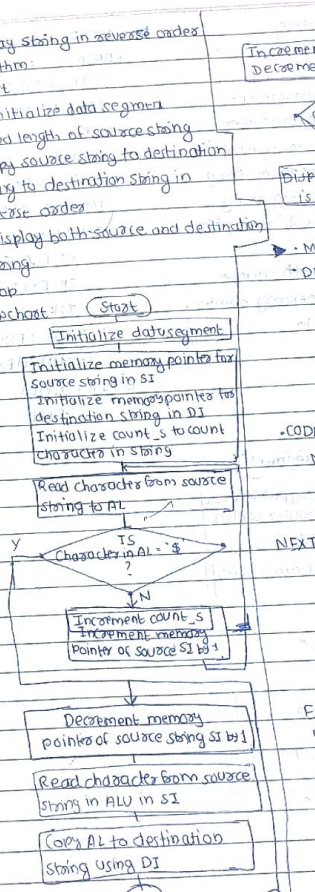
PAGE NO. 187

15 Display string in reverse order

Algorithm

1) Start
2) Initialize data segment
3) Find length of source string
4) Copy source string to destination string to destination string in reverse order
5) Display both source and destination string
6) Stop

Flowchart



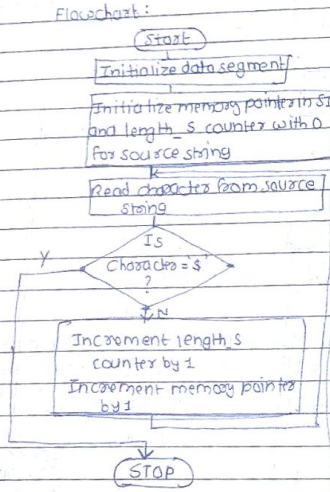

```
.MODEL SMALL
.DATA
STR_S DB 'COMPUTER DEPARTMENT$'
STR_D DB 50 DUP('$')
MSG1 DB 10,13,'The source string$'
MSG2 DB 10,13,'The string after reverse$'
COUNT DB 0
.CODE
MOV AX, @DATA
MOV DS, AX
MOV SI, OFFSET STR_S
NEXT: MOV AL, [SI]
CMP AL, '$'
JE EXIT
INC SI
INC COUNT
JMP NEXT
EXIT: MOV DI, OFFSET STR_D
UP: DEC SI
MOV AL, [SI]
MOV DI, [AL]
INC DI
DEC COUNT
JNZ UP
MOV AH, 09H
LEA DX, MSG1
INT 21H
MOV AH, 09H
LEA DX, STR_S
INT 21H
MOV AH, 09H
LEA DX, STR_D
INT 21H
MOV AH, 4CH
INT 21H
ENDS
END
```

16 Find length of string

Program 31: Algorithm

1) Start
2) Initialize data segment
3) Initialize length counter
4) Initialize memory pointer to read character from string
5) Read character from string
6) If character is '$' then goto step 10
7) Increment length counter
8) Increment memory pointer to next character
9) Goto step 5
10) Stop

Flowchart:



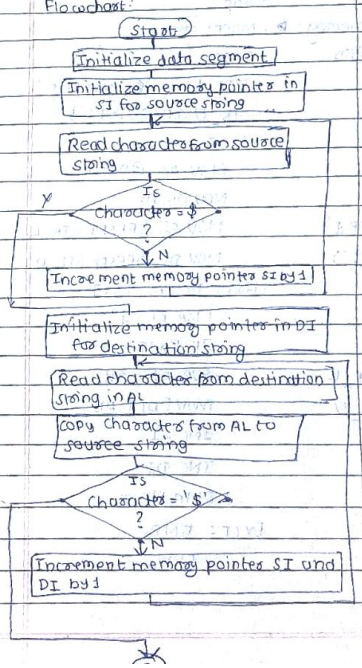

```
.MODEL SMALL
.DATA
    STR_S DB 'COMPUTER$'
    LENGTH DB 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV SI, OFFSET STR_S
NEXT: MOV AL, [SI]
    CMP AL, '$'
    JE EXIT
    INC SI
    INC LENGTH
    JMP NEXT
EXIT: ENDS
    END
```

17 Concatenation of two strings

Program 32: Algorithm

1) Start
2) Initialize data segment
3) Initialize memory pointer for source and destination
4) Move memory pointer of source string to end of string
5) Copy characters from destination string to source string
6) Stop

Flowchart:



C

Copy '$' to at the end of source string

Display source string after concatenation

STOP

```
.MODEL SMALL
.DATA
    STR_S DB 'COMPUTER ENGINEERING$'
    STR_D DB '_INFORMATION TECHNOLOGY$'
    MSG1 DB 'AFTER CONCATENATION ...:$'
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV SI, OFFSET STR_S
NEXT:
    MOV AL, [SI]
    CMP AL, '$'
    JE EXIT
    INC SI
    JMP NEXT
EXIT: MOV DI, OFFSET STR_D
UP: MOV AL, [DI]
    CMP AL, '$'
    JE EXIT1
    MOV [SI], AL
    INC SI
    INC DI
    JUMP UP
EXIT1: MOV AL, '$'
    MOV [SI], AL
    MOV AH, 09H
    LEA DX, MSG1
    INT 21H
    MOV AH, 09H
    LEA DX, STR_D
    INT 21H
    MOVAH, 4CH
    INT 21H
    ENDS
    END
```

18 Convert lower case string to upper case

Algorithm:
1) Start
2) Initialize data segment
3) Initialize memory pointer for source and destination
4) Read character from source memory
5) If end of the string, goto step 10
6) Convert from lower to upper case
7) Store into destination memory
8) Increement memory pointers by 1
9) Goto step 4
10) Stop

```
.MODEL SMALL
.DATA
    STR_L DB 'COMPUTER$'
    STR_U DB 20 DUP('$')
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV SI, OFFSET STR_L
    MOV DI, OFFSET STR_U
NEXT: MOV AL, [SI]
    CMP AL, '$'
    JE EXIT
    SUB AL, 20H
    MOV DI, [AL]
    INC SI
    INC DI
    JMP NEXT
EXIT: ENDS END
```

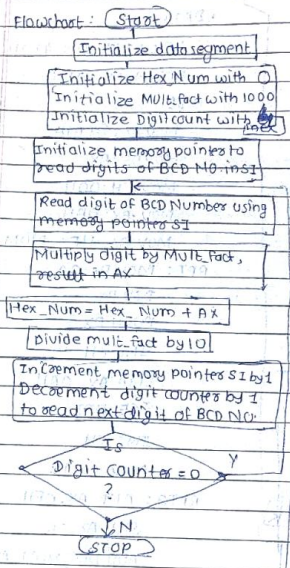
19 Convert uppercase to lowercase

Algorithm: 1) start
2) Initialize data segment
3) Initialize memory pointer for source and destination
4) Read character from source memory
5) If end of string goto step 10
6) Convert from uppers to lowers case
7) Store into destination memory
8) Increement memory pointers by 1
9) Goto step 4
10) Stop

```
.MODEL SMALL
.DATA
    STR_L DB 'COMPUTER$'
    STR_U DB 20 DUP('$')
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV SI, OFFSET STR_L
    MOV DI, OFFSET STR_U
NEXT: MOV AL, [SI]
    CMP AL, '$'
    JE EXIT
    SUB AL, 20H
    MOV [DI], AL
    INC SI
    INC DI
    JMP NEXT
EXIT: ENDS
    END
```


20 Convert BCD NO. to Hexadecimal

Flowchart:




```
.MODEL SMALL
.DATA
    DEC_NUM DB '1234'
    HEX_NUM DW 0
    MULT_FACTOR DW 3E8H
    DIGIT COUNT DW 4
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV BX, 0AH
    MOV CX, DIGIT_COUNT
    MOV SI, OFFSET DEC_NUM
UP: MOV AL, [SI]
    AND AX, 000FH
    MUL MULT_FACTOR
    ADD HEX_NUM, AX
    MOV AX, MULT_FACTOR
    MOV DX, 00
    DIV BX
    MOV MULT_FACTOR, AX
    INC SI
    LOOP UP
    ENDS
    END
```

PAGE NO. 192

21 Convert Hexadecimal number to BCD


```
.MODEL SMALL
.DATA
    HEX_NUM  DW 4D2H
    DEC_NUM  DW 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CH, 00H
    MOV BX, 0AH
    MOV bx, HEX_NUM
RPT: MOV Dx, 00
    DIV BX
    PUSH DX
    INC CH
    CMP AX, 0AH
    JGE RPT
    INC CH
    MOV CL, 12
RPT3: AND AX, 0FH
    SHL AX, CL
    ADD DEC_NUM, AX
    SUB CL, 4
    POP AX
    DEC CH
    JNZ RPT3
    MOV AH, 4CH
    INT 21H
    ENDS
    END
```

PAGE NO. 193

22 BCD to ASCII conversion

Algorithm:

1) Start
2) Initialize data segment
3) Initialize memory pointers for source and destination
4) Read BCD number source memory
5) Convert to ASCII
6) Store into destination memory
7) Increment memory pointers by 1
8) Decrement byte counter by 1
9) If byte counter ≠ 0 then goto step 4
10) Stop

```
.MODEL SMALL
.DATA
    BCD_NO  DB 1,2,3,4,5,6,7,8,9,0
    ASC_NO  DB 10 DUP (0)
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CX, 10
    MOV SI, OFFSET BCD_NO
    MOV DI, OFFSET ASC_NO
UP: MOV AL, [SI]
    ADD AL, 30H
    MOV DI, [AL]
    INC SI
    INC DI
    LOOP UP
    ENDS
    END
```

23 ASCII to BCD conversion

Algorithm

1) Start
2) Initialize data segment
3) Initialize memory pointers for source and destination
4) Read ASCII number from src memory
5) Convert to BCD
6) Store into destination memory
7) Increment memory pointers by 1
8) Decrement byte counter by 1
9) If byte counter ≠ 0 then goto step 4
10) Stop

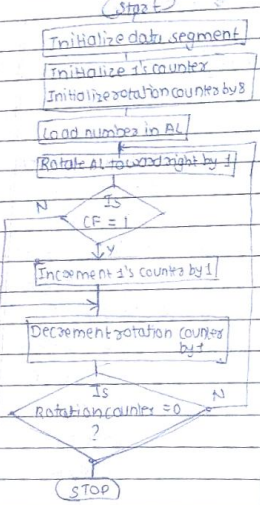
```
.MODEL SMALL
.DATA
    ASC_NO  DB '1','2','3','4','5','6',
            '7','8','9','0'
    BCD_NO  DB 10 DUP (0)
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CX, 10
    MOV SI, OFFSET ASC_NO
    MOV DI, OFFSET BCD_NO
UP: MOV AL, [SI]
    SUB AL, [30H]
    MOV [DI], AL
    INC SI
    INC DI
    LOOP UP
    ENDS
    END
```

PAGE NO. 104

24 Count numbers of 1's and 0's in 8 bit or 16 bit number.

Program 36(a) Count no. of 1's in 8 bit number

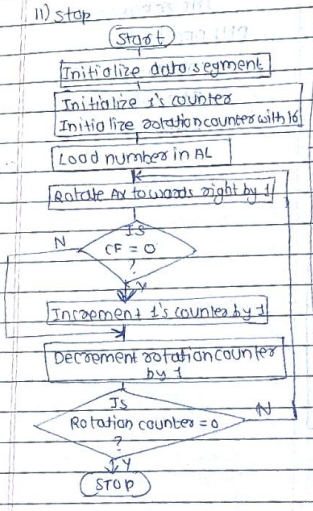
Algorithm: 1) Start
2) Initialize data segment
3) Initialize rotation counter by 8
4) Initialize 1's counter to count 1's
5) Load number
6) Rotate number left or right by 1
7) If CF ≠ 1 then goto step 9
8) Increament 1's counter by 1
9) Decrement rotation counter by 1
10) If rotation counter ≠ 0, goto step 6
11) Stop




```
.MODEL SMALL
.DATA
NUM DB OFFH
ONES DB 0
.CODE
MOV AX, @DATA
MOV DS, AX
MOV CX, 8
MOV AL, NUM
UP: ROR AL, 1
JNC DN
INC ONES
DN: DEC CX
LOOP UP
ENDS
END
```

Program 36(b) Count no. of 1's in 16 bit number

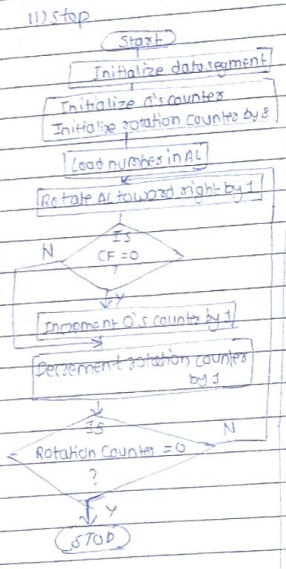
Algorithm: 1) Start
2) Initialize data segment
3) Initialize rotation counter by 16
4) Initialize 1's counter to count 1's
5) Load number
6) Rotate number left or right by 1
7) If CF ≠ 1 then goto step 9
8) Increament 1's counter by 1
9) Decrement rotation counter by 1
10) If rotation counter ≠ 0 then goto step 6
11) Stop




```
.MODEL SMALL
.DATA
NUM DW OFFAAH
ONES DW 0
.CODE
MOV AX, @DATA
MOV DS, AX
MOV CX, 16
MOV AX, NUM
UP: ROR AX, 1
JNC DN
INC ONES
DN: DEC CX
LOOP UP
ENDS
END
```

Program 36(c) Count no. of 0's in 8-bit number

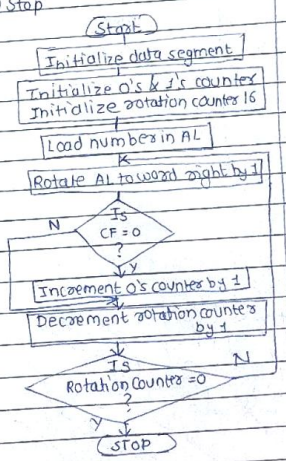
Algorithm: 1) start
2) Initialize data segment
3) Initialize rotation counter by 8
4) Initialize 0's counter to count 0's
5) Load number
6) Rotate number left or right by 1
7) If CF≠0 then goto step 9
8) Increment zero's counter by 1
9) Decrement rotation counter by 1
10) If rotation counter ≠0 then goto step 6
11) Stop




```
.MODEL SMALL
.DATA
    NUM DB 0AAH
    ZEROS DB 0
.CODE
    MOV AX,@DATA
    MOV DS, AX
    MOV CX, 8
    MOV AL, NUM
UP: ROR AL, 1
    JC DN
    INC ZEROS
DN: DEC CX
    LOOP UP
    ENDS
    END
```

Program 37(a) Count numbers of 0's and 1's in 16 bit number

Algorithm: 1) Start
2) Initialize data segment
3) Initialize rotation counter by 16
4) Initialize zero's counter to count 0's
5) Initialize one's counter to count 1's
6) Load number
7) Rotate number left or right by 1
8) If CF≠0 then goto step 11
9) Increament zeroes counter by 1
10) Go to step 12
11) Increament one's counter by 1
12) Decrement rotation counter by 1
13) If rotation counter ≠0 then goto step 5
14) Stop




```
.MODEL SMALL
.DATA
    NUM DW 0FFAAH
    ZEROS DW 0
    ONES DW 0
.CODE
    MOV AX,@DATA
    MOV DS, AX
    MOV CX, 16
    MOV AX,NUM
UP: ROR AX, 1
    JC DN
    INC ZEROS
    JMP NEXT
DN: INC ONES
NEXT: DEC CX
    LOOP UP
    ENDS
    END
```

Program to multiply the contents of AX by six using shift instruction

```
.MODEL SMALL
.DATA
    NUM1 DW 0002H
    NUM2 DB 6
    RESULT DW 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, NUM1
    MOV DL, NUM2
    MOV BX, 0
    MOV CX, 8
UP: ADD BX, AX
    SHL DL, 1
    JNC DN
    ADD BX, AX
DN: DEC CX
    LOOP UP
    MOV RESULT, BX
    MOV AH, 4CH
    INT 21H
    ENDS
    END
```

ALP to count number of 0's in AL register

```
.MODEL SMALL
.DATA
    NUM DB 0AAH
    ZEROS DB 0
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV CX, 8
    MOV AL, NUM
UP: ROR AL, 1
    JC DN
    INC ZEROS
DN: DEC CX
    LOOP UP
    ENDS
    END
```