DATE :

EXPERIMENT NO – 3.2(i)
IMPLEMENT ADDITION OF 16 BIT NUMBERS (KIT)

AIM

To perform addition of 16 – bit numbers using kit .

ALGORITHM

1. Start the program
2. Set the Source Index (SI) register to point to memory address 3000.
3. Set the Destination Index (DI) register to point to memory address 4000.
4. Load the value stored at the memory address pointed to by SI into the AX register.
5. Increment the SI register by 2 bytes (assuming SI is pointing to a word or 2 bytes).
6. Load the value stored at the updated memory address pointed to by SI into the BX register.
7. Add the values stored in AX and BX registers and store the result in the AX register.
8. Check if there was no carry generated during the addition.
   - If no carry occurred, store the value 0001 at the memory address pointed to by DI.
   - If a carry occurred, store the value 0000 at the memory address pointed to by DI.
9. Increment the DI register to point to the next memory address.
10. Store the contents of the AX register at the memory address pointed to by DI.
11. Halt the execution of the program.

TEST CASE – 1

INPUT

3000 : B0
3001 : FA
3002 : B0
3003: 12

OUTPUT

4000 : 01
4001 : 60
4002 : 0D

TEST CASE – 2

INPUT

3000 : 10
3001 : EA
3002 : 31
3003: 04

OUTPUT

4000 : 00
4001 : 41
4002 : EE

PROGRAM

```
2000  MOV SI,3000
2003  MOV DI,4000
2006  MOV AX,[SI]
2008  INC SI
2009  INC SI
200A  MOV BX,[SI]
200C  ADD AX,BX
200E  JNC 2015
2010  MOV [DI],0001
2013  JMP 2018
2015  MOV [DI],0000
2018  INC DI
2019  MOV [DI],AX
201B  HLT
```

RESULT

Assembly program to perform addition of 16 – bit numbers have been implemented successfully .

DATE :

EXPERIMENT NO – 3.2(ii)
IMPLEMENT SUBTRACTION OF 16 BIT NUMBERS (KIT)

AIM

To perform subtraction of 16 – bit numbers using kit .

ALGORITHM

1. Start the program
2. Set SI register to the memory address 3000.
3. Set DI register to the memory address 4000.
4. Load the value at the memory address stored in SI into the AX register.
5. Increment the SI register by 2 (assuming SI points to a word or 2 bytes).
6. Load the value at the updated memory address stored in SI into the BX register.
7. Compare the values in AX and BX.
    - If AX is less than BX, jump to memory address 2017.
    - Otherwise, continue to the next instruction.
8. Subtract BX from AX.
9. Store the value 0000 at the memory address stored in DI.
10. Jump to memory address 2022.
11. If AX was less than BX:
    - Move the value in AX to the CX register.
    - Move the value in BX to AX.
    - Move the value in CX to BX.
    - Subtract BX from AX.
    - Store the value 0001 at the memory address stored in DI.
12. Increment the DI register by 1.
13. Store the value in AX at the memory address stored in DI.
14. Halt the program.

PROGRAM

```
2000  MOV SI,3000
2003  MOV DI,4000
2006  MOV AX,[SI]
2008  INC SI
2009  INC SI
200A  MOV BX,[SI]
200C  CMP AX,BX
200E  JC 2017
```

TEST CASE – 1

INPUT

3000 : 10
3001 : EA
3002 : 31
3003: 04

OUTPUT

4000 : 00
4001 : DF
4002 : E5

TEST CASE – 2

INPUT

3000 : 05
3001 : 01
3002 : 31
3003: 04

OUTPUT

4000 : 01
4001 : 2C
4002 : 03

```
2010  SUB AX,BX
2012  MOV [DI],0000
2015  JMP 2022
2017  MOV CX,AX
2019  MOV AX,BX
201B  MOV BX,CX
201D  SUB AX,BX
201F  MOV [DI],0001
2022  INC DI
2023  MOV [DI],AX
2025  HLT
```

RESULT

Assembly program to perform subtraction of 16 – bit numbers have been implemented successfully .

INPUT

3000 : 12
3001 : 12
3002 : 10
3003: 01

OUTPUT

4000 : 20
4001 : 33
4002 : 13
4003 : 00

DATE :

EXPERIMENT NO – 3.2(iii)
IMPLEMENT MULTIPLICATION OF 16 BIT NUMBERS (KIT)

AIM

To perform multiplication of 16 – bit numbers using kit .

ALGORITHM

1. Start the program
2. Set the source index register SI to point to memory address 3000.
3. Set the destination index register DI to point to memory address 4000.
4. Load the value at the memory address pointed by SI into the AX register.
5. Increment SI by 2 (assuming SI points to a word or 2 bytes in memory).
6. Load the value at the updated memory address pointed by SI into the BX register.
7. Multiply the value in AX by the value in BX, storing the result in AX (product) and DX (high-order bits of the product).
8. Store the value in AX at the memory address pointed by DI.
9. Increment DI by 2 (assuming DI points to a word or 2 bytes in memory).
10. Store the value in DX at the updated memory address pointed by DI.
11. Halt the program.

PROGRAM

```
2000  MOV SI,3000
2003  MOV DI,4000
2006  MOV AX,[SI]
2008  INC SI
2009  INC SI
200A  MOV BX,[SI]
200C  MUL BX
200E  MOV [DI],AX
2010  INC DI
2011  INC DI
2012  MOV [DI],DX
2014  HLT
```

RESULT

Assembly program to perform multiplicatio of 16 – bit numbers have been implemented successfully .

INPUT

3000 : 12
3001 : 04
3002 : 23
3003: 01

OUTPUT

4000 : 03
4001 : 00
4002 : A9
4003 : 00

DATE :

EXPERIMENT NO – 3.2(iv)
IMPLEMENT DIVISION OF 16 BIT NUMBERS (KIT)

AIM

To perform division of 16 – bit numbers using kit .

ALGORITHM

1. Start the program
2. Set SI (source index) register to point to memory address 3000.
3. Set DI (destination index) register to point to memory address 4000.
4. Load the value at the memory address pointed by SI into register AX.
5. Increment the SI register by 2, assuming it's pointing to a word (2 bytes) in memory.
6. Load the value at the updated memory address pointed by SI into register BX.
7. Divide the value in AX by the value in BX, storing the quotient in AX and the remainder in DX.
8. Store the value in AX at the memory address pointed by DI.
9. Increment the DI register by 2, assuming it's pointing to a word in memory.
10. Store the value in DX at the updated memory address pointed by DI.
11. Halt the program.

PROGRAM

```
2000  MOV SI,3000
2003  MOV DI,4000
2006  MOV AX,[SI]
2008  INC SI
2009  INC SI
200A  MOV BX,[SI]
200C  DIV BX
200E  MOV [DI],AX
2010  INC DI
2011  INC DI
2012  MOV [DI],DX
2014  HLT
```

RESULT

Assembly program to division of 16 – bit numbers have been implemented successfully .

DATE :

EXPERIMENT NO – 3.3
IMPLEMENT SORTING OF 16 – BIT NUMBERS (KIT)

AIM

To perform sorting of 16 – bit numbers using kit .

ALGORITHM

1. Start
2. Load the length of the array into BX.
3. Decrement BX by 1 to represent the number of iterations needed.
4. Load the length of the array into CX.
5. Decrement CX by 1 to represent the number of comparisons within an iteration.
6. Start a loop:
   - Load the value at memory address pointed by SI into AX.
   - Increment SI to point to the next element.
   - Compare AX with the value at the next memory address pointed by SI.
   - If AX is not greater than or equal to the value at the next memory address, repeat step 'c'.
   - If AX is greater than the next value:
     - Swap the values by using XCHG and store the greater value at the current address.
     - Move back SI to the previous position.
     - Store the swapped value at the current address.
7. Increment SI twice to point to the next element in the array.
8. Repeat the loop until CX becomes 0.
9. Decrement BX to track the remaining iterations.
10. If BX is not zero, repeat steps 3 to 7.
11. Halt the execution.

PROGRAM

```
0400  MOV BX,[3000]
0404  DEC BX
0405  MOV CX,[3000]
0409  DEC CX
040A  MOV SI,3002
040D  MOV AX,[SI]
040F  INC SI
```

INPUT

3000 : 06
3001 : 00
3002 : 02
3003 : 11
3004 : 35
3005 : 07
3006 : 23
3007 : 10
3008 : 00
3009 : 7A
300A : 72
300B : F0
300C : 10
300D : B0

OUTPUT

3002 : 35
3003 : 07
3004 : 23
3005 : 10
3006 : 02
3007 : 11
3008 : 00
3009 : 7A
300A : 10
300B : B0
300C : 72
300D : F0

```
0410    INC SI
0411    CMP AX,[SI]
0413    JNA 0410
0415    XCHG AX,[SI]
0417    DEC SI
0418    DEC SI
0419    MOV [SI],AX
041B    INC SI
041C    INC SI
041D    LOOP 040D
041F    DEC BX
0420    JNZ 0405
0422    HLT
```

RESULT

Assembly programs to perform sorting of 16 – bit numbers have been implemented successfully .

DATE :

EXPERIMENT NO – 3.4
IMPLEMENT SEARCHING OF 16 – BIT NUMBERS (EMULATOR)

AIM

To perform searching of 16 – bit numbers using 8086 emulator .

ALGORITHM

1. Define the data segment:
   - STRING1 holds an array of words.
   - MSG1 and MSG2 are strings to be printed based on the search result.
   - SE holds the value to be searched in the array.
2. Define a PRINT macro that displays a message using DOS interrupts.
3. In the code segment:
   - Set up the segment registers.
   - Move the address of the data segment to DS.
   - Set AX to the value to be searched (SE).
   - Load the address of the array STRING1 to SI.
   - Set the loop counter CX to 4 (assuming each word in the array is 2 bytes).
4. Start a loop:
   - Load a word from the memory location pointed to by SI into BX.
   - Compare AX with the value in BX.
   - If they match, jump to a label FO (Found), else continue the loop.
   - Increment SI to point to the next element.
   - Decrement the loop counter CX.
   - If CX is not zero, repeat the loop.
5. If the value was not found (CX became zero), print the message MSG2 (NOT FOUND).
6. If the value was found, print the message MSG1 (FOUND).
7. Terminate the program.

PROGRAM

```
DATA SEGMENT
STRING1 DW 1111H,2222H,3333H,4444H,5555H
MSG1 DB "Element FOUND$"
MSG2 DB "Element NOT FOUND$"
SE DW 3333H
DATA ENDS
```

OUTPUT

Element FOUND

```
PRINT MACRO MSG
MOV AH, 09H
LEA DX, MSG
INT 21H
MOV AH,4CH
INT 21H
ENDM

CODE SEGMENT
ASSUME CS:CODE, DS:DATA
START:
MOV AX, DATA
MOV DS, AX
MOV AX, SE
LEA SI, STRING1
MOV CX, 04H

UP:
MOV BX,[SI]
CMP AX, BX
JZ FO
INC SI
INC SI
DEC CX
JNZ UP
PRINT MSG2
JMP END1

FO:
PRINT MSG1

END1: MOV AH,4CH
INT 21H
CODE ENDS
END START
```

RESULT

Assembly program to perform searching has been implemented successfully using emulator .

DATE :

EXPERIMENT NO – 3.5
IMPLEMENT STRING MANIPULATION – PALINDROME (EMULATOR)

AIM

To write an assembly program to check if a string is palindrome or not using 8086 emulator .

ALGORITHM

1. Define the DATA segment:
   - MSG1: Message prompting the user to enter a string.
   - MSG2: Message indicating that the entered string is a palindrome.
   - MSG3: Message indicating that the entered string is not a palindrome.
   - STR1: Buffer to store the input string, initialized with 50 bytes of zeros.
2. Define the CODE segment and assume segment associations (CS:code, DS:data).
3. Define a label START:
   - Move the address of the DATA segment into AX.
   - Move the value in AX to the DS register.
   - Load the effective address of MSG1 into DX.
   - Set AH to 09H (print string function).
   - Trigger interrupt 21H (INT 21H) to display MSG1 to prompt the user for input.
   - Load the effective address of STR1 into SI and DI.
   - Set AH to 01H (input character function).
4. Start a loop labeled NEXT:
   - Trigger interrupt 21H to input a character from the user.
   - Compare the input character with carriage return (0DH).
   - If it's a carriage return, jump to label TERMINATE.
   - Store the input character at the memory location pointed by DI.
   - Increment DI and continue the loop NEXT.
5. When a carriage return is encountered (label TERMINATE):
   - Store '$' (string terminator) at the memory location pointed by DI.
6. Start a loop labeled DOTHIS:
   - Decrement DI to point to the end of the entered string.
   - Load a character from the beginning of the string using SI into AL.
   - Compare the character at DI with AL.
   - If they are not equal, jump to label NOTPALINDROME.
   - Increment SI, compare SI with DI, and if SI is less than DI, continue the loop.
7. If the string is a palindrome (label PALINDROME):
   - Display MSG2 indicating that the entered string is a palindrome.

OUTPUT

TEST CASE – 1

ENTER THE STRING:CSA
STRING IS NOT PALINDROME

TEST CASE – 2

ENTER THE STRING:malayalam
STRING IS PALINDROME

- Jump to label XX.

8. If the string is not a palindrome (label NOTPALINDROME):
   - Display MSG3 indicating that the entered string is not a palindrome.
9. Label XX:
   - Set AH to 4CH (exit program function).
   - Trigger interrupt 21H (INT 21H) to terminate the program.

PROGRAM

```
data SEGMENT
  MSG1 DB 10,13,'ENTER THE STRING:$'
  MSG2 DB 10,13,'STRING IS PALINDROME$'
  MSG3 DB 10,13,'STRING IS NOT PALINDROME$'
  STR1 DB 50 DUP(0)
data ENDS

code SEGMENT
  ASSUME CS:code,DS:data
  START:
    MOV AX,DATA
    MOV DS,AX
    LEA DX,MSG1
    MOV AH,09H
    INT 21H
    LEA SI,STR1
    LEA DI,STR1
    MOV AH,01H

  NEXT:
    INT 21H
    CMP AL,0DH
    JE TERMINATE
    MOV [DI],AL
    INC DI
    JMP NEXT

  TERMINATE:
    MOV AL,'$'
    MOV [DI],AL
```

```
  DOTHIS:
    DEC DI
    MOV AL,[SI]
    CMP [DI],AL
    JNE NOTPALINDROME
    INC SI
    CMP SI,DI
    JL DOTHIS

  PALINDROME:
    MOV AH,09H
    LEA DX,MSG2
    INT 21H
    JMP XX

  NOTPALINDROME:
    MOV AH,09H
    LEA DX,MSG3
    INT 21H

  XX:
    MOV AH,4CH
    INT 21H
code ENDS
END START
```

RESULT

Assembly program to check whether the input string is palindrome or not has been successfully implemented using 8086 emulator .