NAME : VIRAJ KRISHNAJI PRABHU CLASS/SECTION :SE-CMPN-C
MICROPROCESSOR ROLL NO :16

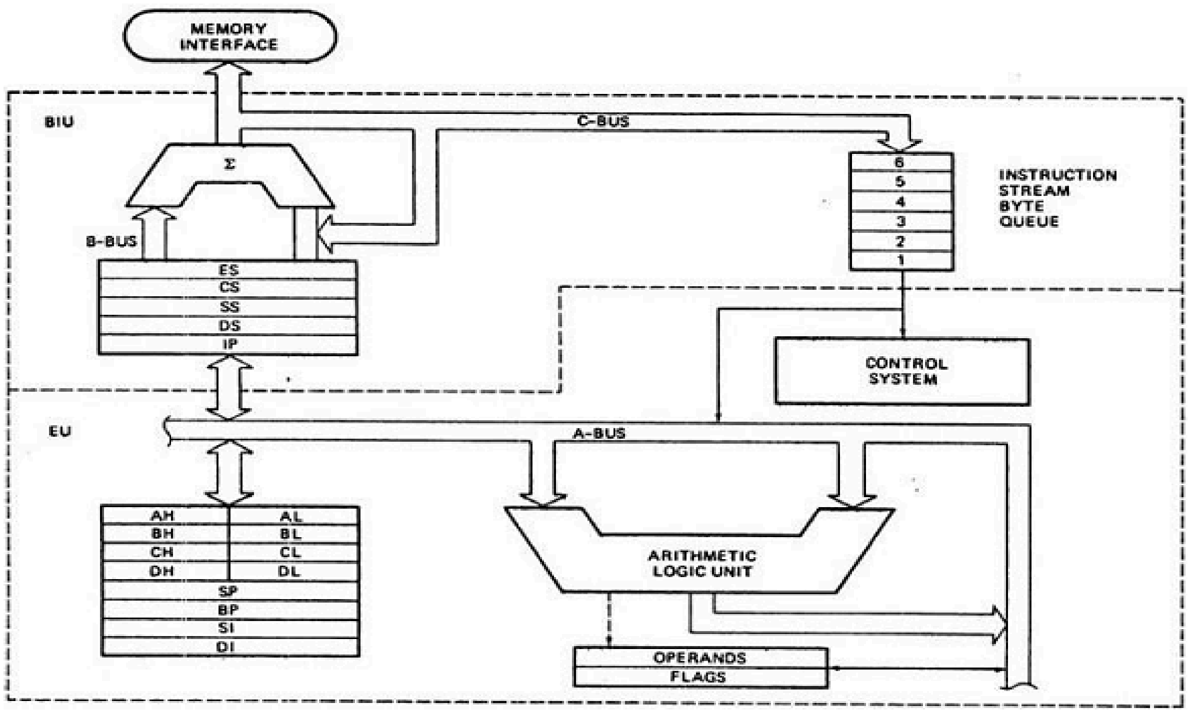
EXPERIMENT 1

Aim : To perform basic Arithmetic operations (8-bits , 16 -bits) using Debug.

Theory :

➢ Architecture of 8086

The 8086 microprocessor is a 16-bit processor, meaning it can process 16 bits of data simultaneously. It was introduced by Intel in 1978 and became a cornerstone for modern computing. The architecture is designed to maximize efficiency and supports a segmented memory model for better program and data management.



Main Features:

- **16-bit Data Bus:** Transfers 16 bits of data at a time.
- **20-bit Address Bus:** Can address up to 1 MB of memory (2^20 addresses).
- **Pipelining:** The BIU prefetches instructions into a 6-byte instruction queue, allowing simultaneous fetching and execution.
- **Segmentation:** Divides memory into segments, each up to 64 KB. This provides better organization and supports modular programming.

Components:

1. **Bus Interface Unit (BIU):**
    - Handles communication between the processor and memory/I/O devices.
    - Performs address generation using segment registers and an offset.
    - Contains a prefetch queue, enabling pipelined execution by fetching the next instruction while the current one is executed.
    - Manages the Code Segment (CS), Data Segment (DS), Stack Segment (SS), and Extra Segment (ES).
2. **Execution Unit (EU):**
    - Executes instructions fetched by the BIU.
    - Includes:
        - Arithmetic Logic Unit (ALU): Performs arithmetic (e.g., addition, subtraction) and logical (e.g., AND, OR) operations.
        - Control Unit: Decodes instructions and orchestrates operations.
        - Registers: Includes general-purpose, segment, and pointer/index registers.
        - Flags: Indicates the status of operations (e.g., Zero Flag, Carry Flag).

Working:

- The BIU fetches instructions and stores them in the queue.
- The EU decodes and executes instructions from the queue.
- Memory addressing is done using segment:offset pairs (e.g., CS:IP for instructions).

➢ 8086 Programmer's Model

The registers in 8086 are essential for controlling operations, storing data, and addressing memory.

General Purpose Registers:

1. **AX (Accumulator)**:
    - Primarily used in arithmetic and I/O operations.
    - Example: In MUL BX, the product is stored in AX.

2. **BX (Base)**:
   - Acts as a base pointer in indirect addressing.
   - Example: MOV AL, [BX] (Moves the value at memory location pointed by BX into AL).
3. **CX (Count)**:
   - Used as a loop counter.
   - Example: LOOP instruction uses CX automatically.
4. **DX (Data)**:
   - Used in division/multiplication and I/O operations.
   - Example: In DIV BX, the remainder is stored in DX.

Segment Registers:

Enable segmented memory access.

1. **CS (Code Segment)**: Points to the memory segment holding the instructions.
2. **DS (Data Segment)**: Holds data used in the program.
3. **SS (Stack Segment)**: Defines the memory segment for the stack.
4. **ES (Extra Segment)**: Used for specific string and memory operations.

Pointer and Index Registers:

1. **SP (Stack Pointer)**: Points to the top of the stack in the SS segment.
2. **BP (Base Pointer)**: Used to access variables within the stack.
3. **SI (Source Index)** and **DI (Destination Index)**: Used in string operations like MOVSB, STOSB.

Flag Register:

Flags represent the state of the processor or the outcome of operations:

- **Zero Flag (ZF)**: Set if the result is zero.
- **Carry Flag (CF)**: Indicates carry or borrow in arithmetic.
- **Sign Flag (SF)**: Reflects the sign of the result.
- **Overflow Flag (OF)**: Set when an arithmetic operation causes overflow.

➢ 8086 Instructions :

8086 instructions have a specific format:

Opcode op1, op2

● Opcode: Specifies the operation to be performed.

● op1: The destination operand (register or memory).

● op2: The source operand (register, memory, or immediate value).

a) Arithmetic Instructions

1. **ADD** (Add):
   - **Syntax:** ADD destination, source
   - **Example:** ADD AX, BX
     Adds the contents of BX to AX. If AX = 1234H and BX = 1111H, then AX = 2345H.

2. **SUB** (Subtract):
   - **Syntax:** SUB destination, source
   - **Example:** SUB AX, CX
     Subtracts CX from AX. If AX = 5678H and CX = 1234H, then AX = 4444H.

3. **MUL** (Multiply, Unsigned):
   - **Syntax:** MUL source
   - **Example:** MUL BX
     Multiplies AX and BX. If AX = 0003H and BX = 0004H, then AX = 000CH.

4. **DIV** (Divide, Unsigned):
   - **Syntax:** DIV source
   - **Example:** DIV BX
     Divides AX by BX. If AX = 1234H and BX = 0010H, then AL = 123H (quotient) and AH = 004H (remainder).

b) Data Transfer Instructions

1. **MOV**:
    - Transfers data between registers, memory, or immediate values.
    - **Syntax:** MOV destination, source
    - **Example :** MOV AX, 1234H (Moves 1234H to AX).

➢ DOS Debug Commands

DEBUG is a DOS utility for examining and manipulating memory, I/O ports, and registers. Some common DEBUG commands are:

1. ? (Help):

- Displays a list of available commands and their usage.
- **Example:** ? displays a brief description of commands.

2. R (Registers):

- Displays or modifies the processor's registers.
- **Example:** R shows the current values of all registers and R AX allows modifying the value of the AX register.

3. A (Assemble):

- Converts assembly language instructions into machine code and stores them in memory.
- **Example:** A 0100 starts assembling instructions at memory location 0100H.

4. T (Trace):

- Executes one instruction at a time, showing the updated register and memory states.
- **Example**: T executes the next instruction and updates the display.

5.G (Go):

- Runs the program from a specified address until a breakpoint or end of execution.
- **Example:** G 0100 (Executes the program starting at address 0100).

NAME : VIRAJ KRISHNAJI PRABHU CLASS/SECTION :SE-CMPN-C
MICROPROCESSOR ROLL NO :16

Code :

- **ADDITION OF TWO 8 - BIT NUMBERS :**

```
-A
073F:0111 MOV AL,05
073F:0113 MOV BL,03
073F:0115 ADD AL,BL
073F:0117
-g 0111

AX=68AC  BX=5678  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0111   NV UP EI PL NZ NA PE NC
073F:0111 B005          MOV     AL,05
-T

AX=6805  BX=5678  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0113   NV UP EI PL NZ NA PE NC
073F:0113 B303          MOV     BL,03
-T

AX=6805  BX=5603  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0115   NV UP EI PL NZ NA PE NC
073F:0115 00D8          ADD     AL,BL
-T

AX=6808  BX=5603  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0117   NV UP EI PL NZ NA PO NC
073F:0117 0000          ADD     [BX+SI],AL                         DS:5603=82
-
```

- **ADDITION OF TWO 16 - BIT NUMBERS :**

```
-A
073F:0109 MOV AX,1234
073F:010C MOV BX,5678
073F:010F ADD AX,BX
073F:0111
-g 0109

AX=1208  BX=5603  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0109   NV UP EI PL NZ NA PO NC
073F:0109 B83412        MOV     AX,1234
-T

AX=1234  BX=5603  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=010C   NV UP EI PL NZ NA PO NC
073F:010C BB7856        MOV     BX,5678
-T

AX=1234  BX=5678  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=010F   NV UP EI PL NZ NA PO NC
073F:010F 01D8          ADD     AX,BX
-T

AX=68AC  BX=5678  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0111   NV UP EI PL NZ NA PE NC
073F:0111 0000          ADD     [BX+SI],AL                         DS:5678=00
-_
```

- **SUBTRACTION OF TWO 8 - BIT NUMBERS :**

```
-A
073F:011F MOV AL,07
073F:0121 MOV BL,02
073F:0123 SUB AL,BL
073F:0125
-g 011F

AX=0002  BX=0003  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=011F   NV UP EI PL NZ NA PO NC
073F:011F B007          MOV     AL,07
-T

AX=0007  BX=0003  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0121   NV UP EI PL NZ NA PO NC
073F:0121 B302          MOV     BL,02
-T

AX=0007  BX=0002  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0123   NV UP EI PL NZ NA PO NC
073F:0123 28D8          SUB     AL,BL
-T

AX=0005  BX=0002  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0125   NV UP EI PL NZ NA PE NC
073F:0125 0000          ADD     [BX+SI],AL                         DS:0002=CA
-
```

- **SUBTRACTION OF TWO 16 - BIT NUMBERS :**

```
-A
073F:0117 MOV AX,0005
073F:011A MOV BX,0003
073F:011D SUB AX,BX
073F:011F
-g 0117

AX=6808  BX=5603  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0117   NV UP EI PL NZ NA PO NC
073F:0117 B80500        MOV     AX,0005
-T

AX=0005  BX=5603  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=011A   NV UP EI PL NZ NA PO NC
073F:011A BB0300        MOV     BX,0003
-T

AX=0005  BX=0003  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=011D   NV UP EI PL NZ NA PO NC
073F:011D 29D8          SUB     AX,BX
-t

AX=0002  BX=0003  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=011F   NV UP EI PL NZ NA PO NC
073F:011F 07            POP     ES
-
```

- **MULTIPLICATION OF TWO 8 - BIT NUMBERS :**

```
073F:0125 MOV AL,05
073F:0127 MOV BL,03
073F:0129 MUL BL
073F:012B
-g 0125

AX=0005  BX=0002  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0125   NV UP EI PL NZ NA PE NC
073F:0125 B005          MOV     AL,05
-T

AX=0005  BX=0002  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0127   NV UP EI PL NZ NA PE NC
073F:0127 B303          MOV     BL,03
-T

AX=0005  BX=0003  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0129   NV UP EI PL NZ NA PE NC
073F:0129 F6E3          MUL     BL
-T

AX=000F  BX=0003  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=012B   NV UP EI PL NZ NA PE NC
073F:012B 0000          ADD     [BX+SI],AL                         DS:0003=A7
-
```

- **MULTIPLICATION OF TWO 16 - BIT NUMBERS :**

```
073F:012B MOV AX,0030
073F:012E MOV BX,0005
073F:0131 MUL BX
073F:0133
-g 012B

AX=000F  BX=0003  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=012B   NV UP EI PL NZ NA PE NC
073F:012B B83000        MOV     AX,0030
-T

AX=0030  BX=0003  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=012E   NV UP EI PL NZ NA PE NC
073F:012E BB0500        MOV     BX,0005
-T

AX=0030  BX=0005  CX=0000  DX=CD08  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0131   NV UP EI PL NZ NA PE NC
073F:0131 F7E3          MUL     BX
-T

AX=00F0  BX=0005  CX=0000  DX=0000  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0133   NV UP EI PL NZ NA PE NC
073F:0133 0000          ADD     [BX+SI],AL                         DS:0005=EA
-_
```

- **DIVISION OF TWO 8 - BIT NUMBERS :**

```
073F:0133 MOV AL,50
073F:0135 MOV BL,02
073F:0137 DIV BL
073F:0139
-g 0133

AX=00F0  BX=0005  CX=0000  DX=0000  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0133   NV UP EI PL NZ NA PE NC
073F:0133 B050          MOV     AL,50
-T

AX=0050  BX=0005  CX=0000  DX=0000  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0135   NV UP EI PL NZ NA PE NC
073F:0135 B302          MOV     BL,02
-T

AX=0050  BX=0002  CX=0000  DX=0000  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0137   NV UP EI PL NZ NA PE NC
073F:0137 F6F3          DIV     BL
-T

AX=0028  BX=0002  CX=0000  DX=0000  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0139   NV UP EI PL NZ NA PE NC
073F:0139 0000          ADD     [BX+SI],AL                         DS:0002=CA
-_
```

- **DIVISION OF TWO 16 - BIT NUMBERS :**

```
073F:0139 MOV AX,1000
073F:013C MOV BX,0050
073F:013F DIV BX
073F:0141
-g 0139

AX=0028  BX=0002  CX=0000  DX=0000  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0139   NV UP EI PL NZ NA PE NC
073F:0139 B80010        MOV     AX,1000
-T

AX=1000  BX=0002  CX=0000  DX=0000  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=013C   NV UP EI PL NZ NA PE NC
073F:013C BB5000        MOV     BX,0050
-T

AX=1000  BX=0050  CX=0000  DX=0000  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=013F   NV UP EI PL NZ NA PE NC
073F:013F F7F3          DIV     BX
-T

AX=0033  BX=0050  CX=0000  DX=0010  SP=0101  BP=0000  SI=0000  DI=0001
DS=073F  ES=B800  SS=073F  CS=073F  IP=0141   NV UP EI PL NZ NA PE NC
073F:0141 0000          ADD     [BX+SI],AL                         DS:0050=CD
-_
```

Conclusion :

This experiment provided valuable insights into the internal workings of the **8086 microprocessor** and its debugging process using the **DOS DEBUG utility**. By interacting directly with the 8086 architecture, we analyzed and manipulated memory, registers, and instructions in real time, reinforcing concepts of assembly programming and microprocessor execution flow.

The debugging process involved:

1. Launching DEBUG in a DOS environment.
2. Using commands such as **R** to view and modify registers, **A** to assemble instructions, and **T** to trace instruction execution.
3. Observing how changes in registers and memory locations reflected the operation of various instructions, offering practical insights.

Advantages of Using DEBUG:

- **Educational Value**: Helps in learning assembly programming and understanding microprocessor internals.
- **Error Detection**: Enables step-by-step execution to identify and fix code errors.
- **Direct Control**: Allows manipulation of memory and registers, crucial for low-level testing.
- **Interactive Learning**: Encourages experimentation, aiding mastery of key concepts like segmentation and instruction execution.

This experiment offered a hands-on approach to understanding the **8086 architecture** and assembly programming. Using DEBUG, we developed foundational skills critical for comprehending modern computing systems and low-level programming.