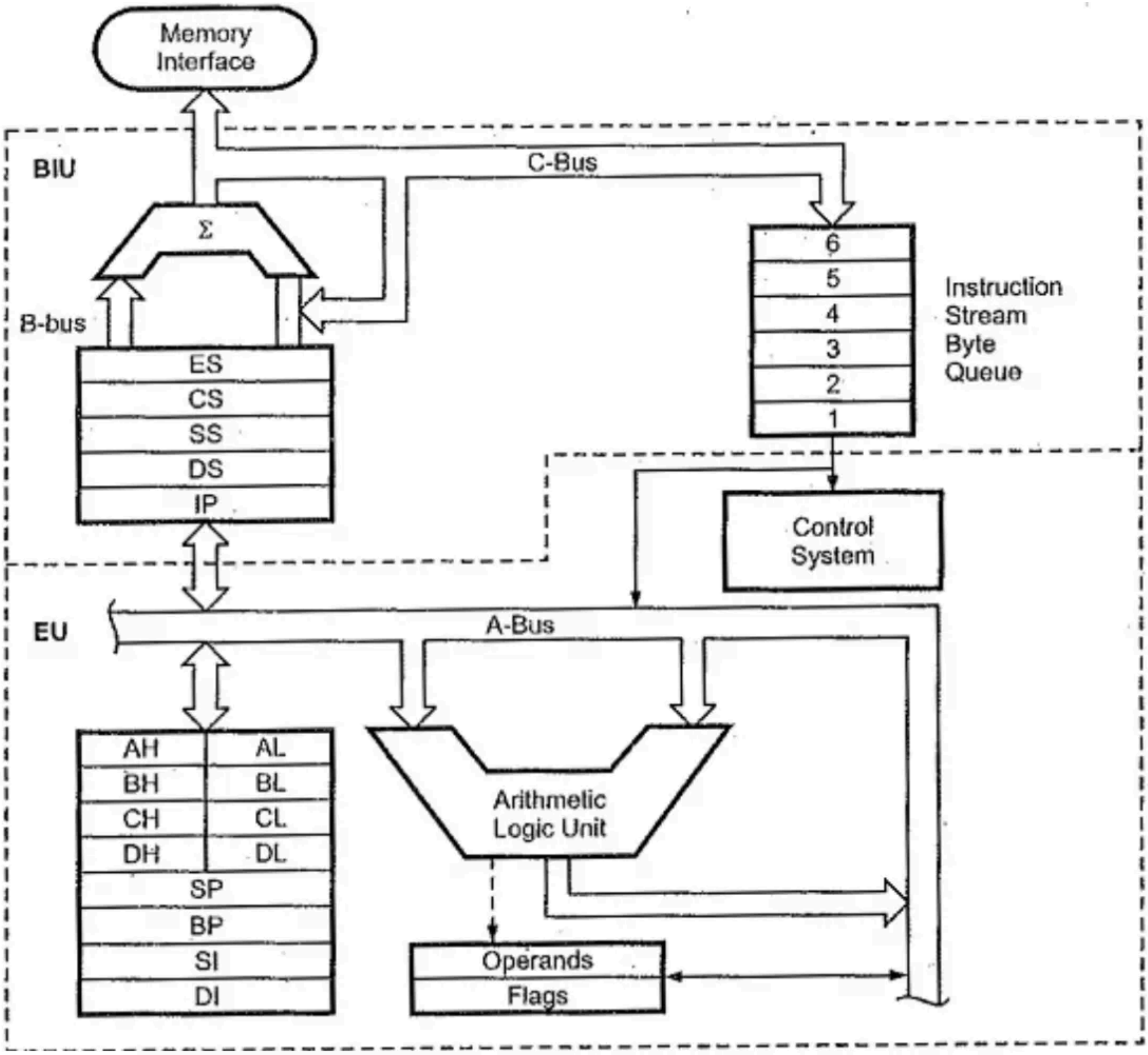
5.1 : Functional Block Diagram of 8086 Microprocessor



Overview of 8086 Architecture

The 8086 microprocessor is divided into two main units:

1. **Bus Interface Unit (BIU)**
2. **Execution Unit (EU)**

1. Bus Interface Unit (BIU)

The **BIU** handles all data and address transfers between the processor and memory/peripherals. It works independently of the EU to improve instruction-fetch efficiency.

Key Components:

1. **Instruction Queue**:
   - Uses a **6-byte prefetch queue** to fetch instructions in advance.
   - Implements pipelining to overlap fetching and execution, speeding up the processor.
   - Reduces memory access delays.
2. **Segment Registers**: The BIU uses **segment registers** to calculate the 20-bit physical address from a 16-bit segment address and a 16-bit offset.
   - **CS (Code Segment)**: Holds the starting address of the code segment.
   - **DS (Data Segment)**: Points to the data segment.
   - **SS (Stack Segment)**: Refers to the stack memory.
   - **ES (Extra Segment)**: Used for string and extra data operations.
3. **Instruction Pointer (IP)**:
   - Holds the offset address of the next instruction to be fetched within the **CS** segment.
4. **Address Generation Logic**:
   - Combines the segment address from a segment register and an offset address to form a 20-bit physical address.
   - Formula: Physical Address=(Segment Address×16)+Offset\text{Physical Address} = (\text{Segment Address} \times 16) + \text{Offset}Physical Address=(Segment Address×16)+Offset
5. **Control Signals**:
   - Generates control signals for data transfer (e.g., memory read/write, I/O read/write).

2. Execution Unit (EU)

The **EU** executes instructions fetched by the BIU. It contains the Arithmetic and Logic Unit (ALU), general-purpose registers, and a control unit.

Key Components:

1. **Arithmetic and Logic Unit (ALU)**:
   - Performs arithmetic operations (addition, subtraction, etc.) and logical operations (AND, OR, NOT, etc.).
2. **General Purpose Registers**: These are used for temporary data storage and processing. The registers are:
   - **AX (Accumulator)**: Primary register for arithmetic/logic operations.
   - **BX (Base Register)**: Used for indexing memory addresses.
   - **CX (Count Register)**: Used for loop and string operations.
   - **DX (Data Register)**: Holds data for multiplication/division and I/O operations.
3. **Pointer and Index Registers**:
   - **SP (Stack Pointer)**: Points to the current top of the stack in the stack segment.
   - **BP (Base Pointer)**: Helps access stack data indirectly.
   - **SI (Source Index)**: Used for source data in string operations.
   - **DI (Destination Index)**: Used for destination data in string operations.
4. **Control Unit (CU)**:
   - Decodes instructions fetched from the queue.
   - Coordinates the flow of data and control signals within the processor.
5. **Flag Register**:
   - Reflects the status of the processor after operations and controls its execution.
   - Flags are divided into:
     - **Status Flags**: Indicate the result of operations (e.g., Carry, Zero, Sign, Overflow).
     - **Control Flags**: Control processor operations (e.g., Direction, Interrupt, Trap).

Interaction Between BIU and EU

- The **BIU** fetches instructions and places them in the instruction queue.
- The **EU** reads the instructions from the queue, decodes them, and executes them.
- The **EU** generates results and may request the BIU for memory or I/O operations during execution.

Advantages of 8086 Architecture

- **Pipelining**: Overlaps instruction fetching with execution for improved speed.
- **Segmentation**: Efficient use of memory and ease of program management.
- **Versatility**: Can operate in both single-processor and multi-processor modes.

5.2 : MOV Instruction formats

The MOV instruction is used to transfer data from one location to another. Its general formats are:

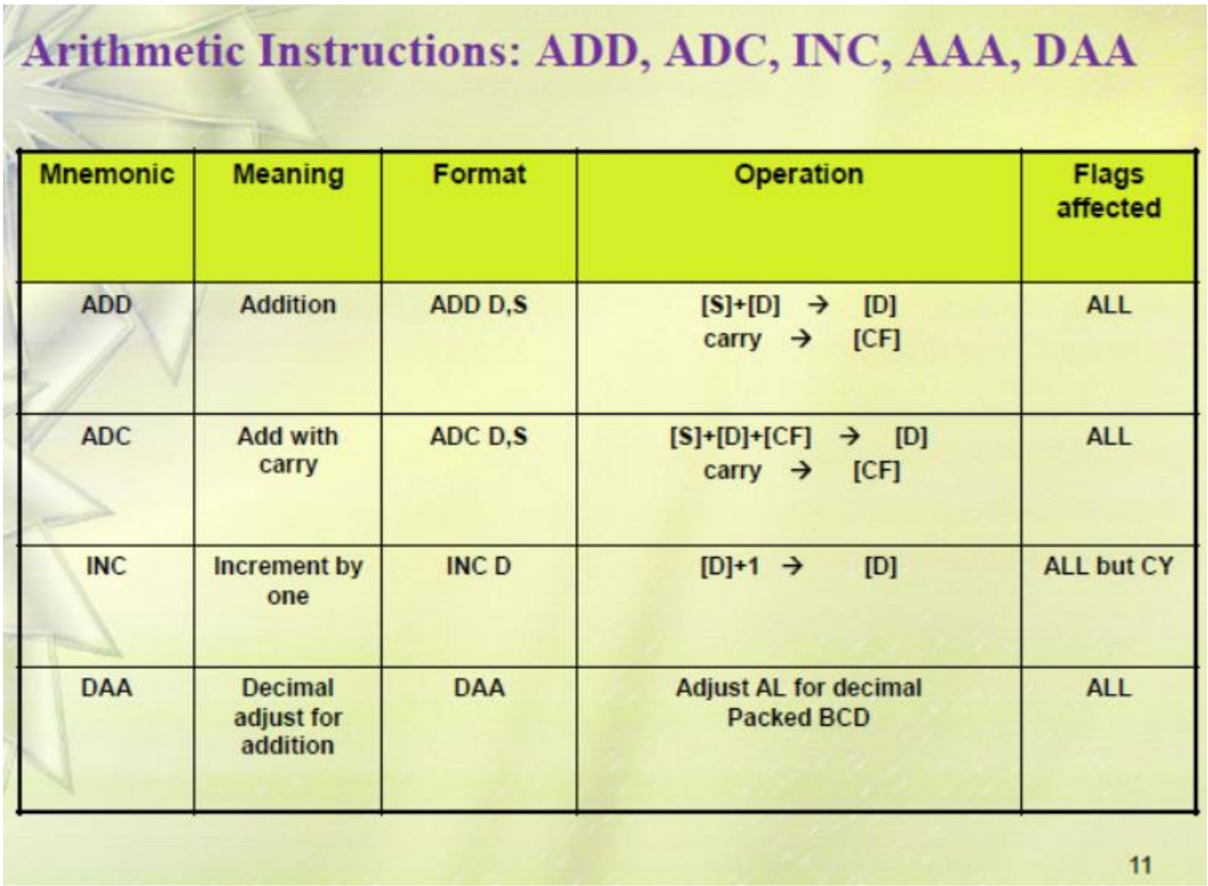
1. **Register to Register**: Transfers data between registers.
    - ○ **Syntax**: MOV destination, source
    - ○ **Example**: MOV AX, BX (copies the content of BX into AX)

2. **Immediate to Register**: Loads a constant value directly into a register.
    - ○ **Syntax**: MOV register, immediate
    - ○ **Example**: MOV AX, 1234H (loads the hexadecimal value 1234 into AX)

3. **Memory to Register**: Loads data from a memory location into a register.
    - ○ **Syntax**: MOV register, [memory address]
    - ○ **Example**: MOV AX, [1234H] (loads the data at address 1234H into AX)

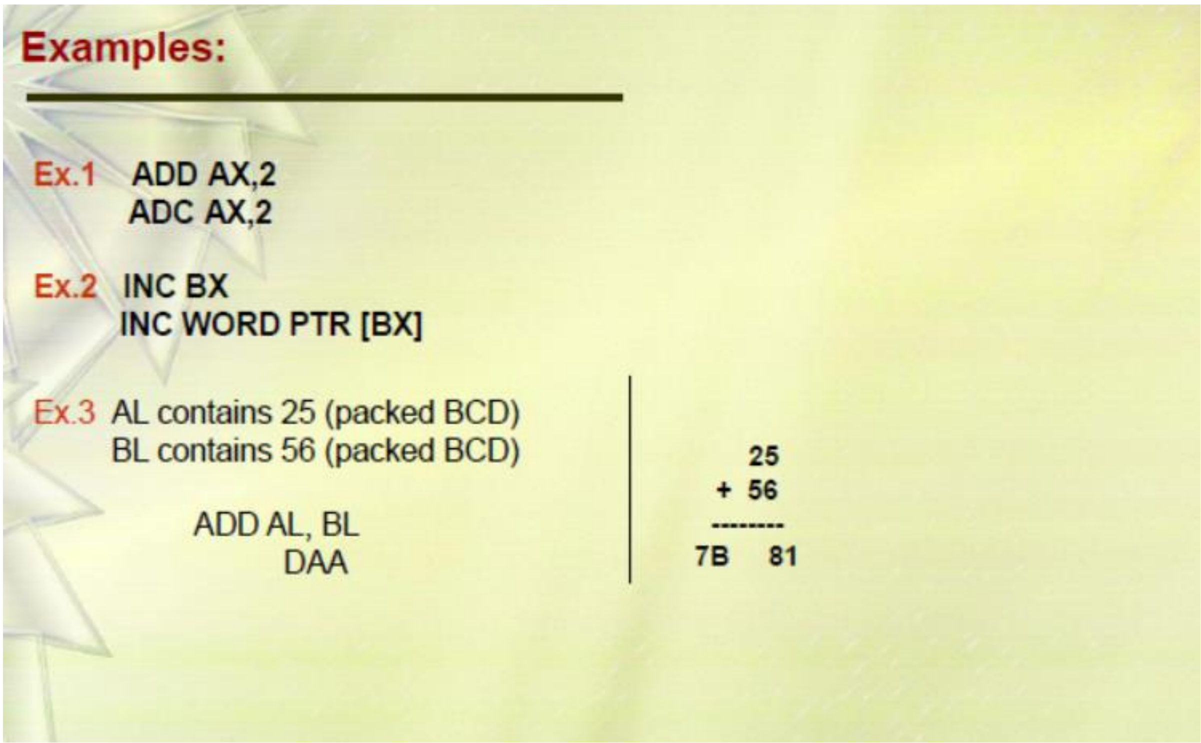
4. **Register to Memory**: Stores data from a register into a memory location.
    - ○ **Syntax**: MOV [memory address], register
    - ○ **Example**: MOV [1234H], AX (stores the content of AX at address 1234H)

5. **Immediate to Memory**: Loads a constant value directly into a memory location.
    - ○ **Syntax**: MOV [memory address], immediate
    - ○ **Example**: MOV [1234H], 5678H (stores 5678H at address 1234H)



Arithmetic Instructions: ADD, ADC, INC, AAA, DAA

Mnemonic	Meaning	Format	Operation	Flags affected
ADD	Addition	ADD D,S	[S]+[D] → [D] carry → [CF]	ALL
ADC	Add with carry	ADC D,S	[S]+[D]+[CF] → [D] carry → [CF]	ALL
INC	Increment by one	INC D	[D]+1 → [D]	ALL but CY
DAA	Decimal adjust for addition	DAA	Adjust AL for decimal Packed BCD	ALL



Examples:

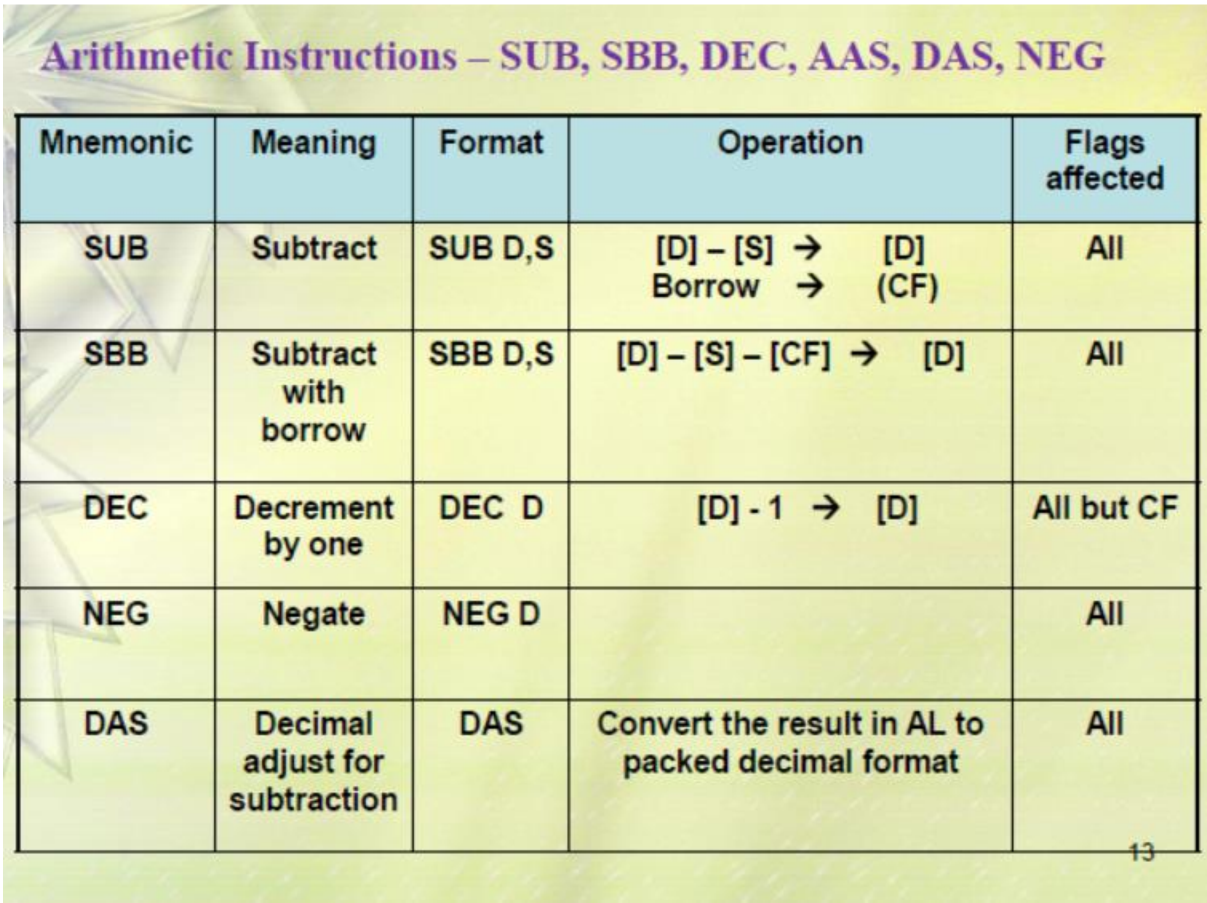
Ex.1 **ADD AX,2**
ADC AX,2

Ex.2 **INC BX**
INC WORD PTR [BX]

Ex.3 AL contains 25 (packed BCD)
BL contains 56 (packed BCD)

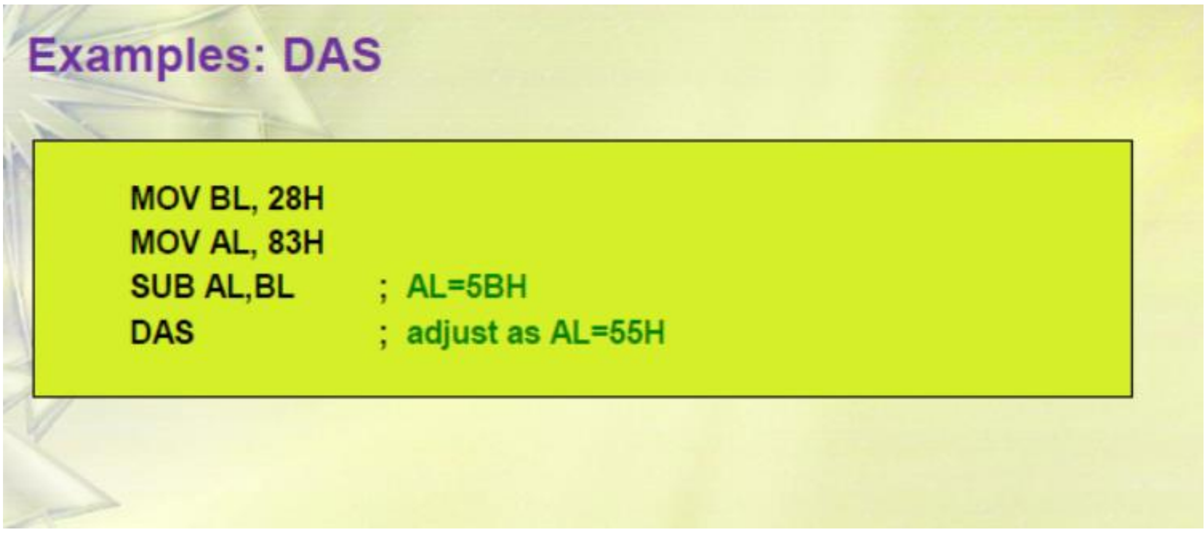
ADD AL, BL
DAA

```
    25
  + 56
 --------
 7B   81
```



Arithmetic Instructions – SUB, SBB, DEC, AAS, DAS, NEG

Mnemonic	Meaning	Format	Operation	Flags affected
SUB	Subtract	SUB D,S	[D] – [S] → [D] Borrow → (CF)	All
SBB	Subtract with borrow	SBB D,S	[D] – [S] – [CF] → [D]	All
DEC	Decrement by one	DEC D	[D] - 1 → [D]	All but CF
NEG	Negate	NEG D		All
DAS	Decimal adjust for subtraction	DAS	Convert the result in AL to packed decimal format	All



Examples: DAS

```
MOV BL, 28H
MOV AL, 83H
SUB AL,BL        ;  AL=5BH
DAS              ;  adjust as AL=55H
```

1) Write an assembly language program to perform the ADDitionof two 8 bit numbers using 8086.

Solution:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
NUM1 DB 12
NUM2 DB 18
SUM DB ?
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
MOV DS,AX
MOV AL, NUM1
ADD AL, NUM2
MOV SUM, AL
INT 03H
CODE ENDS
END START
END.
```

- **Data Segment Declaration**:
  - `DATA SEGMENT`: Defines the beginning of a data segment.
  - `NUM1 DB 12`: Reserves a byte of memory and initializes it with the decimal value `12`.
  - `NUM2 DB 18`: Reserves a byte of memory and initializes it with the decimal value `18`.
  - `SUM DB ?`: Reserves a byte of memory for `SUM` without initializing it (it's meant to store the result of `NUM1 + NUM2`).
  - `DATA ENDS`: Marks the end of the data segment.
- **Code Segment Declaration**:
  - `CODE SEGMENT`: Defines the beginning of a code segment.
- **Execution Start**:
  - `START:` This label is used as the entry point of the program.
  - `MOV AX, DATA`: Loads the address of the `DATA` segment into the `AX` register.
  - `MOV DS, AX`: Moves the address in `AX` to `DS`, setting up the data segment.
- **Addition Operation**:
  - `MOV AL, NUM1`: Loads the value of `NUM1` into the `AL` register.
  - `ADD AL, NUM2`: Adds the value of `NUM2` to the value in `AL`, storing the result in `AL`.
  - `MOV SUM, AL`: Moves the result from `AL` to `SUM`.

- **Interrupt and Program End**:
    - `INT 03H`: Triggers a software interrupt to terminate the program (used here as a breakpoint).
    - `CODE ENDS`: Marks the end of the code segment.
    - `END START`: Specifies `START` as the entry point of the program and marks the end of the assembly source file.

2) Write an ALP for ADDitionof two 16 bit numbers using 8086

Solution:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
NUM1 DW 1234
NUM2 DW 4567
SUM DW ?
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
MOV DS,AX
MOV AX, NUM1
ADD AX, NUM2
MOV SUM, AX
INT 03H
CODE ENDS
END START
END.
```

`NUM1 DW 1234`: Reserves a word (16 bits) of memory for `NUM1`

3) Write an assembly language program to perform the subtract operation of two 8 bit numbers using 8086.

Solution:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
NUM1 DB 18
NUM2 DB 12
```

```
DIFF DB ?
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
MOV DS,AX
MOV AL, NUM1
SUB AL, NUM2
MOV DIFF, AL
INT 03H
CODE ENDS
END START
END.
```

4) Write an assembly language program to perform the subtract operation of two 16 bit numbers using 8086.

Solution:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
NUM1 DW 1835
NUM2 DW 1735
DIFF DW ?
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
MOV DS,AX
MOV AX, NUM1
SUB AX, NUM2
MOV DIFF, AX
INT 03H
CODE ENDS
END START
END.
```