Program to Find LCM and GCD of 48 and 18 (8051 Assembly)

Mathematical Explanation:

GCD (Greatest Common Divisor):

- **Definition**: GCD of two numbers is the largest positive integer that divides both numbers exactly without leaving a remainder.
- **Method Used**: Euclidean Algorithm

Steps to Find GCD of 48 and 18:

1. Divide 48 by 18 → quotient = 2, remainder = 12
2. Divide 18 by 12 → quotient = 1, remainder = 6
3. Divide 12 by 6 → quotient = 2, remainder = 0
4. Since the remainder is 0, the divisor at this step (6) is the GCD.

Result: GCD(48, 18) = 6

Program Listing for GCD:

```
ORG 0000H

MOV 50H, #48      ; First number (initialize)

MOV 51H, #18      ; Second number (initialize)

MOV A, 50H

MOV B, 51H

LOOP:

   MOV R1, B

   DIV AB         ; A/B ? quotient in A, remainder in B

   MOV A, B

   JZ END_GCD     ; If remainder = 0, GCD is in R1

   MOV B, A

   MOV A, R1

   SJMP LOOP

END_GCD:

   MOV A, R1

   MOV 52H, A     ; Store GCD in 52H
```

SJMP $; Stay here forever

Step-by-Step Explanation for GCD:

END1. ORG 0000H

- This sets the **origin (starting address)** of the program to **0000H**, i.e., the program starts execution from memory address 0000H.

---

2. MOV 50H, #48

- Store **first number 48 (decimal)** into memory location 50H.
- MOV 50H, #48 means direct memory address 50H is loaded with value **48**.

---

3. MOV 51H, #18

- Store **second number 18 (decimal)** into memory location 51H.

---

4. MOV A, 50H

- Load **A register** with the **first number** from 50H.
- Now, A = 48.

---

5. MOV B, 51H

- Load **B register** with the **second number** from 51H.
- Now, B = 18.

Start of GCD Loop

6. LOOP: MOV R1, B

- Copy the current value of **B register** to **R1 register**.
- This is done to preserve the old value of B before division.
- Now, R1 = B = 18.

---

7. DIV AB

- Perform **A ÷ B**.
- Result:
  - **Quotient** goes into A.

- **Remainder** goes into B.

- For first iteration:
  - 48 ÷ 18 = 2 quotient, **12 remainder**.
  - After DIV:
    - A = 2 (quotient), B = 12 (remainder).

---

8. MOV A, B

- Move the remainder (now in B) into A.
- Now, A = 12.

---

9. JZ END_GCD

- **Jump if Zero (JZ)**: Check if A (which now has remainder) is 0.
  - If A = 0, jump to label **END_GCD** because it means the GCD is found.
  - Else, continue.
- In first iteration, A = 12 ≠ 0, so jump is NOT taken.

---

10. MOV B, A

- Move current value of A (12) to B.
- Now, B = 12.

---

11. MOV A, R1

- Move the previously saved value in R1 (old B = 18) back into A.
- Now, A = 18.

---

12. SJMP LOOP

- Short jump back to label **LOOP** to repeat the division with new A and B.

Second Iteration

- Now: A = 18, B = 12.
- R1 = 12.

- DIV AB → 18 ÷ 12 = 1 quotient, **6 remainder**.
    - A = 1, B = 6.
- MOV A, B → A = 6.
- JZ END_GCD → Not zero, continue.
- MOV B, A → B = 6.
- MOV A, R1 → A = 12.
- SJMP LOOP.

Third Iteration

- A = 12, B = 6.
- R1 = 6.
- DIV AB → 12 ÷ 6 = 2 quotient, **0 remainder**.
    - A = 2, B = 0.
- MOV A, B → A = 0.
- JZ END_GCD → A = 0 → Jump to END_GCD.

---

13. END_GCD: MOV A, R1

- Move R1 (which has **last non-zero B = 6**) into A.
- A = 6 → GCD found.

---

14. MOV 52H, A

- Store GCD value (6) from A into memory location **52H**.

---

15. SJMP $

- This is an **infinite loop**, it means the program **halts here forever**.
- $ means the current address, so it keeps jumping to itself.

Final Output:

- **GCD of 48 and 18 is 6** → stored at **memory location 52H**.

How to Find LCM of Two Numbers (Example: 48 and 18)

Step 1: Prime Factorization of Each Number

Break each number into its **prime factors**.

- **$48 = 2 \times 2 \times 2 \times 2 \times 3 = 2^4 \times 3^1$**
- **$18 = 2 \times 3 \times 3 = 2^1 \times 3^2$**

Step 2: Choose the Highest Power of Each Prime Number

From both numbers, **pick the biggest exponent (power)** of each prime factor.

Prime Number	Highest Power
2	2^4
3	3^2

Step 3: Multiply the Highest Powers Together

Now multiply them:

$LCM = 2^4 \times 3^2 = 16 \times 9 = 144$

Final Answer: LCM(48, 18) = 144

Program Listing for LCM:

```
ORG 0000H

MOV 50H, #48       ; num1
MOV 51H, #18       ; num2
MOV 52H, #6        ; GCD pre-calculated

MOV A, 50H
MOV B, 52H
DIV AB             ; A = 48 / 6 = 8

MOV B, 51H
MUL AB             ; A = 8 × 18 = 144 = 90H

MOV 53H, A         ; Store LCM = 144 decimal (90H)

SJMP $
END
```

Step-by-Step Explanation for LCM:

1. ORG 0000H

- Sets the **program start address** at memory location **0000H**.
- Instructs the assembler to begin code placement from this location.

2. MOV 50H, #48

- Stores the **first number (48)** into memory location **50H**.
- #48 represents an immediate value of 48 in decimal.

3. MOV 51H, #18

- Stores the **second number (18)** into memory location **51H**.

4. MOV 52H, #6

- Stores the **GCD value (6)** into memory location **52H**.
- This GCD is pre-calculated.

5. MOV A, 50H

- Loads **num1 (48)** from memory into the accumulator **A**.

6. MOV B, 52H

- Loads **GCD (6)** into register **B**.

7. DIV AB

- Performs **A ÷ B**: 48 ÷ 6 = 8.
- After division:
  - A = 8 (quotient)
  - B = 0 (remainder, ignored)

8. MOV B, 51H

- Loads **num2 (18)** into register **B**.

9. MUL AB

- Performs **A × B**: 8 × 18 = 144.
- After multiplication:
  - A = 144 (result), which is **90H** in hexadecimal.
  - B = 0 (ignored)

10. MOV 53H, A

- Stores the **LCM (144)** into memory location **53H**.
- Final result: 53H = 90H (hex) = 144 decimal.

11. SJMP $

- Infinite loop: jumps to the same instruction to halt program safely.

12. END

- Marks the **end of the program** for the assembler.

Final Output for LCM:

Address	Value	Meaning
53H	90H	144 decimal (LCM)

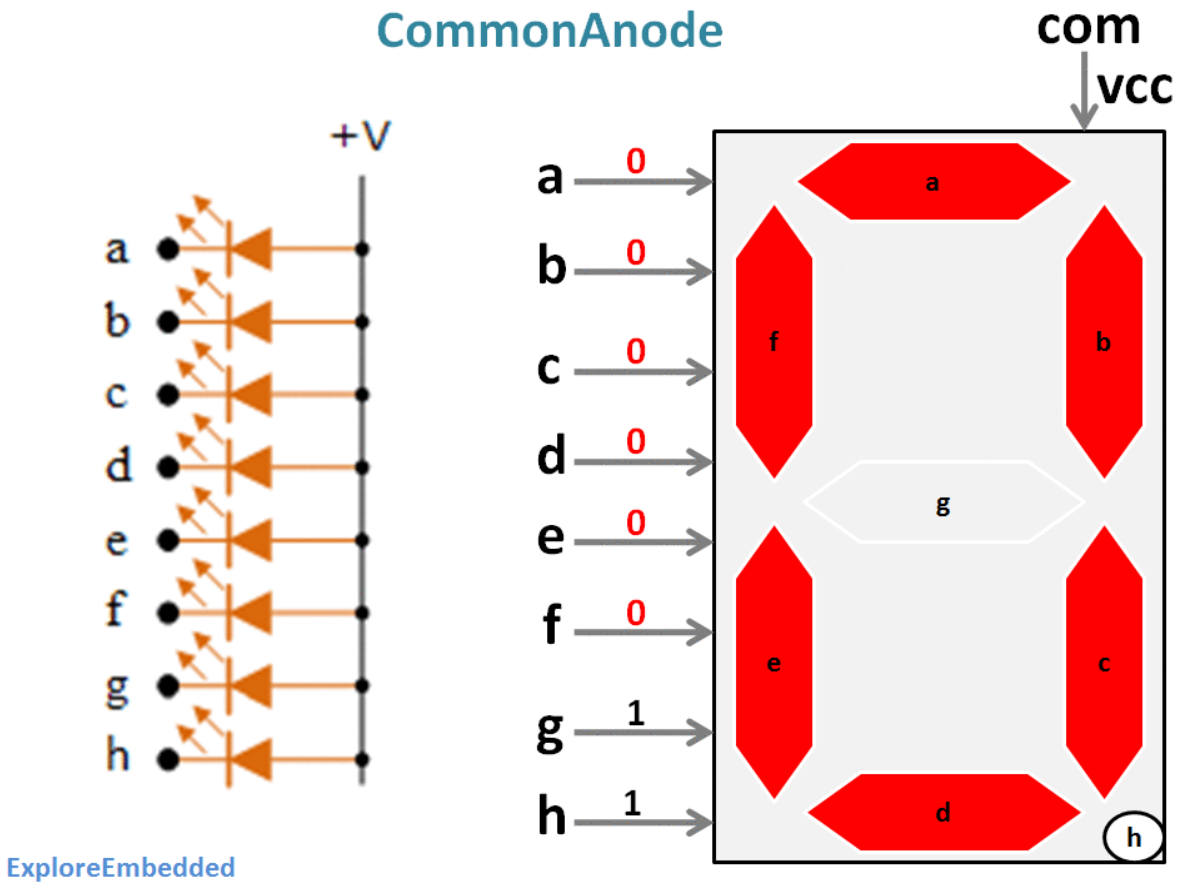
LCM of **48 and 18** is **144** (stored as 90H in memory 53H).

3. BCD to Seven Segment Display Program (for 8051)

Step-by-step Explanation for Students

1 What is a 7-Segment Display?

A **7-segment display** is an electronic display device used to display **decimal numbers (0–9)**.



- It has **7 LEDs**, labeled **a, b, c, d, e, f, g** arranged in a figure 8.
- Each segment can be turned ON/OFF to display digits.
- Common types:

- **Common Cathode**: Ground is common; HIGH signal lights up segment.
- **Common Anode**: VCC is common; LOW signal lights up segment.

◆ Example: To display **digit 3**, segments **a, b, c, d, g** are ON.

---

2 What is BCD?

- **BCD (Binary-Coded Decimal)** represents a decimal digit (0–9) using **4 bits**.
  - Example: Decimal **5 = 0101** in BCD.

---

3 How do we convert BCD to Seven Segment?

We need to convert BCD digit (like 0101) to its **7-segment code**.

Example:

Digit	Segments ON	7-Segment Code (Binary)	Hex Code
0	a b c d e f	0011 1111	3FH
1	b c	0000 0110	06H
2	a b d e g	0101 1011	5BH
3	a b c d g	0100 1111	4FH
4	b c f g	0110 0110	66H
5	a c d f g	0110 1101	6DH
6	a c d e f g	0111 1101	7DH
7	a b c	0000 0111	07H
8	a b c d e f g	0111 1111	7FH
9	a b c d f g	0110 1111	6FH

These **7-segment codes** are stored in a **Look-Up Table (LUT)**.

---

4 Simple 8051 Assembly Program

Goal: Convert BCD number stored in memory to 7-segment display value.

💻 Program: (Explain step by step)

```
ORG 0000H
; Step 1: Initialize Look-Up Table
LUT: DB 3FH, 06H, 5BH, 4FH, 66H, 6DH, 7DH, 07H, 7FH, 6FH
; Step 2: Load BCD number from memory
MOV DPTR, #LUT     ; Point to LUT table
MOV A, 50H         ; Load BCD digit from memory address 50H
; Step 3: Get 7-segment code from LUT
MOVC A, @A+DPTR    ; Move code from LUT to A
; Step 4: Store result in memory or port
MOV 60H, A         ; Store 7-segment code in memory 60H
END
```

5️⃣ Explanation of Steps

Step	What It Does
LUT Table	Stores segment codes for 0–9
MOV DPTR	Points to start of LUT
MOV A, 50H	Gets BCD digit from memory
MOVC A,@A+DPTR	Looks up 7-segment code for BCD digit
MOV 60H, A	Saves 7-segment result to memory
DB	DB = Define Byte • DB is a directive in Assembly used to store data (usually 8-bit values) in memory. • It tells the assembler to allocate memory and initialize it with a byte value.

6 Example

If memory **50H = 05H (BCD 5)**:

- Program finds **LUT[5] = 6DH**
- Result **6DH** = segment pattern for digit **5**
- Output **6DH** is stored at **60H**.

7 Highlight Key Concepts for Students

- Use **Look-Up Table** to map BCD → segment code.
- Use **MOVC A, @A+DPTR** to fetch data from table.
- Understand **memory use**: input at 50H, output at 60H.
- Display result on **7-segment display connected to Port** (e.g., P1).

8 Lab Extension Idea

- Connect **7-segment display to Port1**.
- After MOV 60H, A, add: MOV P1, A to show the digit.