Experiment 9

Code:

```
DATA SEGMENT

Q DB 7H

M DB 3H

RESULT DW ?

DB 100 DUP(00H)

DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START:MOV AX, DATA

MOV DS, AX

MOV BX, 0000H

MOV AX, 0000H

MOV CX, 0008H ; Number of Bits

MOV AL, Q ; Load Dividend in AL

BACK: SHL AX, 1 ; Shift A-Q towards left

SUB AH, M

MOV BH, AH ; Make copy of A in BH

SHL BH, 1 ; Move MSB of A in carry bit

JC NEXT ; Jump if A is less than 0

OR AL, 01H ; Make Q0 bit 1

JMP COUNT ; Jump for Count = Count - 1

NEXT: AND AL, 0FEH ; Make Q0 bit 0

ADD AH, M ; Restore A (A = A + M)

COUNT: LOOP BACK ; Decrement counter

MOV RESULT, AX ; Store result into memory

INT 3h

CODE ENDS
```

END START

Explanation:

DATA SEGMENT

The **data segment** contains the data variables that the program will work with.

1. Q DB 7H
   - Q is a **byte** (8 bits) initialized with the value 7H (which is 7 in hexadecimal, or 7 in decimal). This is the **dividend** for the division operation.
2. M DB 3H
   - M is another **byte** initialized with the value 3H (3 in hexadecimal, or 3 in decimal). This is the **divisor** for the division operation.
3. RESULT DW ?
   - RESULT is defined as a **word** (16 bits), and its value is uninitialized (? means no value assigned yet). This will hold the result of the division.
4. DB 100 DUP(00H)
   - This line defines 100 bytes of memory, all initialized to 00H (0 in hexadecimal). These bytes are unused in the program but are reserved for future use or padding.
5. DATA ENDS
   - Marks the end of the **data segment**. All variables are now defined.

CODE SEGMENT

The **code segment** contains the instructions that define the program's logic.

6. ASSUME CS: CODE, DS: DATA
   - Tells the assembler that the **code segment** (CS) is named CODE and the **data segment** (DS) is named DATA. This helps the assembler know where to find the program's code and where to find the data.
7. START: MOV AX, DATA
   - The label START marks the beginning of the program. The instruction moves the address of the **data segment** (DATA) into the AX register. This is required to set up the **data segment** register.
8. MOV DS, AX

- Moves the value in AX (which now holds the address of the data segment) into the **data segment register** (DS). This prepares the program to access the data segment.

9. MOV BX, 0000H
   - Initializes the BX register to 0000H (0 in hexadecimal). This register will be used later, but it is initially set to zero.
10. MOV AX, 0000H
    - Initializes the AX register to 0000H (0 in hexadecimal). This register will be used to hold intermediate results and calculations.
11. MOV CX, 0008H ; Number of Bits
    - Initializes the CX register to 0008H (8 in hexadecimal), indicating that the program will perform 8 iterations (corresponding to the 8 bits of the byte Q).
12. MOV AL, Q ; Load Dividend in AL
    - Loads the **dividend** (the value of Q, which is 7H or 7 in decimal) into the **lower byte of AX** (AL). Now, AL contains the value of Q.

BACK LOOP:

This section defines the **main loop** of the algorithm, which will run for 8 iterations (as specified in CX).

13. BACK: SHL AX, 1 ; Shift A-Q towards left
    - The label BACK marks the start of the loop. The SHL AX, 1 instruction shifts the contents of the AX register **left** by one bit. This effectively multiplies AX by 2, which is part of the division process (shifting is part of the long division operation).
14. SUB AH, M
    - Subtracts the value of M (the divisor, which is 3H or 3 in decimal) from the high byte of AX (AH). This is part of the long division step, where the divisor is subtracted from the quotient during each iteration.
15. MOV BH, AH ; Make copy of A in BH
    - Copies the value of AH (the high byte of AX) into BH. This is done because we will need to modify the value in AH, and we want to preserve its original value in BH.
16. SHL BH, 1 ; Move MSB of A in carry bit

   - Shifts BH (which now holds the original value of AH) left by one bit. The **most significant bit (MSB)** of AH is moved into the **carry flag**. This operation helps in detecting whether the result of the subtraction was negative, which determines the next step.

17. JC NEXT ; Jump if A is less than 0
   - The JC instruction jumps to the NEXT label if the **carry flag** is set. The carry flag is set if the **MSB** of AH (stored in BH) was moved out during the SHL BH, 1 operation. This indicates that A is less than 0 (i.e., the subtraction result was negative). If A is less than 0, we need to adjust the result.
18. OR AL, 01H ; Make Q0 bit 1
   - If the carry flag was not set (i.e., the subtraction was not negative), we set the **least significant bit (Q0)** of AL to 1. This represents the fact that the division at this step produced a 1 in the quotient.
19. JMP COUNT ; Jump for Count = Count - 1
   - Jumps to the COUNT label to continue the process of decreasing the bit count (CX) and repeating the process.

NEXT:

This section is executed if A is less than 0 after the subtraction (i.e., the subtraction resulted in a negative value).

20. NEXT: AND AL, 0FEH ; Make Q0 bit 0
   - If the subtraction was negative, this clears the **least significant bit (Q0)** of AL by performing a bitwise **AND** with 0FEH (which is 11111110 in binary). This effectively sets Q0 to 0, indicating that no 1 was obtained in the quotient at this step.
21. ADD AH, M ; Restore A (A = A + M)
   - Since A was negative after the subtraction, we **restore** the value of A by adding M back to AH. This corrects the division process.

COUNT:

This section handles the loop control and decrements the bit count.

22. COUNT: LOOP BACK ; Decrement counter

- The LOOP instruction decrements the value in CX (which was initialized to 8 in line 11). If CX is not zero, the program jumps back to the BACK label to continue the loop. This loop will repeat 8 times (for 8 bits).

FINAL STEPS:

23. MOV RESULT, AX ; Store result into memory
    - After all the iterations, the result (which is stored in AX) is moved into RESULT to save the final quotient.
24. INT 3h
    - The INT 3h instruction triggers a **breakpoint interrupt**, which is commonly used for debugging. It stops the program execution.
25. CODE ENDS
    - Marks the end of the **code segment**.
26. END START
    - Marks the end of the program and indicates that execution should start at the START label.