**Practical – 12**

**Aim:** Write an assembly program to perform ASCII adjust of result of addition, subtraction, multiplication and division. Also analyze the result before and after the adjustment.

**Description of instructions used**

1. **AAA** − ASCII adjust after addition

* If the least significant four bits in AL are > 9 or if AF =1, it adds 6 to AL and 1 to AH. – Both CF and AF are set.
* In all cases, the most significant four bits in AL are cleared

1. **AAS** − ASCII adjust after subtraction

* If the least significant four bits in AL are > 9 or if AF =1, it subtracts 6 from AL and 1 from AH. – Both CF and AF are set
* In all cases, the most significant four bits in AL are cleared
* This adjustment is needed only if the result is negative

1. **AAM**− ASCII adjust after multiplication

* The AAM instruction adjusts the result of a MUL instruction.

∗ Multiplication should not be performed on ASCII

» Can be done on unpacked BCD.

* The AAM instruction works as follows

∗ AL is divided by 10

∗ Quotient is stored in AH

∗ Remainder in AL

* AAM does not work with IMUL instruction

1. **AAD** − ASCII adjust before division

* The ADD instruction adjusts the numerator in AX before dividing two unpacked decimal numbers

∗ The denominator is a single unpacked byte

* The ADD instruction works as follows

∗ Multiplies AH by 10 and adds it to AL and sets AH to 0

∗ Example:

» If AX is 0207H before AAD

» AX is changed to 001BH after AAD

* AAD instruction reverses the changes done by AAM
* ADD converts the unpacked BCD number in AX to binary form so that div can be used

**Code:**

1. **AAA:**

SUB AH, AH ; clear AH

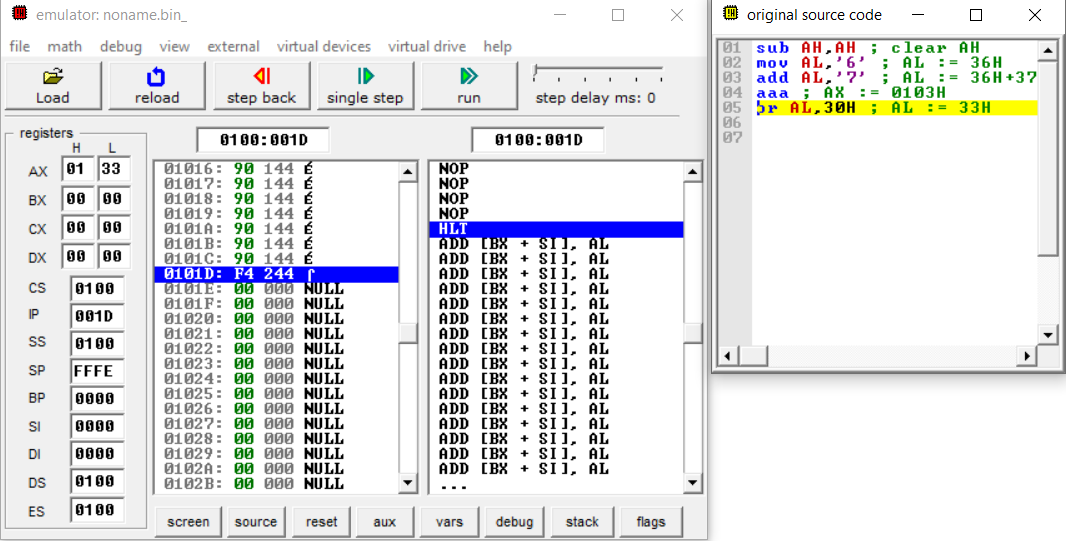
MOV AL,'6' ; AL: 36H

ADD AL,'7' ; AL: 36H+37H = 6DH

AAA ; AX: 0103H

OR AL,30H ; AL: 33H

**Output:**



1. **AAS:**
   1. **POSITIVE RESULT:**

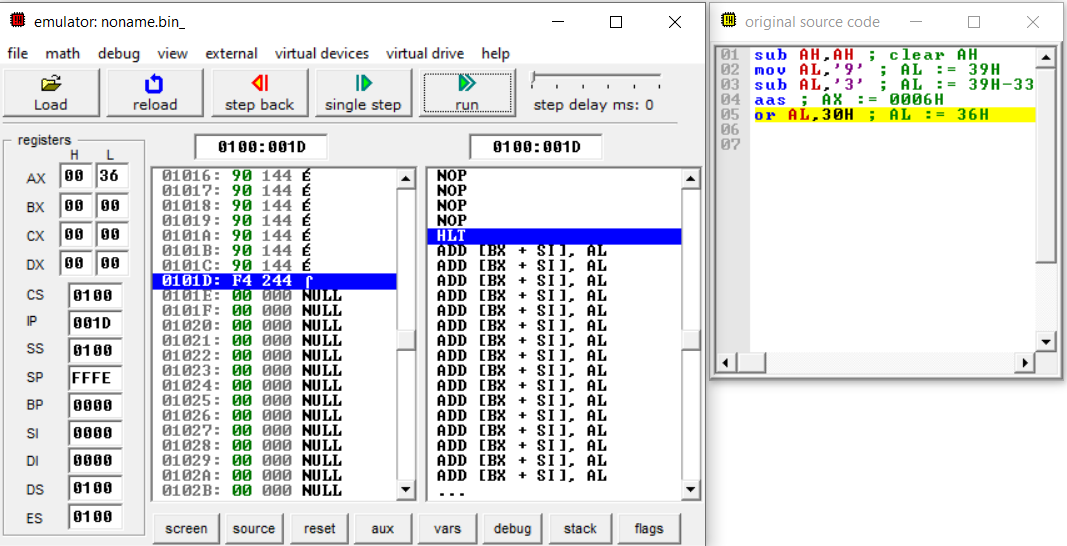
SUB AH, AH ; clear AH

MOV AL,'9' ; AL:39H

SUB AL,'3' ; AL: 39H-33H = 6H

AAS ; AX:0006H

OR AL,30H ; AL:36H

**Output:** 

* 1. **NEGATIVE RESULT:**

SUB AH, AH ; clear AH

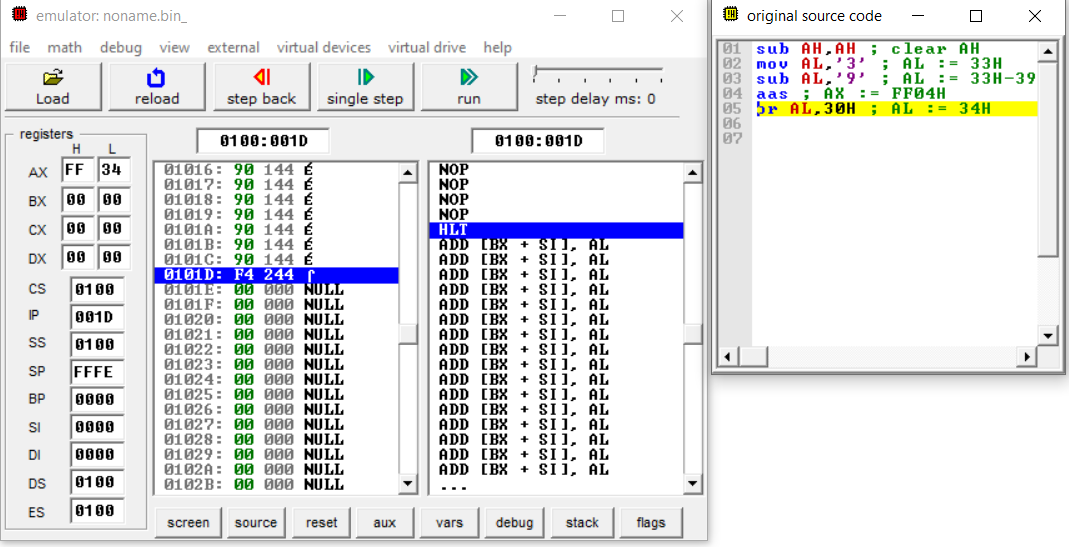
MOV AL,'3' ; AL: 33H

SUB AL,'9' ; AL: 33H-39H = FAH

AAS ; AX: FF04H

OR AL,30H ; AL: 34H

**Output:**



1. **AAM:**
   1. **Example 1:**

MOV AL,3 ; multiplier in unpacked BCD form

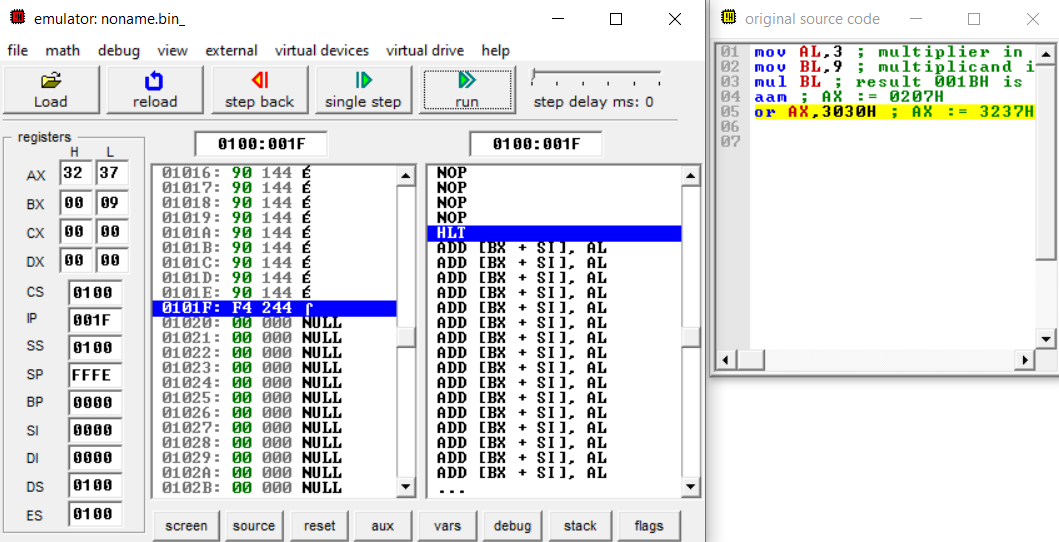
MOV BL,9 ; multiplicand in unpacked BCD form

MUL BL ; result 001BH is in AX

AAM ; AX: 0207H

OR AX,3030H ; AX: 3237H

**Output:**



* 1. **Example 2:**

MOV AL,'3' ; multiplier in ASCII

MOV BL,'9' ; multiplicand in ASCII

AND AL,0FH ; multiplier in unpacked BCD form

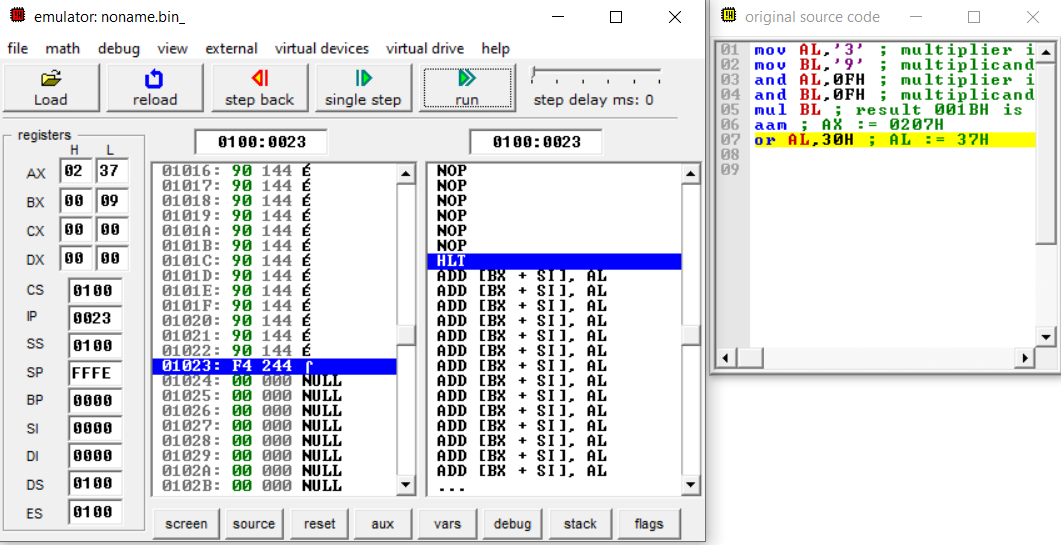
AND BL,0FH ; multiplicand in unpacked BCD form

MUL BL ; result 001BH is in AX

AAM ; AX: 0207H

OR AL,30H ; AL: 37H

**Output:**



1. **AAD:**

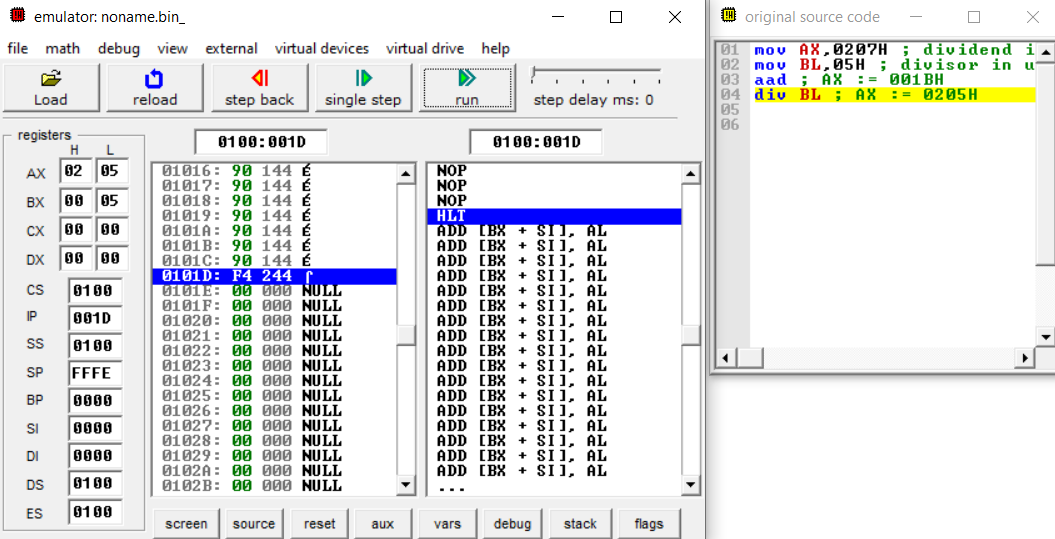
MOV AX,0207H ; dividend in unpacked BCD form

MOV BL,05H ; divisor in unpacked BCD form

AAD ; AX: 001BH

DIV BL ; AX: 0205H

**Output:**



**Practical - 13**

**Aim:** Write an assembly program to perform decimal adjust of result of addition and subtraction. Also analyze the result before and after the adjustment.

**Description of instructions used:**

1. **DAA** − Decimal adjust after addition
   * Used after add or ADC instruction
   * If the least significant four bits in AL are > 9 or if AF =1, it adds 6 to AL and sets AF
   * If the most significant four bits in AL are > 9 or if CF =1, it adds 60H to AL and sets CF
2. **DAS** − Decimal adjust after subtraction
   * Used after sub or SBB instruction
   * If the least significant four bits in AL are > 9 or if AF =1, it subtracts 6 from AL and sets AF
   * If the most significant four bits in AL are > 9 or if CF =1, it subtracts 60H from AL and sets CF

**Code:**

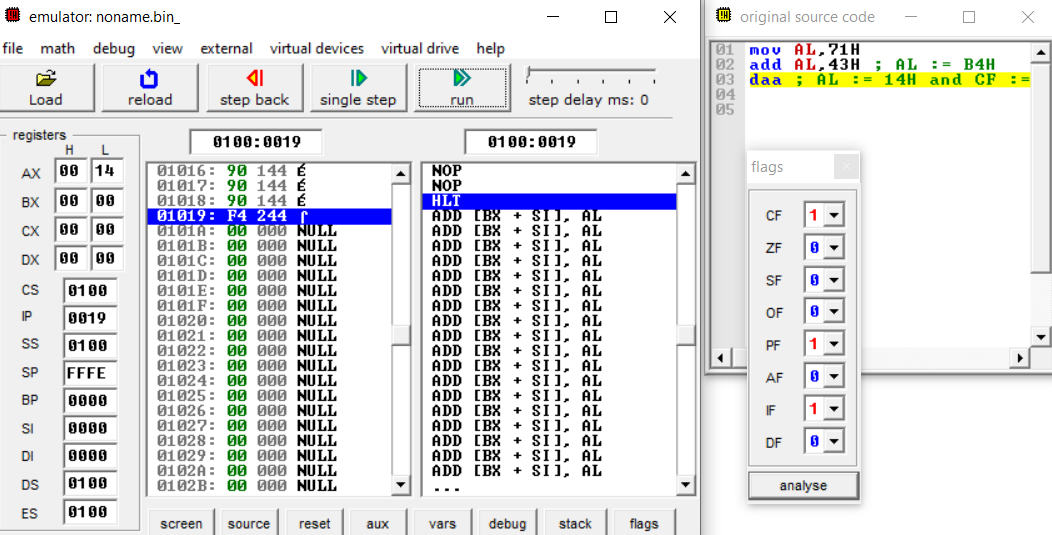
1. **DAA:**

AL,71H

ADD AL,43H ; AL: B4H

DAA ; AL: 14H and CF: 1

**Output:**



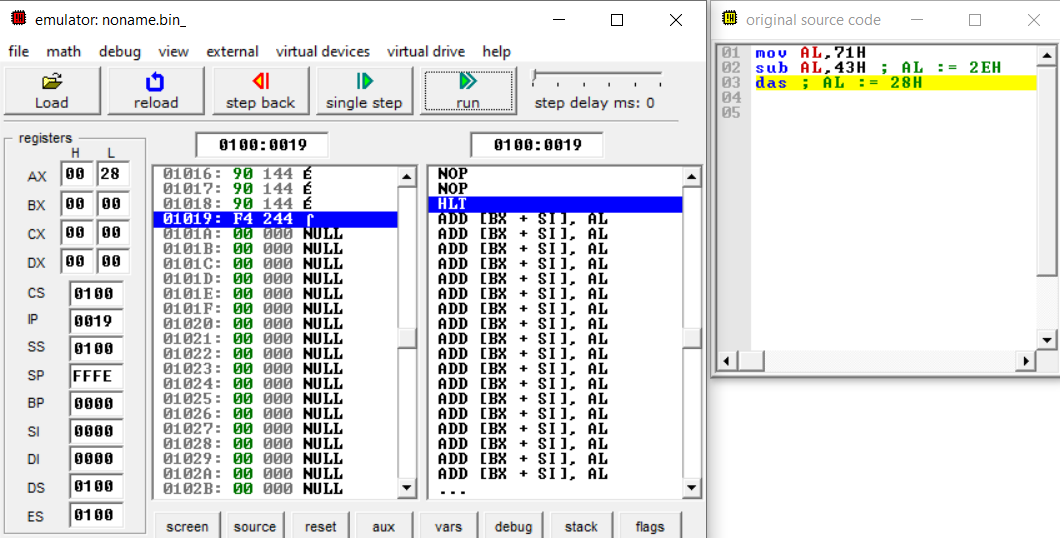
1. **DAS:**

MOV AL,71H

SUB AL,43H ; AL: 2EH

DAS ; AL: 28H

**Output:**



=======================================================================================

**PRACTICAL – 14**

===============================================================================================

**Aim:** Write an assembly program to check given number is even or odd. Also print appropriate message on console.

**Description of instructions used:**

**MSG:**Here MSG stands for message here in place of MSG we will also use STRING.

**DB** **(DEFINE BYTE):**The **DB** directive is used to declare a byte type variable, or a set aside one or more storage locations of type byte in memory.

**MARCO:**A **Macro** is a set of instructions grouped under a single unit. It is another method for implementing modular programming in the **8086** microprocessors (The first one was using Procedures)

**ASSUME:**The ASSUME directive tells the assembler to assume, that a certain register contains the base of some structure (in your case: segments). In your case, CS and DS point to the code segment and the data segmentrespectively, both the one and only of their respective kind.

**DISPLAY:** For display SRTING or in here MSG.

**INT:** INT is an assembly language instruction for x86 processors that generates a software interrupt. It takes the interrupt number formatted as a byte value.

**CMP:** The CMP instruction compares two operands. It is generally used in conditional execution. This instruction basically subtracts one operand from the other for comparing whether the operands are equal or not. It does not disturb the destination or source operands.

**JNE:** The JNE (or JNZ) instruction is a conditional jump that follows a test. It jumps to the specified location if the Zero Flag (ZF) is cleared (0). JNZ is commonly used to explicitly test for something not being equal to zero whereas JNE is commonly found after a CMP instruction.

**JMP:** the JMP instruction performs an unconditional jump. Such an instruction transfers the flow of execution by changing the instruction pointer register.

**Code:**

DATA SEGMENT

INPUTSTR DB 10,13,'ENTER NO $'

EVENSTR DB 10,13,'EVEN NUMBER $'

ODDSTR DB 10,13,'ODD NUMBER $'

DATA ENDS

DISPLAY MACRO MSG

MOV AH,9

LEA DX,MSG

INT 21H

ENDM

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START:

MOV AX, DATA

MOV DS, AX

DISPLAY INPUTSTR

MOV AH,1

INT 21H

MOV AH,0

CHECK:

MOV DL,2

DIV DL

CMP AH,0

JNE ODD ;JUMP IF ZERO FLAG IS CLEARED

EVEN:

DISPLAY EVENSTR

JMP DONE

ODD:

DISPLAY ODDSTR

DONE:

MOV AH,4CH

INT 21H

CODE ENDS

END START

**Output**

