Problem 3a

1. AX = 80B0h.
2. Multiply by CX = 3h.
   80B0h × 3
   80B0h × 2 = 10160h,
   10160h + 80B0h = 18210h.
3. The 32-bit product is 18210h. Split this into high–word and low–word:

   Low word (AX) = 8210h

   High word (DX) = 0001h

If DX Were Not Set to 0 First:
If a product were small enough to fit in AX and later code assumes DX is zero any leftover garbage in DX could lead to an incorrect interpretation of the full 32-bit result.

Problem 3b

The IDIV (signed division) instruction causes an overflow when the computed quotient is too large to be represented in the destination register.

Example:
If I wanted to divide –128 by –1, the mathematically correct quotient is 128, which cannot be represented as an 8-bit signed number (the maximum is +127). Thus, executing IDIV in this case will cause an overflow exception.

Problem 3c

1. mov ax, 0h; mov dx, 0h
   AX = 0000h, DX = 0000h.
2. sub ax, 5h
   Performs AX = 0000h – 0005h = FFFBh. A borrow occurs; therefore, the Carry flag is set.
3. sbb dx, 0
   Subtracts 0 plus the borrow from DX. Thus, DX = 0000h – 0001h = FFFFh

DX:AX is FFFFh:FFFBh.

Purpose:
This sequence propagates the sign from the low word (AX) into the high word (DX). In other words, it is a manual sign extension—from a 16-bit signed result to a 32-bit signed result in DX:AX.

Problem 4

In the data segment a table called CaseTable is defined. Each entry contains a character literal (such as 'A') immediately followed by the address of a procedure (such as Process_A). Constants EntrySize and NumberOfEntries are calculated by the assembler from the table's size, so that the loop later knows exactly how many entries there are and how many bytes to move for each entry. At runtime the program displays a prompt, reads a single character into AL, and then sets EBX to point to the start of the case table. It loops through all table entries- comparing the input character with the lookup byte in each entry. When a match is found, the code uses an indirect call to invoke the corresponding procedure. That procedure loads a pointer to a message into EDX. Then, the message is printed and a new line is output. If no match is found after all entries have been examined, the program simply exits. When the table is expanded manually by entering fixed values for EntrySize and NumberOfEntries, any subsequent change to the layout would require a corresponding update of those values by hand. This increases the risk of error and reduces the flexibility of the code.