```
void CO(){
    // this method will NOT be executed for the UNARY instructions, since the DI() method would have set the execution flag
    // for this method to 'false', and 'true' for TRINARY instructions...
    // here we calculate the operand's EFFECTIVE ADDRESS, setting the EA global var.,
    // for use in the FO() or WO() stages and the EX() stage for the BR’s only.
    // in the FI() method we set cpu.OS to the operand specifier value
    // in the DI() method we set the cpu.MODE to the letter corresponding to the addressing mode
    // 'i' for Immediate, 'd' for Direct, 'n' for iNdirect and 'x' for indeXed
    //
    // IF 'indexed' mode;
    // effective address(EA) is: the value of the operand specifier(cpu.OS) plus the contents of the 'X' reg.
    //
    // IF 'indirect' mode
    // effective address is calculated as follows:
         // 1) use the value of the operand specifier(cpu.OS) in order to retrieve a two byte address from memory
         // making use of the global var NEA to concatenate the two bytes.
         // 2) then the two byte address retrieved from memory IS the effective address(EA)
    //
    // IF 'direct' mode
    // effective address(EA) is: the value of the operand specifier(cpu.OS).
    //
    // IF 'immediate' mode
    // we need to set the FO() flag to false, so we do not attempt to retrieve an operand value from memory.
    // the operand value(cpu.OP) is the value of the operand specifier(cpu.OS).
    // the effective address(EA) is the value of the operand specifier(cpu.OS);
    // we need to set the EA, because the BR’s instructions will use it to alter the cpu.PC during the EX() step.
 }; // end of the CO() method
 //
 void FO(){
    // this method will NOT be executed for:
    // 1) the two types of the STORE instructions(ST/STBYTE), and 2) the BR’s,
    // since the DI() method would have set the execution flag for this method to 'false',
    // and 'true' for the others...
    // here we make use of the EA calculated in the CO() method
    // if an "LDBYTEA" or "LDBYTEX" instruction:
    //  uses the effective address (EA) to fetch ONLY ONE byte from memory, an instruction's Operand's value (1st or ONLY byte)
    //  and stores it into the operand value(cpu.OP's) LOW ORDER byte
    // otherwise, for the rest of the instructions:
    //  uses the effective address (EA) to fetch TWO bytes from memory, an instruction's Operand's value (1st/2nd bytes)
    //  and stores them into the operand value(cpu.OP
};// end of the FO() method
```

```
 void WO(){
       // this method will ONLY be executed for the two types of the STORE instructions(ST/STBYTE),
       // since the DI() method would have set the execution flag for this method to 'true' for ONLY these instructions!!!!
       // and 'false for the rest!!!!
       //
       // here we make use of the EA calculated in the CO() method
       // if a "STBYTEA" or "STBYTEX"  instruction:
       //  uses the effective address (EA) to store one byte(LOW ORDER byte) from the operand value(cpu.OP), to memory.
       // otherwise, for the "STA" or "STX" instruction:
       //  uses the effective address (EA) to store TWO bytes, from the operand value(cpu.OP), to memory
       //
};// end of the WO()
```