

```
//
// start of individual instruction execution methods;
// at this point all we have is the 'shell' for each of the UNIMPLEMENTED instruction methods
// .....
// NOTE: YOU MAY NOT USE ANY ADDITIONAL JAVA LANGUAGE OBJECTS/METHODS FOR THE REMAINDER OF THE CODE!!!!
// start of mem/reg instructions
// ADD/SUB/AND/OR
// These instructions involve the pattern REG = REG 'operation' OPERAND as the only action with the data values...
// In addition ALL these instructions need to simulate the setting of the Status Flags as follows:
/*
```

Clarification regarding the setting of the Status Flags in the coding for the Instruction methods that need to set them:  
When implementing the methods that execute Instructions, please consult the PEP8 instruction specifications chart that were provided and/or that are shown in the PEP8 program, in the HELP menu entry under PEP8/Reference.

For each instruction you will see WHAT status flags need to be set.

IF a flag is to be set then you must provide code that checks for the appropriate condition and EITHER sets it ('1') or resets it ('0').  
Please realize that you need to deal with the fact that we are using an 'int' (4 bytes) to represent the PEP8 registers, which are ONLY two bytes.

- Checking for a 'ZERO' condition is straight forward: if ( A == 0 ) Z=1; else Z=0;
- Checking for a 'NEGATIVE' condition is also simple, when you take into consideration that the correct sign bit is the 15th bit, NOT the 31st bit, so just single it out with a mask and check for it being a 1: if ( ( A & 0x00008000 ) == 0x00008000 ) N=1;
- Checking for an 'OVERFLOW' condition is done by implementing the 'OVERFLOW RULE' which states:  
"If two numbers are added and they are both positive OR both negative, then overflow occurs ONLY IF the result has the opposite sign".

For ADDition, the logic should be (in pseudocode): IF( (A>0 AND OP>0 AND (A + OP)<0 ) OR (A<0 AND OP<0 AND (A + OP)>0 ) V=1;

For SUBtraction, you need to consider the 'SUBTRACTION RULE' first, which states:

"To subtract one number (subtrahend) from another (minuend), NEGATE the subtrahend and ADD it to the minuend".

In other words:  $A - B = A + (-B)$ .

Therefore, checking for an 'OVERFLOW' when SUBtracting, must take this rule into consideration and NEGATE the subtrahend when checking the sign. So for the SUBtraction method, the logic should be (in pseudocode):

IF( (A>0 AND (-OP)>0 AND (A - OP)<0 ) OR (A<0 AND (-OP)<0 AND (A - OP)>0 ) V=1;

- Checking for a 'CARRY' condition when ADDing, needs to take into consideration that when performing and ADD using a two byte register, such as PEP8, the carry is a 'hardware' function that can only be 'SIMULATED' in software.  
So we take advantage of the fact that we are representing the two byte regs(A/X), using a four byte 'int' variable.  
So we perform the ADDition, and IF there was a 'carry', it will show as a 'spillover',  
a '1' in the 16th bit position of the 'int' variable that can be detected using a mask: if ( ( A & 0x00010000 ) == 0x00010000 ) C=1;
- Checking for a 'CARRY' condition when SUBtracting is very complex and we will not attempt it here. I will be willing to explain WHY in class,  
but for this project we will just ignore it and always set the 'C' bit to '0'.

```
*/
//
```

```
// CP is implemented exactly as the SUB, EXCEPT!!!! that it is used only for its SIDE EFFECT, which is the setting of the status flags...
// so this instruction uses the pattern TEMP = REG operation OPERAND, with int TEMP acting in lieu of the REG...
//;
//
// LD/LDBYTE
// These two instructions involve the pattern REG = OPERAND , as the only action with the data values...
// In addition these instructions need to simulate the setting of the Zero and Neg Status Flags
//
// ST/STBYTE
// These two instructions involve the pattern OPERAND = REG , as the only action with the data values...
// These instructions DO NOT change the Status Flags
//
} // class step
```