// start of unary instr. NOT/NEG/ASL/ASR/ROL/ROR
These instructions involve the pattern REG = REG 'operation' as the only action with the data values...
In addition ALL these instructions need to simulate the setting of the appropriate Status Flags...

- 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.
- Checking for a 'CARRY' condition is done by settinf the cpu.C to the corresponding bit value, Low order for Right shift/rotate and High order for Left shift/rotate

```
 void NOT() {
  // use the '~' java operator and then set the Z/N flags
 } // end of NOT()
//
 void NEG() {
// use the '-' java operator and then set the Z/N flags
// for the 'V' flag, just 'think' about how you would set it…
 } // end of NEG()
//
 void ASL() {
   // cøpy/save the value of the leftmost Bit to the cpu.C (the Carry).
   // use the '<<' java operator to shift left one position
   // and then set the Z/N flags
   // for the 'V' flag, just 'think' about how you would set it…
 } // end of ASL()
//
 void ASR() {
   // copy/save the value of the rightmost Bit to the cpu.C (the Carry).
   // copy/save the value of the leftmost Bit (the Sign) to a temp var(int Sign).
   // use the '>>' java operator to shift right one position.
   // use the copied/saved 'Sign' value to SET the leftmost Bit, since we need
   // to PRESERVE the sign value and the shift would have set it to '0'…
   // and then set the Z/N flags
 } // end of ASR()
//
 void ROL() {
  // the PEP8 implementation is different than the 'canonical' rotates;
  // it implements a 'delayed' use of the 'C' bit …
  // copy/save the value of the leftmost Bit to a temp var(int LeftC).
  // use the '<<' java operator to shift left 1 position.
  // use the cpu.C (current Carry) value to set the RIGHTMOST bit.
  // Then, use the copied/saved 'LeftC' value to SET the cpu.C (the Carry),
  } // end of ROL()
//
 void ROR() {
 // the PEP8 implementation is different than the 'canonical' rotates;
  // it implements a 'delayed' use of the 'C' bit …
  // copy/save the value of the rightmost Bit to a temp var(int RightC).
  // use the '>>' java operator to shift right 1 position.
  // use the cpu.C (current Carry) value to set the LEFTMOST bit.
  // Then, use the copied/saved 'RightC' value to SET the cpu.C (the Carry),
  } // end of ROR()
```