```
void EX(){ // the EXECUTE STEP code is structurally similar to the code in the DI() method
          // the difference is that the only ACTION taken here is to INVOKE/CALL the
          // specific method that will 'execute' the particular instruction!!! nothing else...
            //
           OP = cpu.getIS(); // local var OP is a modified Instr Spec. and used to bracket the 'switch'
       if(cpu.getIS() >= 0x70) { OP= (cpu.getIS() & 0xF8);} // if instr. uses the full address mode, then keep
the op code only
          switch (OP) {
          case 0x00: System.out.println("STOP INSTRUCTION EXECUTED");break;
          // start of branches bracket
          case 0x04: case 0x05:   BR();break;
          case 0x06: case 0x07: BRLE();break;
          .............................
          // end of branches bracket
          //
          // start of unary instr; operates on a REG with no operand
          case 0x18: NOTA();break;
          case 0x19: NOTX();break;
          case 0x1A: NEGA();break;
          .............................
          // end of unary instr
          //
          // the following instructions are specified in range increments of 8
          // with the 3 low order bits having the addressing mode set to 0's
          // for example: the range 0x70/0x77 are all ADDA with the 8 different modes
          //
          case 0x70:    ADDA();break;
          case 0x78:    ADDX();break;
          case 0x80:    SUBA();break;
          ..........................................................
          default: Unary=false; cpu.DESCR="INVALID";break;
          //
         }// end of the 'switch' structure
  //
  }; // end of the EX() method
  //
```