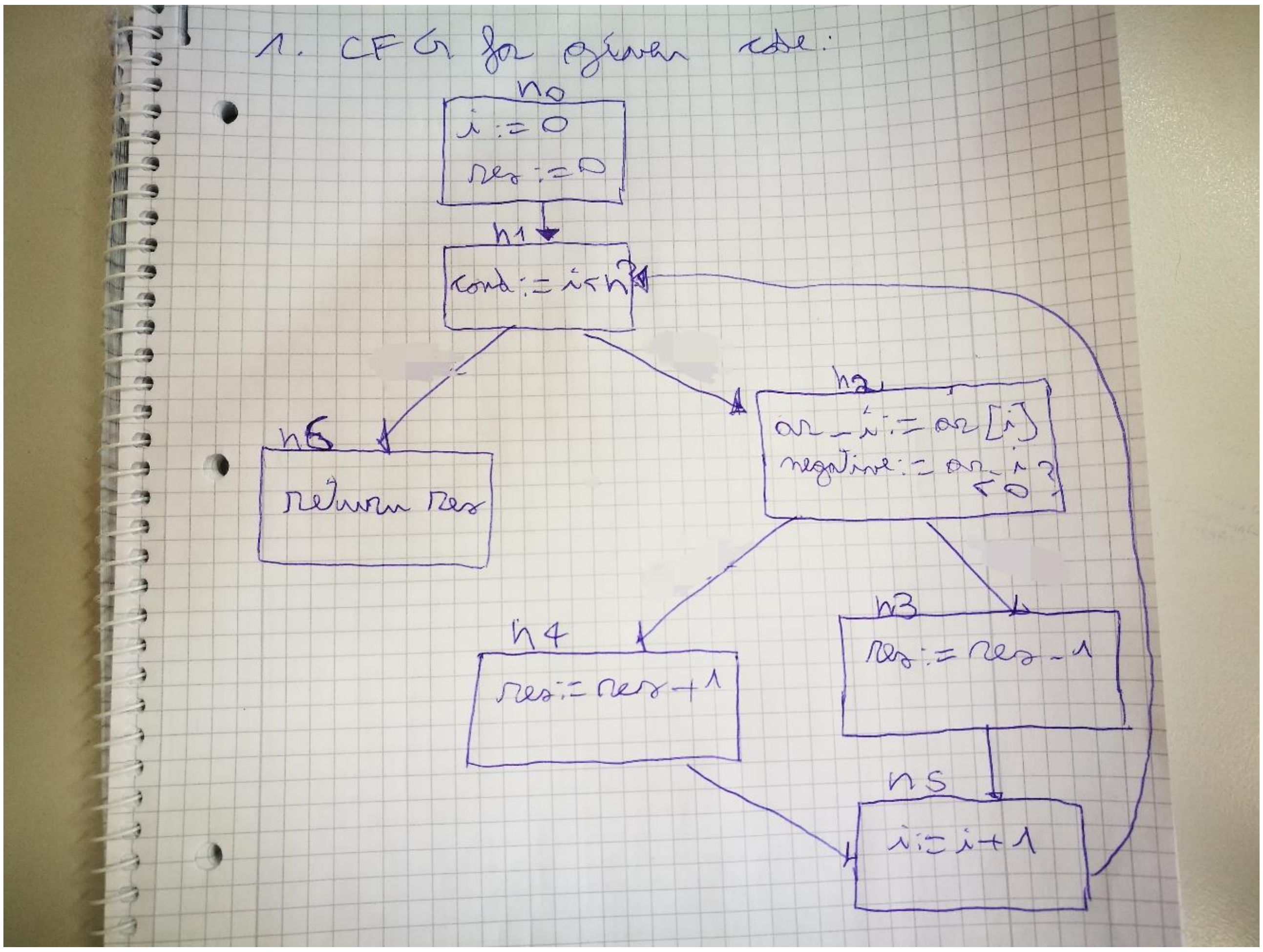
## 2 Theory: SSA-Form

**Task 1:**



**Task 2**: Explain why the code is not in SSA form.

The given code is not in SSA form because:

- In SSA form, variables need to have one (static) definition, whereas in the given code this is not the case. This single (static) definition of a variable may be in a loop that is executed many (dynamic) times [1], as it happens for variable “i” in basic block n5 of the CFG drawn in task 1. However, multiple definitions of a variable are otherwise not allowed. For instance, if defining the same variable in a branch operation (if-else), different variable names need to be used for the two respective TRUE basic block and FALSE basic (i.e: in the given code, both basic blocks n3 and n4 of the “if-else” branch operation in n2 shown in the CFG in task 1 define variable “res”, but according to SSA, this is not allowed and would need to use different variable names).
- Nodes to where two definitions of the same variable converge need to contain φ-functions at the beginning of the block. In the given code, there is no φ-function and hence this property surely does not hold.
  For example, we would need to add a φ-function in basic block n5, where basic blocks n3 and n4 converge and the value of “res” needs to be updated according to the basic block executed.

  [1]: Modern Compiler Implementation in Java. Andrew W. Appel, 1999

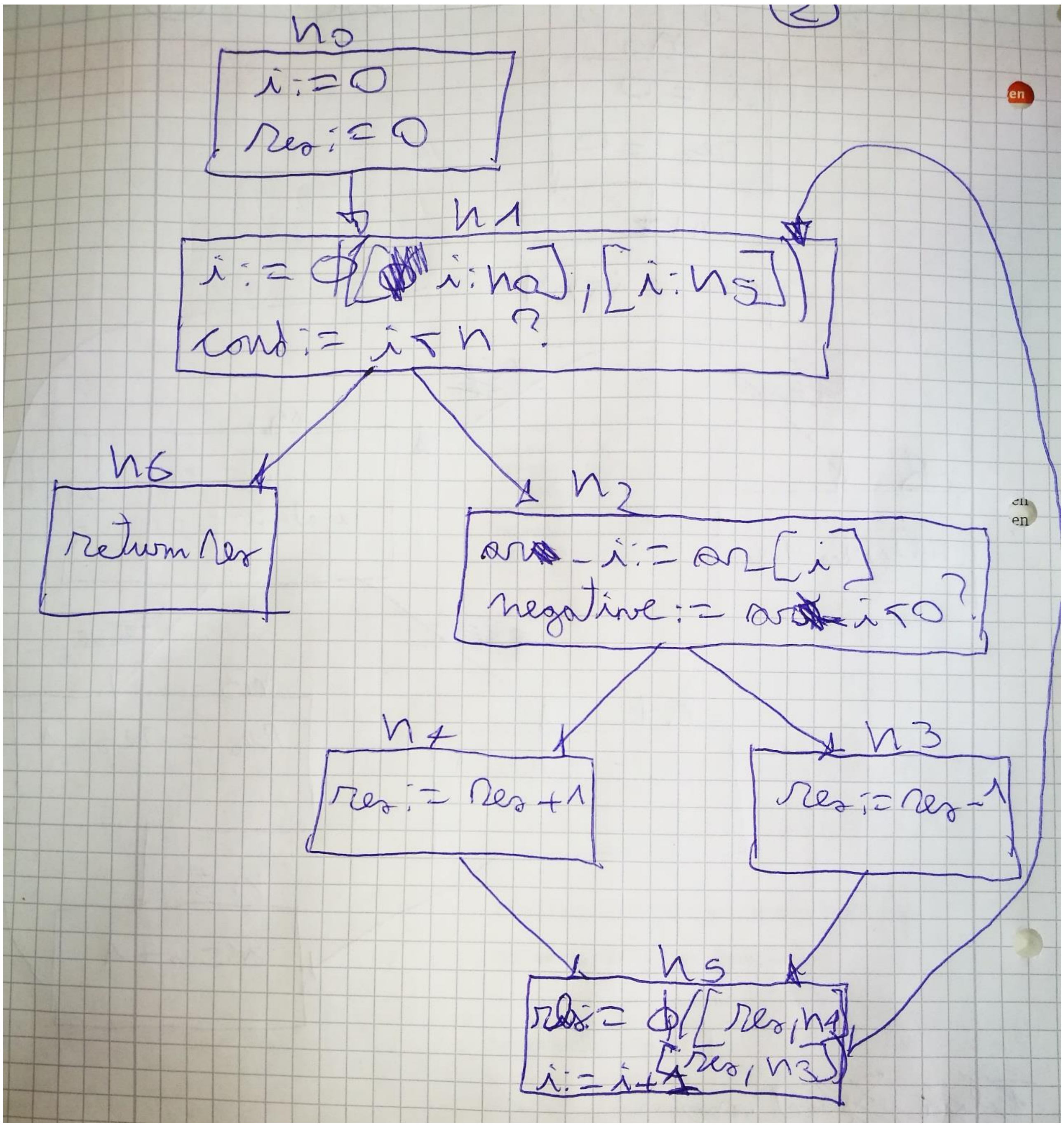
**Task 3:**

Conversion of code to SSA form encompasses the following steps:

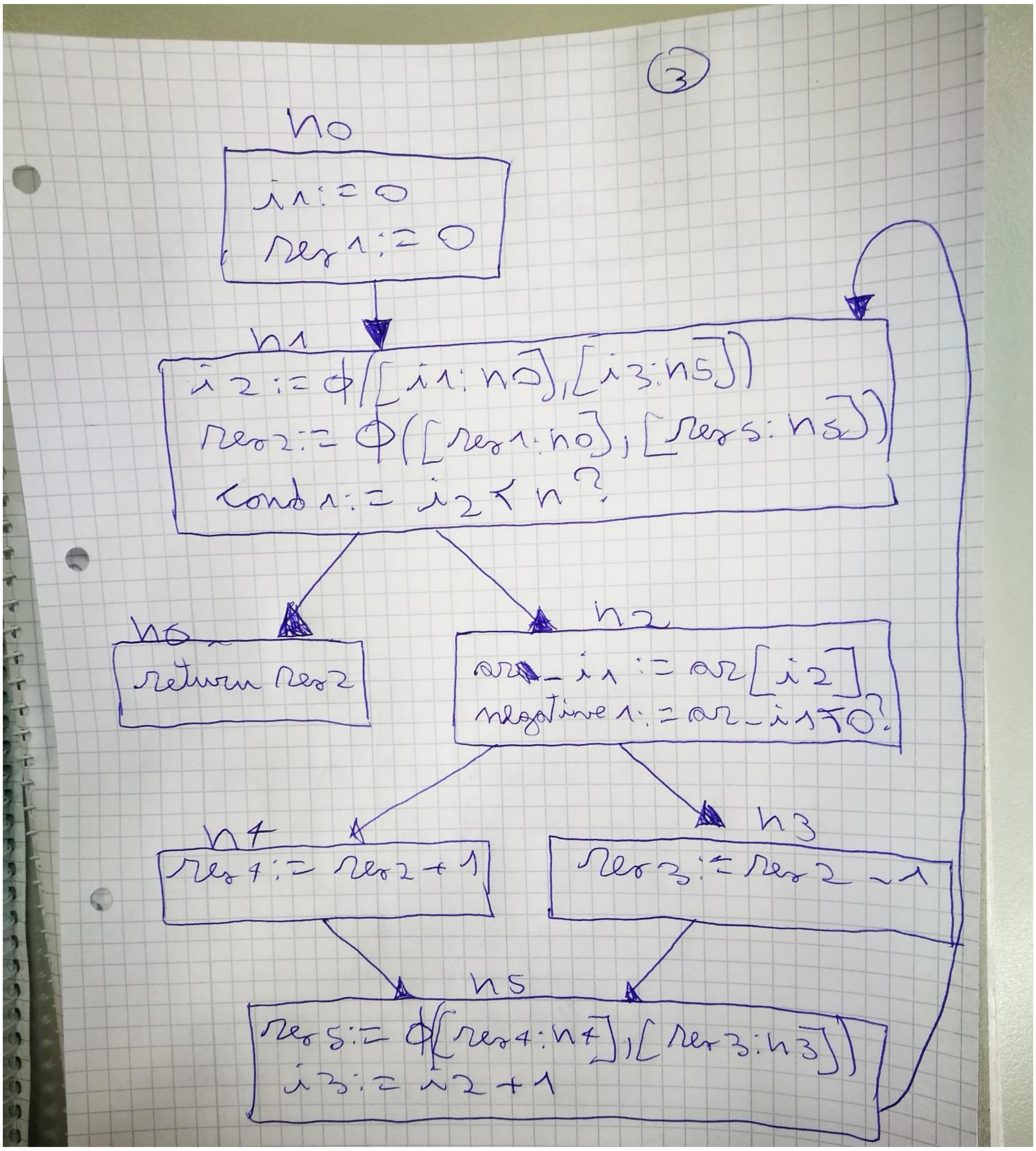
1. Program needs to be converted to a CFG.
2. Insert φ-functions for each variable at each join point (points where two basic blocks converge)
3. Rename variables so that they are in SSA form
4. Perform edge splitting (avoid edges from a node with multiple successors to node with multiple predecessors) and convert back from SSA form

1: See Task 1.

**2. Insert φ-functions** for each variable at each join point



**3. Rename variables** so that they are in SSA form



**4. Edge splitting**: We want avoid edges from a node with multiple successors to a node with multiple predecessors.

Hence respecting the following property:

∃ edge from $l_i$ --> $l_j$, and $|succ[l_i]| > 1$ AND $|pred[l_j]| > 1$

If this holds, we split the edge from $l_i$ --> $l_j$

However, in this case, there are no two edges from $l_i$ to $l_j$ where this property holds and we cannot perform any edge splitting.