A very simple form of conditional execution can be represented by this syntax:

\[
\text{if } \text{condition} \text{ then } \text{dosomething}
\]

\textit{condition} would of course be specified in polish notation, and \textit{dosomething} would be some kind of simple statement. The whole thing would be reduced to assembly language in these steps:

Think of a new label that’s never been used before, call it L
Perform jump to L if the condition is false
Do the something
Put the label L here.

For example:

\[
\text{if } < \text{x } 7 \text{ then set } \text{x } = + \text{x } 1
\]

produces (on the assumption that x is our first local variable):

\[
\begin{align*}
\text{LOAD} & \quad \text{R1, [FP-1]} \\
\text{LOAD} & \quad \text{R2, 7} \\
\text{COMP} & \quad \text{R1, R2} \\
\text{JCOND GEQ, _L123} \\
\text{LOAD} & \quad \text{R1, [FP-1]} \\
\text{LOAD} & \quad \text{R2, 1} \\
\text{ADD} & \quad \text{R1, R2} \\
\text{STORE} & \quad \text{R1, [FP-1]}
\end{align*}
\]

\_L123:

So it all comes down to needing a function that can read a conditional expression in polish notation and generate a jump to a given label if that condition turns out to be false.

To make the reading easier, the example expressions will be in normal infix non-polish notation, considering the various cases one by one.

How to jump to L if false is false:

produce JUMP L

How to jump to L if true is false:

How to jump to L if \textsc{not} A is false:

jump to L if A is true

How to jump to L if A AND B is false:

jump to L if A is false
jump to L if B is false
How to jump to L if A OR B is false:
make up a totally new label, call it M
jump to M if A is true
jump to L if B is false
produce M:

How to jump to L if A < B is false:
use the normal polish function to get the value of A into a register
use the normal polish function to get the value of B into the next register
produce COMP first register, second register
produce JCOND GEQ, L

How to jump to L if A = B is false:
use the normal polish function to get the value of A into a register
use the normal polish function to get the value of B into the next register
produce COMP first register, second register
produce JCOND NEQ, L

How to jump to L if the value of the variable X is false:
load X into a register
produce COMP register, 0
produce JCOND EQL, L

etc.

So a function to read a polish conditional and jump to a label if it is false would look something like this:

```cpp
void jumpifpolishfalse(istringstream & sin, int label, int reg)
{
    get first symbol;
    if first symbol is the reserved word "false"
        output "JUMP _L", label
    else if first symbol is the reserved word "true"
        do nothing
    else if first symbol is the reserved word "not"
        jumpifpolishtrue(sin, label, reg);
    else if first symbol is the reserved word "and"
    { jumpifpolishfalse(sin, label, reg);
        jumpifpolishfalse(sin, label, reg); }
    else if first symbol is the reserved word "or"
    { int mylabel = nextfreelabel;
        nextfreelabel+=1;
        jumpifpolishtrue(sin, mylabel, reg);
        jumpifpolishfalse(sin, label, reg);
        output "_L", mylabel, ":" }
```
else if first symbol is the operator “<”
{ polish(sin, reg);
  polish(sin, reg+1);
  output "COMP R", reg, ", R", reg+1
  output "JCOND GEQ, _L", label }

else
  etc

BUT... what would happen if a boolean expression appeared outside of its natural
environment of ifs and whiles, just in a normal expression, maybe like

    set x = and > a 0 <= a 10

meaning that the variable x is to be set to 1 if a is between 0 and 10, and to zero
otherwise.

The expression in an assignment is always processed by the original polish function, so
the question is really what should polish do if it sees an operator like and, or, >, =?

The answer is quite simple. It must produce code that would get the value 0 or 1 into the
appropriate register depending on the condition, and it has a friend jumpifpolishfalse that can deal with conditional execution, so:

    LOAD  reg, 0
    reserve next free label, L.
    jumpifpolishfalse(sin, L, reg+1);
    LOAD  reg, 1
    L: