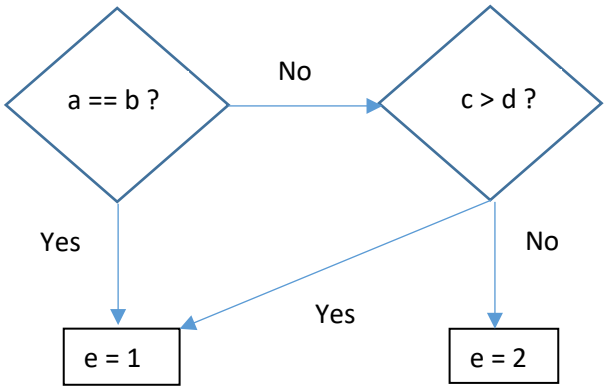
Multiple conditions

To compile if-statements that use multiple conditions joined by AND (&&) or OR (||), we can accomplish this in assembly by using *consecutive branch instructions*.

As an example, consider a HLL if statement featuring two conditions joined by OR.

```
if (a == b || c > d)
   e = 1;
else
   e = 2;
```

Let's depict this logic using a flow chart. This may help inspire the proper branch instructions.



The key is to ask ourselves when we need to evaluate both conditions. Here, when do we need to evaluate both the "a == b" and the "c > d"? Because of the OR between them, it is when the first condition is false. When a == b is false, we need to fall through. Thus, when a == b is true, we need to branch. If a == b is branching, then it has to branch to the "then" clause of the if statement.

To minimize confusion, I recommend that both branch instructions have the same destination. So, our "pseudocode assembly" code now looks like this:

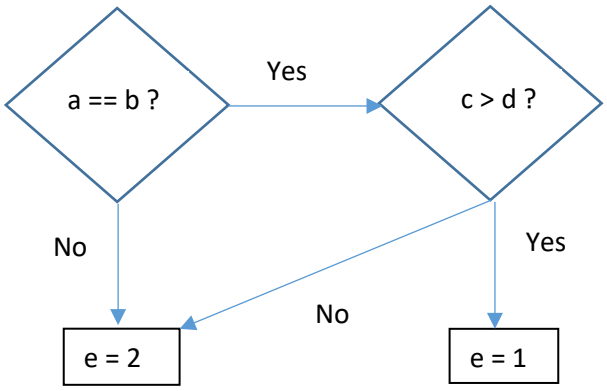
```
        beq a, b, yes
        bgt c, d, yes
        e = 2
        j endif
yes:
        e = 1
endif:
```

Notice that the sense of the two branch instructions is the same as the HLL code. In other words, the original == and > correspond to the branch instructions `beq` and `bgt`. Finally, we can allocate registers s1 through s5 to correspond to a through e. Now our code is:

```
        beq $s1, $s2, yes
        bgt $s3, $s4, yes
        li $s5, 2
        j endif
yes:
        li $s5, 1
endif:
```

What if the two conditions are joined by an && instead of || ? Here is the HLL code and corresponding flow chart:

```
if (a == b && c > d)
   e = 1;
else
   e = 2;
```



We need to ask ourselves: When would we need to evaluate both conditions? Because of the AND between them, it's when the first condition is true. We want to fall through when a == b is true. That means we branch when a == b is false, i.e. when a != b. We want to branch to the "else" part of the code. Therefore, we need to write the first condition using `bne`. The second condition c > d is false when c <= d, so we write the second condition using `ble`. Our pseudocode becomes:

```
        bne a, b, no
        ble c, d, no
        e = 1
        j endif
no:
        e = 2
endif:
```

Notice that in this case, the HLL relational operators have been reversed. The == and > now correspond to the branch instructions `bne` and `ble`. And after allocating registers, we obtain:

```
        bne $s1, $s2, no
        ble $s3, $s4, no
        li $s5, 1
        j endif
no:
        li $s5, 2
endif:
```