Lets look a bit more into what the branch `br` statement means.

We've seen what is called an **unconditional branch** before: unconditional meaning it happens every time. For example, in:

```
iloop: br iloop
```

Here, `iloop` as learned before, `iloop` is a memory label for this instruction, and the `br` statement tells the program to "branch" to the instruction labeled with `iloop`, as opposed to just running the next instruction. Since this *is* `iloop`, it gets stuck in an infinite loop.

A **conditional branch** will only go to the specified instruction if a condition is true. Otherwise, it will go to the instruction that is next in memory.

Let's look at an example of a program that counts from 1 to 4:

```
movi r8, 1
movi r9, 4

myloop: addi r8, r8, 1
                ble r8, r9, myloop

fin: br fin
```

Here we see a new `ble` statement. This is a type of conditional loop that stands for "branch if less than or equal". In this specific case, the code will branch to `myloop` if `r8` is less than or equal to `r9`. Otherwise, it will move on to the next instruction. An equivalent C code for the above would be:

```
int A = 1, B = 4;
do {
        A = A + 1;
} while (A <= 4)
```

Here is the general form of a conditional branch:

```
bXX rA, rB, label
```

Where `rA`, `rB`, are registers and `XX` is one of several branch conditions:

- `beq` branch if equal

- `bne` branch if not equal
- `blt` branch if less than
- `ble` branch if less than or equal
- `bgt` branch if greater than
- `bge` branch if greater than or equal
- `bgeu` branch if greater than or equal, unsigned
- `bleu` branch if less than or equal, unsigned