Think about implementing logic operators `||`, `&&` and the sort in C. How can we do these operations as bitwise operations (`|` and `&` in C) in assembly?

The assembly instructions are:

```
and rC, rA, rB
or rC, rA, rB
xor rC, rA, rB
```

`rC` stores the result of the bitwise logic operation done on `rA` and `rB`.
You can add an `i` after the instruction, for example `addi`, if you would like to compare a register value with an immediate instead of another register value.

These operations are **bitwise**, meaning the register storing the result compares **each bit** of the two comparands, and stores the resulting bit into the appropriate bit of the result.

This begs a question: when using the immediate form of a bitwise comparison operator, remind yourself that the **immediate can only be 16-bits** (this is true for things like `movi` as well). Registers store 32 bits, so, what happens when trying to compare the upper 16 bits? Keep this question in mind.

We will often need to move bits around in words (`<<` or `>>` in C). Here is an example of such an instruction.

```
srli r8, r9, 3
```

The above code shifts the bits in register `r9` right by `3` bits, and stores the result in `r8`.
Similarly we can do this without immediate where

```
srl r8, r9, r10
```

will shift `r9` by `r10` bits.

The shift left instructions are `sli` and `sll`.

This instruction will lose the bits at the end of the register, and replace the bits at the beginning with `0`. To do a rotating shift, use:

- `rol` and `roli` for rotating left shift
- `ror` and `rori` for rotating right shift