Report

(1)

Which MIPS instructions do you think would produce wrong outputs if the ControlUnit signal *RegWrite* is *'stuck at 0'*, i.e., *RegWrite* always has the value 0? In other words, which MIPS instructions depend on the control signal *RegWrite*?

All Mips instructions that store something back into the register won't work anymore.
In the controlUnit we can see all affected operations listed:
`OP_RTYPE`, `OP_LW` and `OP_ADDI`

---

(2)

Explain why a 6-bit address is enough for the instruction and data memory. *(Hint: think about the size of the memory.)*

A 6-bit address can represent $2^6 = 64$ distinct values. In both the `DataMemory` and `InstructionMemory` modules, the memory arrays are declared as `reg [31:0] DataArr [63:0]` and `reg [31:0] InsArr [63:0]`. Thus 6 bits are enough to uniquely address each data and instruction in memory.

(3)

As you might have noticed, there are three different counters used in this lab. One is present in the *snake_patterns.asm* file, the second is in the clock_div module, and the third is the `DispCount` signal for the 7-segment display. Explain the functions of each of these three counters/dividers in a sentence or two each.

`snake_pattern.asm`

This counter keeps track of the position in the loop of snake patterns. Making it go faster, loops through the pattern faster.

`clock_div.v`

This counter is responsible for slowing down the frequency of the processor. It is incremented on every tick of the hardware clock and every x ticks, it sends a *divided* clock output.

```
// Instantiate an internal clock divider that will
// take the 50 MHz FPGA clock and divide it by 5 so that
// We will have a simple 10 MHz clock internally
```

`DispCount`, `top.v`

This counter determines, which seven segment display to turn on (send logical 0). `DispDigit` controls, which specific segment is turned on, but for all AN. For this Reason, we need to turn on only the one that is needed.

```verilog
always @ ( * ) begin
        case (DispCount[15:14])
                2'b00: begin
                        AN = 4'b1110;
                        DispDigit = DispReg[6:0];
                end   // LSB
                2'b01: begin
                        AN = 4'b1101;
                        DispDigit = DispReg[13:7];
                end   // 2nd digit
                2'b10: begin
                        AN = 4'b1011;
                        DispDigit = DispReg[20:14];
                end   // 3rd digit
                2'b11: begin
                        AN = 4'b0111;
                        DispDigit = DispReg[27:21];
                end   // MSB, default
        endcase
end
```