4. (10 points) There are four Verilog code snippets in this section. Only one of these codes is syntactically correct. All others have a problem with the **syntax**. For each code, first state whether or not there is a mistake. If there is a mistake explain how to correct it. *Note: Assume that the behavior as described, is correct*

(a)

```
1  module one (input [1:0] sel, input [3:0] data, output z);
2
3    assign z = sel[1] ? (sel[0] ? data[0] : data[3])
4                      : (sel[0] ? data[2] : data[1]);
5  endmodule
```

Solution: This code is correct. The distribution of the data bits may seem strange, but we are not checking for behviour.

(b)

```
1  module mux2 ( input [1:0] i, input sel, output z);
2
3    assign z= (sel) ? i[1]:i[0];
4
5  endmodule
6
7  module two ( input [3:0] data, input sel1, input sel2, output z);
8
9    mux2 i0 (.i(data[1:0]), .sel(sel1), .z(m[0]) );
10   mux2 i1 (.i(data[3:2]), .sel(sel1), .z(m[1]) );
11   mux2 i2 (.i(m), .sel(sel2), .z(z) );
12
13 endmodule
```

Solution: This code has mistakes. In `module two` there is an additional signal `m` used. This has not been declared, it should be declared as `wire [1:0] m;`.

(c)

```
module three (input [1:0] sel, output reg [7:0] z);

  always @ (sel)
    if      (sel = 2'b01) z=8'b01010101;
    else if (sel = 2'b10) z=8'b10101010;
    else                  z=8'b00000000;

endmodule
```

Solution: This code has mistakes. The condition checking for `sel` has been written as = which is an assignment. It should be == in both instances.

(d)

```
module four (input [1:0] sel, input neg, output reg [3:0] z);

  always @ (neg, sel)
      if (neg)    z = 4'b1111;
      else        z = 4'b0000;
      if (sel[1]) z = 4'b0001;
      if (sel[0]) z = 4'b0010;

endmodule
```

Solution: This code has mistakes. There are 3 seperate `if` statements following `always`. These should be within a `begin ... end` block. Note that, it would not be correct to have three separate `always` statements as this would mean driving the signal `z` from three different processes.