3. This question is about SRAMs. Subquestions can be answered independently. If you have skipped the previous part, and think you need to make some assumptions, please clearly state your assumptions.

   (a) (2 points) For the design of a microprocessor you are given a pre-designed SRAM with 16-bit wide data inputs (`din`) and outputs (`dout`), fourteen address bits(`addr`), as well as the write enable (`we`) signal that is set to 1 when data is written to memory. The Verilog secription of this module is given below:

```
module sram (input [15:0] din,
             input [13:0] addr,
             input we,
             input clk,
             output [15:0] dout);

   // description of the module
endmodule
```

   How many bytes can be stored in this memory (in Kilobytes)?

   Solution: Each address stores $16/8 = 2$ bytes. There are 14 address bits, that allow 2^{14} addresses, which equals to $2^4 \times 2^{10} = 16 \times 1024 = 16k$ addresses. In total this makes $16k \times 2bytes = 32kBytes$.

   (b) (2 points) Using the memory `sram` described in 3.a we want to build a larger memory called `bigram`, that will have the following interface.

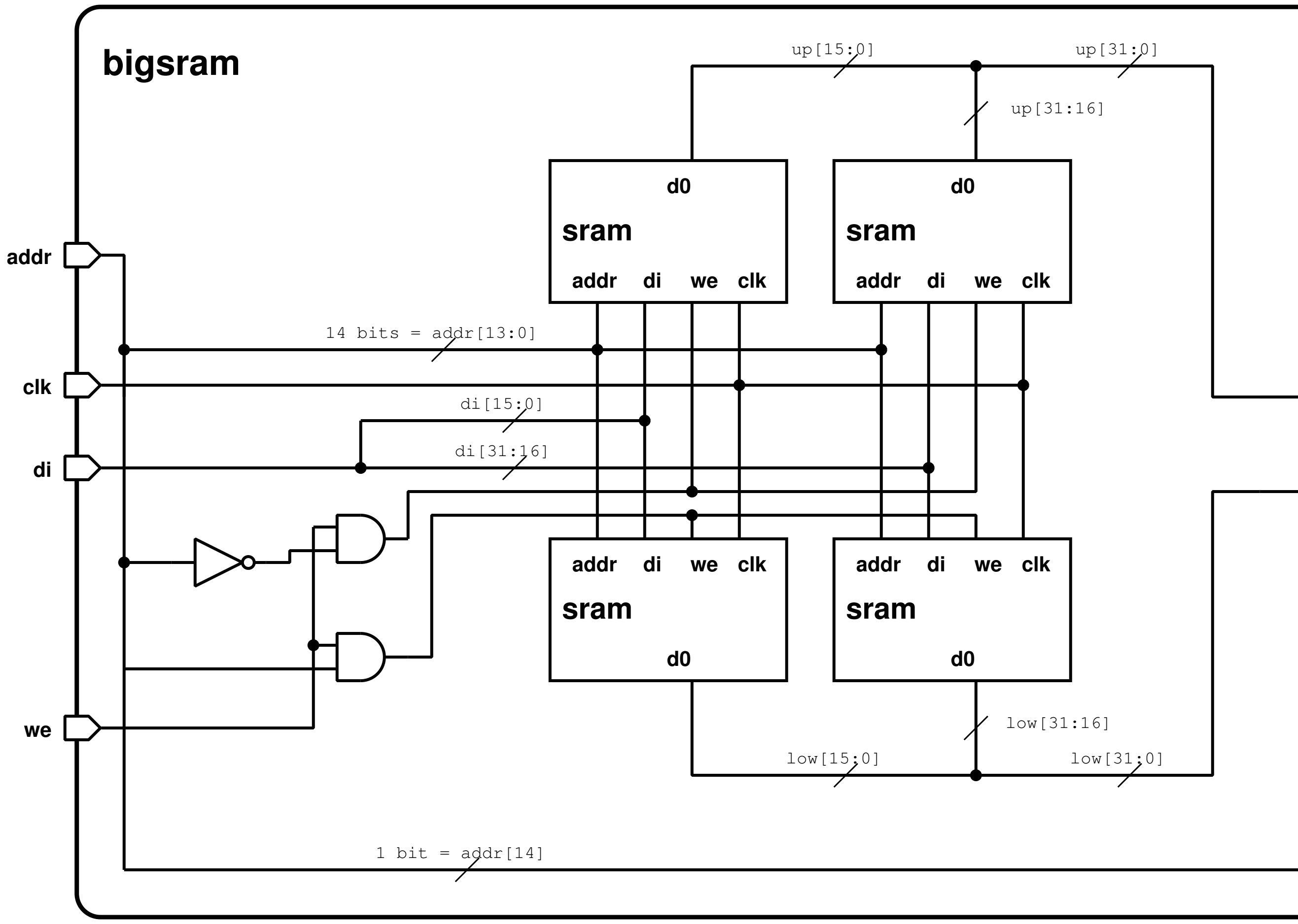
```
module bigram (input [31:0] din,
               input [14:0] addr,
               input we,
               input clk,
               output [31:0] dout);

   // description of the module
endmodule
```

   How many instances of the `sram` module do you need to create to build `bigram`? Why?

   Solution: The `bigram` stores twice the number of bits, 32 instead of 16. So we will need to use two `sram` instances. Then the `bigram` also uses one more bit for the address, doubling the number of locations. So we will need to double the number of `sram` instances once again, for a total of 4.

(c) (10 points) Draw a block diagram of the `bigram` and show in detail how it can be obtained by combining sufficient number of `sram` instances. To make your life easier, consider what needs to be done for the data input, data output and write enable separately.



(d) (10 points) Write the Verilog code that assembles `bigram` from sufficient number of instances of `sram`

```
module bigram (input [31:0] din,
               input [14:0] addr,
               input we,
               input clk,
               output [31:0] dout);

// declare signals if necessary
  wire [31:0]  up, low;
  wire         we_up, we_low;

// instantiate sram instances
  sram up0  (.din(din[15:0]),  .dout( up[15:0]),
             .addr(addr[13:0]), .we(we_up),  .clk(clk) );
  sram up1  (.din(din[31:16]), .dout( up[31:16]),
             .addr(addr[13:0]), .we(we_up),  .clk(clk) );
  sram low0 (.din(din[15:0]),  .dout(low[15:0]),
             .addr(addr[13:0]), .we(we_low), .clk(clk) );
  sram low1 (.din(din[31:16]), .dout(low[31:16]),
             .addr(addr[13:0]), .we(we_low), .clk(clk) );

// write enable
  assign we_up  = addr[14] ? we : 0;
  assign we_low = addr[14] ? 0 : we;

// select output
  assign dout = addr[14] ? up : low;

endmodule
```