# Solutions to Midterm 1

*There were two versions of each question. The values and the answers for both versions are given below.*

## Question 1

Fill the table below with the value of each expression as a Verilog numeric literal including the correct width and the correct value in *hexadecimal* base. Assume the following declarations:

```
logic [7:0] x ;
logic [3:0] y ;
```

and that `x` has the value `8'h83` (or `8'h5A`) and that `y` has the value `4'b0110`. The first row has been filled in as an example. You need not show your work or draw another box around the answer.

<table>
<tr><th><code>expression</code></th><th><code>value</code></th></tr>
<tr><td><code>x[3:0]</code></td><td><code>4'h3 (or 4'hA)</code></td></tr>
<tr><td><code>x - y + 4'h4</code></td><td></td></tr>
<tr><td><code>x[3] ? y : y >> 2</code></td><td></td></tr>
<tr><td><code>x ? {y,x} : {x,y,x}</code></td><td></td></tr>
<tr><td><code>x[3:0] ||  x |  y</code></td><td></td></tr>
<tr><td><code>x != y >= 0</code></td><td></td></tr>
<tr><td><code>x & y ^ 8'hff</code></td><td></td></tr>
<tr><td><code>x * y[3:0] && x</code></td><td></td></tr>
</table>

### Answers

For `x = 8'h83`:

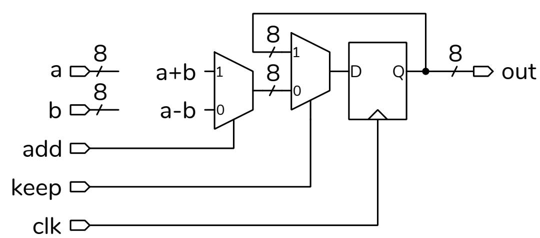
<table>
<tr><td><code>x[3:0]</code></td><td><code>4'h3</code></td></tr>
<tr><td><code>x - y + 4'h4</code></td><td><code>8'h81</code></td></tr>
<tr><td><code>x[3] ? y : y >> 2</code></td><td><code>4'h1</code></td></tr>
<tr><td><code>x ? {y,x} : {x,y,x}</code></td><td><code>20'h683</code></td></tr>
<tr><td><code>x[3:0] ||  x |y</code></td><td><code>1'h1</code></td></tr>
<tr><td><code>x != y >= 0</code></td><td><code>1'h1</code></td></tr>
<tr><td><code>x & y ^ 8'hff</code></td><td><code>8'hfd</code></td></tr>
<tr><td><code>x * y[3:0] && x</code></td><td><code>1'h1</code></td></tr>
</table>

and for `x = 8'h5a`:

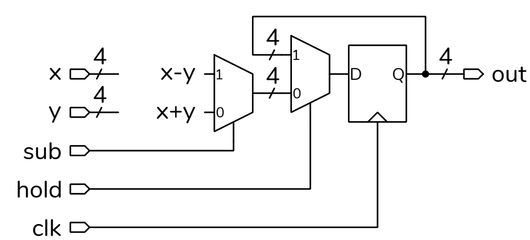
<table>
<tr><td><code>x[3:0]</code></td><td><code>4'ha</code></td></tr>
<tr><td><code>x - y + 4'h4</code></td><td><code>8'h58</code></td></tr>
<tr><td><code>x[3] ? y : y >> 2</code></td><td><code>4'h6</code></td></tr>
<tr><td><code>x ? {y,x} : {x,y,x}</code></td><td><code>20'h65a</code></td></tr>
<tr><td><code>x[3:0] ||  x |y</code></td><td><code>1'h1</code></td></tr>
<tr><td><code>x != y >= 0</code></td><td><code>1'h1</code></td></tr>
<tr><td><code>x & y ^ 8'hff</code></td><td><code>8'hfd</code></td></tr>
<tr><td><code>x * y[3:0] && x</code></td><td><code>1'h1</code></td></tr>
</table>

## Question 2

Write a Verilog module named `alu` (or `addsub`) that implements the following block diagram. The diagram follows the course conventions for block diagrams. Follow the course coding guidelines but omit comments.




(or




)

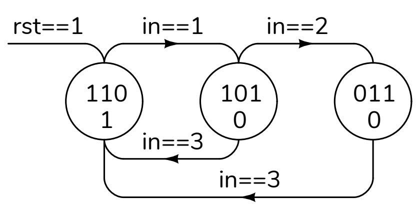
## Answers

```
module alu
  ( output logic [7:0] out,
    input logic [7:0] a, b,
    input logic add, keep, clk ) ;

   always_ff @(posedge clk)
     out <= keep ? out : add ? a+b : a-b ;

endmodule

module addsub
  ( output logic [3:0] out,
    input logic [3:0] x, y,
    input logic sub, hold, clk ) ;


   always_ff @(posedge clk)
     out <= hold ? out : sub ? x-y : x+y ;

endmodule
```

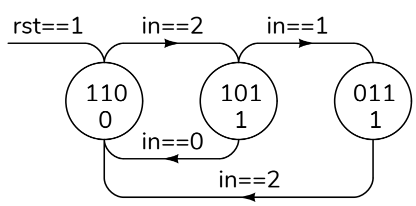
## Question 3

A state machine has a one-bit output named **out**, a two-bit input named **in**, a one-bit input named **rst**, and clock input named **clock**.

Write a Verilog module named **sm1** (or **sm2**) that implements the following state transition diagram. The diagram follows the conventions described in the lecture notes. Include all necessary declarations. Follow the course coding guidelines but omit comments.




(or




)

## Answers

```
module sm1
  ( output logic out,
    input logic [1:0] in,
    input logic rst, clock ) ;

   logic [2:0] state ;

   always_ff @(posedge clock)
     state <= rst ? 3'b110 :
              in == 1 && state == 3'b110 ? 3'b101 :
              in == 2 && state == 3'b101 ? 3'b011 :
              in == 3 && state == 3'b101 ? 3'b110 :
              in == 3 && state == 3'b011 ? 3'b110 :
              state ;

   assign out = state == 3'b110 ;

endmodule

module sm2
  ( output logic out,
    input logic [1:0] in,
    input logic rst, clock ) ;

   logic [2:0] state ;

   always_ff @(posedge clock)
     state <= rst ? 3'b110 :
              in == 2 && state == 3'b110 ? 3'b101 :
              in == 1 && state == 3'b101 ? 3'b011 :
              in == 0 && state == 3'b101 ? 3'b110 :
              in == 2 && state == 3'b011 ? 3'b110 :
              state ;

   assign out = state == 3'b101 || state == 3'b011 ;

endmodule
```