

Show your code and simulation at TA/tutor office hours to get credit. Both partners should be available and should be capable of answering questions related to the assignment. You can use behavioural or structural modelling in any part of the assignment. Test bench will be provided under resources. Please be on the lookout on Piazza.

In this lab, you will learn to design and build an 8-bit Tiny CPU.

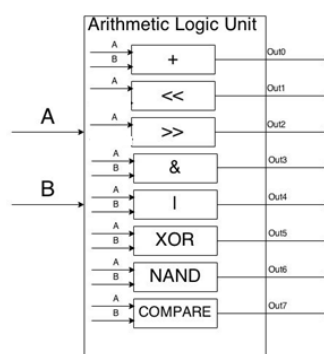
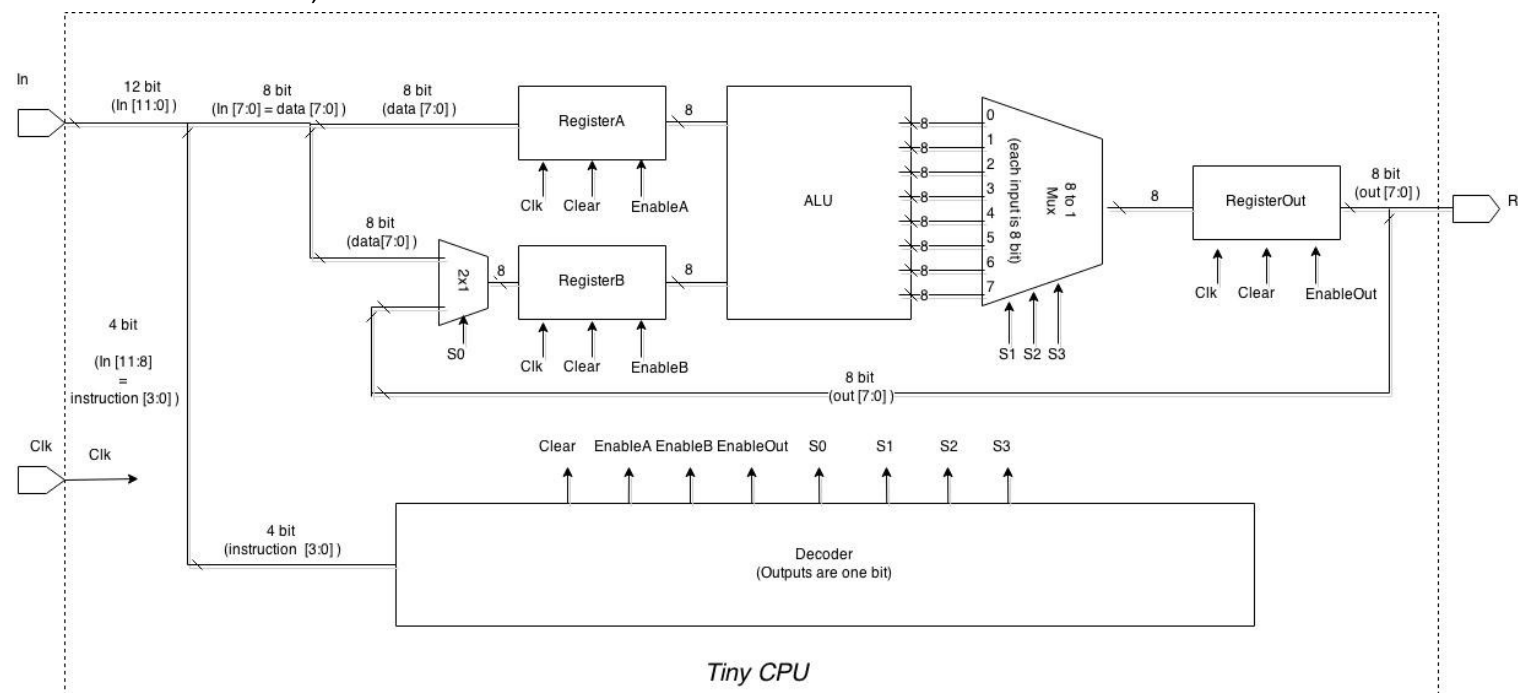
Use the same naming convention as shown in the schematic so that the test bench that we provide can be used effectively. Please use the same number of bits as prescribed in the problem statement. No more, no less. Study the schematic very carefully and make sure you have all the operations running functionally.

Your CPU will perform these basic operations:

8-bit Tiny CPU design high level specs:

- » 8-bit addition (without overflow).
- » Multiply by 2 and divide by 2 (shift left/right operation).
- » It must have two 8-bit registers that hold input values for the calculations.
- » It must have one 8-bit register for output
- » It takes in a 12-bit value as input (4-bit instruction set & 8-bit data).
- » It gives out an 8-bit value as result of the calculations.
- » Each instruction will happen during one clock cycle.

In order to build this CPU, let's understand what it looks like:



Note that all the inputs and outputs in the ALU diagram are 8 bits.

Detail Requirements:

- » 0000 00000000 - clear all the three registers (write "00000000")
- » 0001 (8-bit value)- put (8-bit value) value into RegisterA
- » 0010 (8-bit value) - put (8-bit value) value into RegisterB
- » 0011 00000000 - RegisterB = RegisterOut (write result back to RegisterB)
- » 0100 00000000 - RegisterOut = RegisterA + RegisterB, Neglect any overflows
- » 0101 00000000 - RegisterOut = RegisterA << 1 (The new bit in the LSB should be a zero)
- » 0110 00000000 - RegisterOut = RegisterA >> 1 (The new bit in the RSB should be a zero)
- » 0111 00000000 - RegisterOut = RegisterA & RegisterB (bitwise)
- » 1000 00000000 - RegisterOut = RegisterA | RegisterB (bitwise)
- » 1001 00000000 - RegisterOut = RegisterA XOR RegisterB (bitwise)
- » 1010 00000000 - RegisterOut = RegisterA NAND RegisterB (bitwise)
- » 1011 00000000 - Comparator: if (RegisterA < RegisterB) RegisterOut ="00000000",
else if (RegisterA >= RegisterB) RegisterOut ="11111111"

Note that shifting a binary number to left by one position and filling the LSB with a zero is the same as multiplying by 2 (vice versa for dividing)

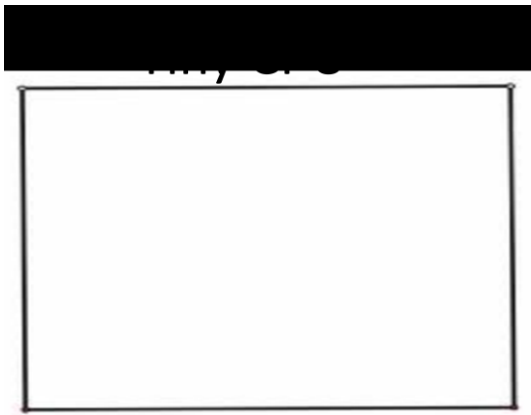
If you give inputs other than the ones that are specified above, your module should not do anything. Instead, the registers should hold on to their previous values.

Use the exact same variable names as defined in the table/diagrams because we will provide you a test bench with the same variable names. For example, you should define your Tiny CPU I/O's as

```
input [11:0] In;
input Clk;
output [7:0] Result;
```

Bellow you can find the truth table of your Decoder

Operation	Instruction	Clear	EnableA	EnableB	EnableOut	S0	S1	S2	S3
Clear	0000	1	1	1	1	X	X	X	X
Move1	0001	0	1	0	0	X	X	X	X
Move 2	0010	0	0	1	0	0	X	X	X
Store out	0011	0	0	1	0	1	X	X	X
+	0100	0	0	0	1	X	0	0	0
<<	0101	0	0	0	1	X	0	0	1
>>	0110	0	0	0	1	X	0	1	0
&	0111	0	0	0	1	X	0	1	1
	1000	0	0	0	1	X	1	0	0
XOR	1001	0	0	0	1	X	1	0	1
NAND	1010	0	0	0	1	X	1	1	0
Compare	1011	0	0	0	1	X	1	1	1
INVALID		0	0	0	0	X	X	X	X



```

module TinyCPU(
  input [11:0] In,
  input Clk,
  output [7:0] Result
);
...
endmodule

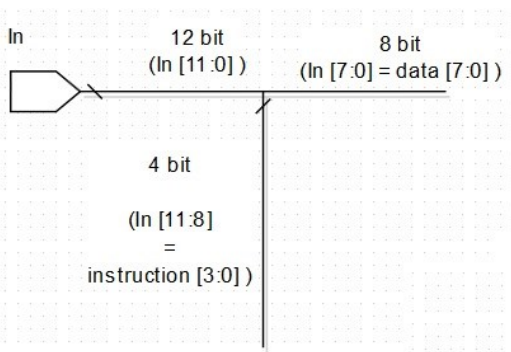
```

Instruction
decoder

```

module InstructionDecoder(
  ...
);
always@(instruction)
  if (...)
    begin
      ...
    end
  else if (...)
    begin
      ...
    end
end
endmodule

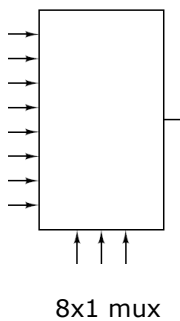
```



```

module BusSplit(
  ...
);
assign data = In[7:0];
assign instruction = In[11:8];
endmodule

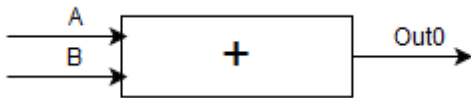
```



```

module Mux8To1(
  input [7:0] A,B,C,D,E,F,G,H;
  input [2:0] Sel,
  output [7:0] Y
);
// if else block...
endmodule

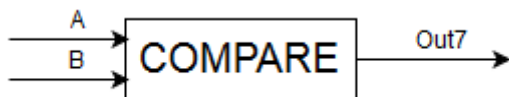
```



```

module Adder(
...
);
always @(A or B)
begin
    Out0 <= A + B;
end
endmodule

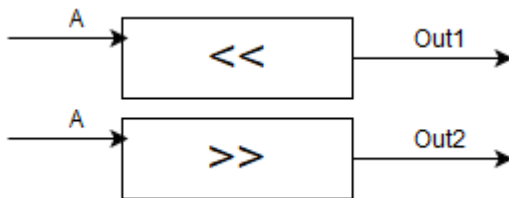
```



```

module Comparator(
...
);
reg[7:0] Out7;
always @(A or B)
begin
    if (A >= B)
        Out7 <= 8'b11111111;
    else
        Out7 <= 8'b00000000;
    end
endmodule

```



```

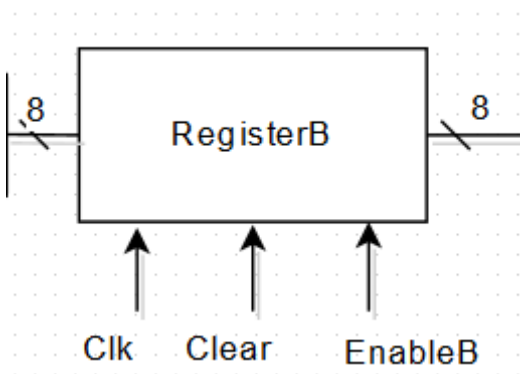
module ShiftRight(
    input [7:0] A,
    output [7:0] Out2
);
assign Out2 = A >> 1;
endmodule

```

```

module ShiftLeft(
    input [7:0] A,
    output [7:0] Out1
);
assign Out1 = A << 1;
endmodule

```



```

module Register8bit(
...
);
always @(posedge CLK)
begin
    if (En)
    begin
        if (CLR)
            Qout <= 8'b00000000;
        else
            Qout <= D;
        end
    end
endmodule

```

Combine Everything Together as Shown in the Diagram

Voila !

Your Very Own Tiny CPU

(The objective here is to learn what are the components of a basic CPU and how to build a CPU by combining them. Just like in a real industrial design, step by step)

You should use the provided test bench in Piazza

Connect the modules explicitly like,

```
Adder M1 (.AAdder(A), .BAdder(B), .Out0Adder(Out0));
```

The easy solution (don't use this for lab 4)