Skeleton codes-

ALU-

```
module alu(Result,DATA1,DATA2,Select);
input [8:0]DATA1,DATA2;
input [3:0] Select;
output [8:0]out;
reg out;
always@(DATA1,DATA2,Select)
begin
case(Select)
//your code here//
endcase
end
endmodule
```

Register File-

```
module regfile8x8a
  (input clk,
   input [3:0] INaddr,
   input [8:0] IN,
   input [3:0] OUT1addr,
   output [8:0] OUT1,
   input [3:0] OUT2addr,
   output [8:0] OUT2);

reg [8:0]              reg0, reg1, reg2, reg3,reg4,reg5,reg6,reg7;

   assign OUT1 = OUT1addr == 0 ? reg0 :
                         OUT1addr == 1 ? reg1 :
                         OUT1addr == 2 ? reg2 :
                         OUT1addr == 3 ? reg3 : 0;
                         // add until 8 //
   assign OUT2 = OUT2addr == 0 ? reg0 :
                         OUT2addr == 1 ? reg1 :
                         OUT2addr == 2 ? reg2 :
                         OUT2addr == 3 ? reg3 : 0;
                         //add until 8//
   always @(negedgeclk) begin
     case(INaddr)
            // your code here
   end // always @ (negedgeclk)
endmodule
```

Control Unit:

```
module CU(instruction, OUT1addr, OUT2addr, INaddr);

Input [31:0] instruction;
Output [2:0] OUT1addr;
Output [2:0] OUT2addr;
Output [2:0] INaddr;

always @(instruction) begin
case(instruction)

//add your code here.
//use the instruction flow given in the last page. Convert them to 32bit binary.
end
end


endmodule
```

Instruction Register-

```
module Instruction_reg (clk, Read_Addr, instruction);

input               clk;
input     [31:0]    Read_Addr;
output    [31:0]    instruction;

// define necessary reg's here//
always @(negedgeclk) begin
          //add your code here//
end

endmodule
```

Program Counter-

```
module counter (clk, reset,Read_addr);

input               clk;
input               reset;
output    [31:0]    Read_addr;

  //      The outputs are defined as registers too
reg       [31:0]    Read_addr;

//        The counter doesn't have any delay since the
//        output is latched when the negedge of the clock happens.

always @(negedgeclk)
          // add code here//
endmodule
```

Instruction flow

Implement the given instruction flow in your processor and see if you get the desired output.

loadi 4, X, 0xFF

loadi 6,X,0xAA

loadi 3,X,0xBB

add 5,6,3

and 1,4,5

or 2,1,6

mov 7,x,2

sub 4,7,3