```verilog
`timescale 1ns / 1ns // `timescale time_unit/time_precision

module ALUwithRegister(input [9:0] SW, input [3:0] KEY, output [6:0]
HEX0,HEX1,HEX2,HEX3,HEX4,HEX5, output [9:0] LEDR);
    wire [3:0] A,B;
    wire [2:0] sel;
    wire clock,reset_b;

    // Input assignment
    assign A = SW[3:0]; // Input data
    assign reset_b = SW[9]; // Reset switch
    // KEY inputs are inverted so should invert it here for proper behaviour
    assign clock = KEY[0]; // Clock input for register
    assign sel = KEY[3:1]; // ALU input selection

    wire [7:0] ALUout,q; // Output for always blocks

    ALU ALU0(.A(A),.B(B),.sel(sel),.ALUoutput(ALUout)); // Instantiate ALU

    flipflop register(.clock(clock),.reset_b(reset_b),.D(ALUout),.Q(q)); // Register

    // Assign lower 4 bits of q to B
    assign B = q[3:0];

    // Output
    assign LEDR[7:0] = q;
    displayHEX h0(.BIN(A),.HEX(HEX0));
    displayHEX h1(.BIN(4'b0000),.HEX(HEX1));
    displayHEX h2(.BIN(4'b0000),.HEX(HEX2));
    displayHEX h3(.BIN(4'b0000),.HEX(HEX3));
    displayHEX h4(.BIN(ALUout[3:0]),.HEX(HEX4));
    displayHEX h5(.BIN(ALUout[7:4]),.HEX(HEX5));
endmodule

// ALU
module ALU(input[3:0] A,B, input[2:0] sel, output[7:0] ALUoutput);
    wire [4:0] adderOut;

    // FUll adder instatiation
    fullAdder4bit fa(.A(A),.B(B),.S(adderOut[3:0]),.Cin(0),.Cout(adderOut[4]));

    reg[7:0] ALUout;
    // ALU
    always @(*)
    begin
        case(sel) // KEY input is inverted so must invert back
            3'b000: // A+B using fullAdder4bit
                ALUout = {3'b000,adderOut};
            3'b001: // A+B using Verilog "+" operator
                ALUout = A+B;
            3'b010: // A NAND B in lower 4 bits, A NOR B in upper 4
                ALUout = {~(A|B),~(A&B)};
            3'b011: // If there is at least one bit that is one output 8'b11000000
                ALUout = (A|B > 0 ? 8'b11000000:8'b00000000);
            3'b100: // If A has exactly 2 1's and B has exactly 3 1's output 8'bb00111111
                ALUout = ((A[0]+A[1]+A[2]+A[3]==2)&(B[0]+B[1]+B[2]+B[3]==3) ?
                8'b00111111:8'b00000000);
            3'b101: // B in most sig 4 bits and A complement in least sig 4 bits
                ALUout = {B,~A};
            3'b110: // A XNOR B in lower 4 bits, A XOR B in upper 4 bits
                ALUout = {A^B,~(A^B)};
            3'b111: // Hold current value of register i.e. do nothing
                ALUout = ALUout;
            default: ALUout = 8'b00000000; // Default just make 0
        endcase
    end

    assign ALUoutput = ALUout;
endmodule
```

```verilog

// Active-low, synchronus reset positive edge triggered flip flop
module flipflop(input clock,reset_b, input[7:0] D, output[7:0] Q);
    reg[7:0] q;

    // Triggered every time clock rises -> on button press
    always @(posedge clock)
    begin
        if(reset_b==0) // Set to 0
            q <= 0;
        else // Pass output of ALU to q
            q <= D;
    end

    assign Q = q;
endmodule

// Full adder for 2 4-bit numbers
module fullAdder4bit(input [3:0] A,B, input Cin, output [3:0] S, output Cout);
    wire C1,C2,C3; // Intermediary wires

    adder bit0(.a(A[0]),.b(B[0]),.s(S[0]),.cin(Cin),.cout(C1));
    adder bit1(.a(A[1]),.b(B[1]),.s(S[1]),.cin(C1),.cout(C2));
    adder bit2(.a(A[2]),.b(B[2]),.s(S[2]),.cin(C2),.cout(C3));
    adder bit3(.a(A[3]),.b(B[3]),.s(S[3]),.cin(C3),.cout(Cout));
endmodule

// Adder bit slice
// digit = a XOR b XOR cin, carry = a*b+a*cin+b*cin
module adder(input a,b,cin, output s,cout);
    assign s = a ^ b ^ cin;
    assign cout = (a&b) | (a&cin) | (b&cin);
endmodule

// Turns decimal to HEX (from lab 2)
module displayHEX(input [3:0] BIN, output [6:0] HEX);
     wire x = BIN[3], y = BIN[2], z = BIN[1], w = BIN[0];
     //assign each segment with appropriate formula
     assign HEX[0] = ~((x|y|z|~w)&(x|~y|z|w)&(~x|y|~z|~w)&(~x|~y|z|~w));
     assign HEX[1] = ~((x|~y|z|~w)&(x|~y|~z|w)&(~x|y|~z|~w)&(~x|~y|z|w)&(~x|~y|~z|w)&(~x|~y|~z|~w));
     assign HEX[2] = ~((x|y|~z|w)&(~x|~y|z|w)&(~x|~y|~z|w)&(~x|~y|~z|~w));
     assign HEX[3] = ~((x|y|z|~w)&(x|~y|z|w)&(x|~y|~z|~w)&(~x|y|z|~w)&(~x|y|~z|w)&(~x|~y|~z|~w));
     assign HEX[4] = ~((x|y|z|~w)&(x|y|~z|~w)&(x|~y|z|w)&(x|~y|z|~w)&(x|~y|~z|~w)&(~x|y|z|~w));
     assign HEX[5] = ~((x|y|z|~w)&(x|y|~z|w)&(x|y|~z|~w)&(x|~y|~z|~w)&(~x|~y|z|~w));
     assign HEX[6] = ~((x|y|z|w)&(x|y|z|~w)&(x|~y|~z|~w)&(~x|~y|z|w));
endmodule

```