```
module fullAdder(A, B, cin, S, cout);
        input A, B, cin;
        output S, cout;
        assign cout = (A & B) | ((A ^ B) & cin);
        assign S = A ^ B ^ cin;
endmodule

module fourBitAdder(A, B, cin, S, cout);
        input [3:0] A;
        input [3:0] B;
        input cin;

        output [3:0] S;
        output cout;

        wire c1;
        wire c2;
        wire c3;

        fullAdder fa1(
                .A(A[0]),
                .B(B[0]),
                .S(S[0]),
                .cin(cin),
                .cout(c1)
        );
        fullAdder fa2(
                .A(A[1]),
                .B(B[1]),
                .S(S[1]),
                .cin(c1),
                .cout(c2)
        );
        fullAdder fa3(
                .A(A[2]),
                .B(B[2]),
                .S(S[2]),
                .cin(c2),
                .cout(c3)
        );
        fullAdder fa4(
                .A(A[3]),
                .B(B[3]),
                .S(S[3]),
                .cin(c3),
                .cout(cout)
        );
endmodule

module alublock(ALUout, A, B, f);
```

```
        output reg [7:0] ALUout;
        input [3:0] A, B;
        input [2:0] f;

        wire [3:0] ab1;
        wire ab2;
endmodule

module sevenhex(hex, in);
    input [3:0] in;
    output [6:0] hex;

    assign hex[0] = (~in[0] & in[1] & ~in[2] & ~in[3]) | (in[0] &
~in[1] & in[2] & in[3]) | (in[0] & in[1] & ~in[2] & in[3]) | (~in[0] &
~in[1] & ~in[2] & in[3]);

    assign hex[1] = (in[0] & in[2] & in[3]) | (in[0] & in[1] & ~in[3])
| (~in[0] & in[1] & ~in[2] & in[3]) | (in[1] & in[2] & ~in[3]);

    assign hex[2] = (~in[0] & ~in[1] & in[2] & ~in[3]) | (in[0] &
in[1] & ~in[3]) | (in[0] & in[1] & in[2]);

    assign hex[3] = (~in[1] & ~in[2] & in[3]) | (in[0] & ~in[1] &
in[2] & ~in[3]) | (in[1] & in[2] & in[3]) | (~in[0] & in[1] & ~in[2] &
~in[3]);

    assign hex[4] = (~in[1] & ~in[2] & in[3]) | (~in[0] & in[1] &
~in[2]) | (~in[0] & in[3]);

    assign hex[5] = (~in[0] & ~in[1] & in[2]) | (~in[0] & ~in[1] &
in[3]) | (in[0] & in[1] & ~in[2] & in[3]) | (~in[0] & in[2] & in[3]);

    assign hex[6] = (~in[0] & in[1] & in[2] & in[3]) | (in[0] & in[1]
& ~in[2] & ~in[3]) | (~in[0] & ~in[1] & ~in[2]);

endmodule

module innerAlu(A, B, func, ALUout);
        input [3:0] A, B;
        input [2:0] func;
        output [7:0] ALUout;

        reg [7:0] Out;
        wire dummy1;
        wire dummy2;

        wire [3:0] w1;
        fourBitAdder
f1(.A(A[3:0]), .B(4'b0001), .cin(1'b0), .S(w1), .cout(dummy1));
```

```
        wire [3:0] w2;
        fourBitAdder
f2(.A(A[3:0]), .B(B[3:0]), .cin(1'b0), .S(w2), .cout(dummy2));


        always @(*)
        begin
                case (func[2:0])
                        3'b000: Out = {4'b0000, w1};
                        3'b001: Out = {4'b0000, w2};
                        3'b010: Out = {4'b0000, A[3:0] + B[3:0]};
                        3'b011: Out = {A[3:0] | B[3:0], A[3:0] ^
B[3:0]};
                        3'b100: Out = {7'b0000000, | {A[3:0],
B[3:0]}};
                        3'b101: Out = {A[3:0], B[3:0]};
                        default: Out = 8'b0000_0000;
                endcase
        end

        assign ALUout[7:0] = Out;

endmodule

module alu(SW, LEDR, KEY, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5);
        input [7:0] SW;
        input [2:0] KEY;
        output [7:0] LEDR;
        output [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;

        wire [3:0] A, B;
        wire [7:0] out;

        innerAlu
alu1(.A(SW[7:4]), .B(SW[3:0]), .func(KEY[2:0]), .ALUout(LEDR[7:0]));

        sevenhex h1(.in(SW[3:0]), .hex(HEX0));
        sevenhex h2(.in(SW[7:4]), .hex(HEX2));
        sevenhex h3(.in(4'b0000), .hex(HEX1));
        sevenhex h4(.in(4'b0000), .hex(HEX3));
        sevenhex h5(.in(LEDR[3:0]), .hex(HEX4));
        sevenhex h6(.in(LEDR[7:4]), .hex(HEX5));

endmodule
```