```
module alu( A, B, ALU_Sel, ALU_Result);
  input [7:0] A,B; // ALU 8-bit Inputs
  input [3:0] ALU_Sel;// ALU Selection
  output [8:0] ALU_Result;// ALU 8-bit Result
  reg [8:0] ALU_Result;

always @(*) begin
  case(ALU_Sel)
4'b0000:ALU_Result = A + B ; // Addition
4'b0001:ALU_Result = A - B ; // Subtraction
//4'b0010:ALU_Result = A * B; // Multiplication
4'b0011: ALU_Result = A/B; // Division
4'b0100: ALU_Result = A<<1;// Logical shift left
4'b0101: ALU_Result = A>>1;// Logical shift right
4'b0110: ALU_Result = {A[6:0],A[7]};// Rotate left
4'b0111: ALU_Result = {A[0],A[7:1]}; // Rotate right
4'b1000: ALU_Result = A & B; // Logical and
4'b1001: ALU_Result = A | B;// Logical or
4'b1010: ALU_Result = A ^ B;// Logical xor
4'b1011: ALU_Result = ~(A | B);// Logical nor
4'b1100: ALU_Result = ~(A & B); // Logical nand
4'b1101: ALU_Result = ~(A ^ B); // Logical xnor
4'b1110: ALU_Result = (A>B)?8'd1:8'd0 ; // Greater comparison
4'b1111: ALU_Result = (A==B)?8'd1:8'd0 ;// Equal comparison
  default: ALU_Result = 0;
endcase
end
endmodule
```