

Kevin Hoser and Alex Schendel

Project 3

27 March 2020

EE 435

DFF w/ Enable (1 bit):

```
module DFF_EN(q, clk, d, reset, en);
```

```
output wire q;
```

```
input  d, clk, reset, en;
```

```
wire dp;
```

```
mux2_1 mux(dp, q, d, en);
```

```
DFF state(q, clk, dp, reset);
```

```
endmodule
```

```
module DFF(q, clk, d, reset);
```

```
output q;
```

```
input  d, clk, reset;
```

```
reg    q;
```

```
// always @(negedge reset or posedge clk)
```

```
always @(negedge reset or posedge clk)
```

```
if (~reset)
```

```
q <= 1'b0;
```

```
else
```

```
q <= d;
```

```
endmodule
```

```
module mux2_1 (z, i0, i1, s);
```

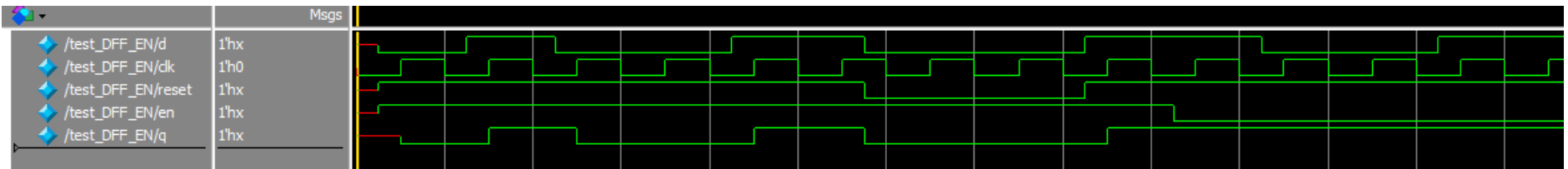
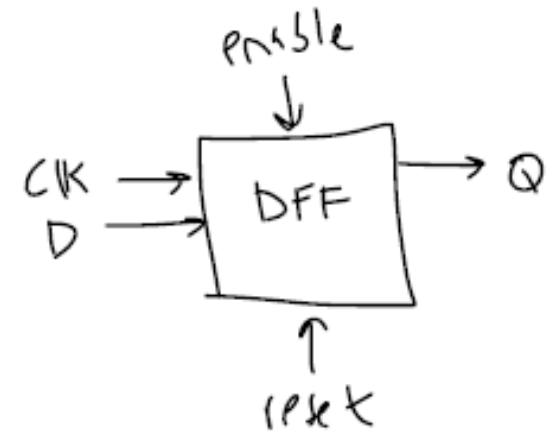
```
output z;
```

```
input i0, i1, s;
```

```
bufif1 (z, i1, s);
```

```
bufif0 (z, i0, s);
```

```
endmodule
```



Register (32 bit):

```
module register32 (q, clk, d, en);
```

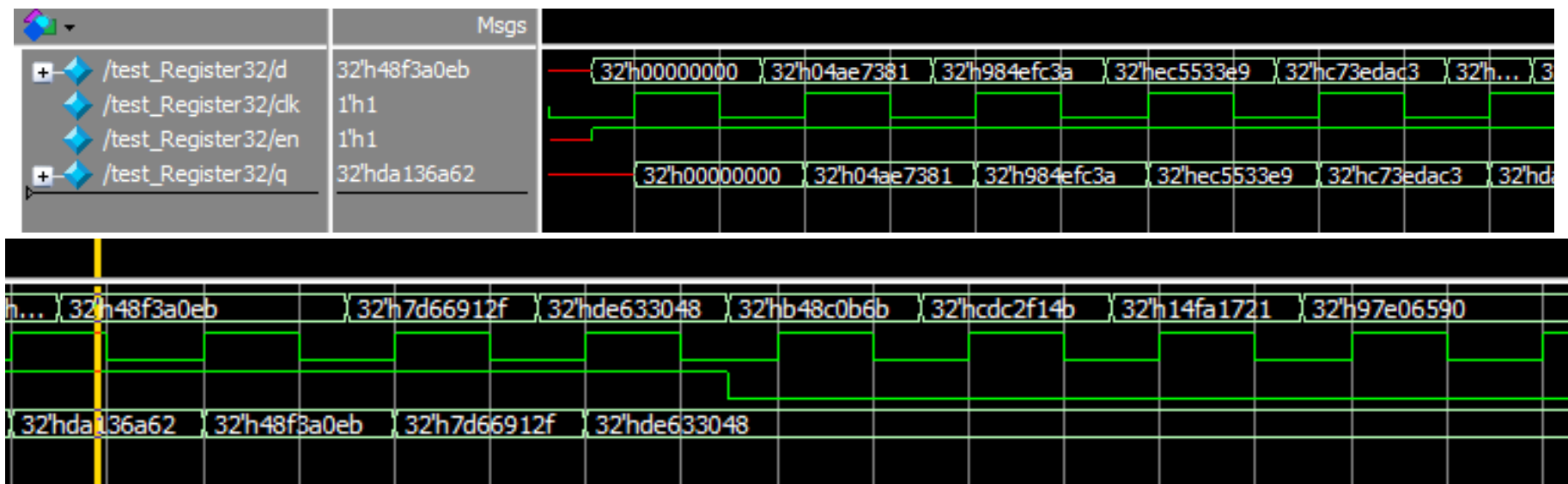
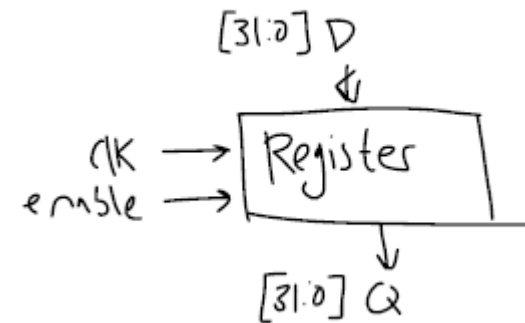
```
output [31:0] q;  
input [31:0] d;  
input clk, en;
```

```

DFF_EN d1[31:0] (q, clk, d, , en);

```

endmodule



32:1 MUX (32 bit):

```
module mux32_32 (z, i0, i1, i2, i3, i4, i5, i6, i7,
                 i8, i9, i10, i11, i12, i13, i14,
                 i15,
                 i16, i17, i18, i19, i20, i21, i22,
                 i23,
                 i24, i25, i26, i27, i28, i29, i30,
                 i31, s);
output [31:0] z;
input [31:0] i0, i1, i2, i3, i4, i5, i6, i7,
            i8, i9, i10, i11, i12, i13, i14, i15,
            i16, i17, i18, i19, i20, i21, i22, i23,
            i24, i25, i26, i27, i28, i29, i30, i31;
input [4:0] s;

wire [31:0] a, b;

mux16_32 a1(a, i0, i1, i2, i3, i4, i5, i6, i7,
            i8, i9, i10, i11, i12, i13, i14, i15,
            s[3:0]);
mux16_32 a2(b, i16, i17, i18, i19, i20, i21, i22, i23,
            i24, i25, i26, i27, i28, i29, i30, i31,
            s[3:0]);
mux2_32 a3(z, a, b, s[4]);

endmodule

module mux16_32 (z, i0, i1, i2, i3, i4, i5, i6, i7,
                 i8, i9, i10, i11, i12, i13, i14,
                 i15, s);
output [31:0] z;
input [31:0] i0, i1, i2, i3, i4, i5, i6, i7,
            i8, i9, i10, i11, i12, i13, i14, i15;
input [3:0] s;

wire [31:0] a, b;

mux8_32 a1(a, i0, i1, i2, i3, i4, i5, i6, i7, s[2:0]);
mux8_32 a2(b, i8, i9, i10, i11, i12, i13, i14, i15,
            s[2:0]);
```

```
mux2_32 a3(z, a, b, s[3]);

endmodule

module mux8_32 (z, i0, i1, i2, i3, i4, i5, i6, i7, s);
output [31:0] z;
input [31:0] i0, i1, i2, i3, i4, i5, i6, i7;
input [2:0] s;

wire [31:0] a, b;

mux4_32 a1(a, i0, i1, i2, i3, s[0], s[1]);
mux4_32 a2(b, i4, i5, i6, i7, s[0], s[1]);
mux2_32 a3(z, a, b, s[2]);

endmodule

module mux4_32 (z, i0, i1, i2, i3, s0, s1);
output [31:0] z;
input [31:0] i0, i1, i2, i3;
input s0, s1;

wire [31:0] a, b;

mux2_32 a1(a, i0, i1, s0);
mux2_32 a2(b, i2, i3, s0);
mux2_32 a3(z, a, b, s1);

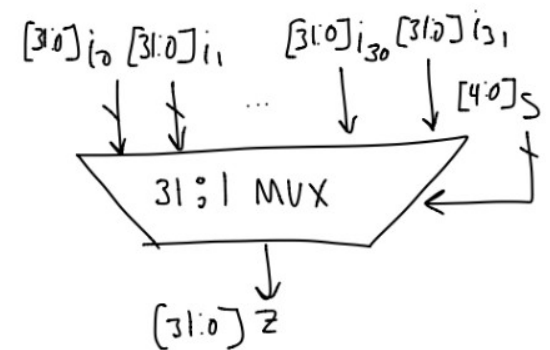
endmodule
```

```
bufif1 buf0[31:0] (z, i1, s);
bufif0 buf1[31:0] (z, i0, s);
```

```
output [31:0] z;
input [31:0] i0, i1;
input s;
```

endmodule

	Mags															
+ /test_mux32_32/z	32h00000000	32h00000000	32h00000001	32h00000002	32h00000003	32h00000004	32h00000005	32h00000006	32h00000007	32h00000008	32h00000009	32h0000000a	32h0000000b	32h0000000c	32h0000000e	32h0000000f
+ /test_mux32_32/0	32h00000000															
+ /test_mux32_32/i1	32h00000001															
+ /test_mux32_32/i2	32h00000002															
+ /test_mux32_32/i3	32h00000003															
+ /test_mux32_32/i4	32h00000004															
+ /test_mux32_32/i5	32h00000005															
+ /test_mux32_32/i6	32h00000006															
+ /test_mux32_32/i7	32h00000007															
+ /test_mux32_32/i8	32h00000008															
+ /test_mux32_32/i9	32h00000009															
+ /test_mux32_32/i10	32h0000000a															
+ /test_mux32_32/i11	32h0000000b															
+ /test_mux32_32/i12	32h0000000c															
+ /test_mux32_32/i13	32h0000000d															
+ /test_mux32_32/i14	32h0000000e															
+ /test_mux32_32/i15	32h0000000f															
+ /test_mux32_32/i16	32h00000010															
+ /test_mux32_32/i17	32h00000011															
+ /test_mux32_32/i18	32h00000012															
+ /test_mux32_32/i19	32h00000013															
+ /test_mux32_32/i20	32h00000014															
+ /test_mux32_32/i21	32h00000015															
+ /test_mux32_32/i22	32h00000016															
+ /test_mux32_32/i23	32h00000017															
+ /test_mux32_32/i24	32h00000018															
+ /test_mux32_32/i25	32h00000019															
+ /test_mux32_32/i26	32h0000001a															
+ /test_mux32_32/i27	32h0000001b															
+ /test_mux32_32/i28	32h0000001c															
+ /test_mux32_32/i29	32h0000001d															
+ /test_mux32_32/i30	32h0000001e															
+ /test_mux32_32/i31	32h0000001f															
+ /test_mux32_32/s	5h00	5h00	5h01	5h02	5h03	5h04	5h05	5h06	5h07	5h08	5h09	5h0a	5h0b	5h0c	5h0e	5h0f



5:32 Decoder (1 bit):

```
output [31:0] d;
input [4:0] i;
input en;

wire i4_not;
wire [31:0] e;

not a1(i4_not, i[4]);

decoder4_16 a2(e[15:0], i[3:0], i4_not);
decoder4_16 a3(e[31:16], i[3:0], i[4]);

and a4(d[0], e[0], en);
and a5(d[1], e[1], en);
and a6(d[2], e[2], en);
and a7(d[3], e[3], en);
and a8(d[4], e[4], en);
and a9(d[5], e[5], en);
and a10(d[6], e[6], en);
and a11(d[7], e[7], en);
and a12(d[8], e[8], en);
and a13(d[9], e[9], en);
and a14(d[10], e[10], en);
and a15(d[11], e[11], en);
and a16(d[12], e[12], en);
and a17(d[13], e[13], en);
and a18(d[14], e[14], en);
and a19(d[15], e[15], en);

and a20(d[16], e[16], en);
and a21(d[17], e[17], en);
and a22(d[18], e[18], en);
and a23(d[19], e[19], en);
and a24(d[20], e[20], en);
and a25(d[21], e[21], en);
and a26(d[22], e[22], en);
and a27(d[23], e[23], en);
and a28(d[24], e[24], en);
and a29(d[25], e[25], en);
```

```
and a30(d[26], e[26], en);
and a31(d[27], e[27], en);
and a32(d[28], e[28], en);
and a33(d[29], e[29], en);
and a34(d[30], e[30], en);
and a35(d[31], e[31], en);

endmodule

module decoder4_16 (d, i, en);

output [15:0] d;
input [3:0] i;
input en;

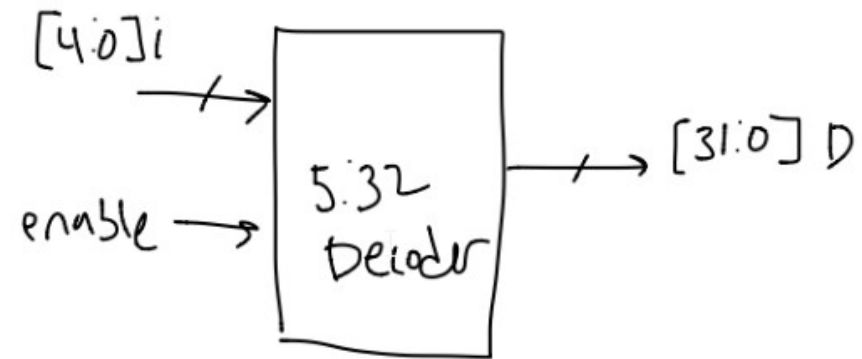
wire i3_not;
wire [15:0] e;

not a1(i3_not, i[3]);

decoder3_8 a2(e[7:0], i[2:0], i3_not);
decoder3_8 a3(e[15:8], i[2:0], i[3]);

and a4(d[0], e[0], en);
and a5(d[1], e[1], en);
and a6(d[2], e[2], en);
and a7(d[3], e[3], en);
and a8(d[4], e[4], en);
and a9(d[5], e[5], en);
and a10(d[6], e[6], en);
and a11(d[7], e[7], en);
and a12(d[8], e[8], en);
and a13(d[9], e[9], en);
and a14(d[10], e[10], en);
and a15(d[11], e[11], en);
and a16(d[12], e[12], en);
and a17(d[13], e[13], en);
and a18(d[14], e[14], en);
and a19(d[15], e[15], en);
```

5h06	5h07	5h08	5h09	5h0a	5h0b	5h0c	5h0d
32h00000040	32h00000080	32h00000100	32h00000200	32h00000400	32h00000800	32h00001000	32h00002000



Register File (32 bit):

```
module registerFile32 (a, b, clk, d, w_sel, w_en, a_sel, b_sel);
    output [31:0] a, b;
    input [31:0] d;
    input clk, w_en;
    input [4:0] w_sel, a_sel, b_sel;

    wire [31:0] q [31:0];
    wire [31:0] en_int;

    decoder5_32 enable_select (en_int, w_sel, w_en);
    register32 reg0 (q[0], clk, d, en_int[0]);
    register32 reg1 (q[1], clk, d, en_int[1]);
    register32 reg2 (q[2], clk, d, en_int[2]);
    register32 reg3 (q[3], clk, d, en_int[3]);
    register32 reg4 (q[4], clk, d, en_int[4]);
    register32 reg5 (q[5], clk, d, en_int[5]);
    register32 reg6 (q[6], clk, d, en_int[6]);
    register32 reg7 (q[7], clk, d, en_int[7]);
    register32 reg8 (q[8], clk, d, en_int[8]);
    register32 reg9 (q[9], clk, d, en_int[9]);
    register32 reg10 (q[10], clk, d, en_int[10]);
    register32 reg11 (q[11], clk, d, en_int[11]);
    register32 reg12 (q[12], clk, d, en_int[12]);
    register32 reg13 (q[13], clk, d, en_int[13]);
    register32 reg14 (q[14], clk, d, en_int[14]);
    register32 reg15 (q[15], clk, d, en_int[15]);
    register32 reg16 (q[16], clk, d, en_int[16]);
    register32 reg17 (q[17], clk, d, en_int[17]);
    register32 reg18 (q[18], clk, d, en_int[18]);
    register32 reg19 (q[19], clk, d, en_int[19]);
    register32 reg20 (q[20], clk, d, en_int[20]);
    register32 reg21 (q[21], clk, d, en_int[21]);
    register32 reg22 (q[22], clk, d, en_int[22]);
    register32 reg23 (q[23], clk, d, en_int[23]);
    register32 reg24 (q[24], clk, d, en_int[24]);
    register32 reg25 (q[25], clk, d, en_int[25]);
    register32 reg26 (q[26], clk, d, en_int[26]);
    register32 reg27 (q[27], clk, d, en_int[27]);
    register32 reg28 (q[28], clk, d, en_int[28]);
    register32 reg29 (q[29], clk, d, en_int[29]);
    register32 reg30 (q[30], clk, d, en_int[30]);
    register32 reg31 (q[31], clk, d, en_int[31]);

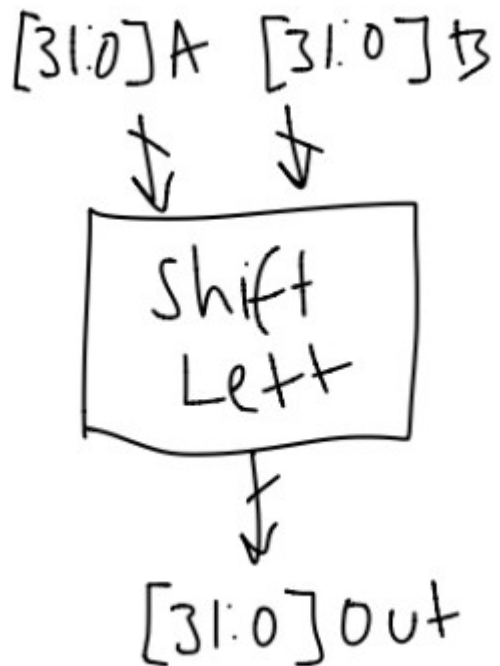
    mux32_32 muxA (a, q[0], q[1], q[2], q[3], q[4], q[5],
        q[6], q[7], q[8], q[9], q[10], q[11], q[12],
        q[13], q[14], q[15], q[16], q[17], q[18], q[19], q[20],
        q[21], q[22], q[23], q[24], q[25], q[26], q[27], q[28],
        q[29], q[30], q[31], a_sel);
    mux32_32 muxB (b, q[0], q[1], q[2], q[3], q[4], q[5],
        q[6], q[7], q[8], q[9], q[10], q[11], q[12],
        q[13], q[14], q[15], q[16], q[17], q[18], q[19], q[20],
        q[21], q[22], q[23], q[24], q[25], q[26], q[27], q[28],
        q[29], q[30], q[31], b_sel);

endmodule
```


Shift Left (32 bit):

```
module shiftLeft32 (out, a, b);  
  
output [31:0] out;  
input [31:0] a, b;  
  
assign out = a << b;  
  
endmodule
```

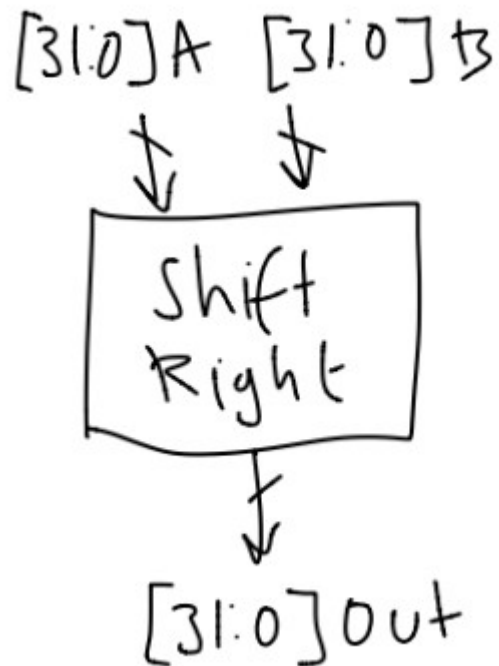
	Msgs										
+ /test_shiftLeft32/out	32'h...	32'h22222220	32'h00000000	32'h00000100	32'h00010000	32'h00000000					
+ /test_shiftLeft32/a	32'h...	32'h11111110	32'hfffffffe	32'h00000001	32'h00010000						
+ /test_shiftLeft32/b	32'h...	32'h00000001	32'h0000001f	32'h00000008	32'h00000000	32'h0000001f					



Shift Right (32 bit):

```
module shiftRight32 (out, a, b);  
  
output [31:0] out;  
input [31:0] a, b;  
  
assign out = a >> b;  
  
endmodule
```

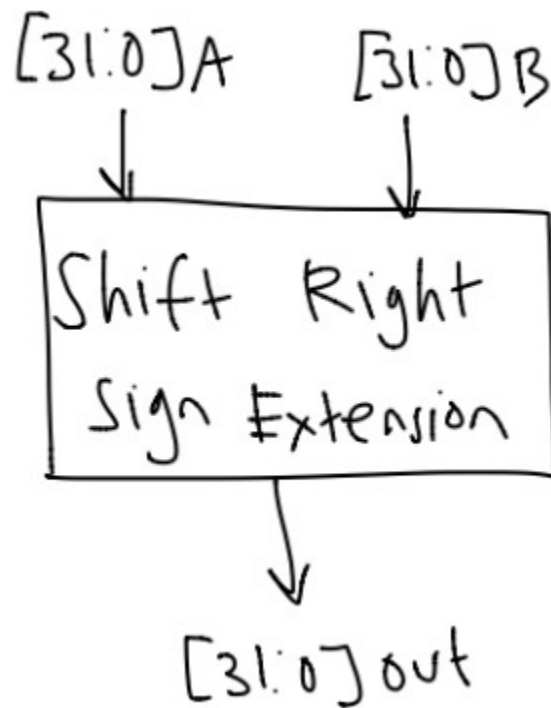
		Msgs						
+ /test_shiftRight32Arithmetic/out	32'h08888888	32'h08888888	32'hffffffff	32'h00000000	32'h00010000	32'h00000000	32'hffff0002	32'hffffe002
+ /test_shiftRight32Arithmetic/a	32'h11111110	32'h11111110	32'hffffffffffe	32'h00000001	32'h00010000		32'h80010000	32'hf0010000
+ /test_shiftRight32Arithmetic/b	32'h00000001	32'h00000001	32'h0000001f	32'h00000008	32'h00000000	32'h0000001f	32'h0000000f	



Shift Right with Sign Extension (32 bit):

```
module shiftRight32Arithmetic (out, a, b);  
  
output [31:0] out;  
input signed [31:0] a, b;  
  
assign out = a >>> b;  
  
endmodule
```

		Msgs							
+ /test_shiftRight32Arithmetic/out	32'h08888888	32'h08888888	32'hffffff	32'h00000000	32'h00010000	32'h00000000	32'hffff0002	32'hffffe002	
+ /test_shiftRight32Arithmetic/a	32'h11111110	32'h11111110	32'hfffffffe	32'h00000001	32'h00010000		32'h80010000	32'hf0010000	
+ /test_shiftRight32Arithmetic/b	32'h00000001	32'h00000001	32'h0000001f	32'h00000008	32'h00000000	32'h0000001f	32'h0000000f		



ALU (32 bit):

```

module ALU(out, a, b, op);

output [31:0] out;
input [31:0] a, b;
input [3:0] op;

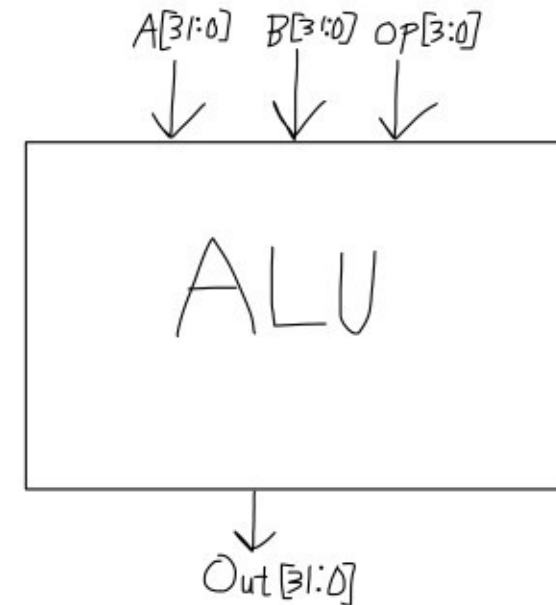
wire [31:0] add, sub, my_and, my_or, my_xor, my_xnor,
            my_shl, my_shr, my_sra;

full_adder_32 adder(, add, a, b, 1'b0);
full_subtractor_32 subber(, sub, a, b, 1'b0);
and32 ander(my_and, a, b);
or32 orer(my_or, a, b);
xor32 xorer(my_xor, a, b);
xnor32 xnorer(my_xnor, a, b);
shiftLeft32 shiftLeftter(my_shl, a, b);
shiftRight32 shiftRighter(my_shr, a, b);
shiftRight32Arithmetic shiftRightAer(my_sra, a, b);

mux16_32 muxer(out, add, sub, , , , , ,
              my_and, my_or, my_xor, my_xnor, my_shl, my_shr, my_sra, , op);

endmodule

```



Msgs									
/ALU_tb/out	32'hxxxxxxxx			32'hde3cc3ed	32'ha7c42e34	32'h10101000	32'h1dcfebdd	32'hb1d513c4	
/ALU_tb/a	32'hxxxxxxxx			32'hbc157222	32'h00000000	32'h10101010	32'h0189abdd	32'he9eec208	
/ALU_tb/b	32'hxxxxxxxx			32'h222751cb	32'h583bd1cc		32'h1dc74a94	32'h583bd1cc	
/ALU_tb/op	4'hx			4'h0	4'h1	4'h8	4'h9	4'ha	

32'hc08d7f59						32'hffffeff0	32'h007fffbf	32'hffffeffe	
32'h1fbc8148	32'h2416e099	32'h687f3b5a	32'h555984ab	32'hffffefff	32'hffff7fff	32'h80000000	32'h23337af9		
32'h20ce01ee	32'h3ef9d5ec	32'h71252c6f	32'h6c886316	32'h00000004	32'h00000009	32'h0000001e	32'h89bfe491		
4'hb	4'h2	4'h3	4'h4	4'hc	4'hd	4'he	4'hf		