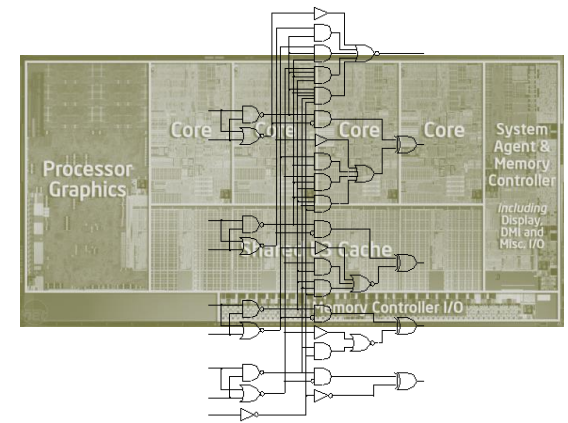




고려대학교
KOREA UNIVERSITY



COSE221, COMP211 Logic Design

Lecture 8. ALU, Shifter, Counter, Shift Register

Prof. Taeweon Suh

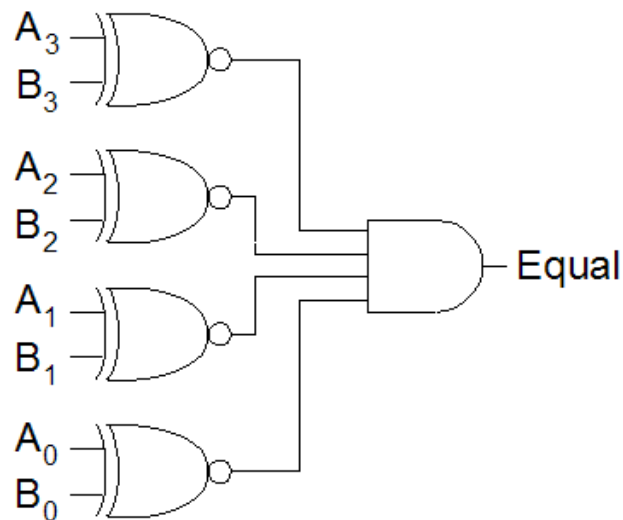
Computer Science & Engineering

Korea University

Comparator

- Comparator determines whether two binary numbers are equal or if one is greater or less than the other
- An equality comparator produces a single output indicating whether A is equal to B

Example: 4-bit equality comparator

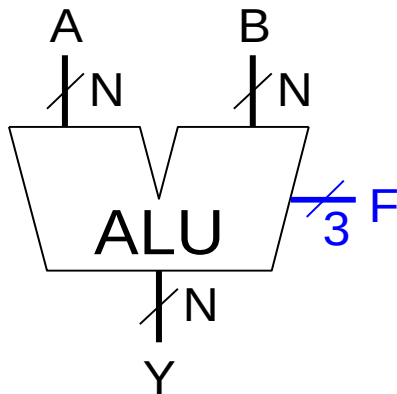


General Comparison

- From the subtraction slides, we know that we can do comparison of unsigned numbers and signed numbers by checking flags (N, C, Z, V) after subtraction
 - N is set to MSB of the result
 - C is set when there is an end carry-out
 - Z is set when the result is zero
 - V is set if signed overflow occurs
- **Unsigned** number comparison
 - CPU does $A - B$
 - If C (carry-out) is 1, then $A \geq B$
 - If C (carry-out) is 0, then $A < B$
- **Signed** number comparison
 - CPU does $A - B$
 - If $(V == N)$, then $A \geq B$
 - If $(V \neq N)$, then $A < B$

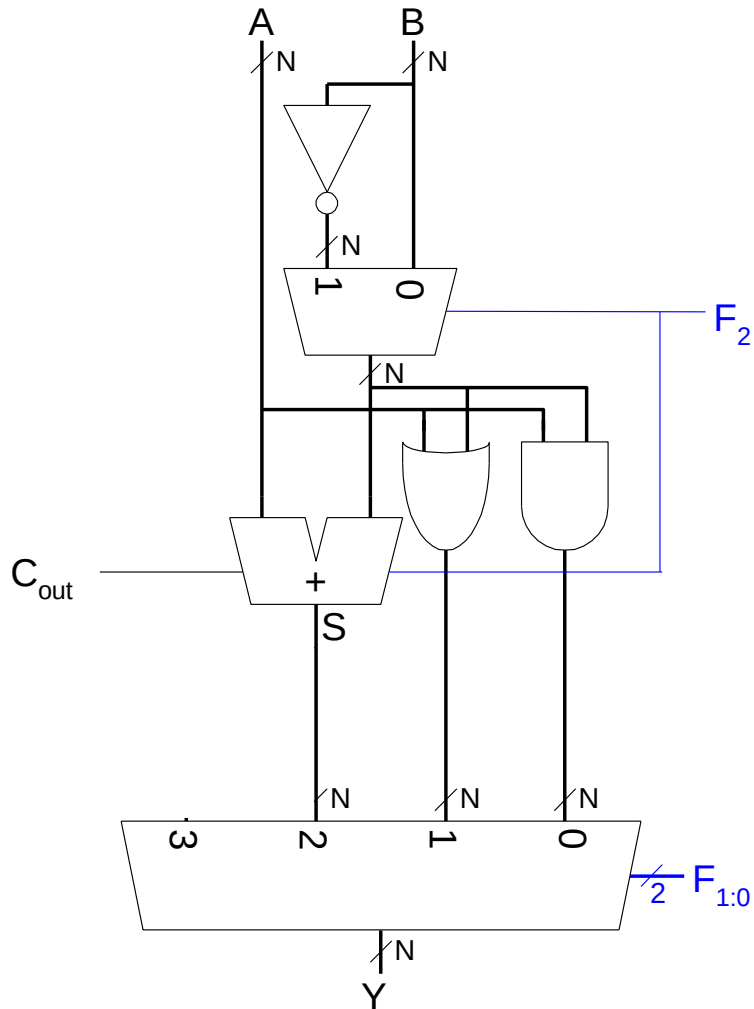
Arithmetic Logic Unit (ALU)

- ALU is a digital circuit that performs arithmetic and logical operations
- ALU is a fundamental building block of CPU (Central Processing Unit) in computers



$F_{2:0}$	Function
000	$A \& B$
001	$A \mid B$
010	$A + B$
011	not used
100	$A \& \sim B$
101	$A \mid \sim B$
110	$A - B$
111	not used

ALU Design Example



$F_{2:0}$	Function
000	A & B
001	A B
010	A + B
011	not used
100	A & $\sim B$
101	A $\sim B$
110	A - B
111	not used

Basic Shifting

- Shift types
 - Logical (or unsigned) shift
 - Arithmetic (or signed) shift
- Shift directions
 - Left (multiply by powers of 2)
 - Right (divide by powers of 2)
 - Take floor value if the result is not an integer
 - Floor value of X (or $\lfloor X \rfloor$) is the greatest integer less than or equal to X
 - $\lfloor 5/2 \rfloor = 2$
 - $\lfloor -3/2 \rfloor = -2$

Logical Shift

- Logical shift left
 - MSB: shifted out
 - LSB: shifted in with a 0
 - Examples:
 - $(11001011 \ll 1) = 10010110$
 - $(11001011 \ll 3) = 01011000$
- Logical shift right
 - MSB: shifted in with a 0
 - LSB: shifted out
 - Examples:
 - $(11001011 \gg 1) = 01100101$
 - $(11001011 \gg 3) = 00011001$
- Logic shifts are useful to perform multiplication or division of **unsigned** integer by powers of two
 - Logical shift right takes floor value if the result is not integer

Arithmetic Shift

- Arithmetic shift left
 - MSB: shifted out, however, be aware of overflow/underflow
 - LSB: shifted in with a 0
 - Examples:
 - $(1100 \ll 1) = 1000$
 - $(1100 \ll 3) = 0000$ (Incorrect!) $\Rightarrow$ Underflow
- Arithmetic shift right
 - MSB: Retain its sign bit
 - LSB: Shifted out
 - Examples:
 - $(1100 \gg 1) = 1110$ (Retain sign bit)
 - $(1100 \gg 3) = 1111$ ($\lfloor -4/8 \rfloor = -1$) $\Rightarrow$ Floor value of -0.5
- Arithmetic shifts can be useful as efficient ways of performing multiplication or division of **signed** integers by powers of two
 - Arithmetic shift right takes floor value if the result is not integer

Examples of Arithmetic Shift

1111 1011 Arithmetic shift **right** by 1
→ **1**111 1101

1111 1011 Arithmetic shift **left** by 1
→ 1111 011**0**

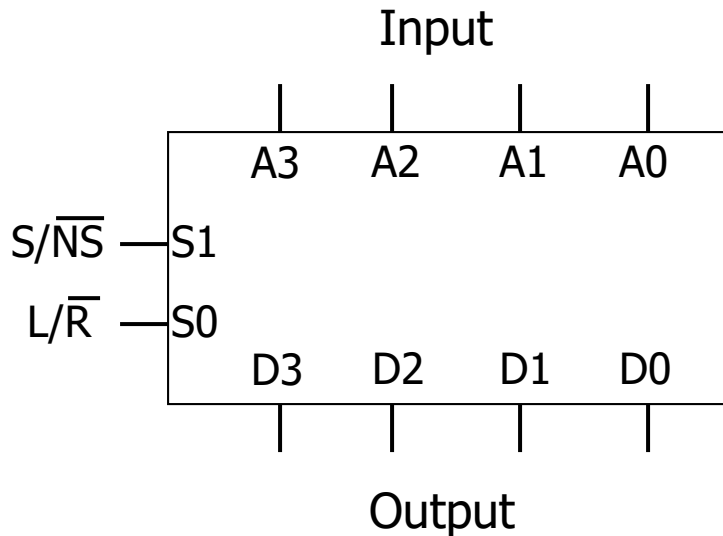
1011 1111 (= -65) Arithmetic shift **left** by 1 (i.e. x2)
→ 0111 111**0** (= +126 ≠ -130) ⇒ Underflow !

Overflow/Underflow



0100 0010 (= +66) Arithmetic shift **left** by 1 (i.e. x2)
→ 1000 010**0** (= -124 ≠ +132) ⇒ Overflow !

4-bit Logical Shifter



S1	S0	D3	D2	D1	D0
0	X	A3	A2	A1	A0
1	0	0	A3	A2	A1
1	1	A2	A1	A0	0

Legend	S: Shift	L: Left
	NS: No Shift	R: Right

$$D_3 = \overline{S_1}A_3 + S_1S_0A_2$$

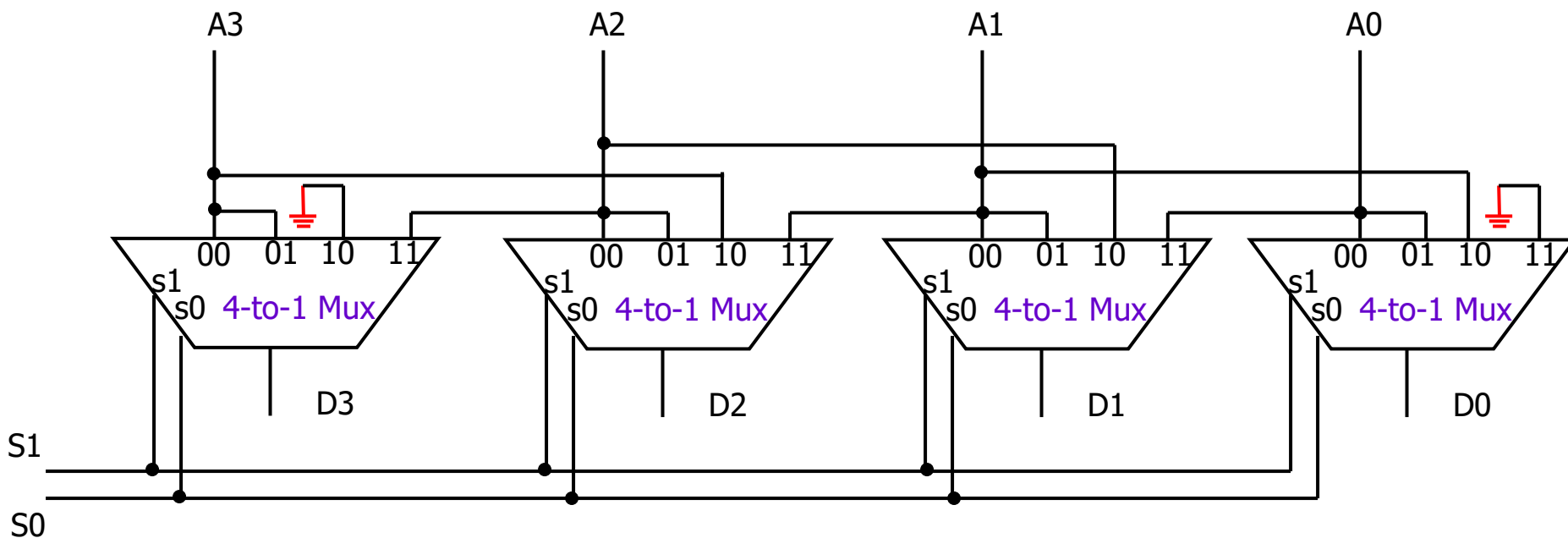
$$D_2 = \overline{S_1}A_2 + S_1\overline{S_0}A_3 + S_1S_0A_1$$

$$D_1 = \overline{S_1}A_1 + S_1\overline{S_0}A_2 + S_1S_0A_0$$

$$D_0 = \overline{S_1}A_0 + S_1\overline{S_0}A_1$$

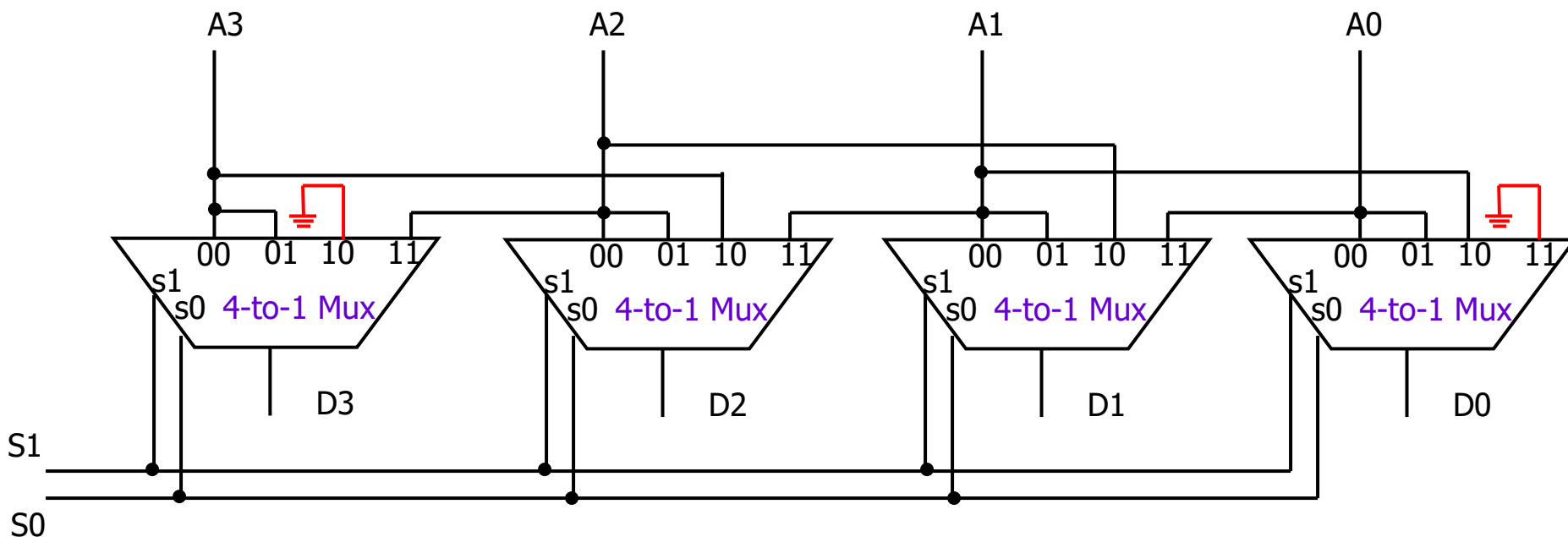
4-bit Logical Shifter using 4:1 Mux

	S1	S0	D3	D2	D1	D0
	0	X	A3	A2	A1	A0
Right Shift →	1	0	0	A3	A2	A1
Left Shift →	1	1	A2	A1	A0	0

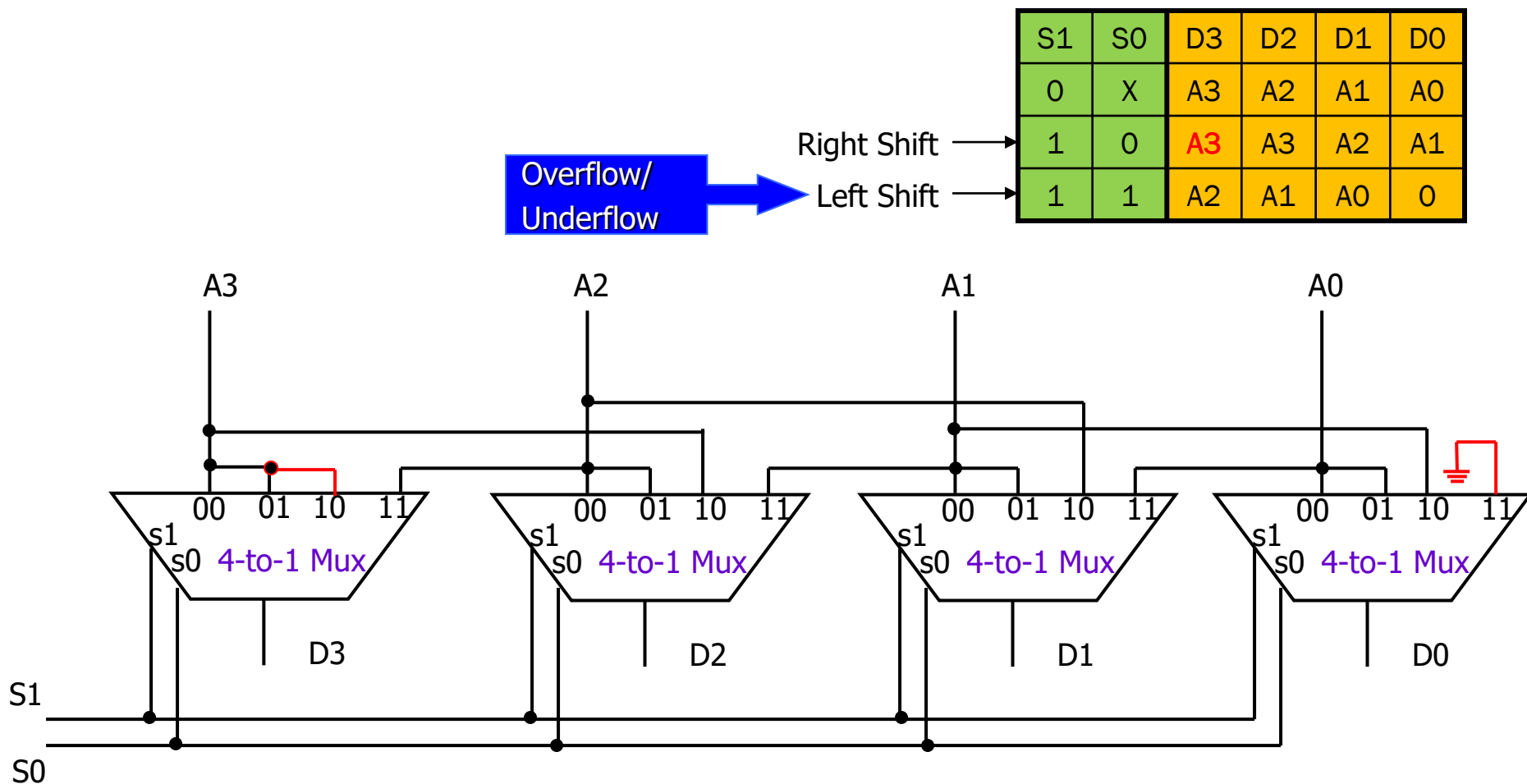


4-bit Arithmetic Shifter using 4:1 Mux

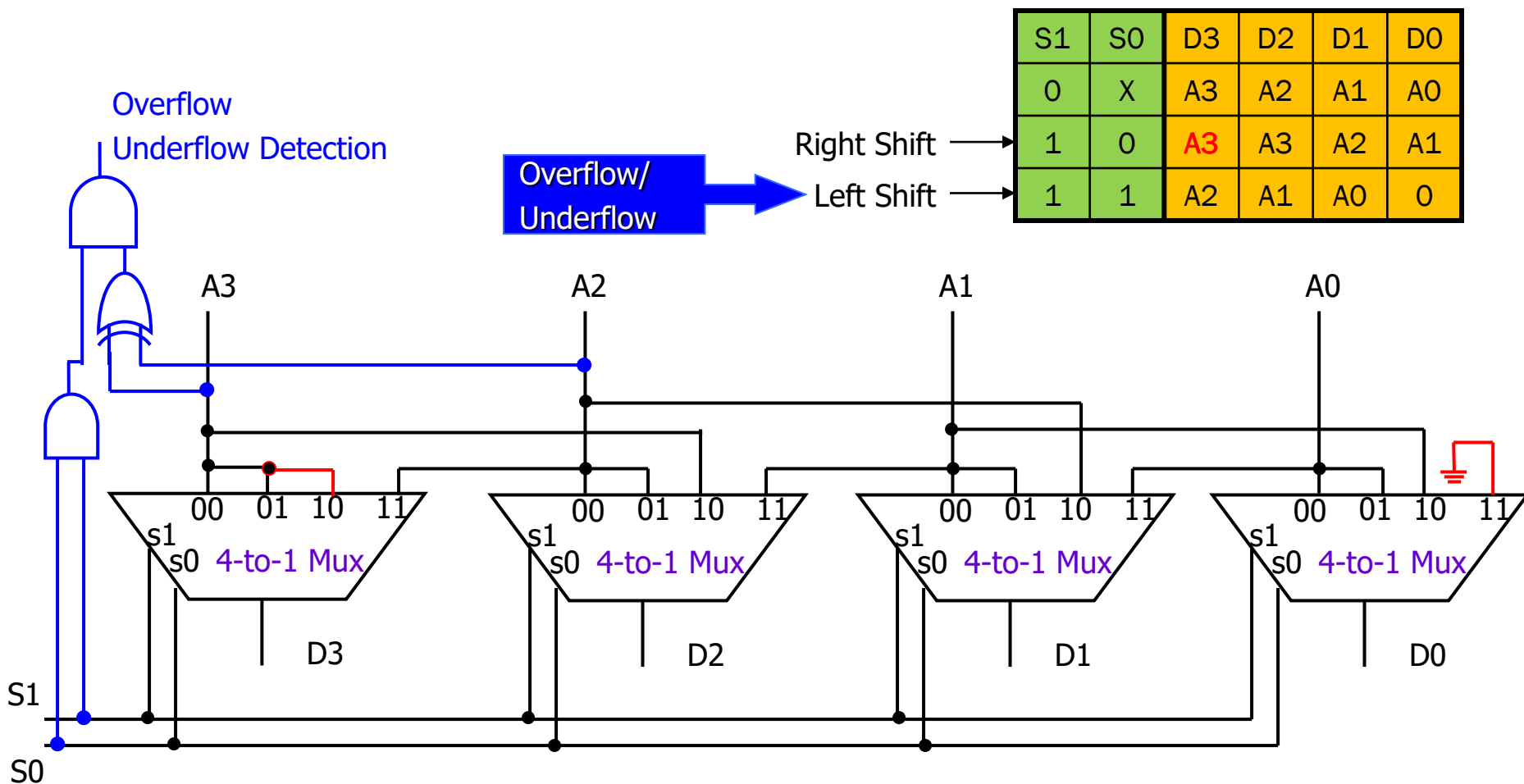
	S1	S0	D3	D2	D1	D0
	0	X	A3	A2	A1	A0
Right Shift →	1	0	A3	A3	A2	A1
Left Shift →	1	1	A2	A1	A0	0



4-bit Arithmetic Shifter using 4:1 Mux



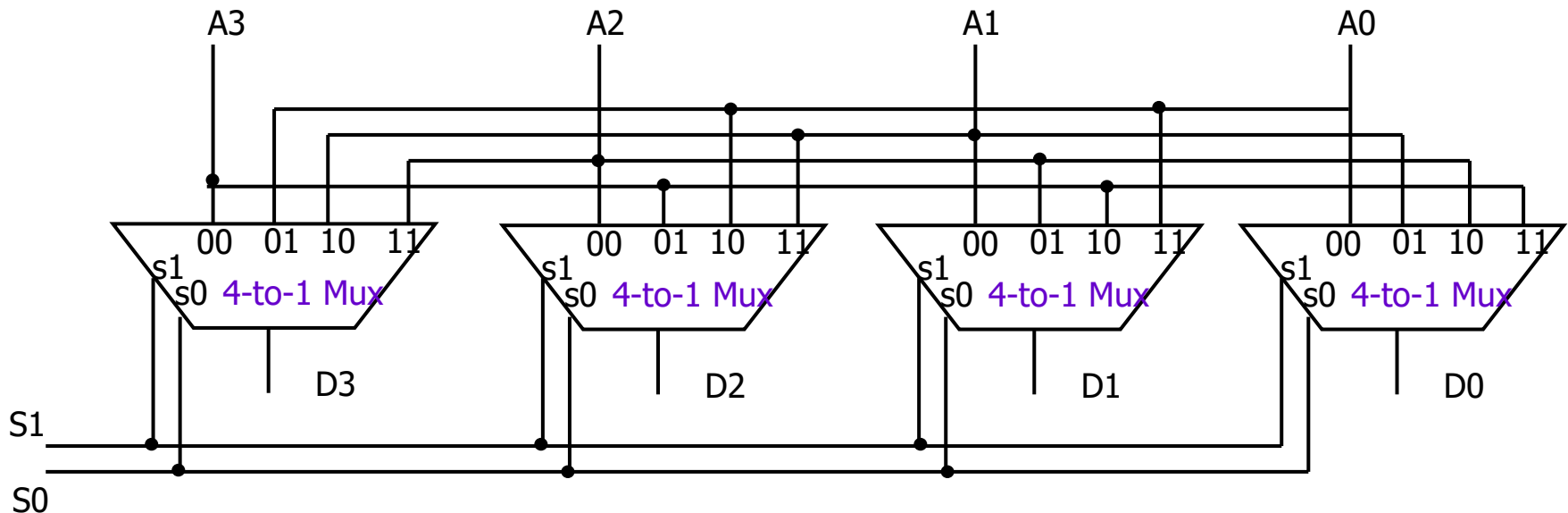
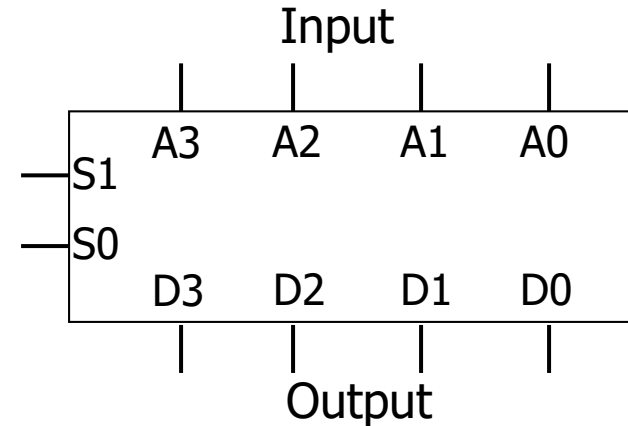
4-bit Arithmetic Shifter using 4:1 Mux



Rotator

- Rotate right

S1	S0	D3	D2	D1	D0
0	0	A3	A2	A1	A0
0	1	A0	A3	A2	A1
1	0	A1	A0	A3	A2
1	1	A2	A1	A0	A3



Barrel Shifter

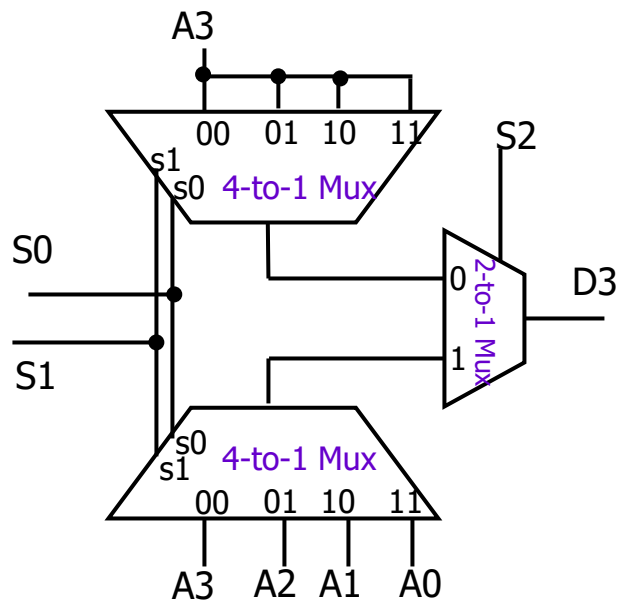
- In many applications, data should be shifted more than one bit position in a single clock cycle
- Barrel shifter is one form of combinational circuit that shifts or rotates the input data bits by the number of bits

Right Shift

Left Shift

S2	S1	S0	D3	D2	D1	D0
0	0	0	A3	A2	A1	A0
0	0	1	A3	A3	A2	A1
0	1	0	A3	A3	A3	A2
0	1	1	A3	A3	A3	A3
1	0	0	A3	A2	A1	A0
1	0	1	A2	A1	A0	0
1	1	0	A1	A0	0	0
1	1	1	A0	0	0	0

Barrel Shifter Design w/ Mux (D3)



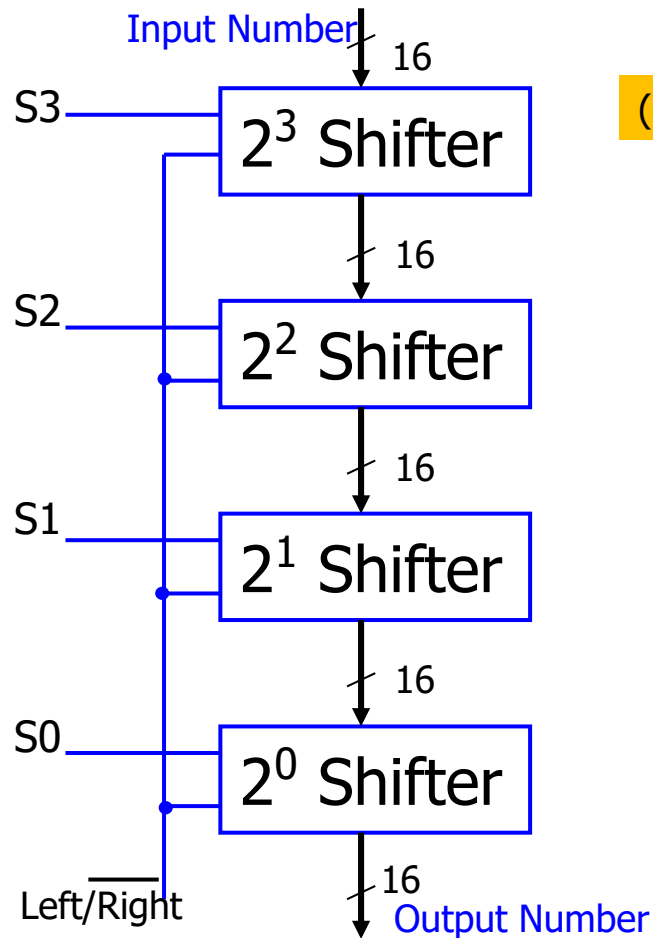
Right Shift

Left Shift

S2	S1	S0	D3	D2	D1	D0
0	0	0	A3	A2	A1	A0
0	0	1	A3	A3	A2	A1
0	1	0	A3	A3	A3	A2
0	1	1	A3	A3	A3	A3
1	0	0	A3	A2	A1	A0
1	0	1	A2	A1	A0	0
1	1	0	A1	A0	0	0
1	1	1	A0	0	0	0

Replicate and change wiring of the two 4-to-1 Muxes for D2, D1 and D0

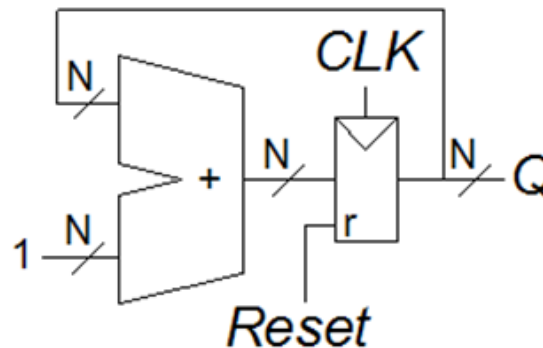
Barrel Shifter Design Alternative (16-bit)



(S3 S2 S1 S0) specifies the "shift amount" in binary

Counters

- An N-bit binary counter is a sequential arithmetic circuit with clock, reset, and an N-bit output
 - Increment output on each clock edge
 - Used to count cycles via numbers
 - For example: 000, 001, 010, 011, 100, 101, 110, 111, 000, 001...
 - Counters are used in many digital systems
 - Digital clock displays
 - Program counter (PC) register is used in computers to keep track of the current instruction CPU is executing

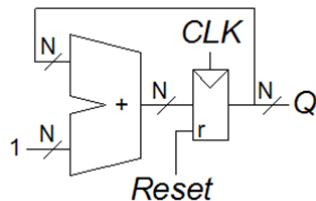


Verilog Representation

```
module counter #(parameter N = 8)
    (input          clk,
     input          reset,
     output reg [N-1:0] q);

    always @(posedge clk or posedge reset)
    begin
        if (reset) q <= 0;
        else      q <= q + 1;
    end

endmodule
```



```
`timescale 1ns / 1ns

module counter_tb( );
    reg  clk;
    reg  reset;

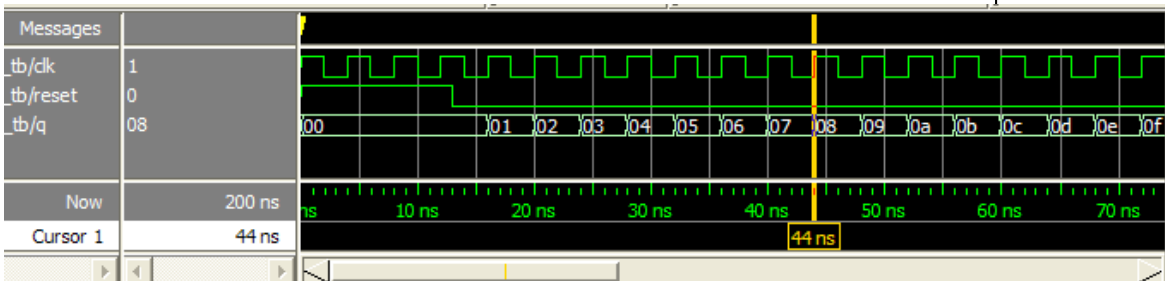
    wire [7:0] q;
    parameter clk_period = 5;

    counter counter_uut(.clk (clk),
                       .reset (reset),
                       .q (q));

    always
    begin
        clk = 1;
        forever #(clk_period/2) clk = ~clk;
    end

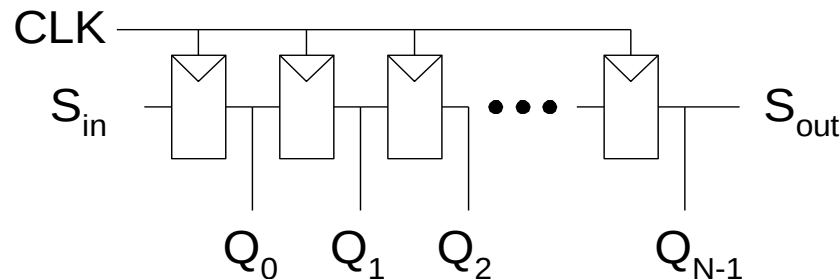
    initial
    begin
        reset = 1'b1; #(clk_period*2+3);
        reset = 1'b0;
    end

end
endmodule
```



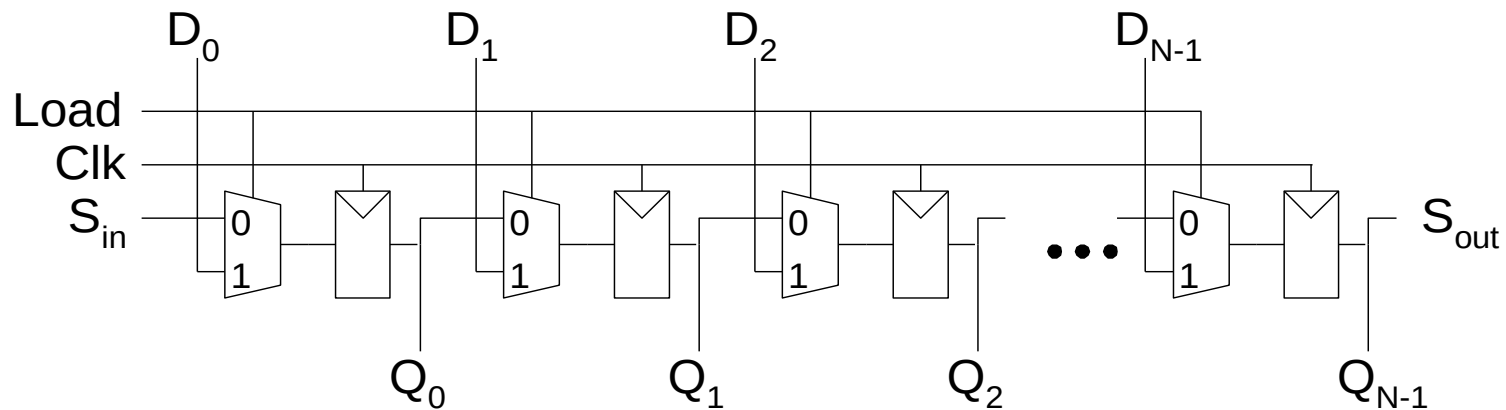
Shift Register

- Shift register has a clock, a serial input (S_{in}), a serial output (S_{out}), and N parallel outputs ($Q[N-1:0]$)
- A new bit is shifted in from S_{in} on each clock edge
 - All the subsequent contents are shifted forward
 - The last bit in the shift register is available at S_{out}
- Used as **serial-to-parallel converter**
 - Converts serial input (S_{in}) to parallel output ($Q[N-1:0]$)
- Don't be confused with shifters, which are combinational logic blocks that shift an input by a specified amount



Shift Register with Parallel Load

- Parallel-to-serial converter
 - When Load = 1, acts as a normal N-bit register
 - When Load = 0, acts as a shift register
- Now a shift register can act as
 - Serial-to-parallel converter (S_{in} to $Q[N-1:0]$)
 - Parallel-to-serial converter ($D[N-1:0]$ to S_{out})



Verilog Representation

```

module shiftreg #(parameter N = 8)
    (input          clk,
     input          reset, load,
     input          sin,
     input          [N-1:0] d,
     output reg [N-1:0] q,
     output          sout);

    always @(posedge clk or posedge reset)
    begin
        if (reset) q <= 0;
        else
            if (load) q[N-1:0] <= d;
            else      q[N-1:0] <= {q[N-2:0], sin};
        end

        assign sout = q[N-1];
    endmodule

```

```

`timescale 1ns / 1ns

module shiftreg_tb();
    reg clk, reset, load, sin,
    reg [7:0] d;

    wire [7:0] q;
    wire      sout;

    parameter clk_period = 10;

    shiftreg shiftreg_uut
        (.clk (clk), .reset (reset), .load (load),
         .sin (sin), .d (d), .q (q), .sout (sout));

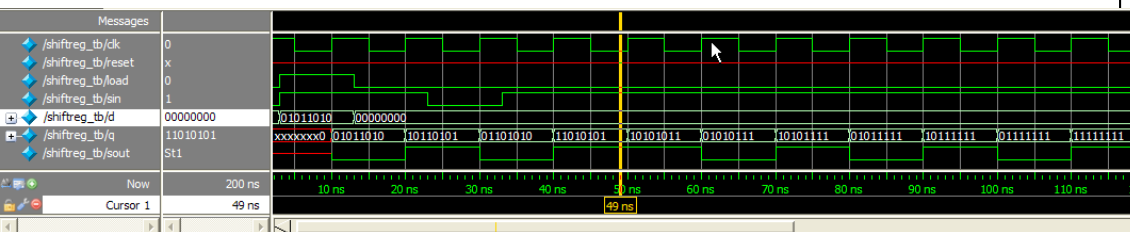
    always
    begin
        clk = 1;
        forever #(clk_period/2) clk = ~clk;
    end

    initial
    begin
        load = 1'b0; d = 8'h00; #3;
        load = 1'b1; d = 8'h5A; #(clk_period);
        load = 1'b0; d = 8'h00;

    end

    initial
    begin
        sin = 1'b0; #3;
        sin = 1'b1; #(clk_period);
        sin = 1'b1; #(clk_period);
        sin = 1'b0; #(clk_period);
        sin = 1'b1; #(clk_period);
        sin = 1'b1; #(clk_period);
        sin = 1'b1; #(clk_period);
    end
endmodule

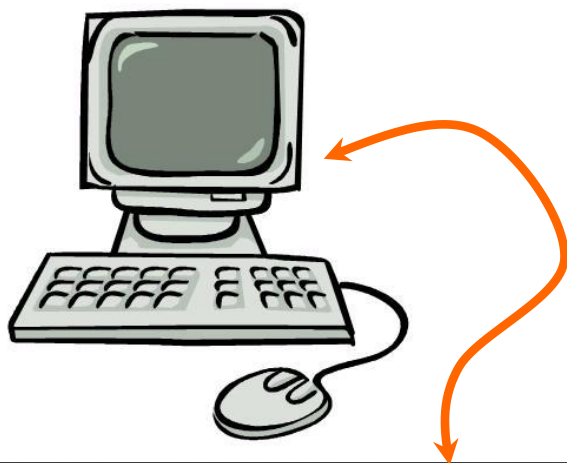
```



Backup Slides

Serial-to-Parallel Application

- RS-232 or UART communication



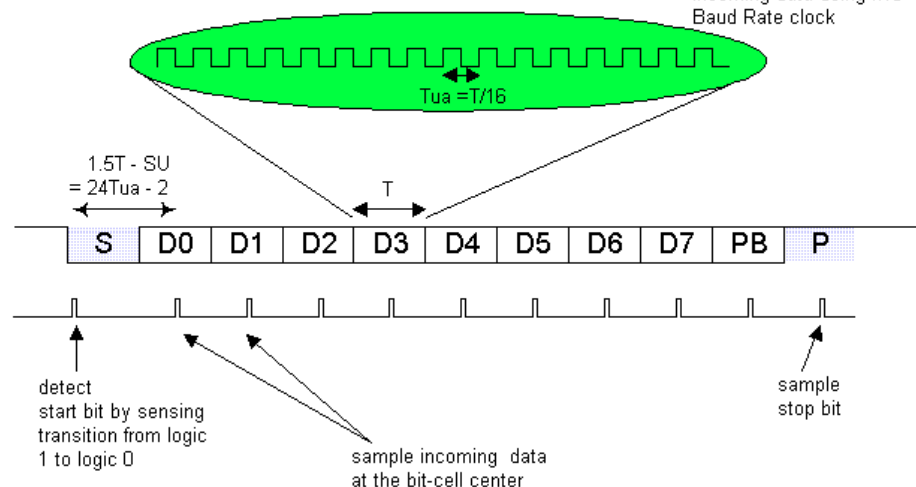
RxD

TxD



$T = 1/\text{Baud Rate}$. Bit-cell period.
 $T_{ua} = T/16$. Oversampled bit-cell period.
 $SU = 2 T_{ua}$. Worst case synchronizer uncertainty.

UART receiver samples the incoming data using $\times 16$ Baud Rate clock



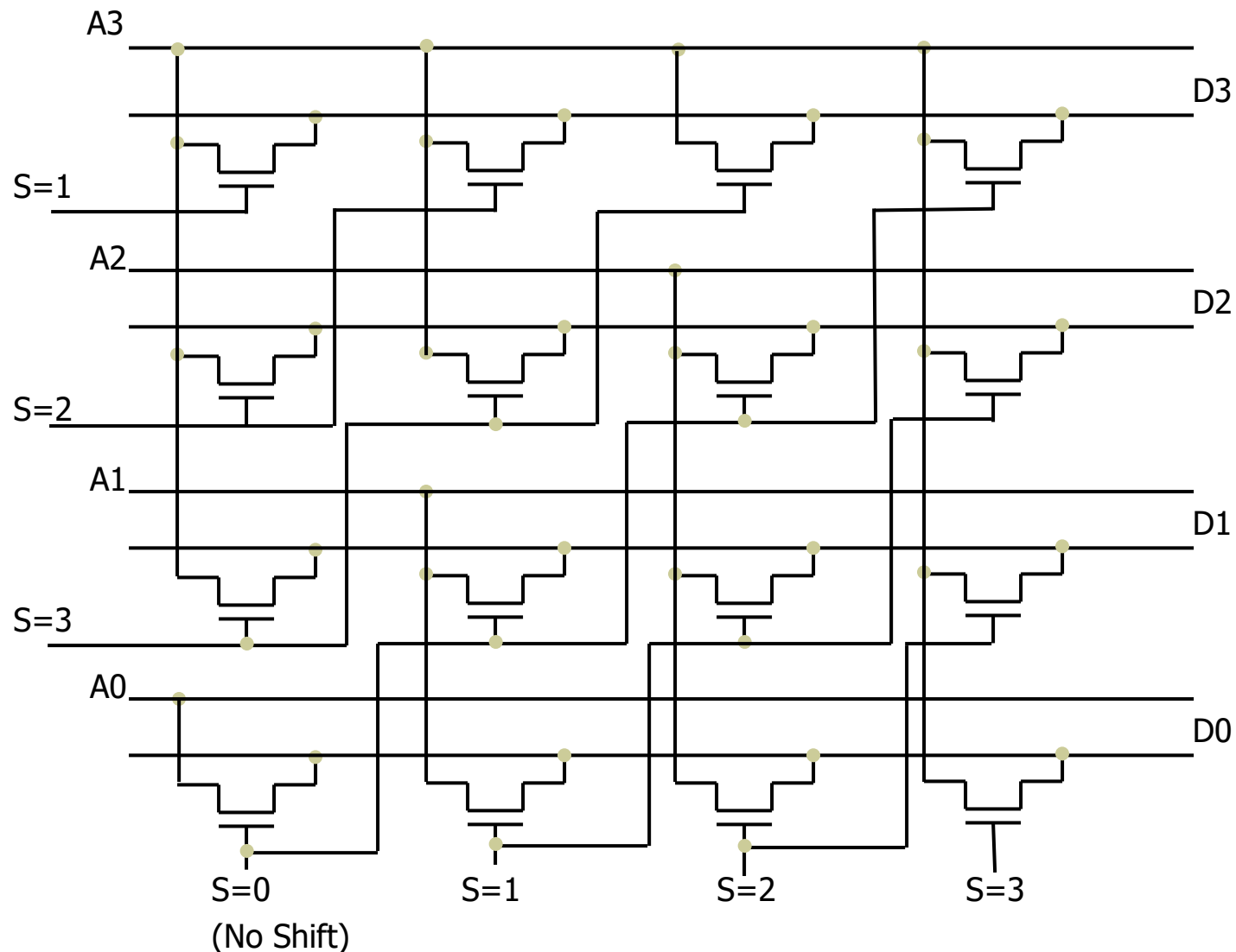
Floor and Ceiling Functions

- Floor function maps a real number to the next smallest integer
 - $\text{Floor}(x)$ is the largest integer, not greater than x
- Ceiling function maps a real number to the next largest integer
 - $\text{Ceiling}(x)$ is the smallest integer, not less than x

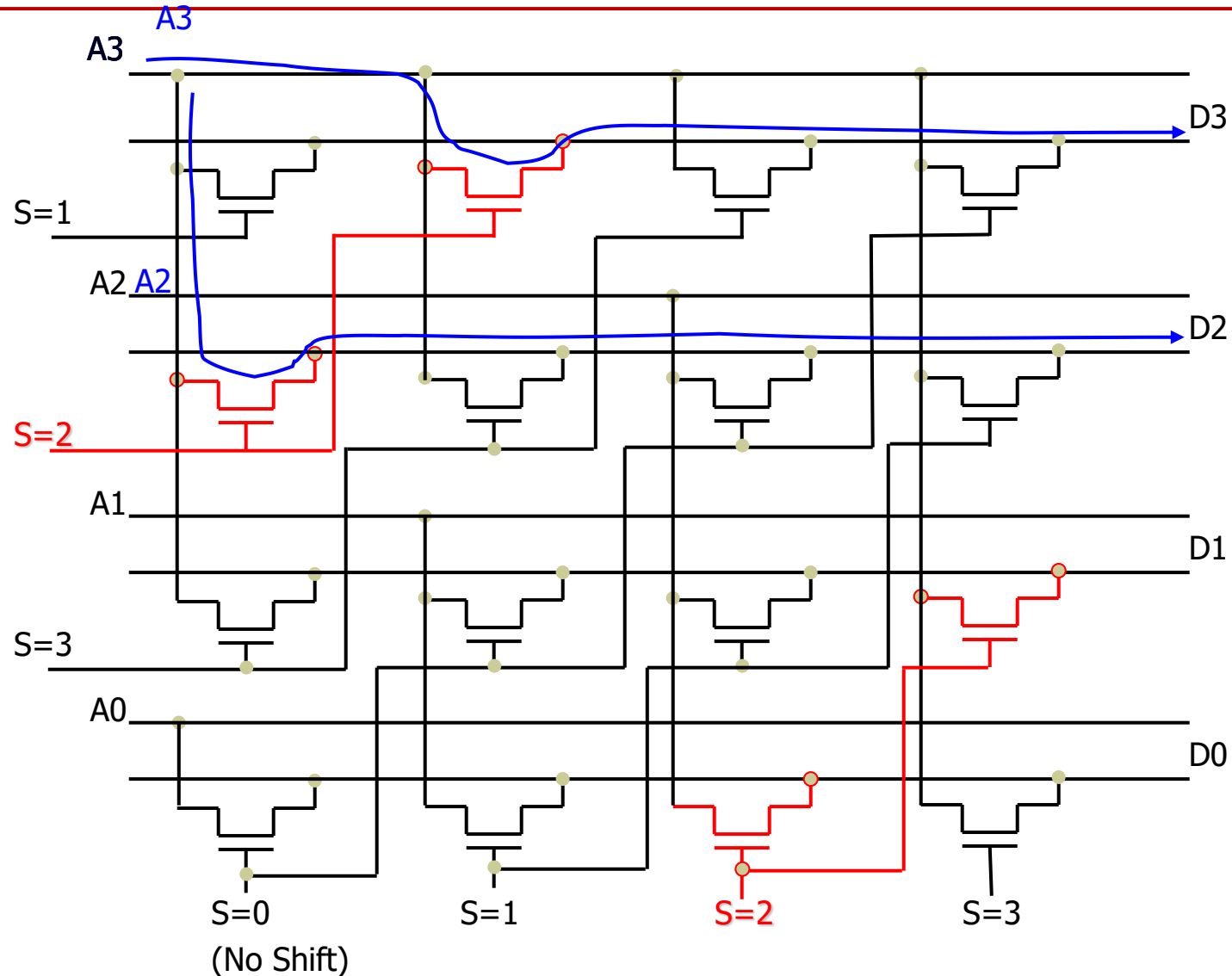
Example

Sample X	Floor $\lfloor X \rfloor$	Ceiling $\lceil X \rceil$
-2.7	-3	-2
-2	-2	-2
12/5	2	3
2.7	2	3

Barrel Shifter Design w/ nMOSFET



Barrel Shifter Design w/ nMOSFET



Barrel Shifter Design w/ nMOSFET

