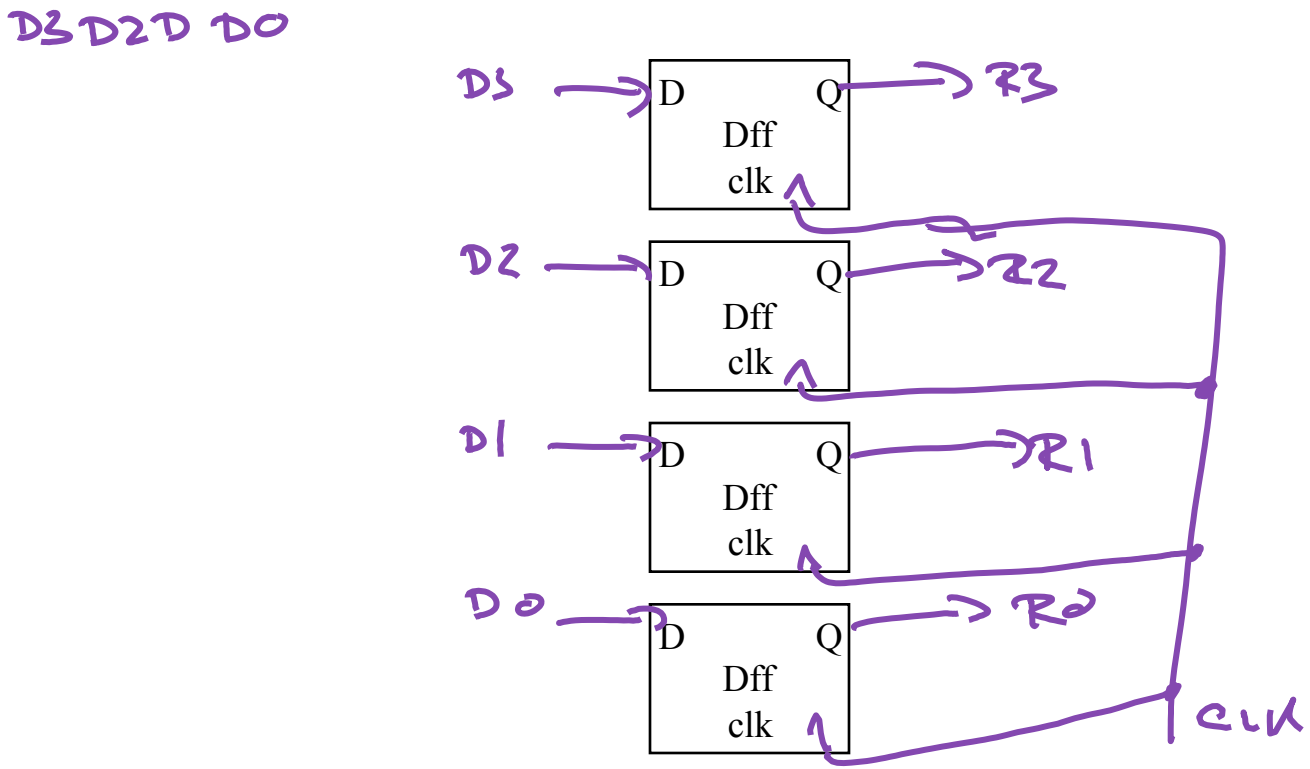
Registers

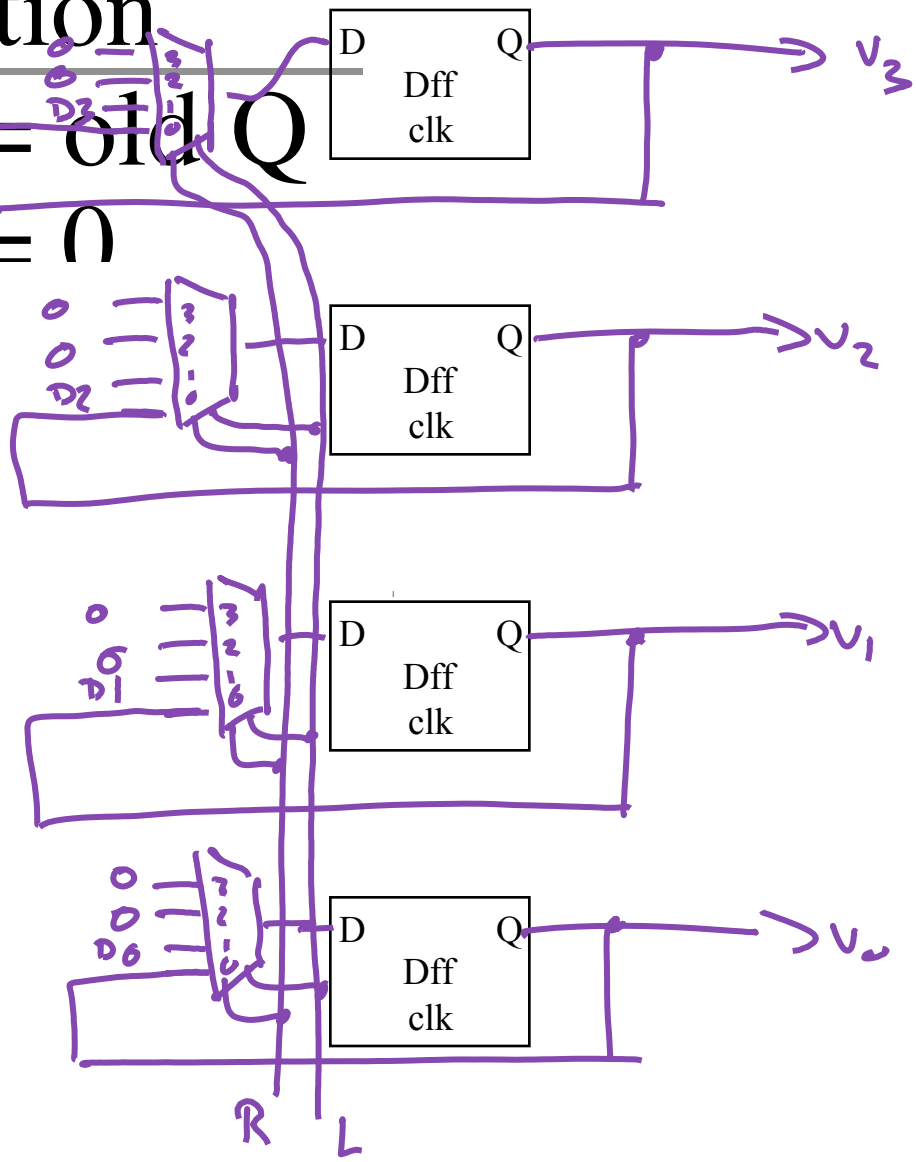
- Readings: 5.8-5.9.3
- Storage unit. Can hold an n-bit value
- Composed of a group of n flip-flops
  - Each flip-flop stores 1 bit of information



Controlled Register

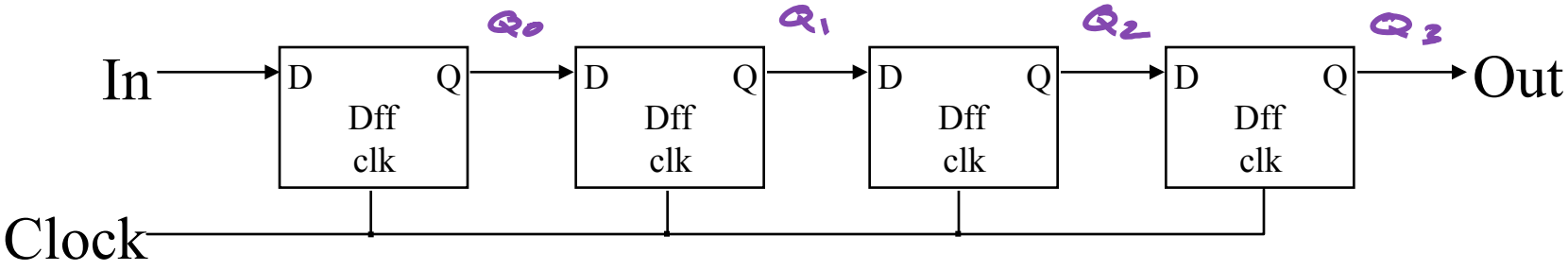
Reset	Load	Action
0	0	Q = old Q
1	0	Q = 0

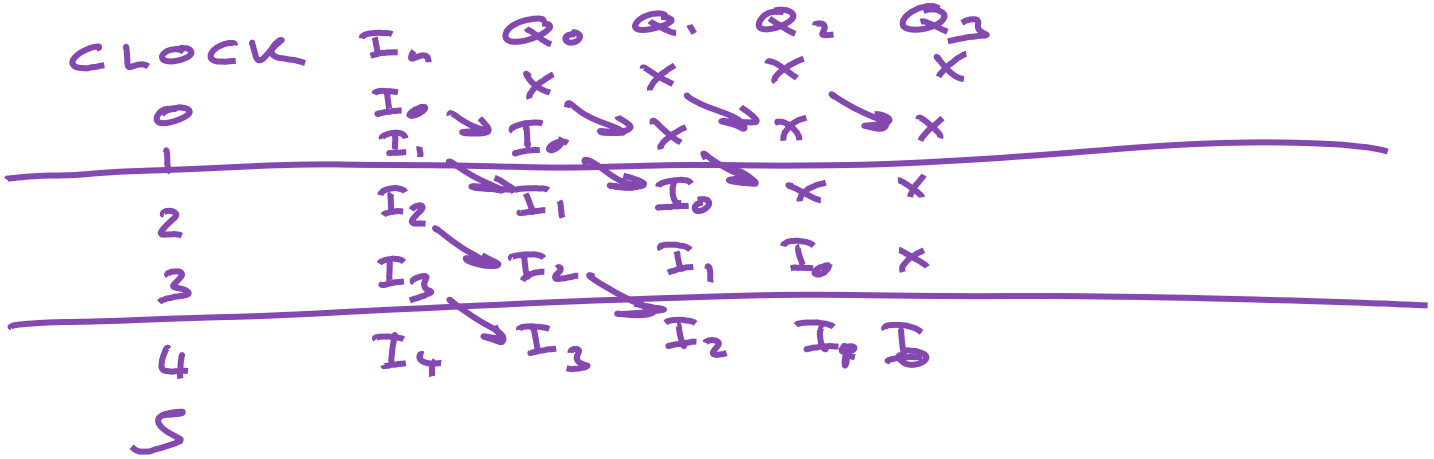
11 → RESET



Shift Register

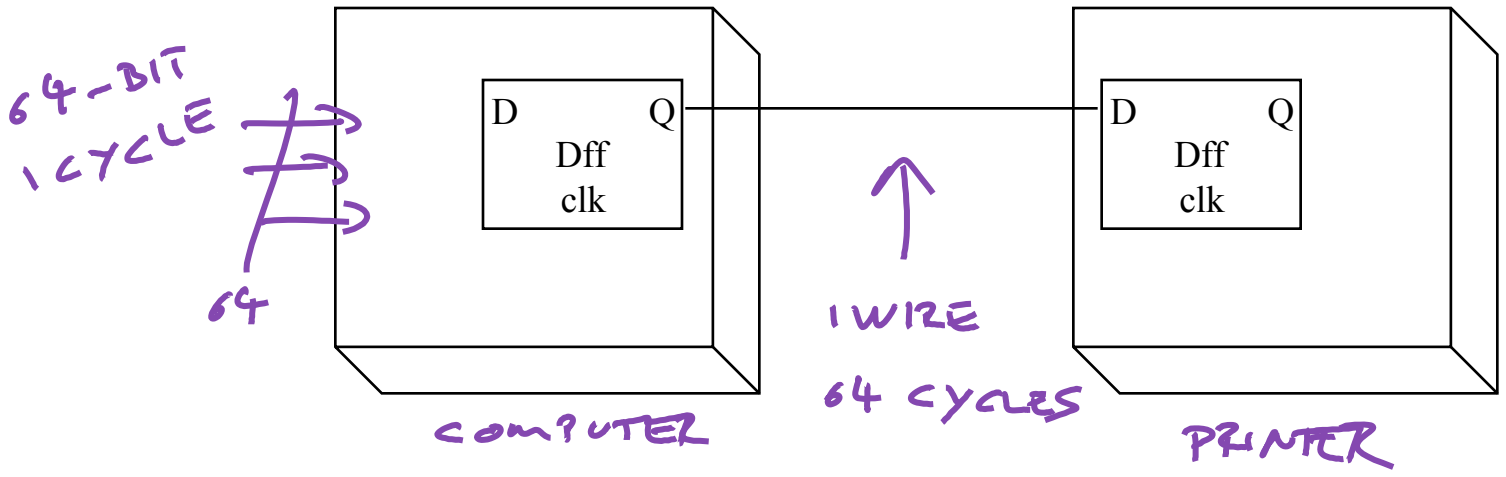
- Register that shifts the binary values in one or both directions





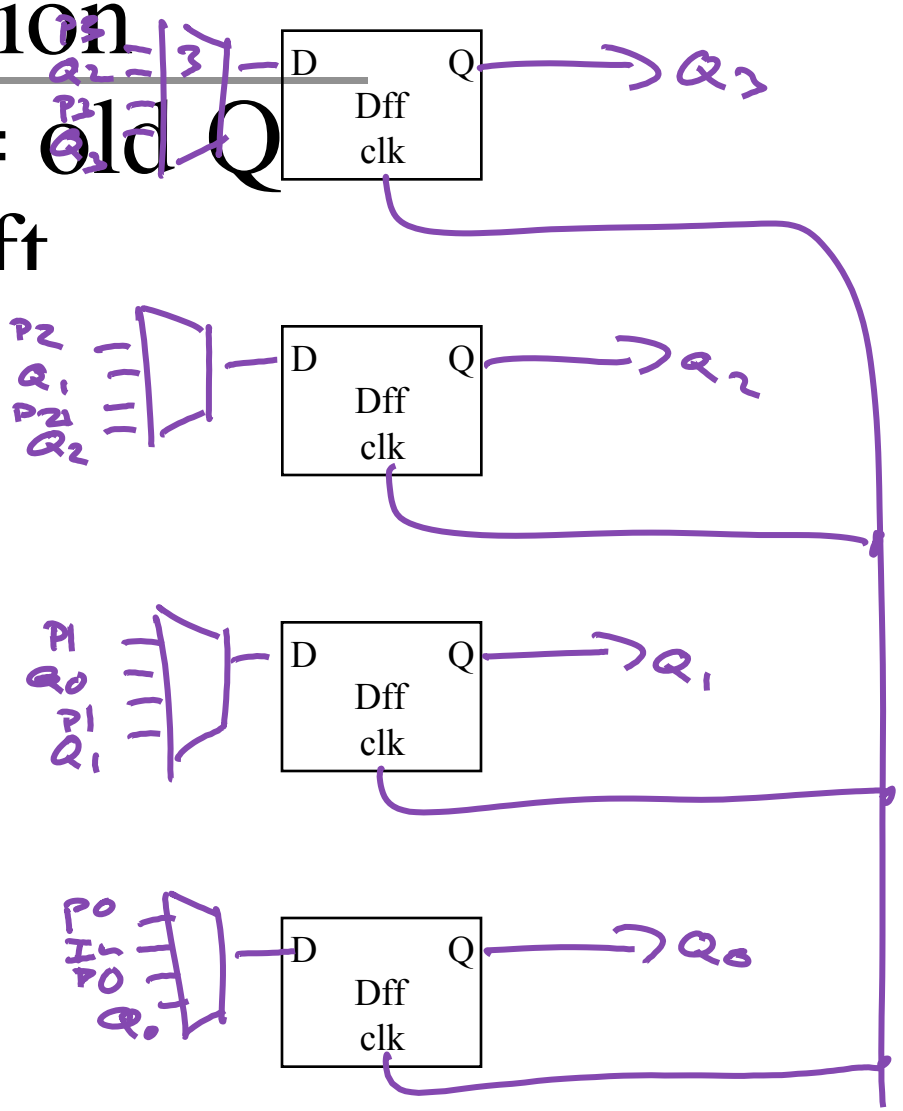
Transfer of Data

- 2 modes of communication: Parallel vs. Serial
  - Parallel: all bits transferred at the same time
  - Serial: one bit transferred at a time
- Shift register can be used for serial transfer

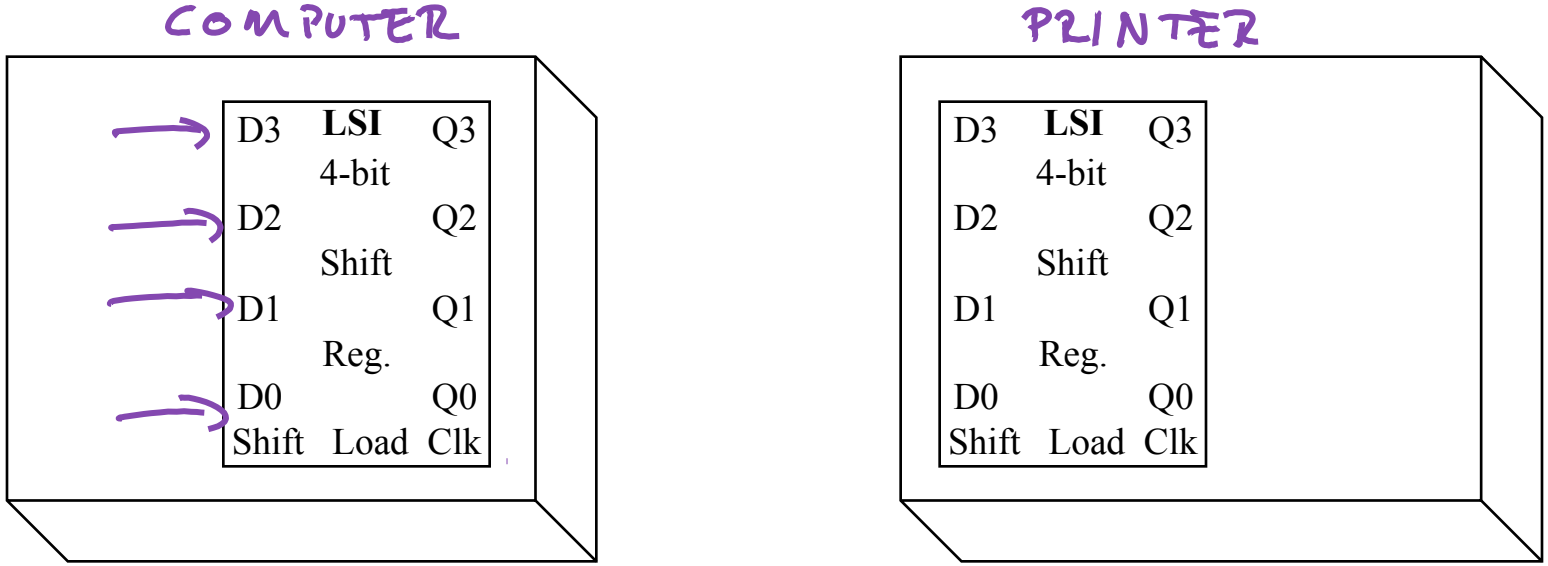


Shift Register w/Parallel Load

Shift	Load	Action
0	0	Q = old Q
1	0	Shift



Conversion between Parallel & Serial

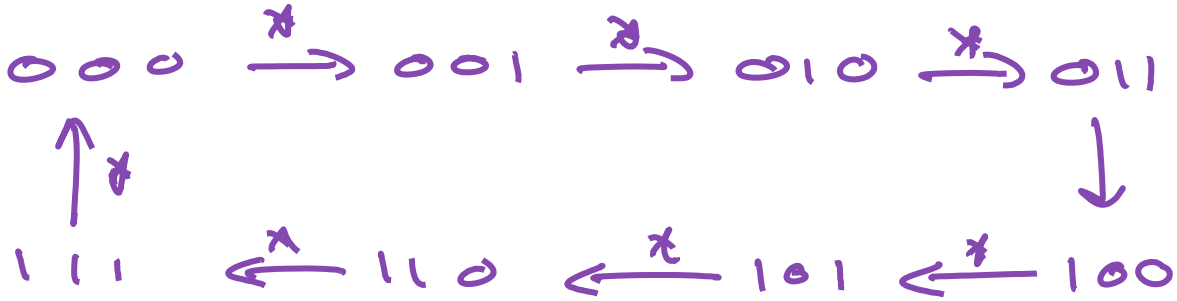


CLOCK	MODE	Q_0	Q_1	Q_2	Q_3
0	LOAD	X	X	X	X
1	SHIFT	D_0	D_1	D_2	D_3
2	SHIFT	X	D_0	D_1	D_2
3	SHIFT	X	X	D_0	D_1
4	LOAD	X	X	X	D_0
5	SHIFT	D_0'	D_1'	D_2'	D_3'
6	SHIFT				
7					

MODE	Q_0	Q_1	Q_2	Q_3
	X	X	X	X
SHIFT	X	X	X	X
SHIFT	D_2	X	X	X
SHIFT	D_2	D_3	X	X
SHIFT	D_1	D_2	D_3	X
	D_0	D_1	D_2	D_3

Counters

- A reg. that goes through a specific state sequence
- *n-bit Binary Counter:* counts from 0 to 2^N-1 in binary
- *Up Counter:* Binary value increases by 1
- *Down Counter:* Binary value decreases by 1
- 3-bit binary up counter state diagram:



PS_2 PS_1 PS_0	NS_2 NS_1 NS_0
0 0 0	0 0 1

$NS_0 = \overline{PS_0}$

$NS_1 = PS_1 \oplus PS_0$

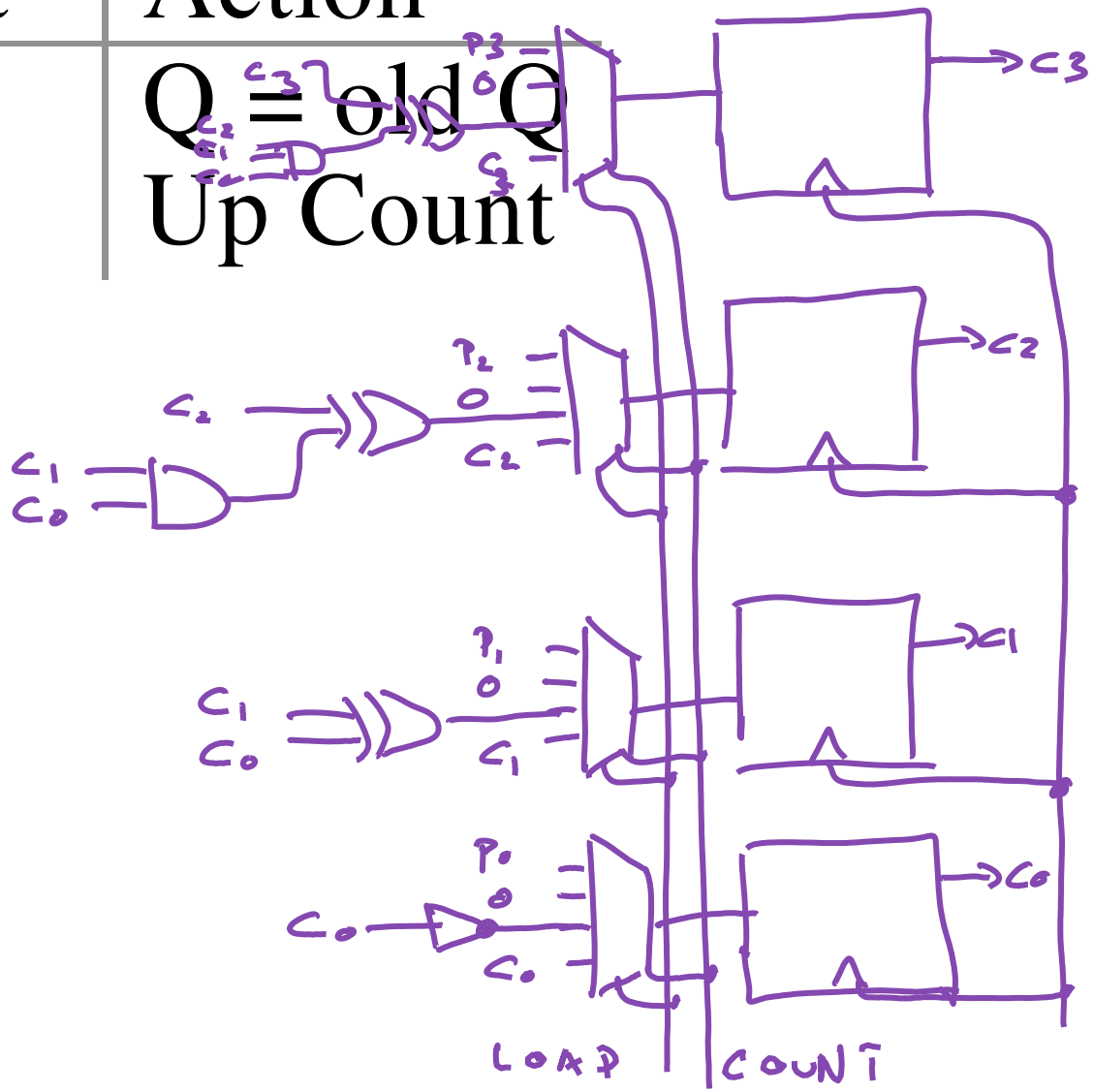
$NS_2 = PS_2 \oplus PS_1 PS_0$

$NS_i = PS_i \oplus (PS_{i-1} \cdots PS_0)$

Binary Up-Counter Imp.

Complex Binary Counter

Load	Count	Action
0	0	Q ≡ old Q
0	1	Up Count



Arbitrary Sequence Counters

- Design a 3-bit count that goes through the sequence 000->010->100->101->111->110->001->011->000->...

=> STATE DIAGRAM => STATE TABLE

=> SOLVE

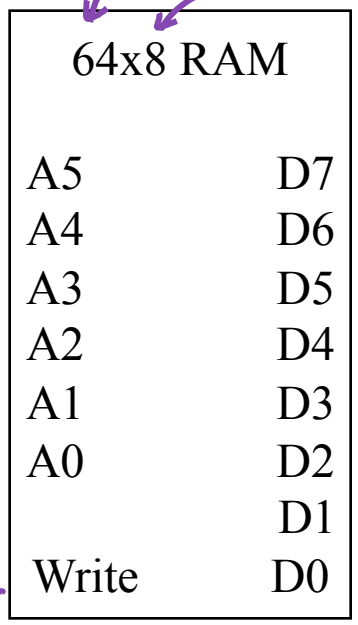
Counters in Verilog

```
module upcounter #(parameter WIDTH=8)
  (out, incr, reset, clk);

  output reg [WIDTH-1:0] out;
  input                  incr, reset, clk;

  always @(posedg  clk) begin
     if (reset)
        out <= 0;
     else if (incr)
        out <= out + 1;
     else
        out <= out;

     end

endmodule
```

Memory

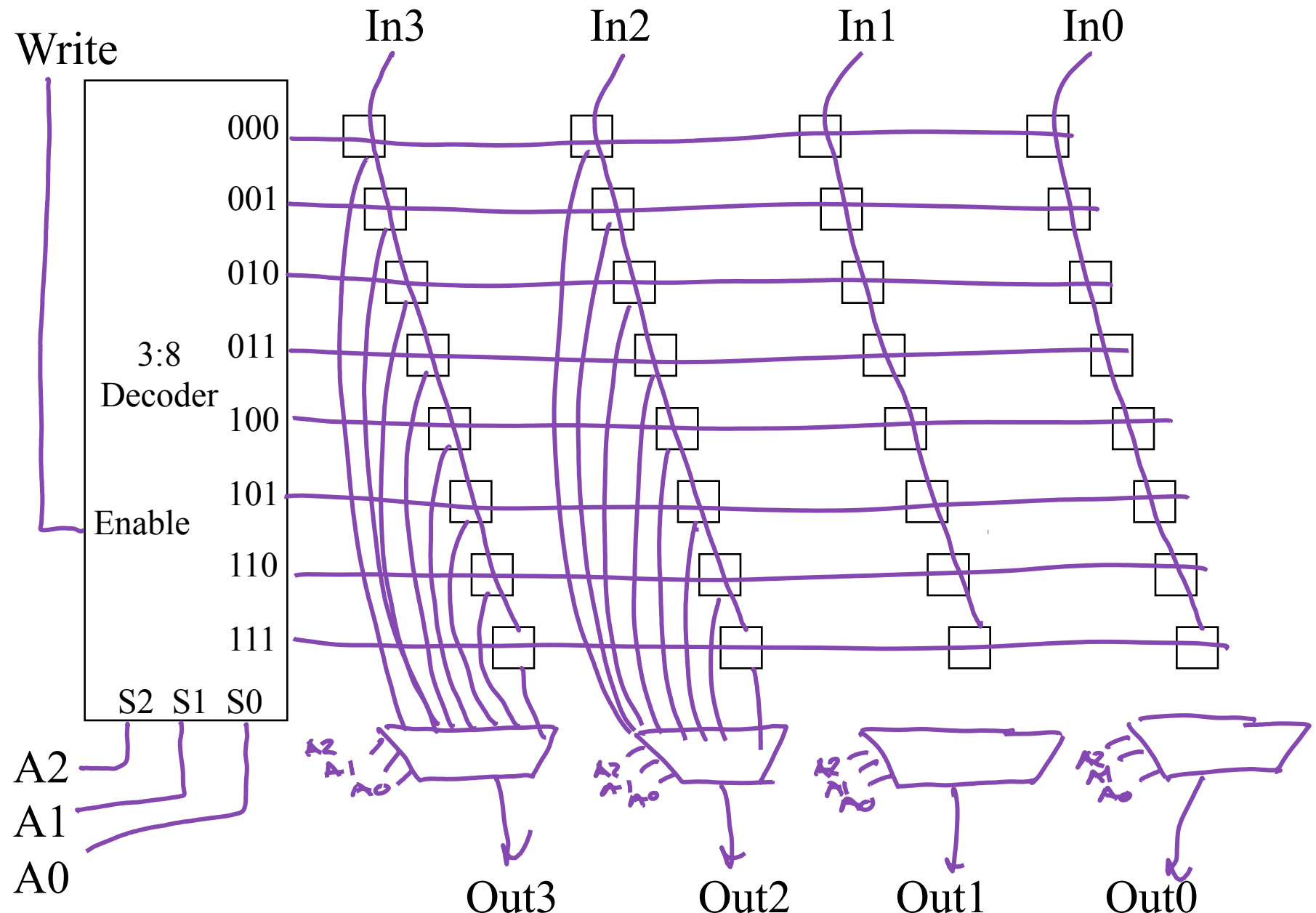
- Need method for storing large amounts of data
  - Computer programs, data, pictures, etc.

Address	Data
000000	00111110
000001	01101011
000010	01011101
000011	01100011
000100	00111110
000101	00000000
000110	11111111
000111	01010101



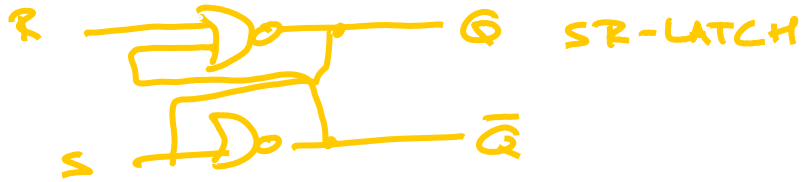
- RAM: Random Access Memory, Read/Write
- ROM: Read-only Memory

8x4 RAM

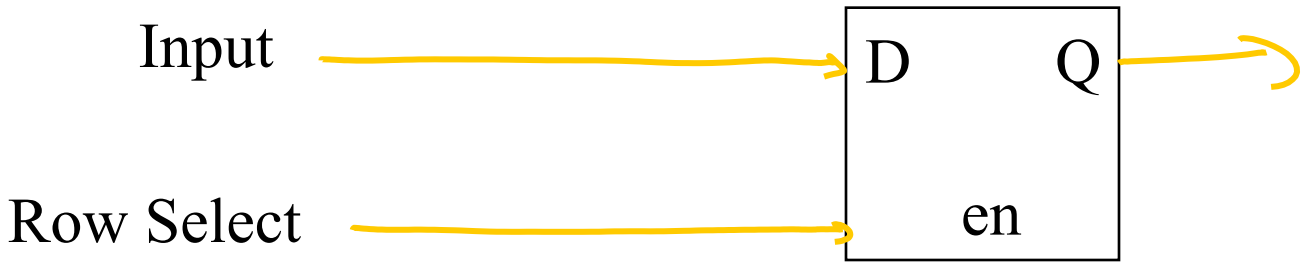

Write
In3
In2
In1
In0
000
001
010
011
100
101
110
111
3:8
Decoder
Enable
S2
S1
S0
A2
A1
A0
Out3
Out2
Out1
Out0

RAM Cell

- Requirements:
  - Store one bit of data
  - Change data based on input when row is selected



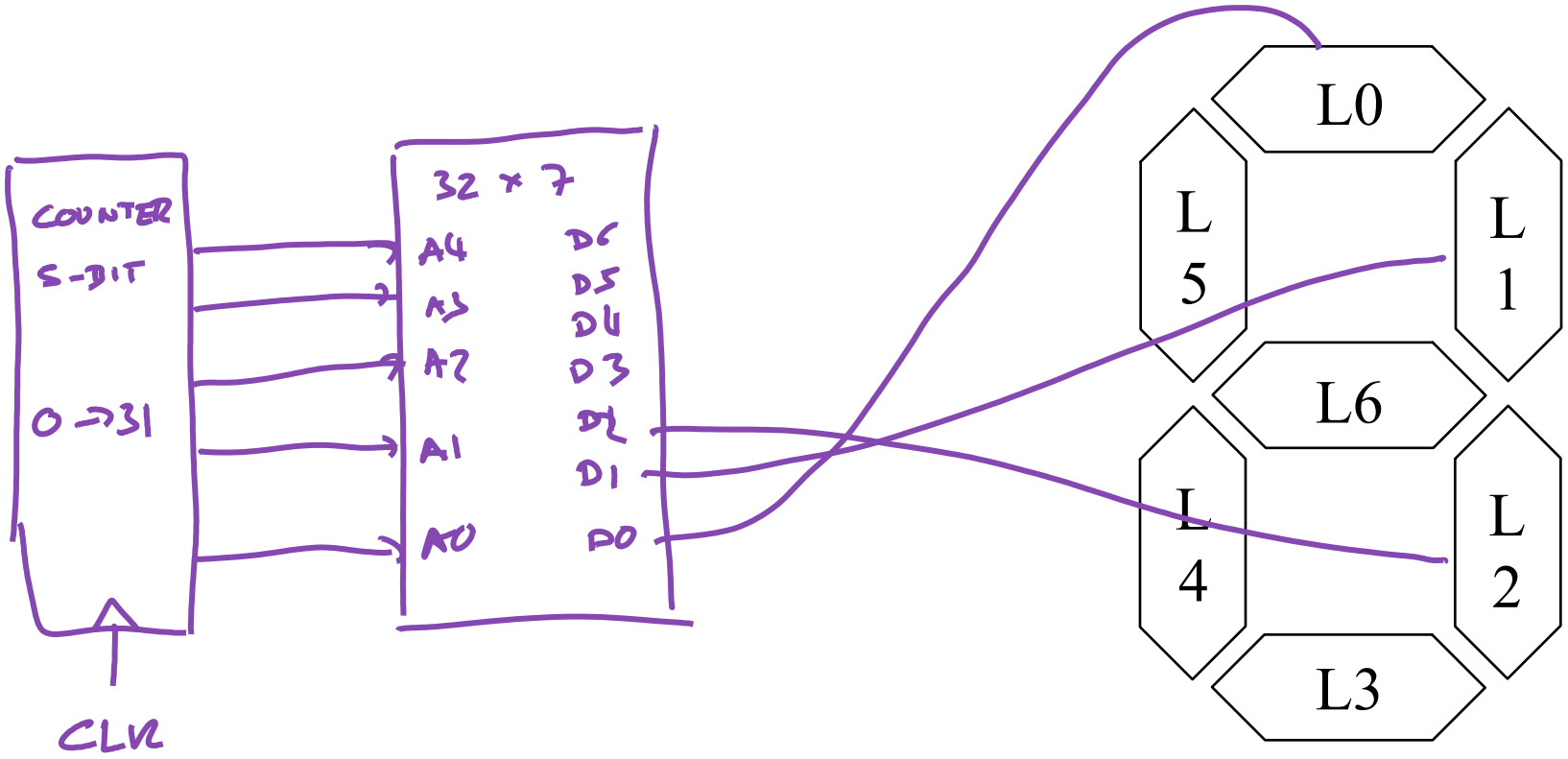
SR	Q
00	HOLD
01	0
10	1
11	FORBIDDEN



W	ROW	S	R	Q
0	0	0	0	HOLD
0	1	0	1	0
1	0	0	0	HOLD
1	1	1	0	1

RAM example

- Use a memory to do a programmable 32-picture animation on a 7-segment display



Verilog Memories

```
module memory16x6 (data_out, data_in, addr, we, clk);
  output reg [5:0] data_out;
  input      [5:0] data_in;
  input      [3:0] addr;
  input            we, clk;

  reg        [5:0] mem       [15:0];

  always @(*)
    data_out = mem[addr];

  always @(posedge clk)
    if (we)
      mem[addr] <= data_in;

endmodule
```