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

```
  D        Q
  Dff      clk

  D        Q
  Dff      clk

  D        Q
  Dff      clk

  D        Q
  Dff      clk
```
## Controlled Register

<table>
<thead>
<tr>
<th>Reset</th>
<th>Load</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q = old Q</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Q = 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Q = D</td>
</tr>
</tbody>
</table>

![Diagram of controlled register]
Shift Register

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

Diagrams and symbols are used to depict the shift register, showing the flow of data from input to output with the clock signal.
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

![Diagram showing parallel and serial communication](image)
Shift Register w/Parallel Load

<table>
<thead>
<tr>
<th>Shift</th>
<th>Load</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q = old Q</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Shift</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>Parallel Load</td>
</tr>
</tbody>
</table>

![Diagram](Dff clk)

![Diagram](Dff clk)

![Diagram](Dff clk)

![Diagram](Dff clk)
Conversion between Parallel & Serial

Diagram:

- D3 LSI Q3
- 4-bit
- D2 Q2
- Shift
- D1 Q1
- Reg.
- D0 Q0
- Shift Load Clk

Diagram:

- D3 LSI Q3
- 4-bit
- D2 Q2
- Shift
- D1 Q1
- Reg.
- D0 Q0
- Shift Load Clk
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:
Binary Up-Counter Imp.
## Complex Binary Counter

<table>
<thead>
<tr>
<th>Load</th>
<th>Count</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$Q = \text{old } Q$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Up Count</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Reset</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Load Parallel</td>
</tr>
</tbody>
</table>
Arbitrary Sequence Counters

- Design a 3-bit count that goes through the sequence 000->010->100->101->111->110->001->011->000->...
module upcounter #(parameter WIDTH=8)
    (out, incr, reset, clk);

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

endmodule
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