Latches, Flip-flops, Registers, Memory

**Sequential logic:** elements to store values
Output depends on inputs *and stored values.*

(vs. combinational logic: output depends only on inputs)
Processor: Data Path Components

Instruction Fetch and Decode

Registers

ALU

Memory

1

2

3
Bistable latches

Suppose we somehow get a 1 (or a 0?) on here.
**SR latch**

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>Q'</th>
<th>Q (stable)</th>
<th>Q' (stable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
<td>?</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>?</td>
<td>?</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
SR latch
if $C = 0$, then SR latch stores current value of $Q$.

if $C = 1$, then $D$ flows to $Q$:

if $D = 0$, then $R = 1$ and $S = 0$, $Q = 0$

if $D = 1$, then $R = 0$ and $S = 1$, $Q = 1$
Time matters!

Assume Q has an initial state of 0
Clocks

**Clock:** free-running signal with fixed **cycle time** = **clock period** = \( T \).

**Clock frequency** = \( 1 / \text{clock period} \)

A clock controls when to update a sequential logic element's state.
Synchronous systems

Inputs to state elements must be **valid** on active clock edge.
D flip-flop with falling-edge trigger

Clock

Can still read \(Q_{\text{now}}\)

Next becomes \(Q_{\text{now}}\)

Leader stores D as E

Follower stores E as Q

Time
Time matters!

Assume Q and E have an initial state of 0
Reading and writing in the same cycle

Assume Q is initially 0.
D flip-flop = one bit of storage
A 1-nybble* register
(a 4-bit hardware storage cell)

*Half a byte!
Array of registers, with register selectors, write/read control, input port for writing data, output ports for reading data.
Read ports
(data out)

FIGURE C.8.7 A register file with two read ports and one write port has five inputs and two outputs. The control input Write is shown in color.

FIGURE C.8.8 The implementation of two read ports for a register file with \( n \) registers can be done with a pair of \( n \)-to-1 multiplexors, each 32 bits wide. The register read number signal is used as the multiplexor selector signal. Figure C.8.9 shows how the write port is implemented.
FIGURE C.8.9 The write port for a register file is implemented with a decoder that is used with the write signal to generate the \( C \) input to the registers. All three inputs (the register number, the data, and the write signal) will have setup and hold-time constraints that ensure that the correct data is written into the register file.

Write port (data in)

- **Write control**
- **Clock**
- **Register number**
- **Incoming data**

\( n \)-to-\( 2^n \) decoder

\( C \)
\( D \)

Register 0
Register 1
Register \( n-2 \)
Register \( n-1 \)
RAM (Random Access Memory)

Similar to register file, except...
16 x 4 RAM

4-bit address

1101

4 to 16 decoder

data out