Registers and Register Files
State – the central concept of computing

Computer can do 2 things
1) Store state (How do we actually store bits?)
2) Manipulate state
Today’s lecture

- More D Flip flops
  - Asynchronous reset
  - Enable

- Random Access Memory (RAM)
  - Addressable storage

- Register Files
  - Registers
  - Decoders
Asynchronous reset immediately resets a flip-flop to 0

- **Asynchronous** = pertaining to operation without the use of fixed time intervals (opposed to **synchronous**).
Asynchronous Reset implementation

One example possible implementation

DLatch w/Enable

SRLatch w/Enable

Ignores inputs and current state.

Forces Q output to zero.

(Not required material)
When enable is 0, the flip flop doesn’t change on the rising edge

- Behaves normally when enable=1
Use multiple flip flops to build registers

- Example 4-bit register made of four D flip flops
  - All control signals use the same input
Random Access Memory (RAM) is the hardware equivalent of an array

<table>
<thead>
<tr>
<th>Addr</th>
<th>idx</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ RAM stores <strong>data</strong> at addresses</td>
<td>▪ Arrays store <strong>data</strong> at indices</td>
</tr>
<tr>
<td>▪ All <strong>data</strong> has the same bit width</td>
<td>▪ All <strong>data</strong> has the same type</td>
</tr>
<tr>
<td>▪ Use brackets to access <strong>data</strong> at any address “immediately”</td>
<td>▪ Use brackets to access <strong>data</strong> at any index “immediately”</td>
</tr>
</tbody>
</table>

M[Addr]

Array[idx]
RAM is the hardware equivalent of an array

- The address is an array index.
  - A $k$-bit address can specify one of $2^k$ words
- Each address refers to one word of data.
  - Each word can store $N$ bits
A RAM should be able to

1. Store many words, one per address
2. Read the word that was saved at a particular address ($??? = M[Addr]$)
3. Change the word that’s saved at a particular address ($M[Addr] = ???$)
A Register File is a synchronous RAM

Use the letter R to indicate that the RAM is a register file rather than a generic memory (M)

R[Addr]
Our MIPS register file will enable single cycle operations on multiple data:

- 2 read ports, so we can read two values simultaneously
- 1 write port
Let’s build a 2-word memory with 4-bit words

A register file has 3 parts

1. **The Storage**: An array of registers
2. **The Read Ports**: Output the data of the register indicated by read addresses
3. **The Write Port**: Selectively write data to the register indicated by write address
Step 1: Allocate 1 register per address ($2^1 \times 4$)

- Wire clocks and resets together to maintain synchronization
Step 2: Read ports use the **address** to **select** one register’s **data** to output
Step 3: Write ports decode the **address** to _enable_ writing to exactly one register.
Decoders receive a binary code to generate control signals

- A 1-to-2 Binary decoder receives a 1-bit unsigned binary code to enable one of two devices

<table>
<thead>
<tr>
<th>EN</th>
<th>A0</th>
<th>(E1, E0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>(0, 0)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>(0, 1)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>(1, 0)</td>
</tr>
</tbody>
</table>
n-to-2\(^n\) Binary decoders receive \(n\)-bit unsigned binary codes to enable one of \(2^n\)

<table>
<thead>
<tr>
<th>EN</th>
<th>A[1:0]</th>
<th>E[3:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>(0,0)</td>
<td>(0,0,0,1)</td>
</tr>
<tr>
<td>1</td>
<td>(0,1)</td>
<td>(0,0,1,0)</td>
</tr>
<tr>
<td>1</td>
<td>(1,0)</td>
<td>(0,1,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>(1,1)</td>
<td>(1,0,0,0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EN</th>
<th>A[4:0]</th>
<th>E[31:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>0x0000</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0x0001</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0x0002</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>30</td>
<td>0x4000</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
<td>0x8000</td>
</tr>
</tbody>
</table>
What does it do?
What do you get if you connect a register to an adder?
Implementing counters