Exam 1 is live

Registers and Register Files

Pick up handout.
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

D Latch w/Enable

SR Latch 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

![Diagram of flip-flop behavior](image)
\[2^2 = 4\]

Address bits \(\times\) Data bits

- Address: 00101011
- Data: 10001111, 10101011, 01001000

00, 01, 10, 11
3 Address bits \times 8 Data bits
2 Address bits \times 8 Data bits

00101011
00

10001111
01

10101011
10

01001000
11
2 Address bits × 16 Data bits
\( \frac{2}{\text{Address bits}} \times \frac{32}{\text{Data bits}} \)
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

- **Addr**
  - RAM stores data at addresses
  - All data has the same bit width
  - Use brackets to access data at any address
    - $M[\text{Addr}]$

- **idx**
  - Arrays store data at indices
  - All data has the same type
  - Use brackets to access data at any index
    - $\text{Array}[\text{idx}]$
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)

2^k x N
register file

\[ R[\text{Addr}] \]
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
clicker question

We need to build a RAM that can store 128, 64-bit words and has one write port and three read ports.
How many *address* bits does my RAM need for its *write* port?

A) 1
B) 6
C) 7
D) 64
E) 128
We need to build a RAM that can store 128, 64-bit words and has one write port and three read ports. How many data bits does my RAM need for each read port?

A) 3  
B) 6  
C) 7  
D) 64  
E) 128
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

A data

B data

A address

B address

write address

write data

write enable

2^1 x 4 register file
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 and an enable signal to enable one of two devices.
**n-to-2^n** Binary decoders receive **n**-bit unsigned binary codes to enable one of **2^n** devices.

<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>
$2^1 \times 4$-bit register file (only 1 read port fully built)
What does it do?

- **clk**: 0 1 0 1 0 1 0 1
- **reset**: 0 1 0 1 0 1 0 1
- **W_addr**: 0 1 0 1 0 1 0 1
- **W_data**: 0xa 0x1 0x5 0xf 0xe
- **W_en**: 0 1 0 1 0 1 0 1
- **R_addrA**: 0 1 0 1 0 1 0 1
- **R_dataA**: φ A φ A F
- **R_addrB**: 0 1 0 1 0 1 0 1
- **R_dataB**: φ φ 5

Timing diagram:
- a) 0x0
- b) 0xa
- c) 0x1
- d) 0x5
- e) 0xf
What will Q[3:0] be during the next clock cycle?

a) 0  
b) 1  
c) 3  
d) 4  
e) 5
a) 0x0
b) 0x2
c) 0x4
d) 0x6
e) 0x8
Implementing counters

Half Adder
Cout Sum

D Q
en reset

Qn

reset clk enable

Qn