State

CS 3410
Computer Science
Cornell University

[K. Bala, A. Bracy, E. Sirer, and H. Weatherspoon]
Stateful Components

Combinational logic

• Output computed directly from inputs
• System has no internal state
• Nothing depends on the past!

Need:

• to record data
• to build stateful circuits
• a state-holding device
Instructions live in **Program Memory**

**PC** = Program Counter, address of current instruction

“Next Instruction Address” = PC + 4

When should we update the PC?

*As fast and as often as possible?*
Clocks

Clock helps coordinate state changes

- Fixed period
- Frequency = 1/period
**Edge Triggered State Changes**

State changes at clock edge

- **positive edge-triggered**

- **negative edge-triggered**

Need to design edge-triggered storage

Positive edge-triggered D Flip-Flop:

- Data captured when clock low
- Output changes only on rising edge
  (could also design it to be negative edge-triggered)
Clock Methodology
Signals must be stable prior to rising edge
Positive edge-triggered D Flip-Flop:
• Output changes only on rising edge
• Data captured when clock low

![Diagram of clock methodology with DFFs and combinational circuit](image-url)
State Examples: Program Memory & PC

(a 32 bit encoding of a subtract instruction)
If we wanted to make the clock faster, what would we need to speed up?

(A) the +4 adder
(B) the time it takes to read Program Memory
(C) the time it takes to execute an instruction
(D) B or C
(E) A, B & C
Goals for Today

Clocks

State

- Storing 1 bit
- Storing N bits:
  - Registers
  - Memory
Registers

- D flip-flops in parallel
- Shared clock
- Additional (optional) inputs: writeEnable, reset, ...

![Diagram of registers with D flip-flops and a 4-bit register with clk input and output]

```plaintext
D0 -> DFF
D1 -> DFF
D2 -> DFF
D3 -> DFF
clk
```

4-bit reg

```plaintext
clk
```
Register File

- N read/write registers
- Indexed by register number

Register File

- Single-Read-Port
- Single-Write-Port

32 x 32
Writing to the Register File (1)

Register File
- N read/write registers
- Indexed by register number

addi $r5, r0, 10

How to write to **one** register in the register file?
- Need a decoder
Aside: 3-to-8 decoder truth table & circuit

<table>
<thead>
<tr>
<th>i2</th>
<th>i1</th>
<th>i0</th>
<th>o0</th>
<th>o1</th>
<th>o2</th>
<th>o3</th>
<th>o4</th>
<th>o5</th>
<th>o6</th>
<th>o7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

3-to-8 decoder
Aside: 3-to-8 decoder truth table & circuit

<table>
<thead>
<tr>
<th>i2</th>
<th>i1</th>
<th>i0</th>
<th>o0</th>
<th>o1</th>
<th>o2</th>
<th>o3</th>
<th>o4</th>
<th>o5</th>
<th>o6</th>
<th>o7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3-to-8 decoder

3
101
R_w

i2
i1
i0

o0

i2
i1
i0

o5
Writing to the Register File (2)

Register File
- N read/write registers
- Indexed by register number

addi r5, r0, 10

How to write to **one** register in the register file?
- Need a decoder
- Write enable signal prevents unintended writes
Reading from the Register File

Register File

- N read/write registers
- Indexed by register number

How to read from one register? Need:

(A) Encoder
(B) Decoder
(C) Or Gate
(D) Multiplexor
Reading from the Register File

Register File
- N read/write registers
- Indexed by register number

How to read from one register?
- Need a multiplexor
Reading from the Register File

Register File

• N read/write registers
• Indexed by register number

How to read from two registers?
• Need 2 multiplexors!
Complete Register File

Register File

- N read/write registers
- Indexed by register number

Implementation:

- D flip flops to store bits
- Decoder for each write port
- Mux for each read port
Register File

- N read/write registers
- Indexed by register number

Implementation:
- D flip flops to store bits
- Decoder for each write port
- Mux for each read port
MIPS Register file

MIPS register file

- 32 x 32-bit registers
- r0 wired to zero
- Write port indexed via \( R_W \)
  - on falling edge when WE=1
- Read ports indexed via \( R_A, R_B \)

Registers

- Numbered from 0 to 31.
- Can be referred by number: $0, $1, $2, ... $31
- Convention, each register also has a name:
  - $16 - $23 \( \rightarrow \) $s0 - $s7, $8 - $15 \( \rightarrow \) $t0 - $t7
iClicker Question

If we wanted to support 64 registers, what would change?

(A) $W, A, B \ 32 \rightarrow 64$
(B) $R_w, R_a, R_b \ 5 \rightarrow 6$
(C) $W \ 32 \rightarrow 64, \ R_w \ 5 \rightarrow 6$
(D) A & B only
Tradeoffs

Register File tradeoffs
+ Very fast (a few gate delays for both read and write)
+ Adding extra ports is straightforward
– Doesn’t scale
  e.g. 32Mb register file with 32 bit registers (1M registers)
  Need 32x 1M-to-1 multiplexor and 32x 20-to-1M decoder
  How many logic gates/transistors?

Diagram: 8-to-1 multiplexer with inputs labeled a, b, c, d, e, f, g, h.
goals for today

Clocks

State

• Storing 1 bit

• Storing N bits:
  – Registers
  – Memory
Memory

- Storage Cells + bus
- Inputs: Address, Data (for writes)
- Outputs: Data (for reads)
- Also need R/W signal (not shown)

- $N$ address bits $\rightarrow 2^N$ words total
- $M$ data bits $\rightarrow$ each word $M$ bits
Memory

- Storage Cells + bus
- Decoder selects a word line
- R/W selector determines access type
- Word line is then coupled to the data lines
  note: w/ a tri-state buffer, not a huge mux!
E.g. How do we design a 4 x 2 Memory Module?

(i.e. 4 word lines that are each 2 bits wide)?
E.g. How do we design a 4 x 2 Memory Module?

(i.e. 4 word lines that are each 2 bits wide)?
E.g. How do we design a 4 x 2 Memory Module?

(i.e. 4 word lines that are each 2 bits wide)?
E.g. How do we design a 4 x 2 Memory Module?

(i.e. 4 word lines that are each 2 bits wide)?
MIPS Memory

- 32-bit address
- 32-bit data (but byte addressed)
- Enable + 2 bit memory control (mc)

00: read word (4 byte aligned)
01: write byte
10: write halfword (2 byte aligned)
11: write word (4 byte aligned)
In past semesters we have covered the rest of this lecture in the beginning of the Caches Lecture. So if you have no recollection of covering this, it might be because once again we didn’t.

😊
Each cell stores one bit, and requires 4 – 8 transistors (6 is typical)
SRAM Summary

SRAM

• A few transistors (~6) per cell
• Used for working memory (caches)

• But for even higher density...
Dynamic RAM: DRAM

Dynamic-RAM (DRAM)

- Data values require constant refresh

Each cell stores one bit, and requires 1 transistor.
Dynamic RAM: DRAM

Dynamic-RAM (DRAM)
- Data values require constant refresh

Each cell stores one bit, and requires 1 transistors
DRAM vs. SRAM

Single transistor vs. many gates

- Denser, cheaper ($30/1GB vs. $30/2MB)
- But more complicated, and has analog sensing

Also needs refresh

- Read and write back...
- ...every few milliseconds
- Organized in 2D grid, so can do rows at a time
- Chip can do refresh internally

Hence... slower and energy inefficient
Memory

Register File tradeoffs
  + Very fast (a few gate delays for both read and write)
  + Adding extra ports is straightforward
  – Expensive, doesn’t scale
  – Volatile

Volatile Memory alternatives: SRAM, DRAM, ...
  – Slower
  + Cheaper, and scales well
  – Volatile

Non-Volatile Memory (NV-RAM): Flash, EEPROM, ...
  + Scales well
  – Limited lifetime; degrades after 100000 to 1M writes