Digital Logic Structures
Basic Logic Gates

- **NOT**
  - Input: A
  - Output: \( \overline{A} \)

- **OR**
  - Inputs: A, B
  - Output: A + B

- **NOR**
  - Inputs: A, B
  - Output: \( \overline{A + B} \)

- **AND**
  - Inputs: A, B
  - Output: AB

- **NAND**
  - Inputs: A, B
  - Output: \( \overline{AB} \)

- **XOR**
  - Inputs: A, B
  - Output: A \( \oplus \) B
More Than Two Inputs?

- AND and OR gates can take any number of inputs
  - AND gives 1 if all inputs are 1
  - OR gives 1 if any input is 1
- NAND?? NOR?? XOR??
  - Not associative!
### One-Bit Full Adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C&lt;sub&gt;in&lt;/sub&gt;</th>
<th>C&lt;sub&gt;out&lt;/sub&gt;</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

![Logic Diagram](image-url)
Four-Bit Full Adder

Ripple-carry adder

1. Look ahead
More Logic Structure

• As we start to build more complex structures we need ways to control parts of them
  – To select signals
  – To activate certain outputs
Signal Selection

adder

ANDER

?.

odder or ANDER
Two-Way Multiplexer
Two-Way Multiplexer

2-way multiplexer: the output is equal to one of the two inputs, based on a selector

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Four-Way Multiplexer

- $n$-bit selector and $2^n$ inputs, one output
  - output equals one of the inputs, depending on selector
- “Four-to-one mux”

A, if $S=00$
B, if $S=01$
C, if $S=10$
D, if $S=11$
Two-to-Four Decoder

- $n$ inputs, $2^n$ outputs
  - exactly one output is 1 for each possible input pattern
- Generates a walking-ones pattern
- Uses:
  - Convert memory or register address to a control line
  - Convert an opcode to one of $n$ control lines
  - We will get to this in the LC-3 material
Building functions from logic gates

• Combinational Logic Circuit
  – Output depends only on the current inputs
  – Stateless (memoryless)

• Sequential Logic Circuit
  – Output depends on the sequence of inputs (past and present)
  – Stores information (state) from past inputs

3 + 3 + 3
Combinational vs. Sequential
Two types of “combination” locks

**Combinational**
Success depends only on the values, not the order in which they are set.

**Sequential**
Success depends on the sequence of values (e.g., R-13, L-22, R-3).
Combinational vs. Sequential

• Combinational circuit
  – Always gives the same output for a given set of inputs
  – Example: Adder always generates sum and carry, regardless of previous inputs

• Sequential circuit
  – Remembers previous input
  – Output depends on state and input
Synchronization of Sequential Circuits

• These are real devices and require time to compute.
• If we want proper results we need a way to ensure consistent timing.
• One way of doing this is a clock
  – Repeating signal at certain frequency
  – When you buy a computer this is the number in gigahertz
• All our actions take place in relation to this clock
Memory
- storing 0 or 1
- magnetic core memory

1
0
N/S
SN
Toroids
Speed of light $\approx 3 \cdot 10^8 \text{ m/s}$

If $3.2 \text{ GHz} = 3125 \text{ nanoseconds} = 3.1 \cdot 10^{-10} \text{ seconds} = 3 \cdot 10^{-2}$
D-Flip-Flop (the one for Lab)

- Basic Memory Device
- Stores the value of D when conditions are met and outputs it on Q
- Otherwise Q holds the last value of D

- D-flip-flop is edge-triggered (changes only on the edge of the clock)
- This can be both edges or a single type (up or down)
D-Flip Flop: Timing Diagram

Diagram showing the timing and logic of a D-Flip Flop.
D-Flip-Flop with Write Enable

- Same idea as a Flip-Flop but adds another input.
- Instead of changing on clock edges. You can only change on a clock edge when WE is high.
D-Flip Flop: Timing Diagram (up)
Register

- A register stores a multi-bit value
- Common WE which latches the n-bit value
Memory

Now that we know how to store bits, we can build a memory – a logical $k \times m$ array of stored bits.

Address Space:
number of locations
(usually a power of 2)

Addressability:
number of bits per location
(e.g., byte-addressable)
1kB 1000 bytes
128k bytes

Pixel XL 3GB or Rom
64 GB+5
Memory
READ

Diagram of a memory circuit with inputs WE, Clk, and outputs Q and D.
Memory

WRITE

Diagram of a memory circuit with inputs WE (write enable), D0 and D1, and clocks CK.
State Machine

The basic type of sequential circuit

- Combines combinational logic with storage
- “Remembers” state, and changes output (and state) based on inputs and current state
Task: $x - y + z + <$

Combinational

$A$

$M$

$x - y$

$A$

Memory

$SM$

$x + 0 = x$

$x + y = ++y$

$(x + y) + z = x + y + z$

Costs
Representing Multi-bit Values

- Number bits from right (0) to left (n-1)
  - just a convention -- could be left to right, but must be 
    **consistent**
- Use brackets to denote range:
  \( D[l:r] \) denotes bit \( l \) to bit \( r \), from left to right

\[ A = \overline{0101001101010101} \]

\[ A[14:9] = 101001 \]
\[ A[2:0] = 101 \]

May also see \( A<14:9> \), especially in hardware block diagrams.