

# Introduction to the Architecture of Computers

Professor Hugh C. Lauer

CS-2011, Machine Organization and Assembly Language

(Slides include copyright materials from *Computer Systems: A Programmer's Perspective*, by Bryant and O'Hallaron, and from *The C Programming Language*, by Kernighan and Ritchie)

# Today

- Before electronic computers
- Logic and gates
- Latches and Registers

# Before electronic computers



← Data values represented  
by positions of beads

Arithmetic by  
manual algorithm

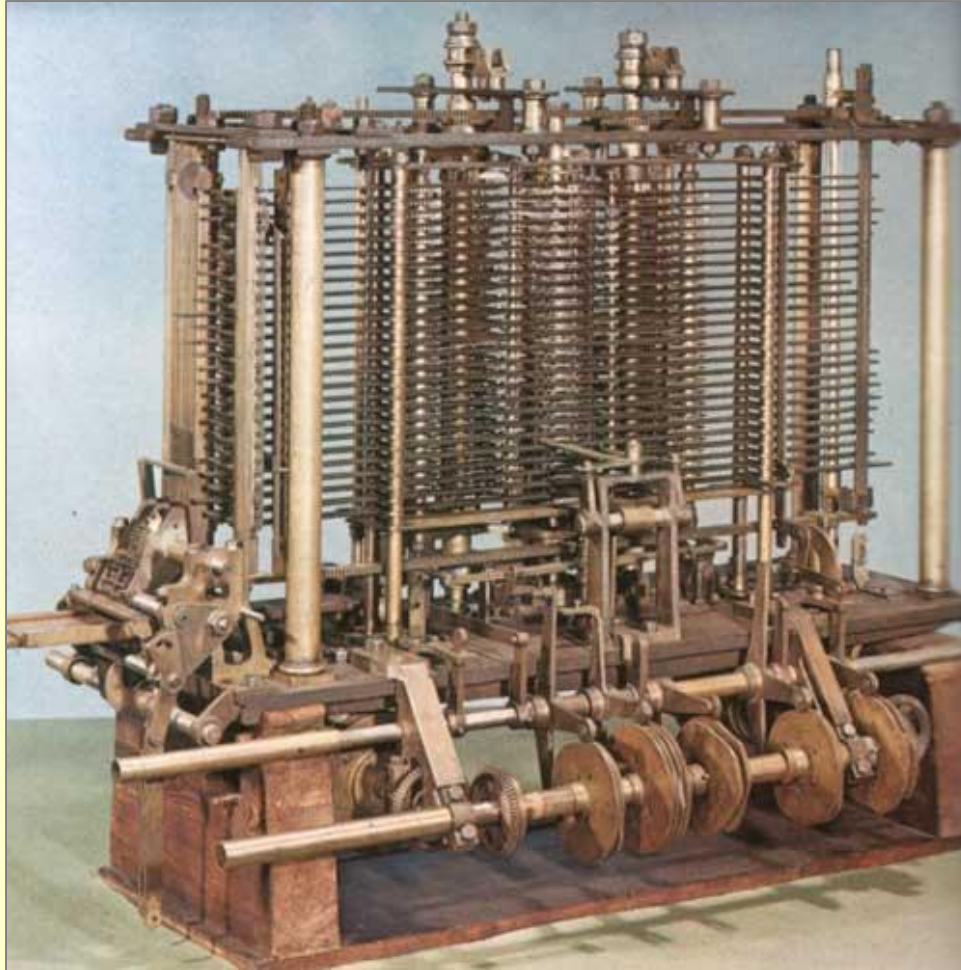


← Data values represented  
by rotational positions of  
wheels and dials

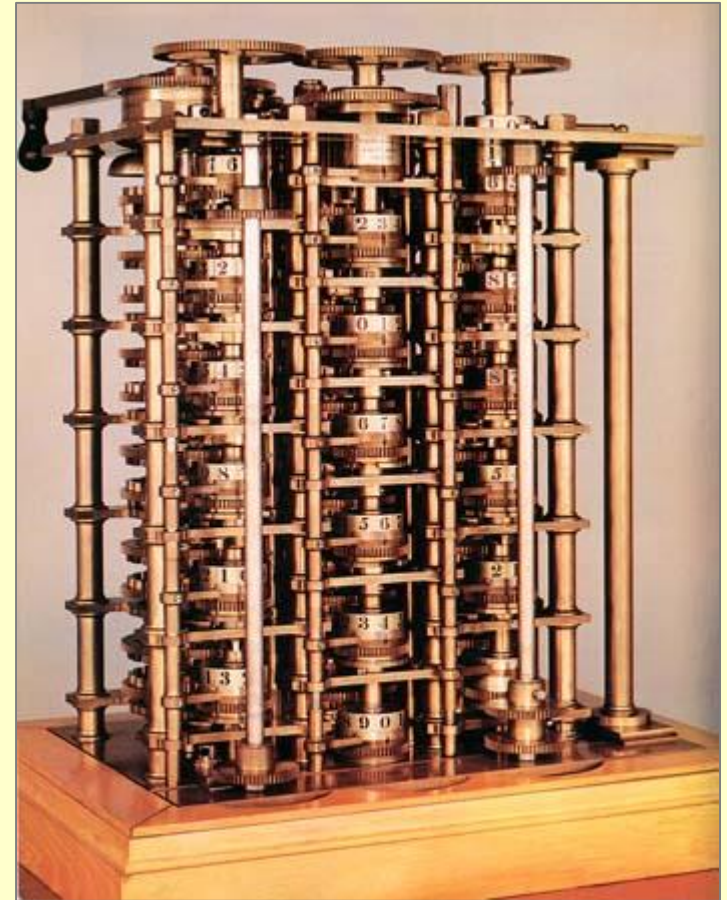


← Arithmetic by  
rotating wheels  
and gears

# Charles Babbage “engines”



Analytical engine

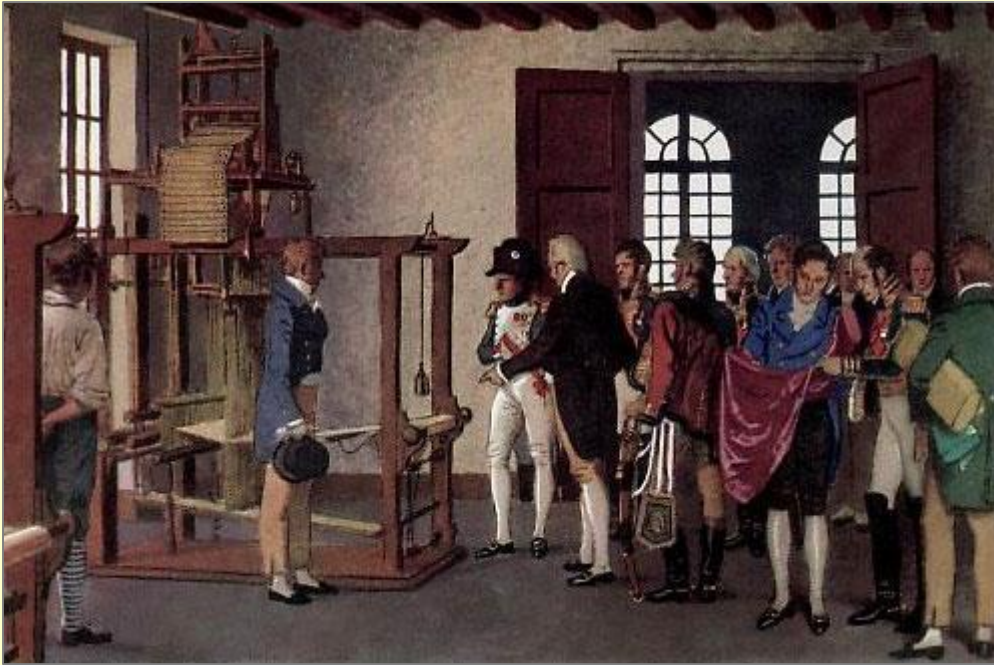


Difference engine

Data values represented  
by rotational positions of  
wheels and dials



# Jacquard Loom



Punched cards for controlling  
patterns of woven cloth



Punched cards were part of  
Babbage's design for data  
entry and program control

# Punched card tabulating equipment



Late 19<sup>th</sup> century



Mid 20<sup>th</sup> century

Data stored in trays (i.e., “files”) of punched cards  
algorithms coded into plug-boards to operate on  
data, punch new cards, etc.

# Today

- Before electronic computers
- **Logic and gates**
- Latches and Registers

**Reading Assignment: §4.2**

# Overview of Logic Design

## ■ Fundamental Hardware Requirements

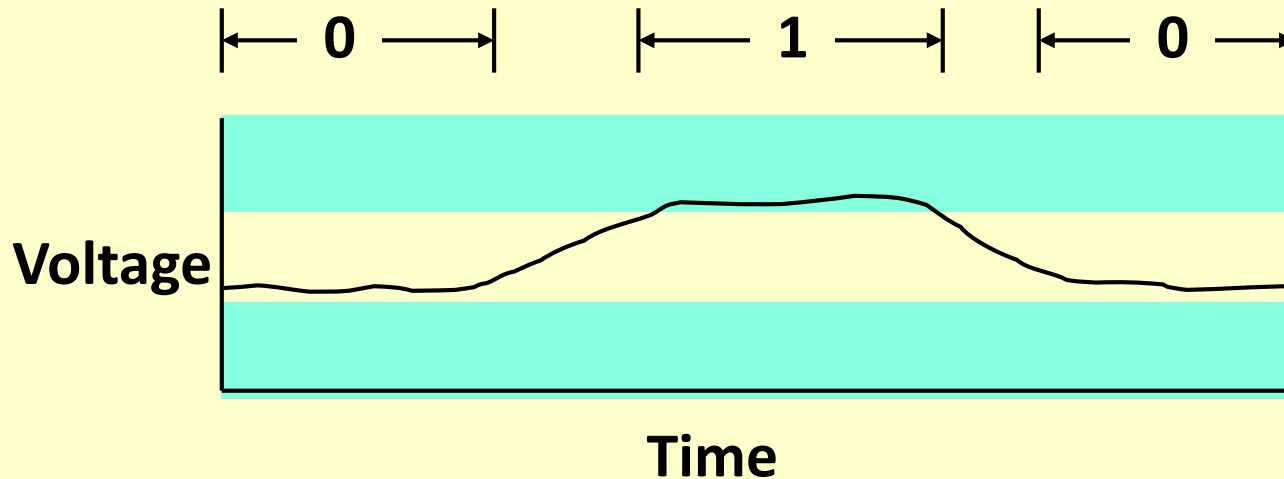
- Communication
  - How to get values from one place to another
- Computation
- Storage

## ■ Bits are Our Friends

- Everything expressed in terms of values 0 and 1
- Communication
  - Low or high voltage on wire
- Computation
  - Compute Boolean functions
- Storage
  - Store bits of information

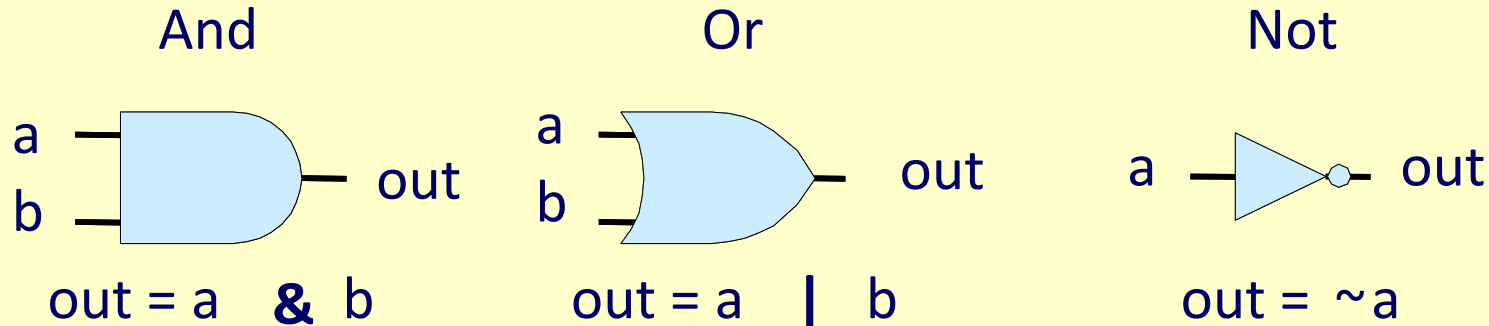


# Digital Signals

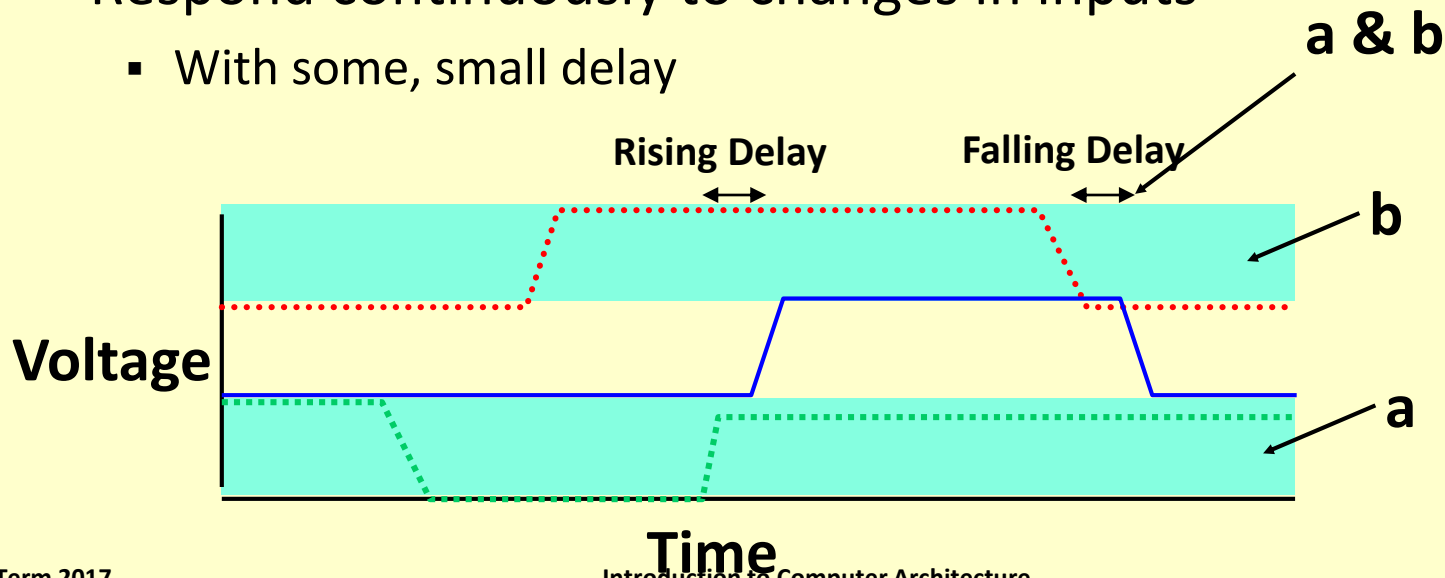


- Use voltage thresholds to extract discrete values from continuous signal
- Simplest version: 1-bit signal
  - Either high range (1) or low range (0)
  - With guard range between them
- Not strongly affected by noise or low quality circuit elements
  - Can make circuits simple, small, and fast

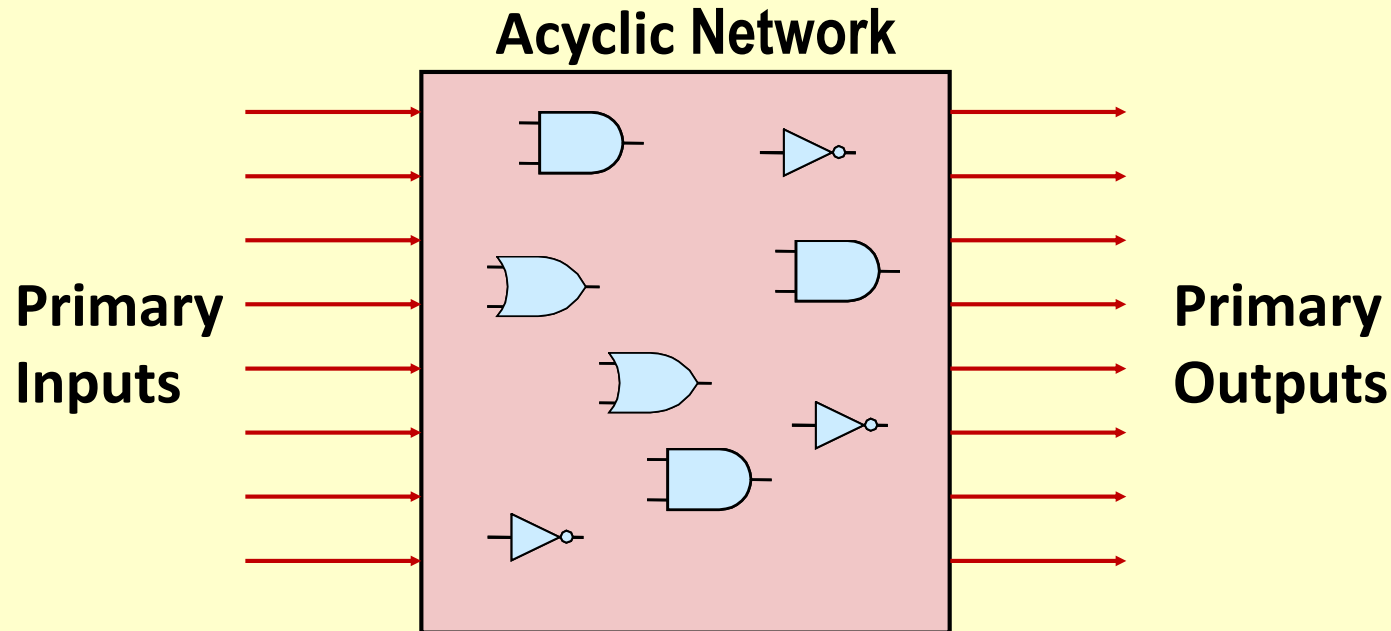
# Computing with Logic Gates



- Outputs are Boolean functions of inputs
- Respond continuously to changes in inputs
  - With some, small delay



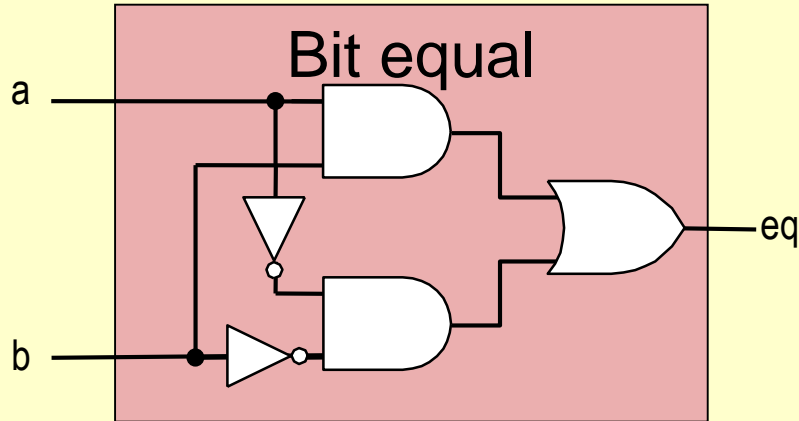
# Combinational Circuits



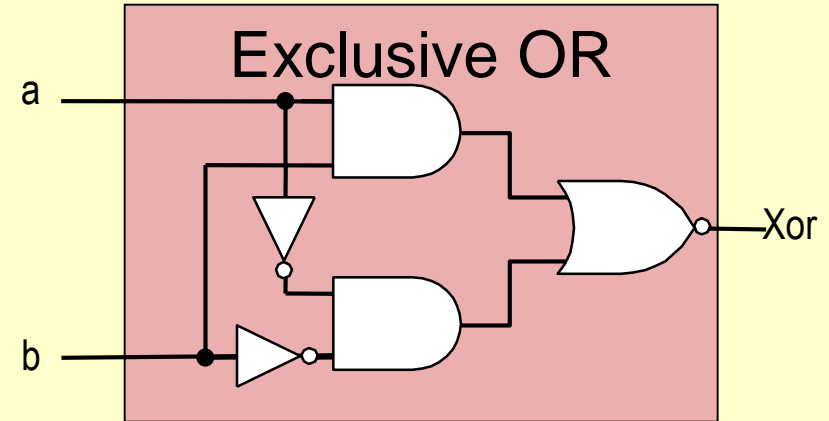
## ■ Acyclic Network of Logic Gates

- Continuously responds to changes on primary inputs
- Primary outputs become (after some delay) Boolean functions of primary inputs

# Bit Equality and Exclusive OR

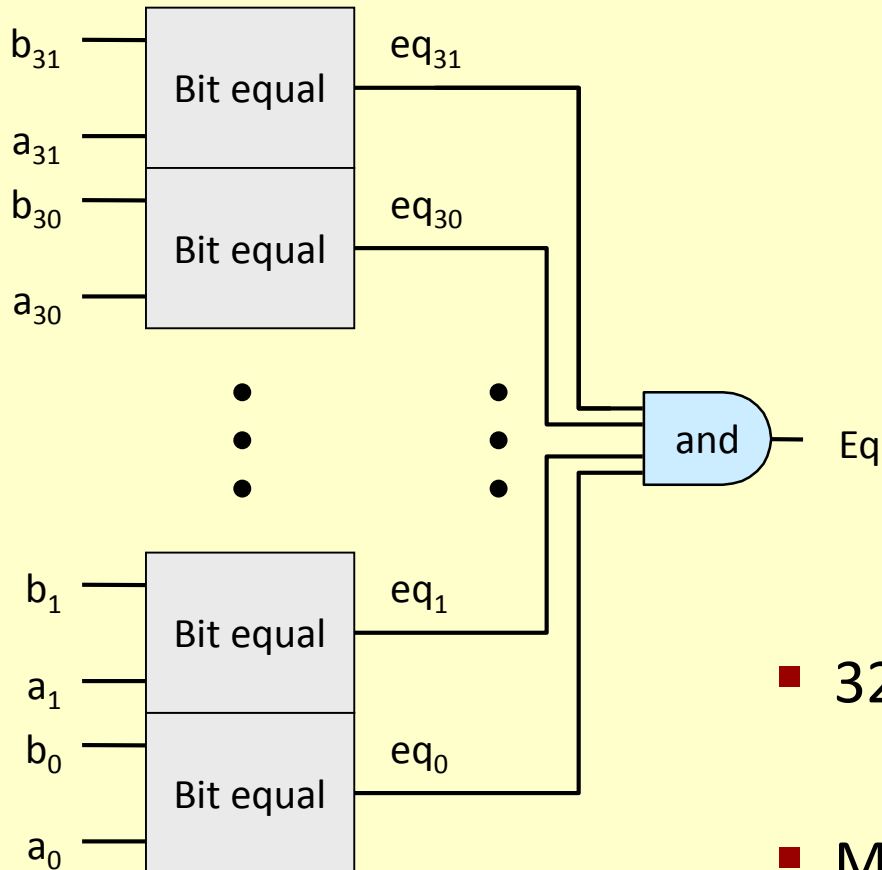


- Generate 1 if a and b are equal

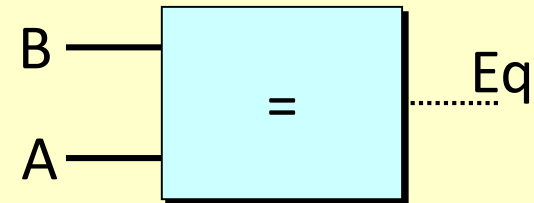


- Generate 1 if a and b are *not* equal

# Word Equality

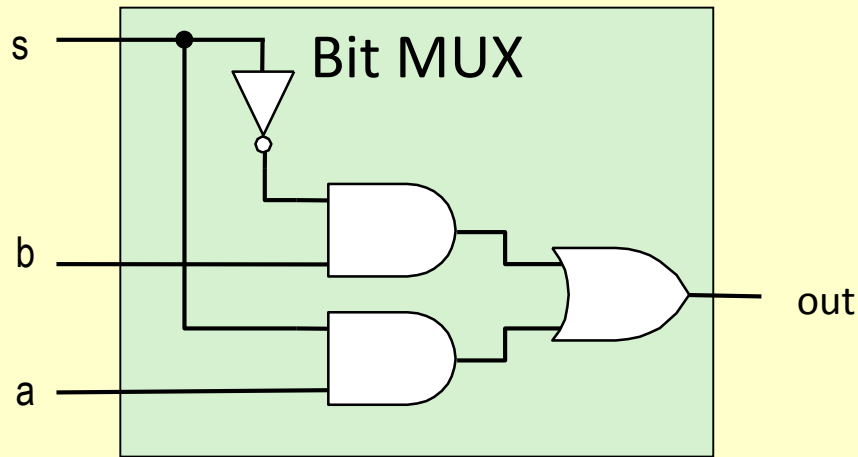


## Word-Level Representation



- 32-bit word size
- May be adapted to any word size

# Bit-Level Multiplexor

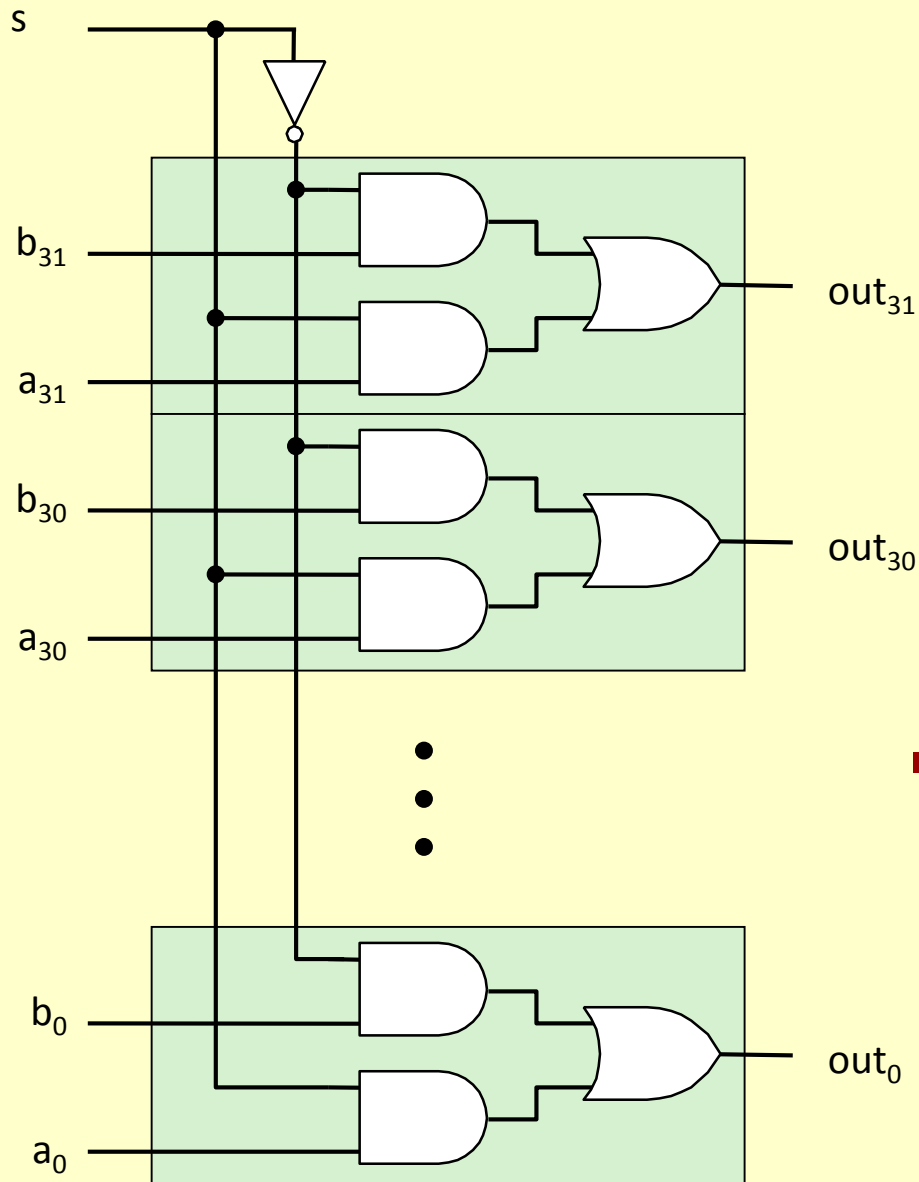


```
bool out = (s&a) || (!s&b)
```

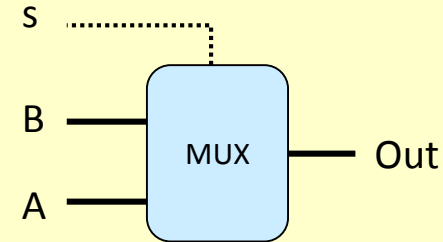
- Control signal *s*
- Data signals *a* and *b*
- Output *a* when *s*=1, *b* when *s*=0



# Word Multiplexor



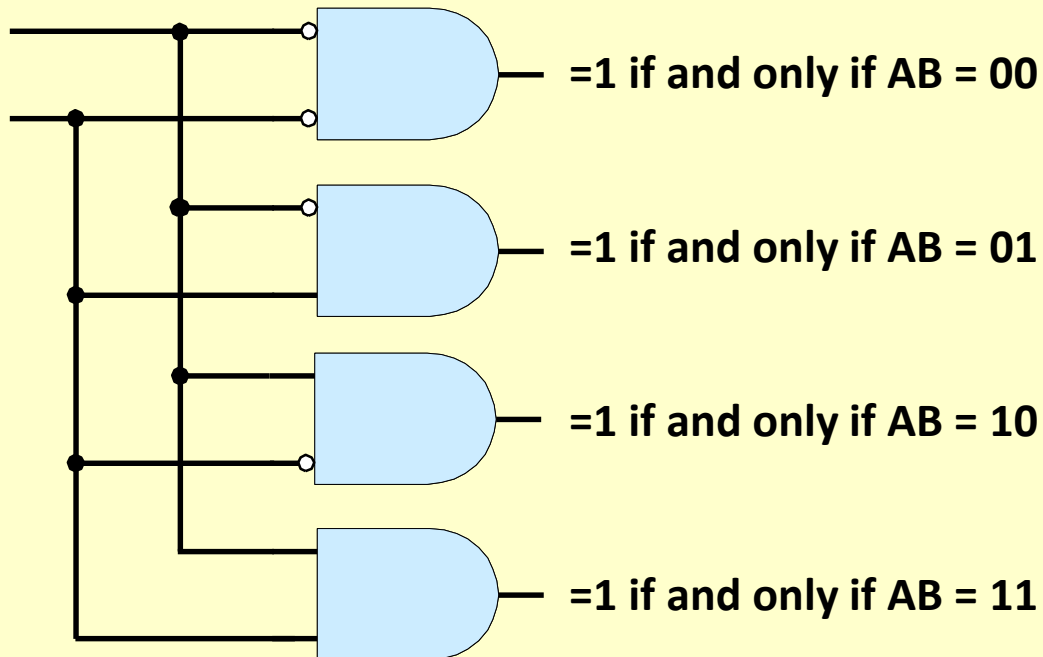
## Word-Level Representation



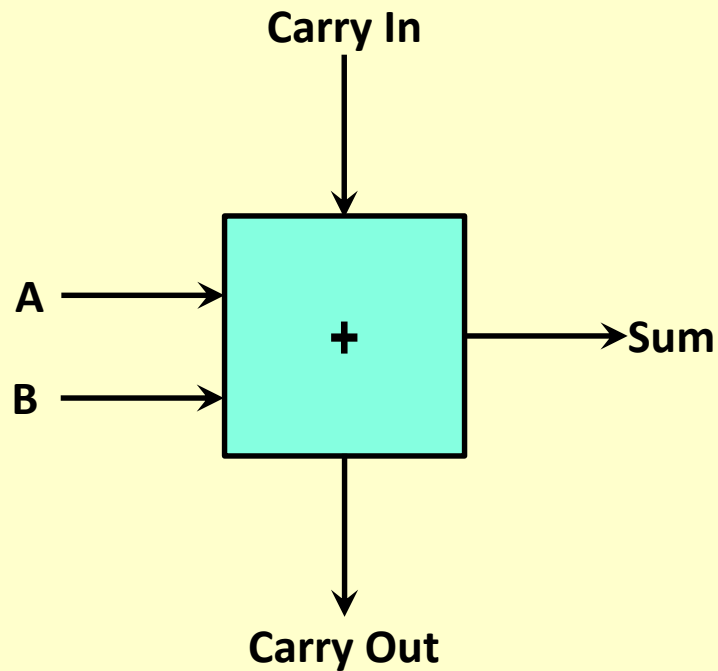
- Select input word  $A$  or  $B$  depending on control signal  $s$

# Decoder

- Opposite of Multiplexor
- Selects one of  $2^n$  outputs from  $n$  inputs



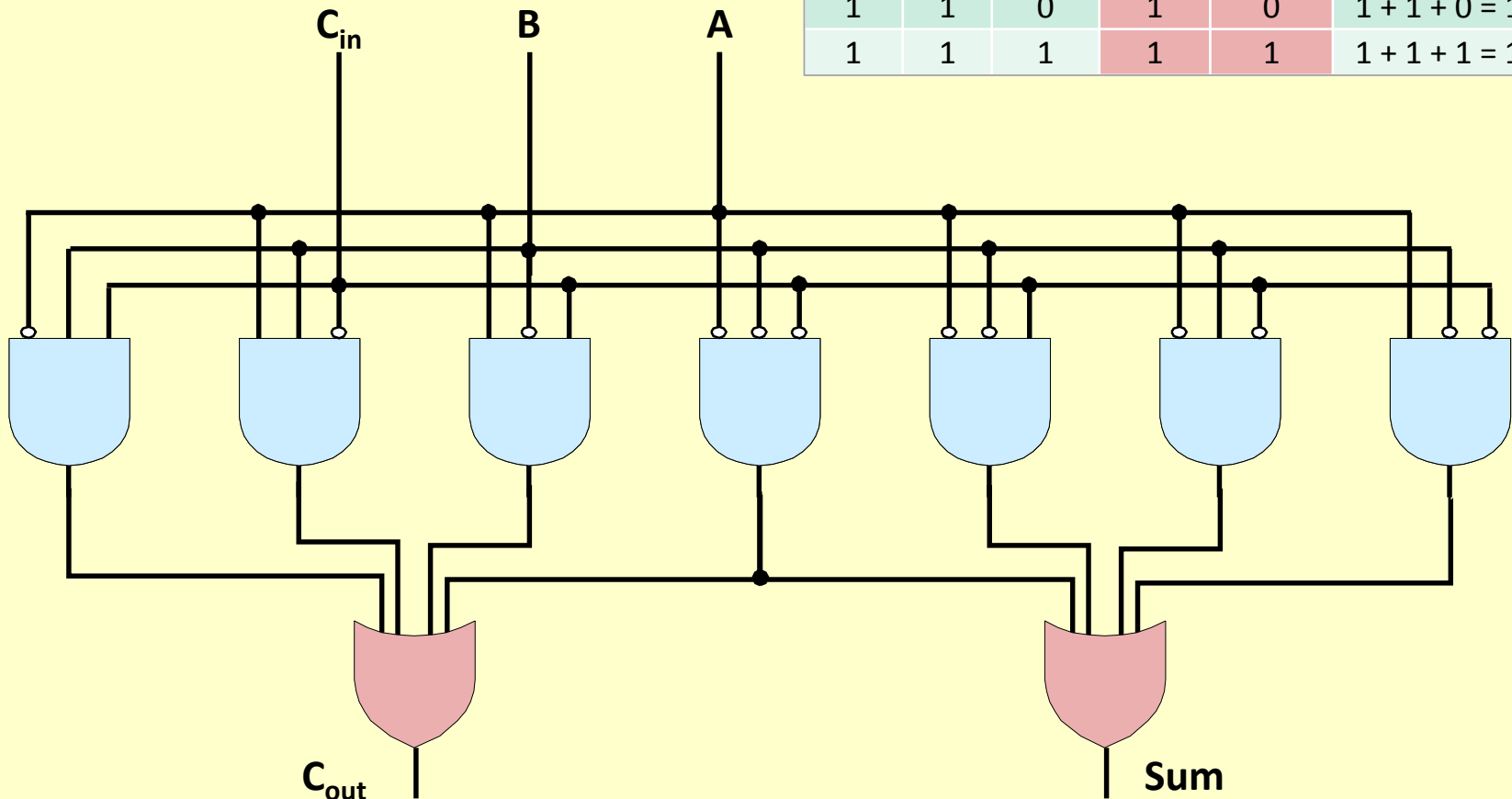
# Single-bit adder



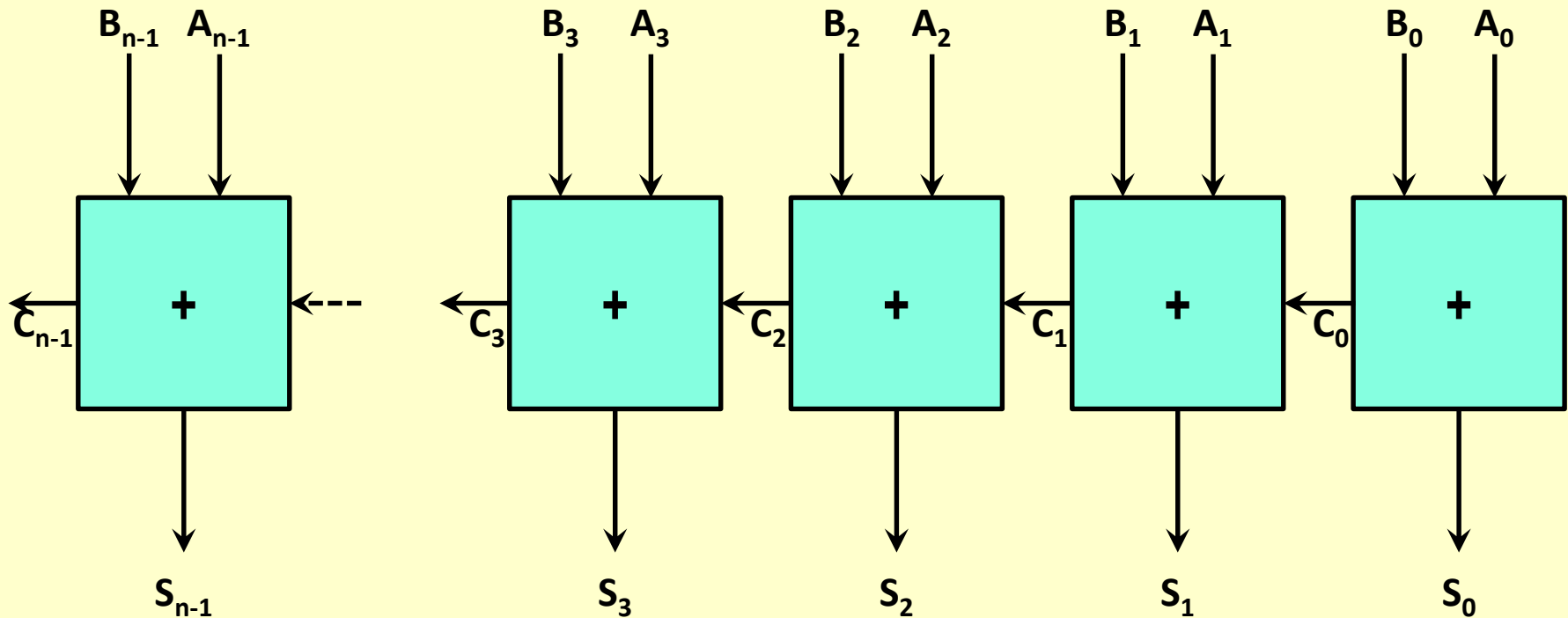
A	B	Carry In	Carry Out	Sum	
0	0	0	0	0	$0 + 0 + 0 = 00_2$
0	0	1	0	1	$0 + 0 + 1 = 01_2$
0	1	0	0	1	$0 + 1 + 0 = 01_2$
0	1	1	1	0	$0 + 1 + 1 = 10_2$
1	0	0	0	1	$1 + 0 + 0 = 01_2$
1	0	1	1	0	$1 + 0 + 1 = 10_2$
1	1	0	1	0	$1 + 1 + 0 = 10_2$
1	1	1	1	1	$1 + 1 + 1 = 11_2$

# Single-bit adder (cont.)

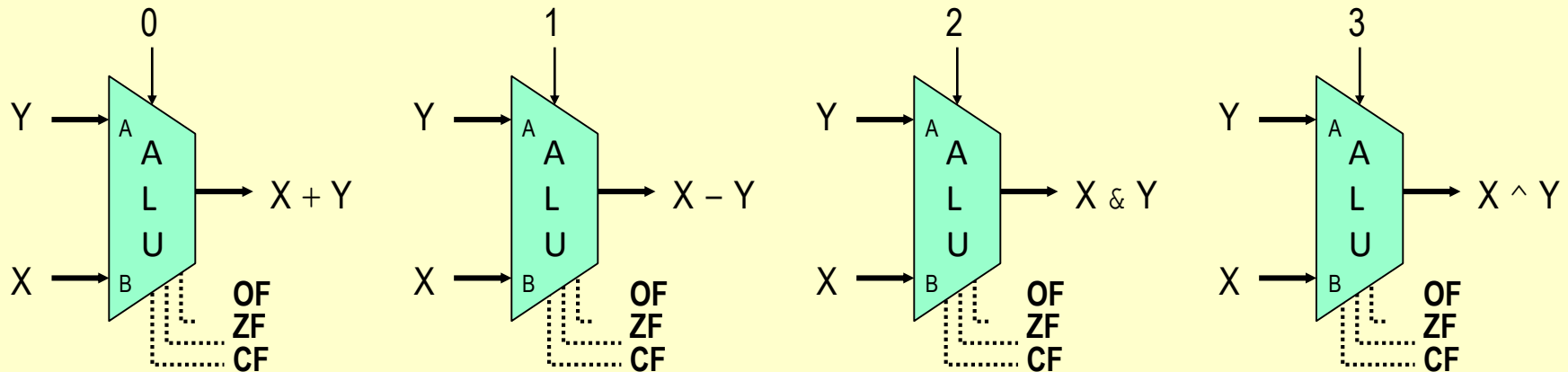
A	B	Carry In	Carry Out	Sum	
0	0	0	0	0	$0 + 0 + 0 = 00_2$
0	0	1	0	1	$0 + 0 + 1 = 01_2$
0	1	0	0	1	$0 + 1 + 0 = 01_2$
0	1	1	1	0	$0 + 1 + 1 = 10_2$
1	0	0	0	1	$1 + 0 + 0 = 01_2$
1	0	1	1	0	$1 + 0 + 1 = 10_2$
1	1	0	1	0	$1 + 1 + 0 = 10_2$
1	1	1	1	1	$1 + 1 + 1 = 11_2$



# Multi-bit adder



# Arithmetic Logic Unit (single-bit example)



- Combinational logic
  - Continuously responding to inputs
- Control signal selects function computed
  - Corresponding to four basic arithmetic/logical operations
- Also computes values for condition codes

**Figure 4.15**  
(Not in our textbook)



# Modern Arithmetic-Logic Unit (ALU)

## ■ Combines

- Add
- Subtract
- And
- Or
- Not
- Xor
- Equality
- <
- >
- <<
- >>
- ...

## ■ Outputs

- Result
- CF — Carry flag
- ZF — Zero flag
- SF — Sign flag
- OF — Overflow flag

Result developed within one cycle (300 ps)

Conspicuously absent:— multiplication & division!

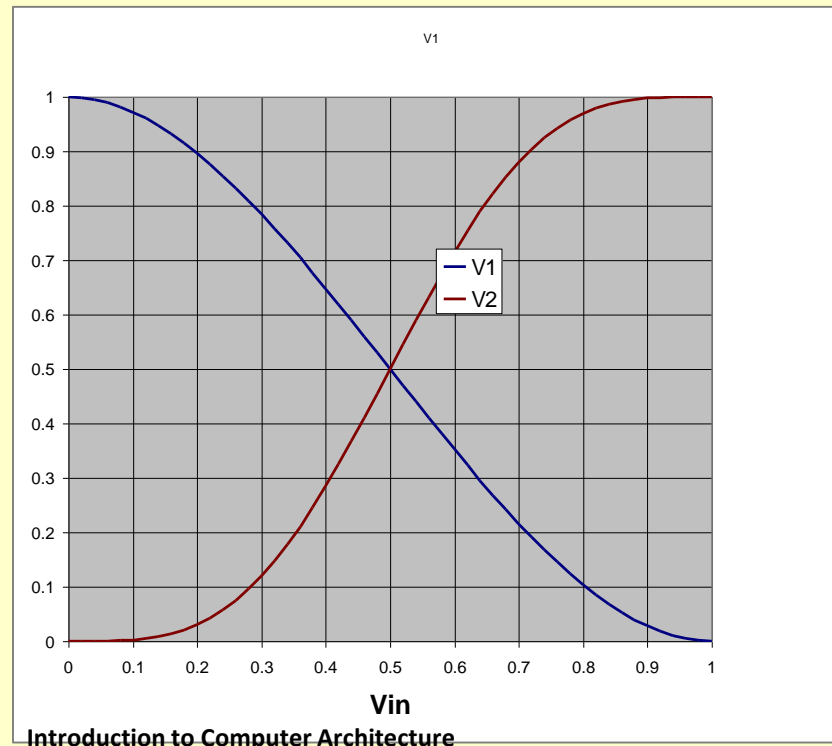
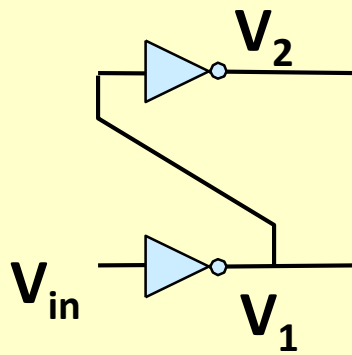
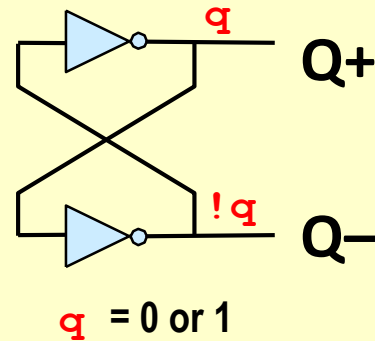
# Questions?

# Today

- Before electronic computers
- Logic and gates
- **Latches and Registers**

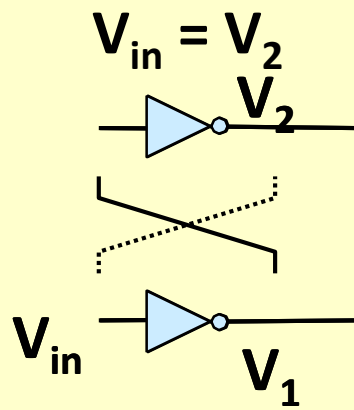
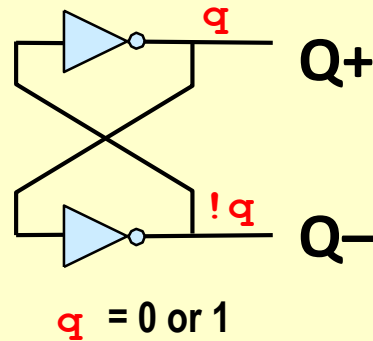
# Storing 1 Bit

## Bistable Element

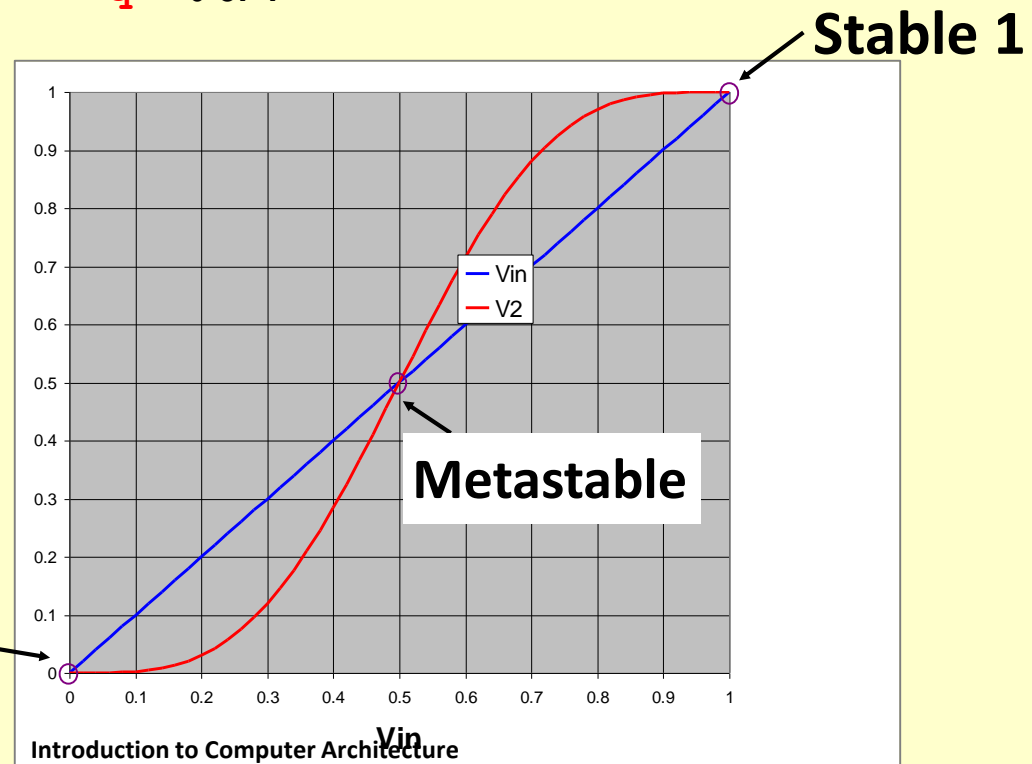


# Storing 1 Bit (continued)

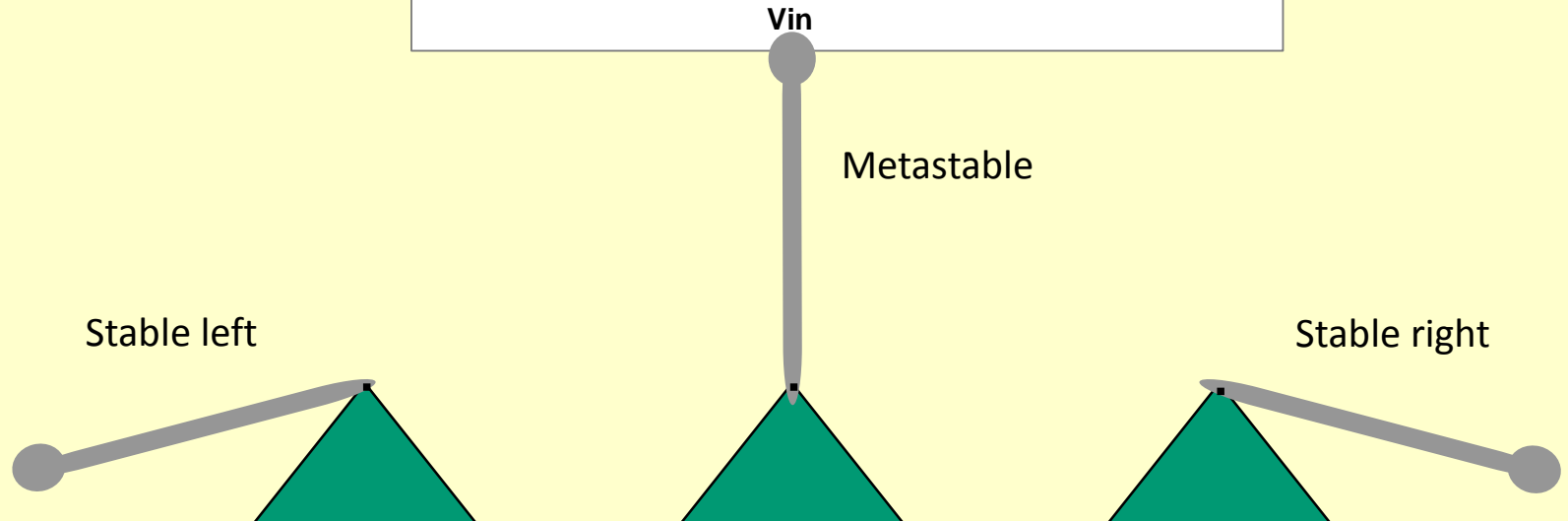
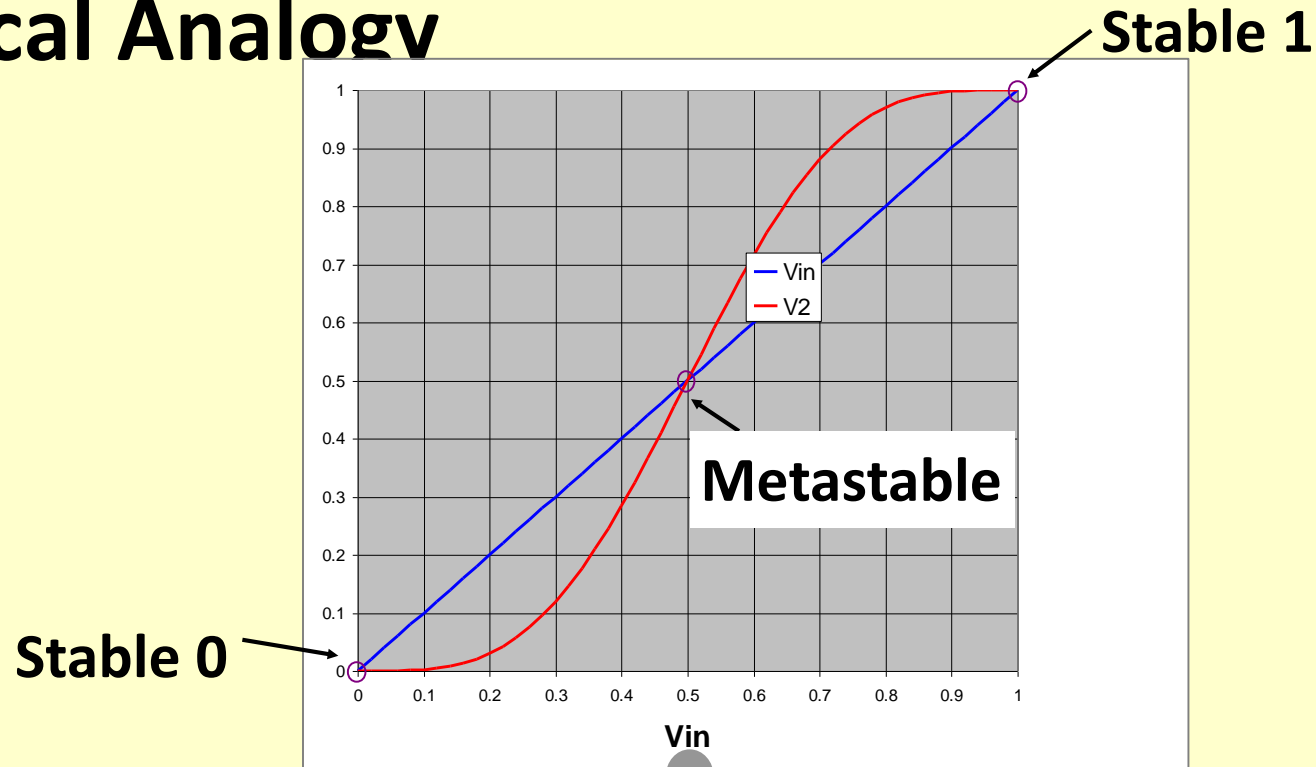
## Bistable Element



Stable 0



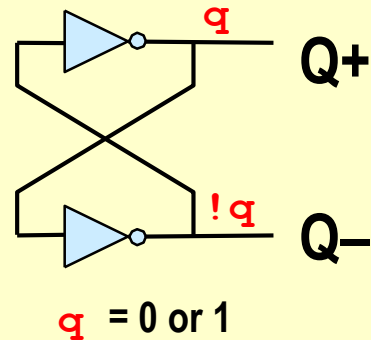
# Physical Analogy



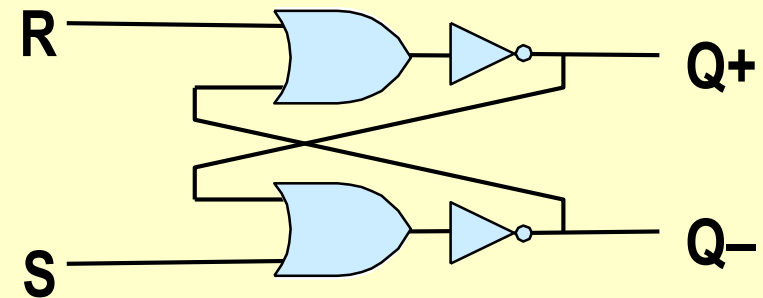


# Storing and Accessing 1 Bit

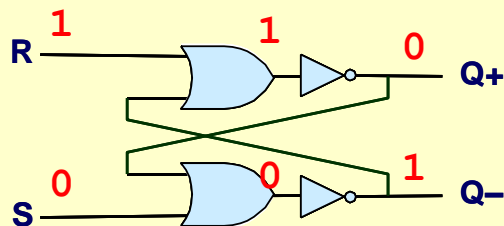
## Bistable Element



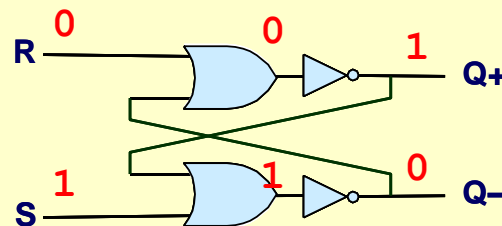
## R-S Latch



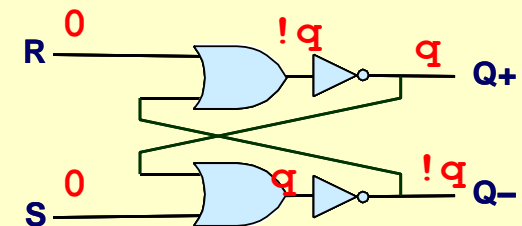
## Resetting



## Setting

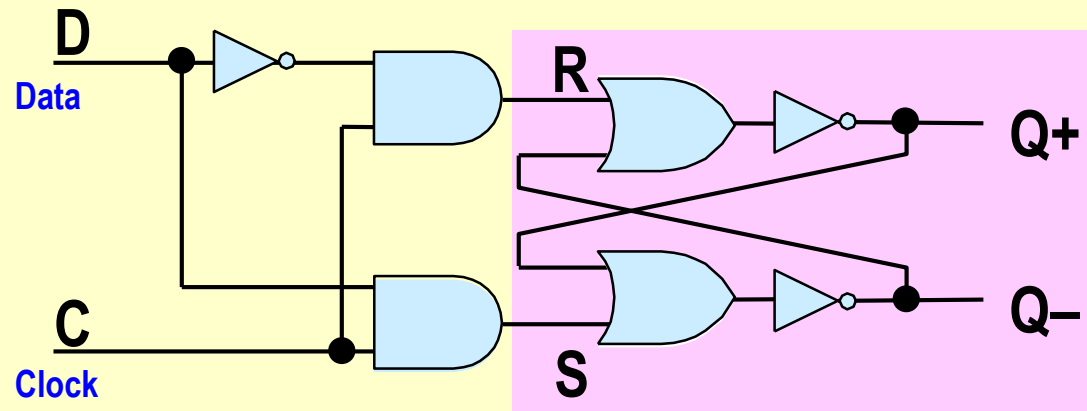


## Storing

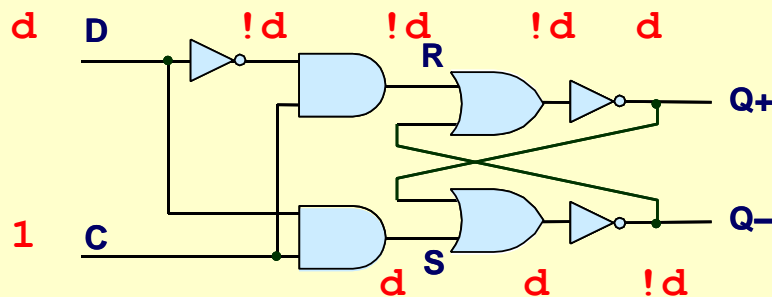


# 1-Bit Latch

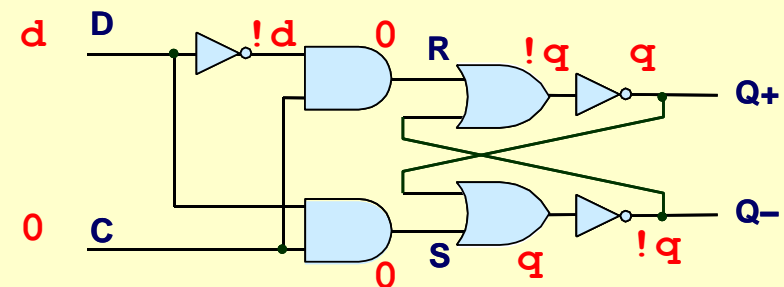
D Latch



Latching



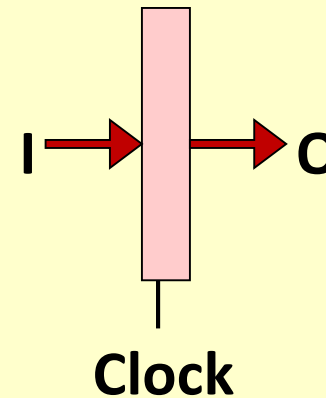
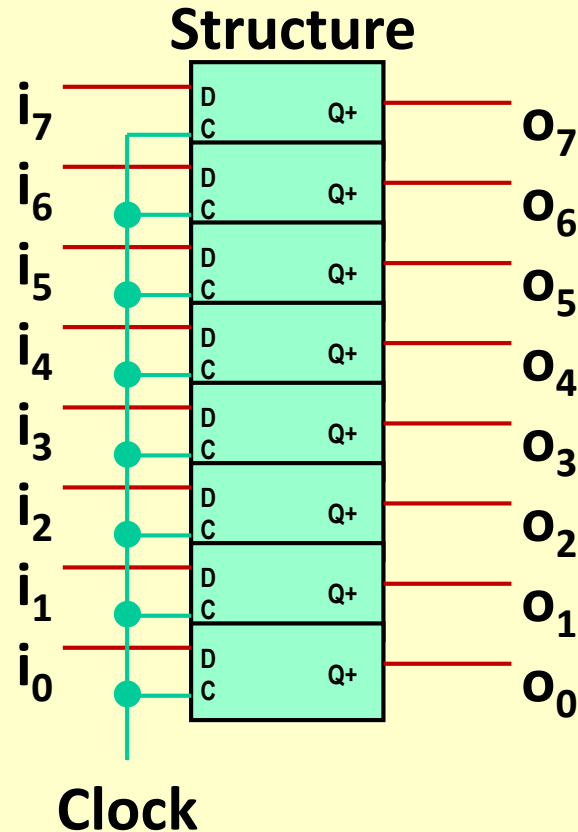
Storing



# Definition — *Clock*

- A periodic signal consisting of an alternating sequence of ones and zeros at regular intervals
  - E.g., 333 picoseconds in a modern 3 GHz processor
  
- Purpose:— to control *when* to capture the result of a logic circuit
  - E.g., an ALU consisting of AND's, OR's, and NOT's
  - E.g., a register or memory or other device

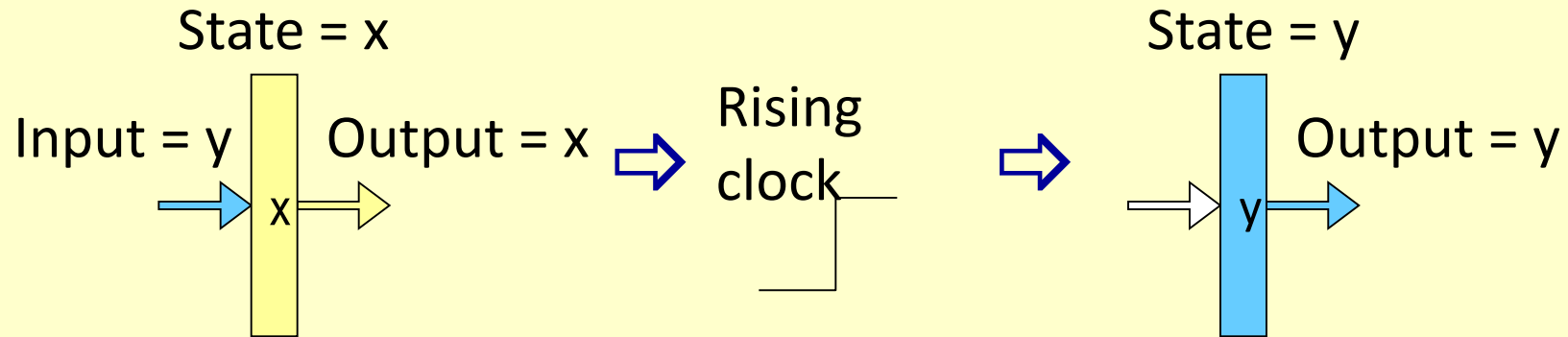
# Register



- Stores word of data
  - Different from *program registers* seen in assembly code
- Collection of edge-triggered latches
- Loads input on rising edge of clock



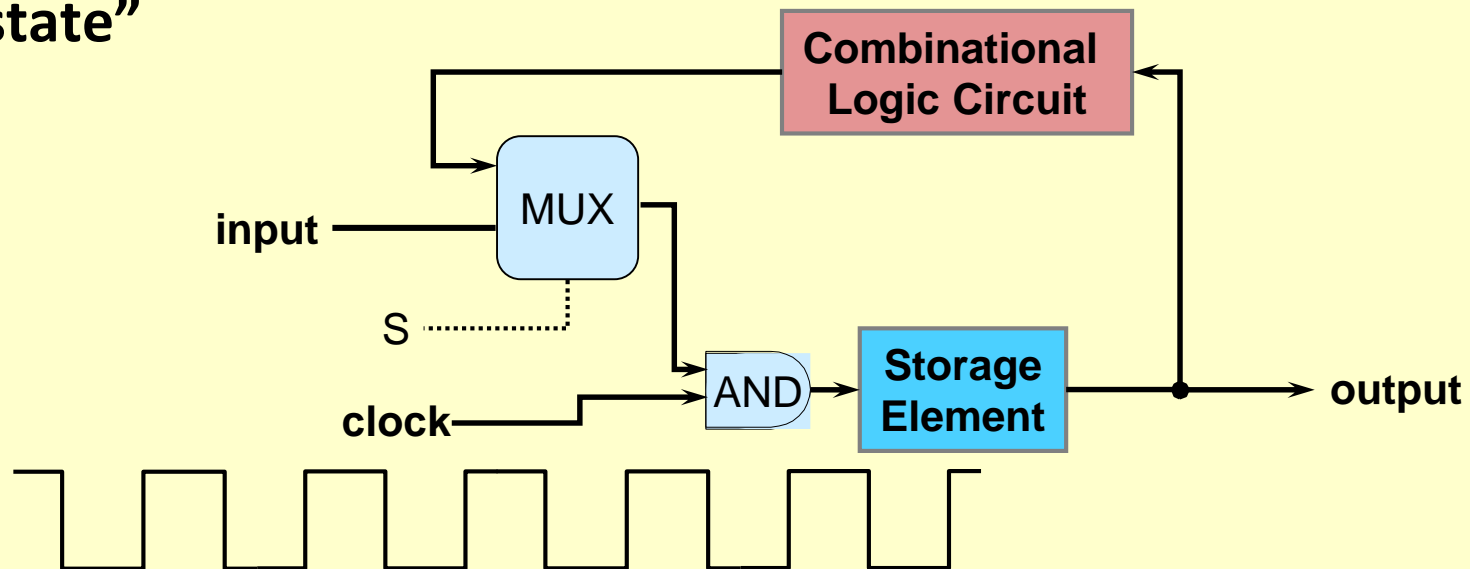
# Register Operation



- Stores data bits
- For most of time acts as barrier between input and output
- As clock rises, loads input

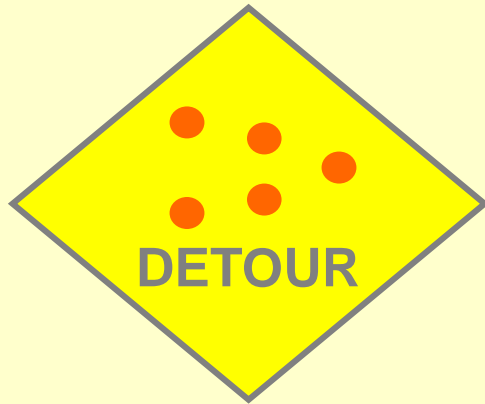
# (Finite) State Machine

- A register or latch of  $n$  bits representing the “state” of the circuit
- An acyclic network of combinatorial logic to compute a new value of  $n$  bits based on the existing value of  $n$  bits
- A clock signal to effect the update of the “state” to a new “state”

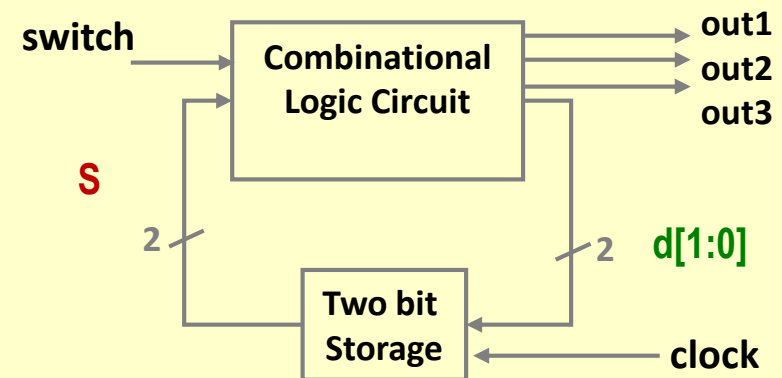
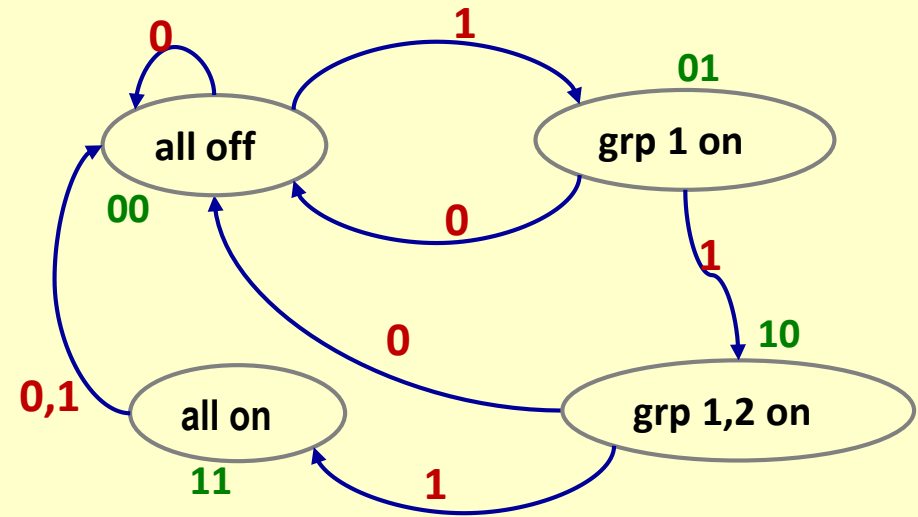




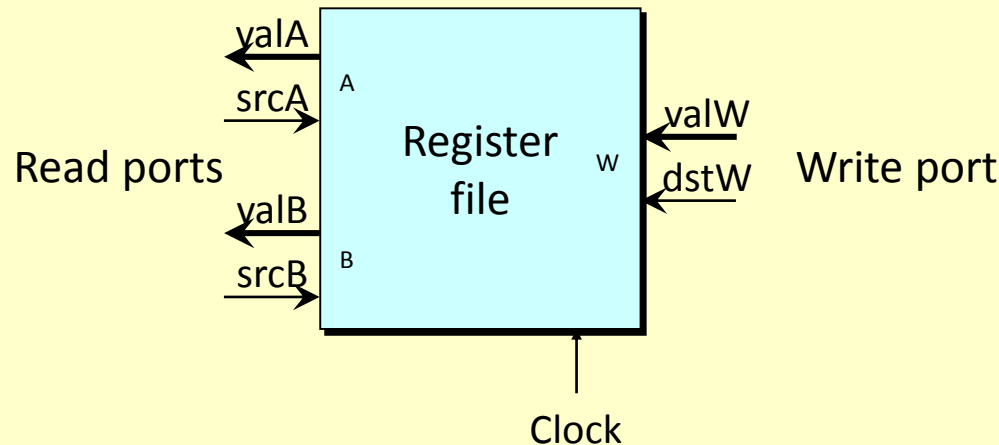
# Finite State Machine Example



- Three groups of lights to be lit in a sequence: group 1 on, groups 1 & 2 on, all groups on, all off.
- The lights are on only if the main switch is on.
- Four states: so we need two bits to identify each state.



# Register File



- Stores multiple words of memory
  - Address input specifies which word to read or write
- Register file
  - Holds values of program registers
  - `%rax`, `%rsp`, etc.
  - Register identifier serves as address
    - ID 15 (0xF) implies no read or write performed
- Multiple Ports
  - Can read and/or write multiple words in one cycle
    - Each has separate address and data input/output



# Summary

- **Data values stored as bits**
  - On wires, in memory cells, etc.
- ***Gates* are logic elements that combine values of bits to produce other bits**
  - And, Or, Xor, addition, subtraction, comparison, etc.
- **Latches capture bit values on wires and keep them until reset**
  - So long as power stays on
- **Setting of latches is triggered by a clock, which allow data into the latches only when the results of combinatorial logic elements has stabilized.**

**Reading Assignment: §4.2**

# Questions?