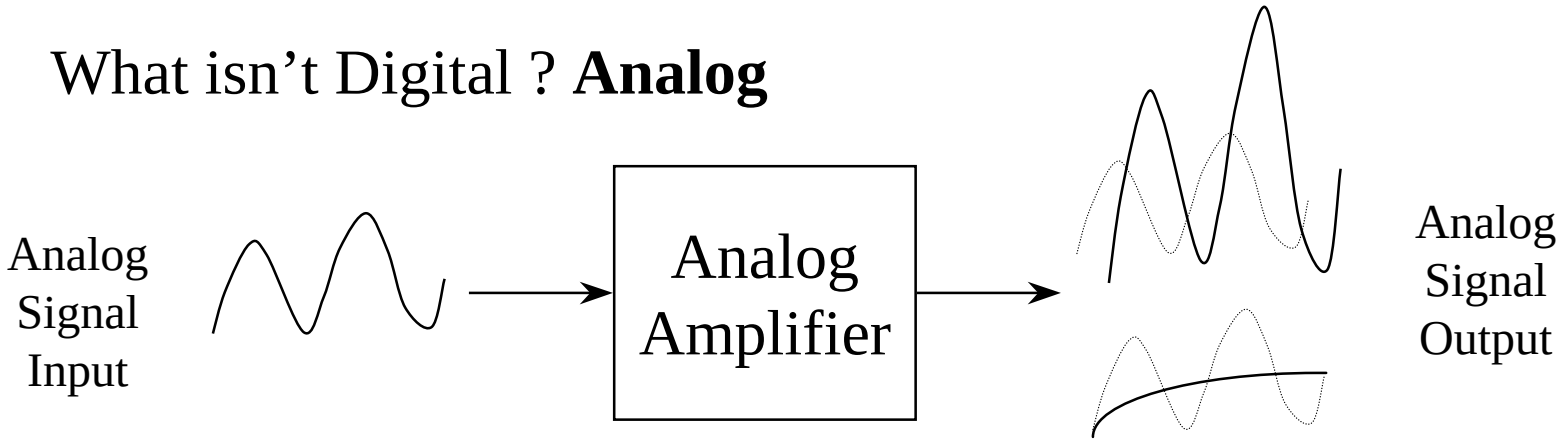
CSE370: Introduction to Digital Design

- Course staff
  - Larry Arnstein, ...
- Course web
  - www.cs.washington.edu/education/courses/370/99au/
- This week: Introduction
  - Keywords "digital" and "design"
  - Examples and basics
- Content and Grading
  - Weekly homeworks = 40%
  - Class participation = 10%
  - Midterm = 25%
  - Final (comprehensive) = 25%
- Software: DesignWorks

What is Digital?

What isn't Digital ? **Analog**



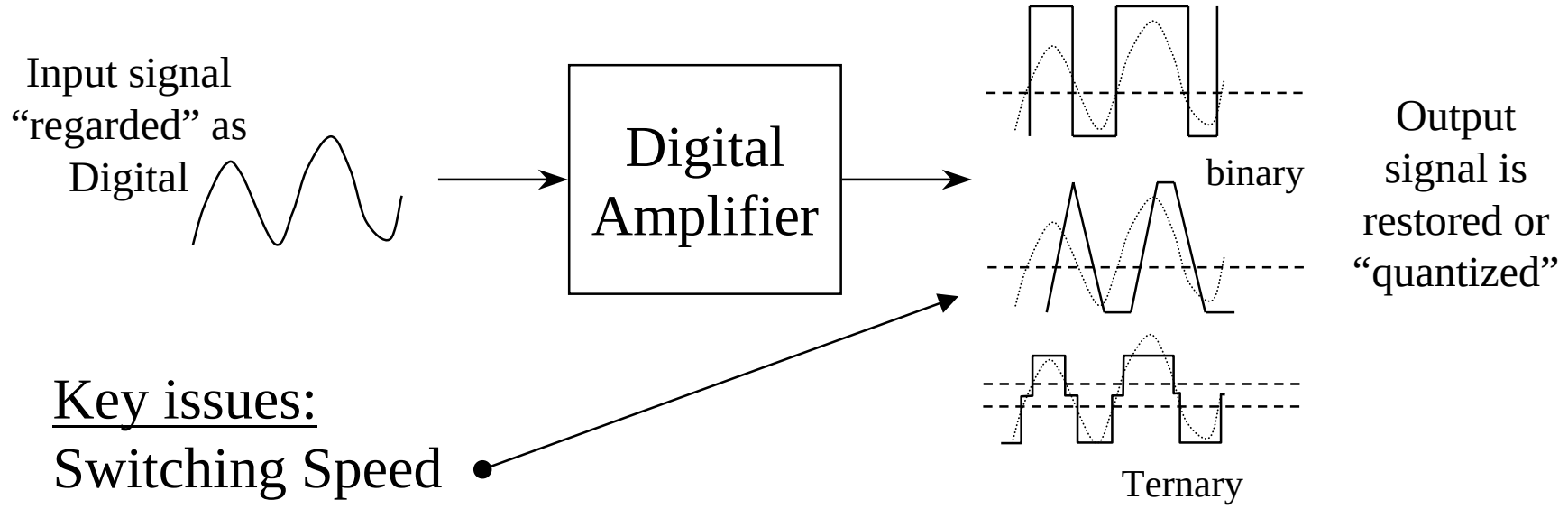
Key Design Issue:
Frequency response

Examples:
- Bullhorn
- Noise Filter

Strengths
Compact, and Error Tolerant
One wire can carry a Symphony

Weaknesses
"Lossy"-- Subject to distortion, good for media, not computation

What's Digital



Key issues:
Switching Speed
and Delay

Weaknesses

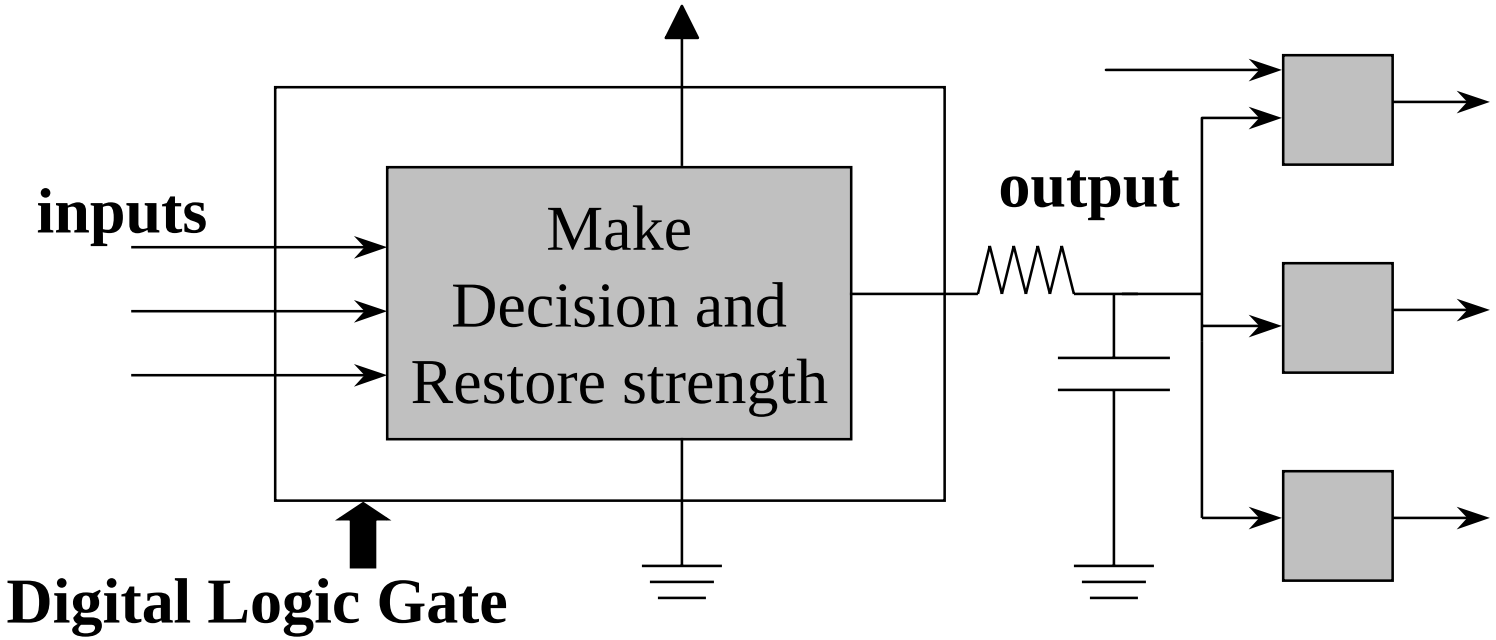
- **Inefficient**
- **Rely on parallelism to get things done**
- **(Sega Dreamcast is a 128 Bit Machine)**
- **Error intolerant**

Strengths

- **Lossless --Only loss is quantization.**
- **Great for computation!**
- **Highly reproducible and shrinkable**

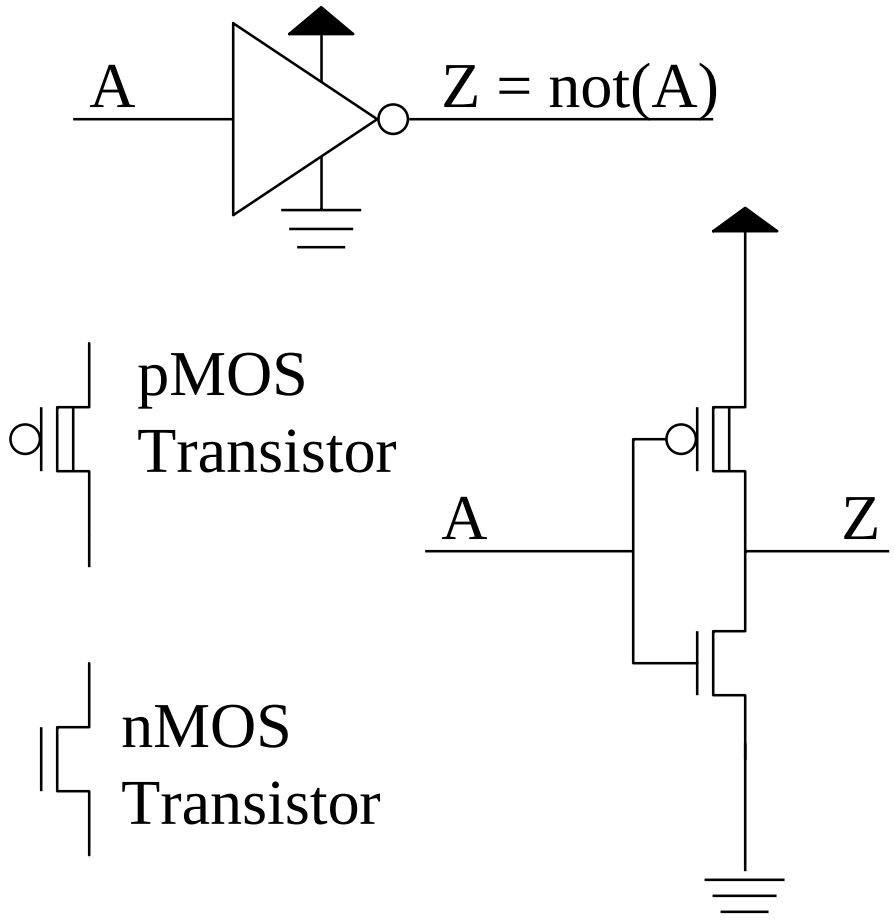
The Digital Gate (Electronic)

- **Value of output determined by values of inputs**
- **The output strength (voltage or current) must be high enough to activate several similar devices.**



Allows complex computation with no degradation of information

An Example: The CMOS Inverter



nMOS + pMOS = CMOS

MOS Transistors

The resistance of the "channel" is determined by the voltage of the gate.

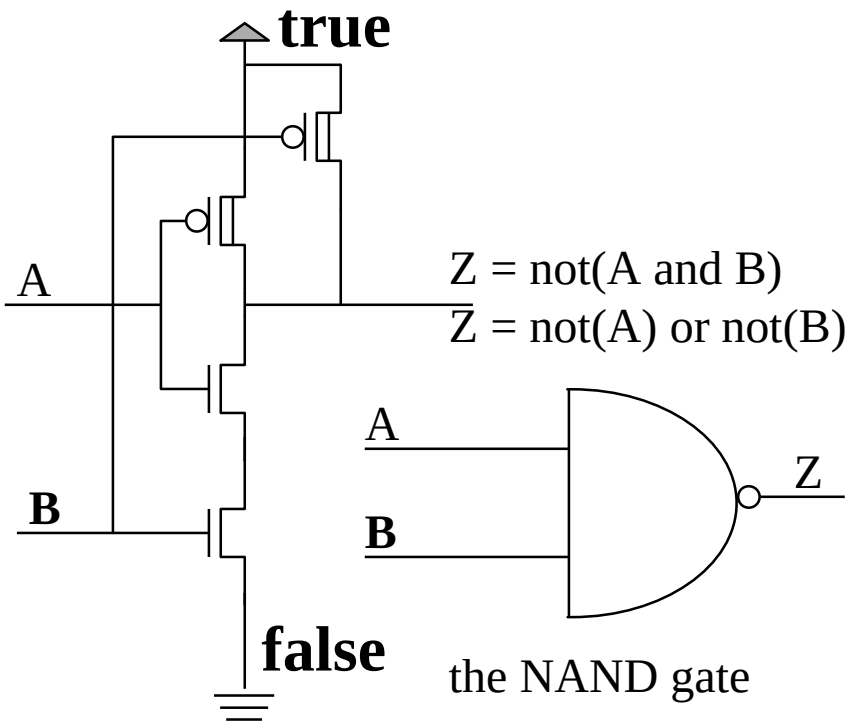
A positive voltage on the input reduces the resistance of the nMOS channel, and increases the resistance of the pMOS channel.

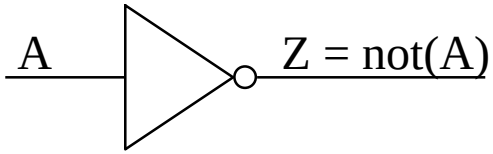
Key properties:

Very small input current (leakage only), very high "gain" amplifier

Weak input → Strong Output

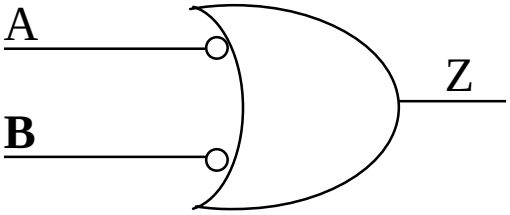
Basic Units of Digital Computation in CMOS

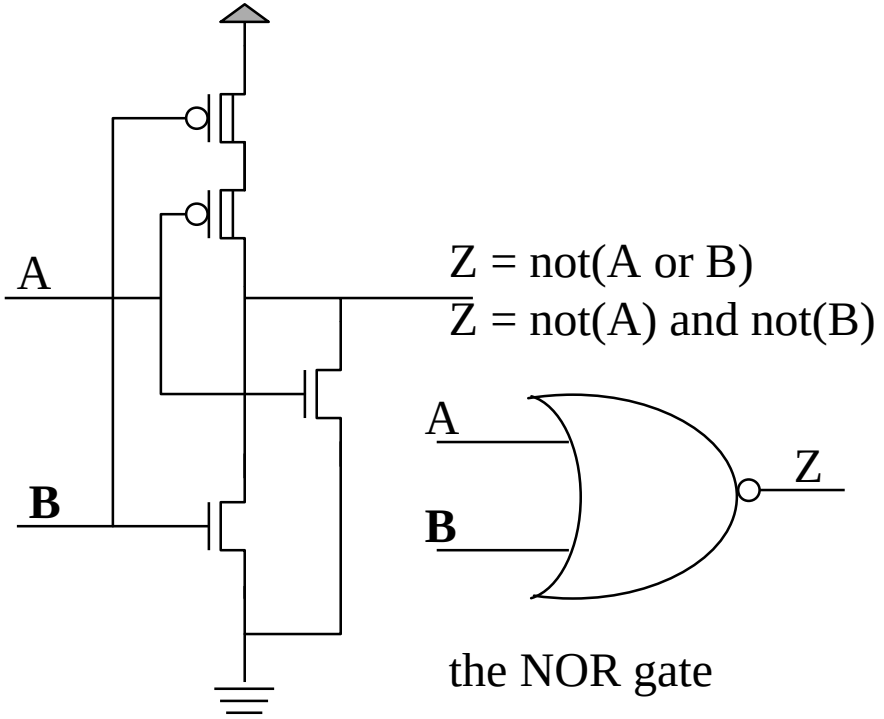




the NOT gate

the NAND gate





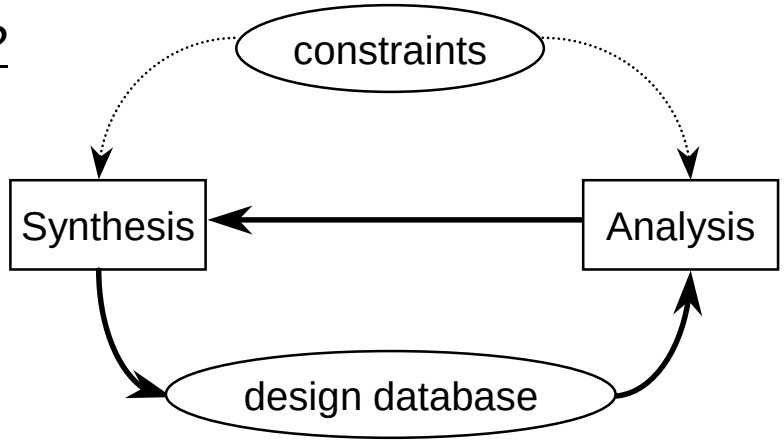
the NOR gate

Other forms of digital hardware

- First Digital System: Molecular Structure such as DNA
- Beads -- the abacus (~500 BC)
- Mechanisms -- the position of a gear or shaft (1600's -- Babbage 1800's)
- Electromechanisms -- relays (Zuse 1900's)
- Electrical Currents (Tubes, Bipolar Transistors 1930's-1970's)
- **Voltages (CMOS 1980's-present)**
- Fiber Optics -- Light or Light off
- Dynamic RAM -- Charged or Discharged capacitor (on MOS gate)
- Nonvolatile memory (erasable) -- Trapped electrons or no trapped electrons
- Programmable ROM -- Fuses
- Bubble memory -- Presence or absence of a magnetic bubble
- Magnetic disk -- Direction of magnetic flux
- Compact disc -- the Presence or absence of a pit

What is design (cont'd)?

- What is engineering design?



- What is logic design?
  - Constraints =

    Function, Cost, Performance, Power, Servicability, Safety, Project Schedule, etc.
  - Database =

    Schematic, Layout, Algorithm, Mechanical Drawing -- whatever the next person in line needs (mfg'r, technician).
  - Analysis = S

    imulate it, Cost it out, Total up the power, Build a prototype, Prove that it works (formal or otherwise)
  - Synthesis = Making engineering decisions based on available information in a reasonable amount of time + Lots of creativity!

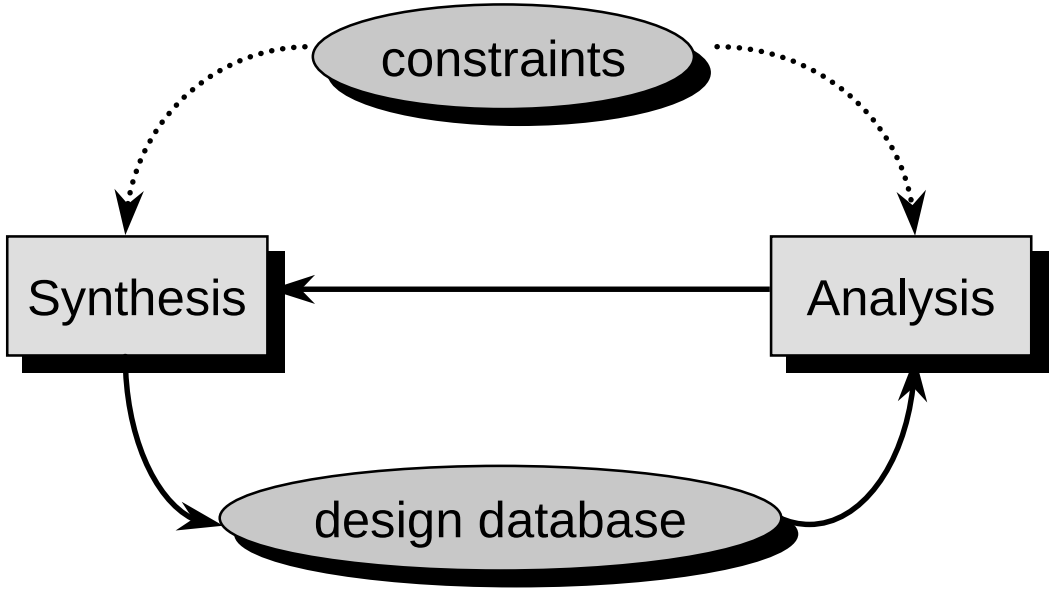
What can go wrong?

Inefficiency due to*:*

- ***lack of automation***
- ***lack of design re-use***

Iteration due to:

- ***inexperience (expertise intensive)***
- ***sub-optimal tool performance***



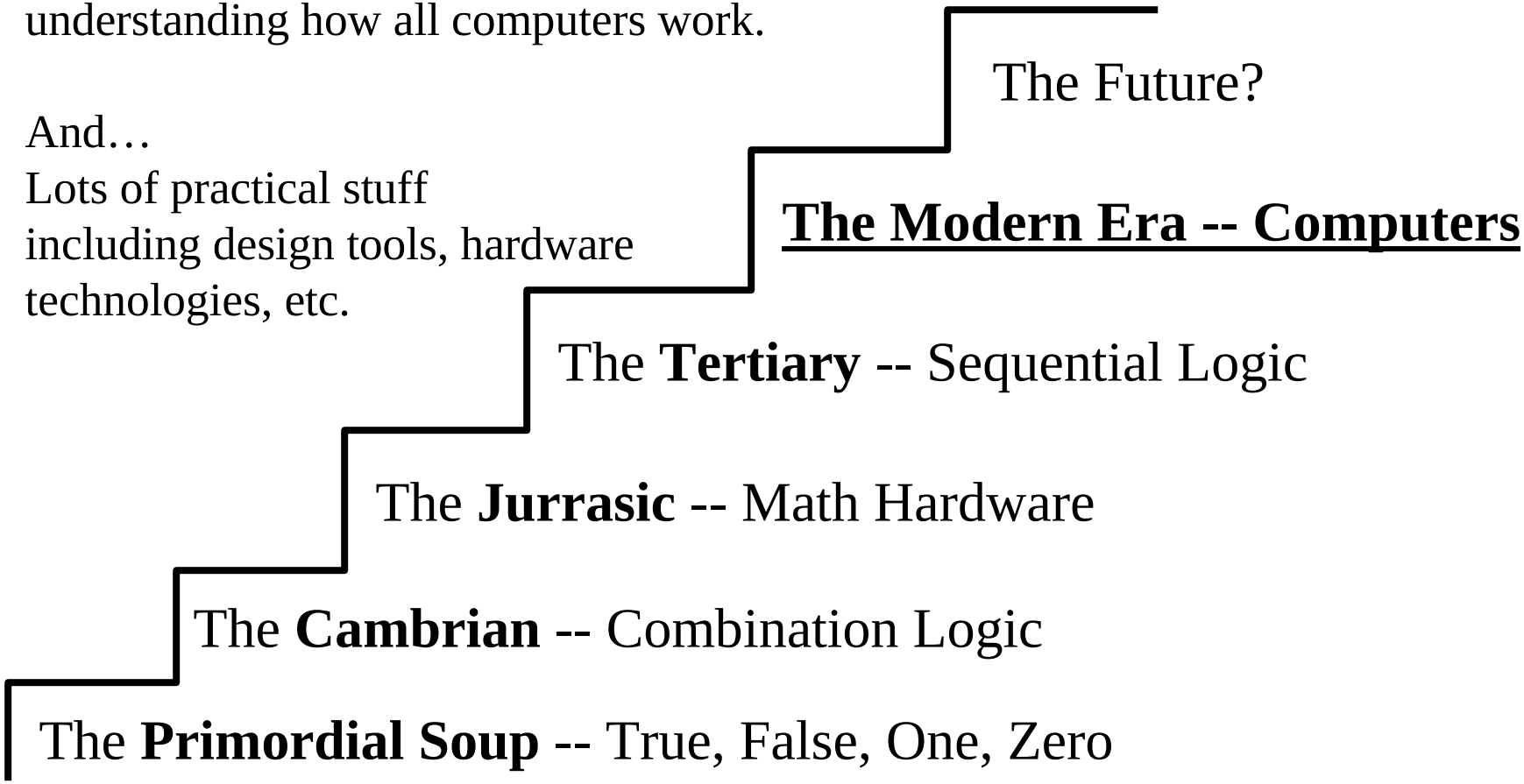
Failure due to*:*

- ***Unreasonable constraints***
- ***Limited access to the design space***

What we will learn in 370

This is ***the watershed*** course for understanding how all computers work.

And…
Lots of practical stuff including design tools, hardware technologies, etc.



Applications of logic design

- Conventional computer design
  - CPUs, busses, peripherals
- Networking and communications
  - phones, modems, routers
- Embedded products
  - in cars, toys, appliances, entertainment devices
- Scientific equipment
  - testing, sensing, reporting
- The world of computing is much much bigger than just PCs!

CSE 370: concepts/skills/abilities

- Understanding the basics of logic design (concepts)
- Understanding sound design methodologies (concepts)
- Modern specification methods (concepts)
- Familiarity with a full set of CAD tools (skills)
- Appreciation for the differences and similarities (abilities) in hardware and software design

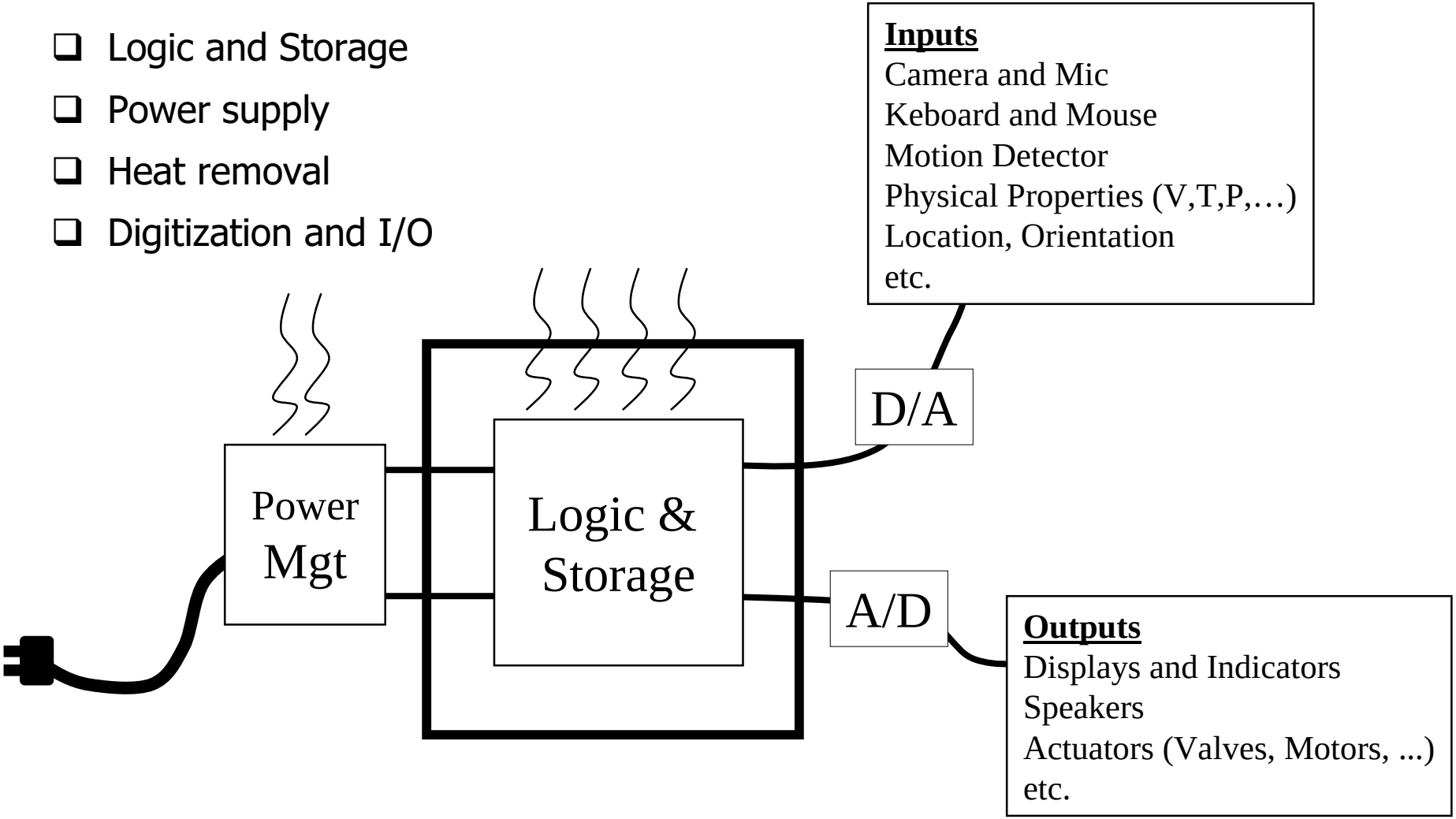
New ability: Understand the basic tools and techniques for designing digital systems, including special purpose hardware and general purpose computers, using a variety of implementation technologies such as LSI and programmable logic

Computation: abstract vs. implementation

- Up to now, computation has been a mental exercise (paper, programs)
- This class is about physically implementing computation using physical devices that use voltages to represent logical valuesdealing with constraints
- Basic units of computation are:
  - representation: "0", "1" or True/False
  - assignment: x = y
  - data operations: x + y – 5
  - control:
    - sequential statements: A; B; C
    - conditionals: if x == 1 then y
    - loops: for (i = 1 ; i == 10, i++)
    - procedures: A; proc(...); B;
- We will study how each of these are implemented in hardware and composed into computational structures

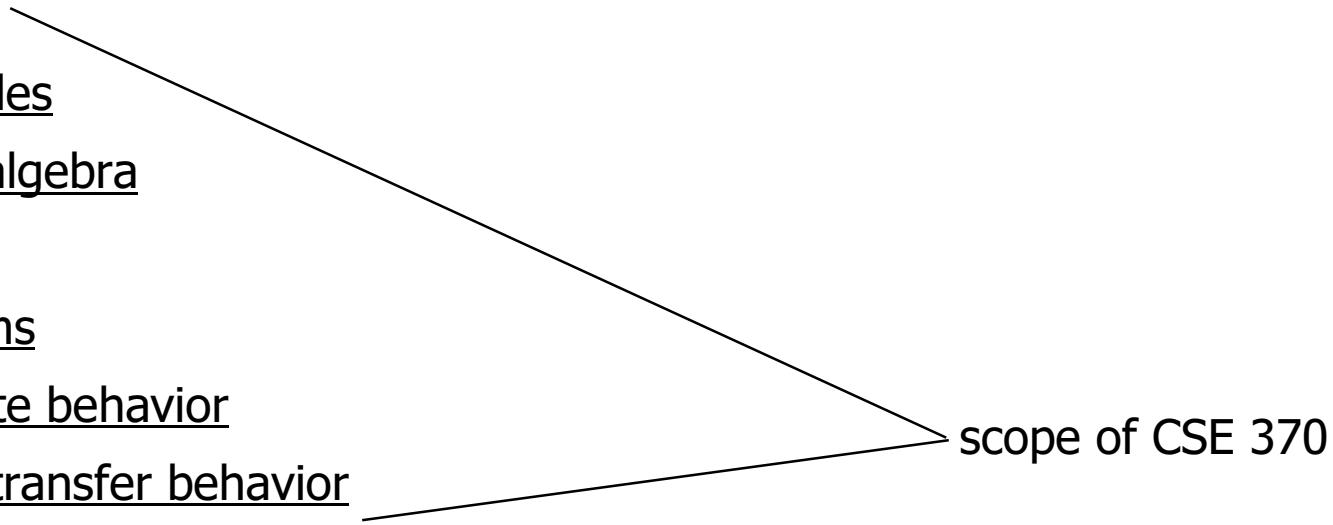
System Components

- Logic and Storage
- Power supply
- Heat removal
- Digitization and I/O



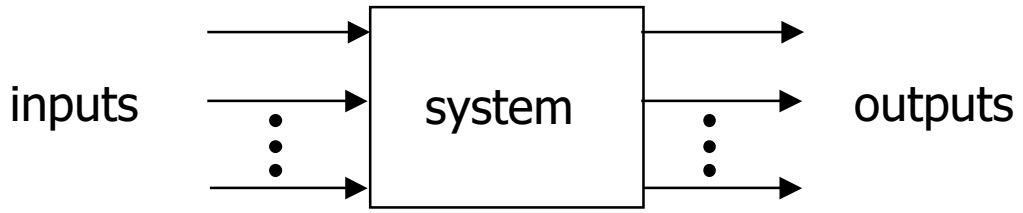
Representation of digital designs

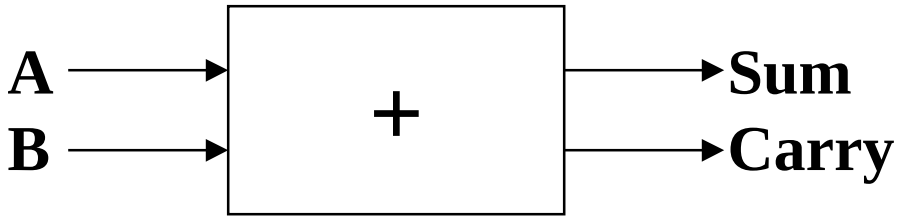
- Physical devices (transistors, relays)
- Switches
- Truth tables
- Boolean algebra
- Gates
- Waveforms
- Finite state behavior
- Register-transfer behavior
- Concurrent abstract specifications



Combinational vs. sequential digital circuits

- A simple model of a digital system is a unit with inputs and outputs:

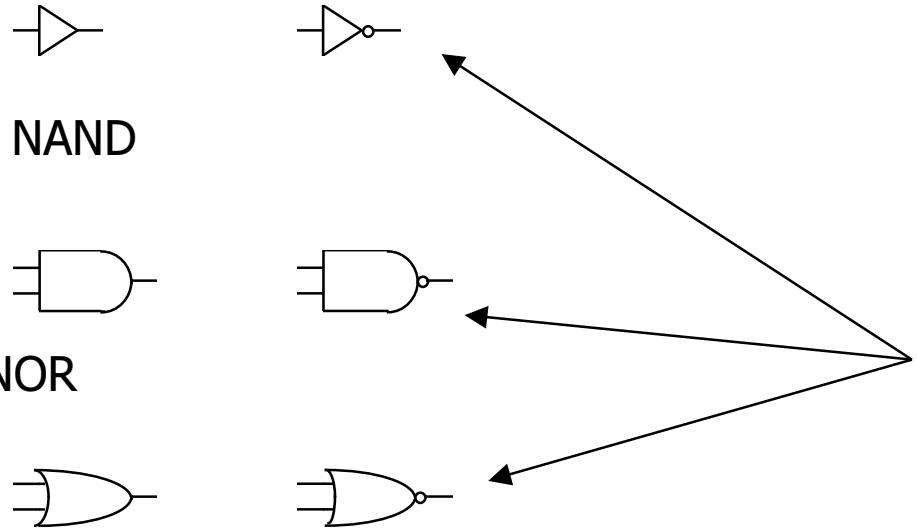




- Combinational means "memory-less"
  - a digital circuit is combinational if its output values only depend on its input values
  - A given input will also have the same steady state output, no matter what has come before

Combinational logic symbols

- Common combinational logic systems have standard symbols called logic gates
  - Buffer, NOT
  - AND, NAND
  - OR, NOR



easy to implement with CMOS transistors (the switches we have available and use most)

Sequential logic

- Sequential systems
  - exhibit behaviors (output values) that depend not only on the current input values, but also on previous (sequence of) input values
- In reality, all real circuits are sequential
  - because the outputs do not change instantaneously after an input change

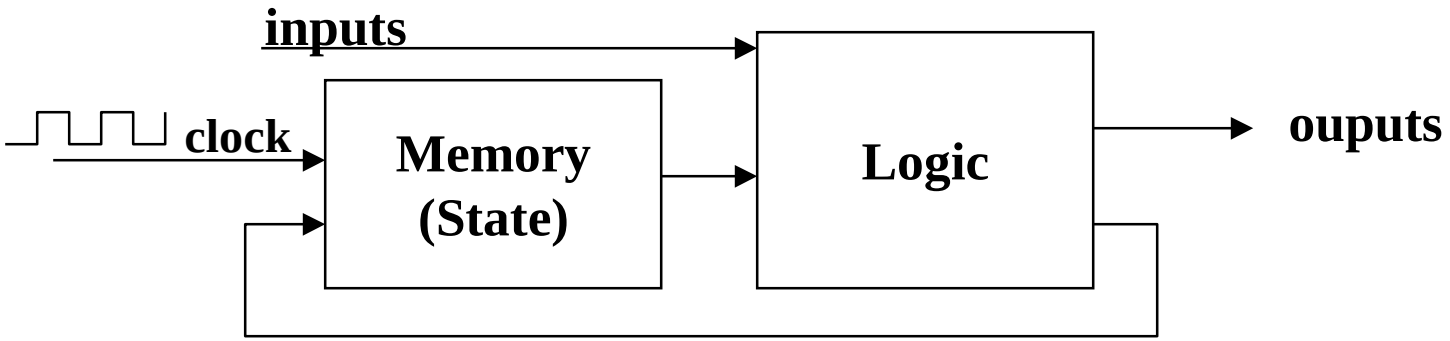
why not, and why is it then sequential?

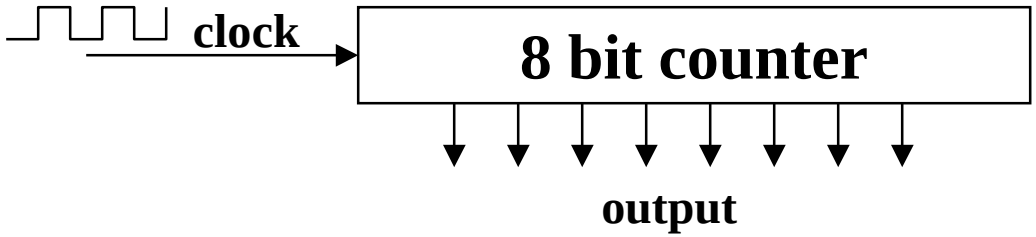
- A fundamental abstraction of digital design is to reason (mostly) about steady-state behaviors
  - look at the outputs only after sufficient time has elapsed for the system to make its required changes and settle down

Output depends on who got there first -- sequence!
Problem: How to guarantee GrantA&GrantB == 0

One at a Time...Synchronous Sequential

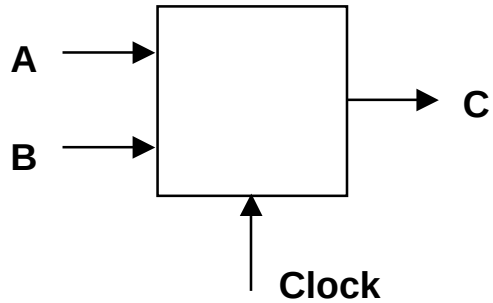
- Ignore changes on inputs except during a specific time window
  - Special signal -- the clock determines when to consider new inputs
- Isolate "memory" from "logic"
  - during a clock transition, compute new state and outputs using inputs and old state.
  - Assume inputs and state are stable during the clock transition





Summary

- Combinational:
  - input A, B
  - wait for clock edge
  - observe C
  - wait for another clock edge
  - observe C again: will stay the same
- Sequential:
  - input A, B
  - wait for clock edge
  - observe C
  - wait for another clock edge
  - observe C again: may be different



Abstractions and Simplifications

- How we make life easy for ourselves
  - Quantization and Amplification -- Lossless propagation of data in exchange for low bandwidth
  - Generally stick to binary represntation
  - Basic logic gates as building blocks in exchange for complex transistor networks (density)
  - Clearly defined state transitions on clock edges. Assume inputs stable during clock transistions.
- Tools for managing complexity that we will see
  - Truth tables and Boolean algebra to represent combinational logic
  - Binary arithmetic to manipulate groups of signals
  - State diagrams to represent sequential logic
  - Hardware description languages to represent digital logic
  - Waveforms to represent temporal behavior

An example

- Calendar subsystem: number of days in a month (to control watch display)
  - used in controlling the display of a wrist-watch LCD screen
  - inputs: month, leap year flag
  - outputs: number of days
  - No need to remember previous entries or results to give correct output

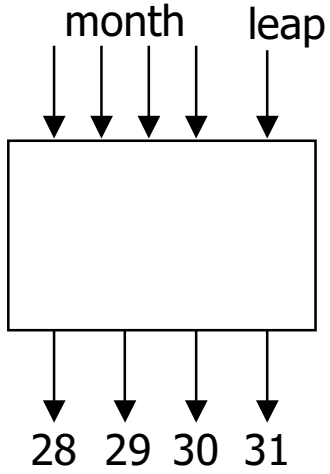
Implementation in software

```
integer number_of_days ( month, leap_year_flag) {
    switch (month) {
        case 1: return (31);
        case 2: if (leap_year_flag == 1) then return (29)
          else return (28);
        case 3: return (31);
        ...
        case 12: return (31);
        default: return (0);
    }
}
```

Clue: Return value is unrelated to previous result call

Implementation as a combinational system

- Encoding:
  - how many bits for each input/output?
  - binary number for month
  - four wires for 28, 29, 30, and 31
- Behavior:
  - combinational
  - truth table specification



month	leap	28	29	30	31
0001	–	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	–	0	0	0	1
0100	–	0	0	1	0
...					
1100	–	0	0	0	1
1101	–	0	0	0	0
111–	–	0	0	0	0
0000	–	0	0	0	0

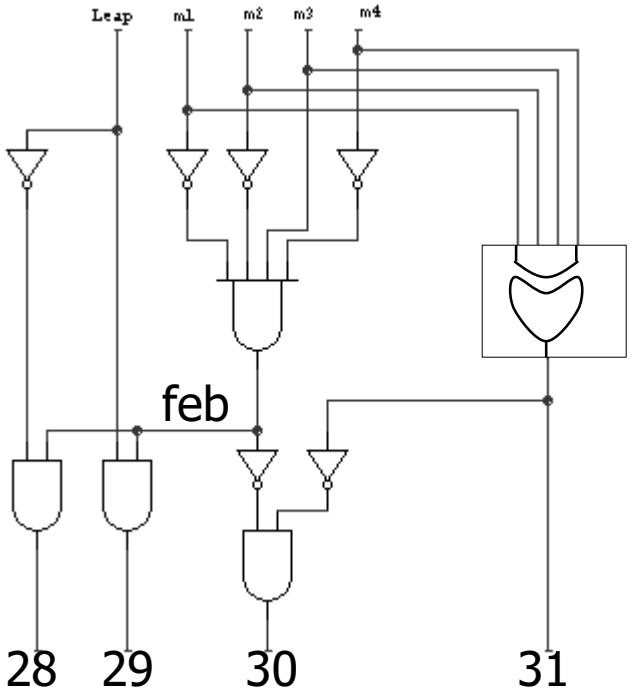
Combinational example (cont'd)

- Truth-table to logic
  - 28 = 1 when month=0010 and leap=0
  - 28 = m1'•m2'•m3•m4'•leap'
  - 31 = 1 when month=0001 or month=0011 or ... month=1100
  - 31 = (m1'•m2'•m3'•m4) + (m1'•m2'•m3•m4) + ... (m1•m2•m3'•m4')
  - 31 = can we simplify more?

symbol for or

symbol for and

symbol for not



month	leap	28	29	30	31
0001	–	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	–	0	0	0	1
0100	–	0	0	1	0
...					
1100	–	0	0	0	1
1101	–	–	–	–	–
111–	–	–	–	–	–
0000	–	–	–	–	–

Example of "multilevel logic" speed v. cost, paralle v. serial

Another example

- Door combination lock:
  - Enter three values, if they match the combination then the door opens;
  - inputs: values V1, V2, V3
  - outputs: door open/close
- Unspecified
  - What happens if wrong combination is entered?
  - How do we know when a new combination is starting?
  - Are V1-V3 on different sets of wires or are the sequenced on the same set of wires? If sequenced how do we know when a new number has been entered?
  - How does it get locked again after being opened?

Refinement of Specification

- All values come in on a single set of wires:

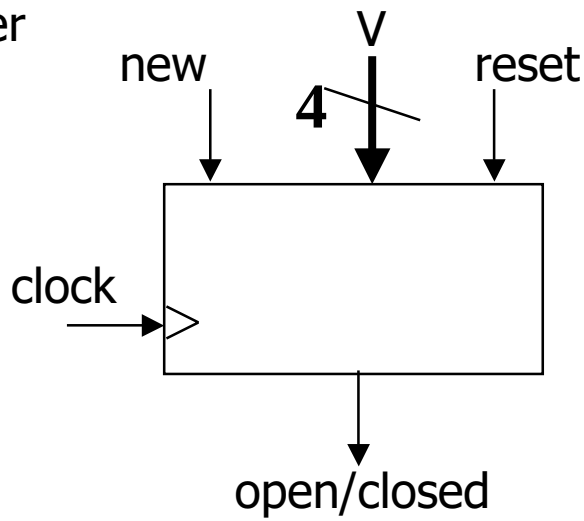
  So its sequential so we need a clock to control the sequence.

- Any other problems?

  Yes -- might not get new value every clock cycle – need another input NEW – we could use new for clock, but usually want our subsystem to be in synch with other sub-systems so we don't have user provided clocks.

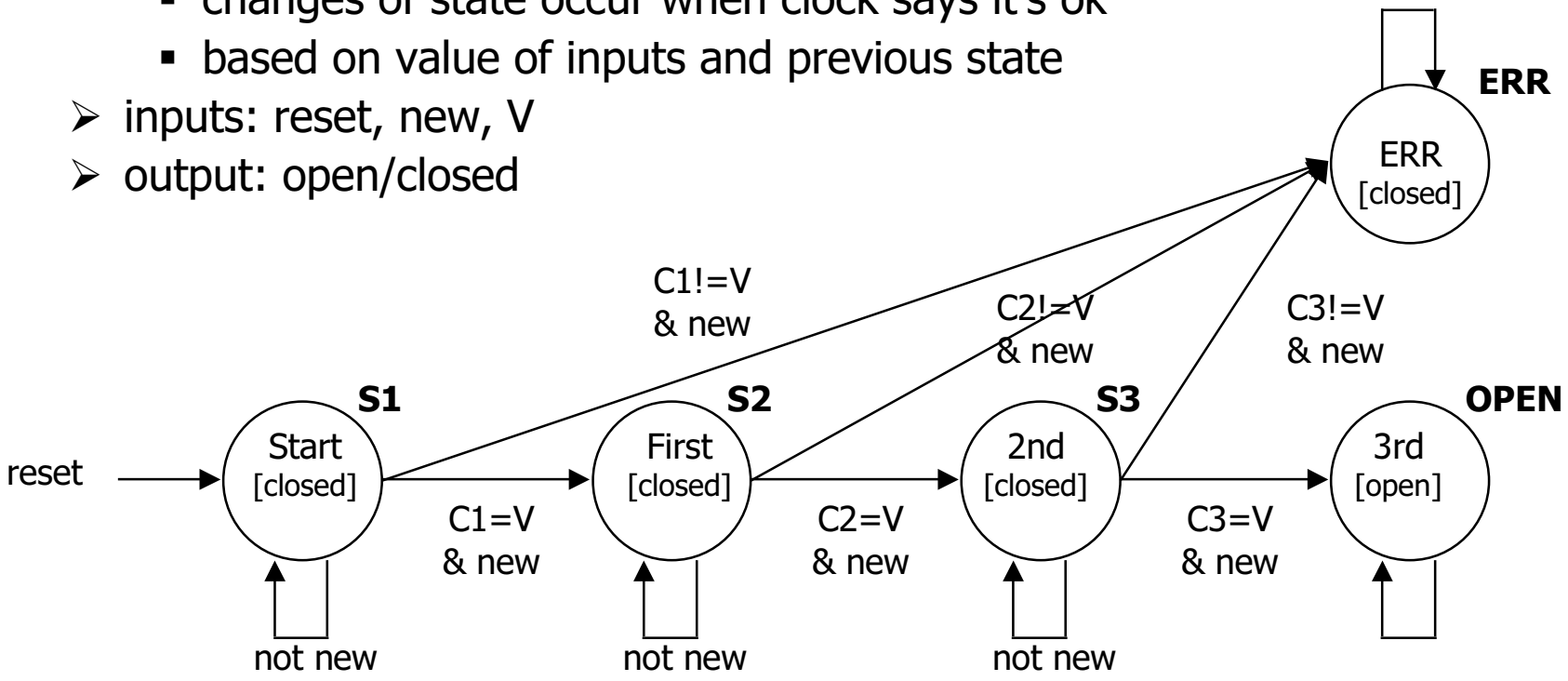
- How to handle wrong combination and relocking?

  Add a reset input



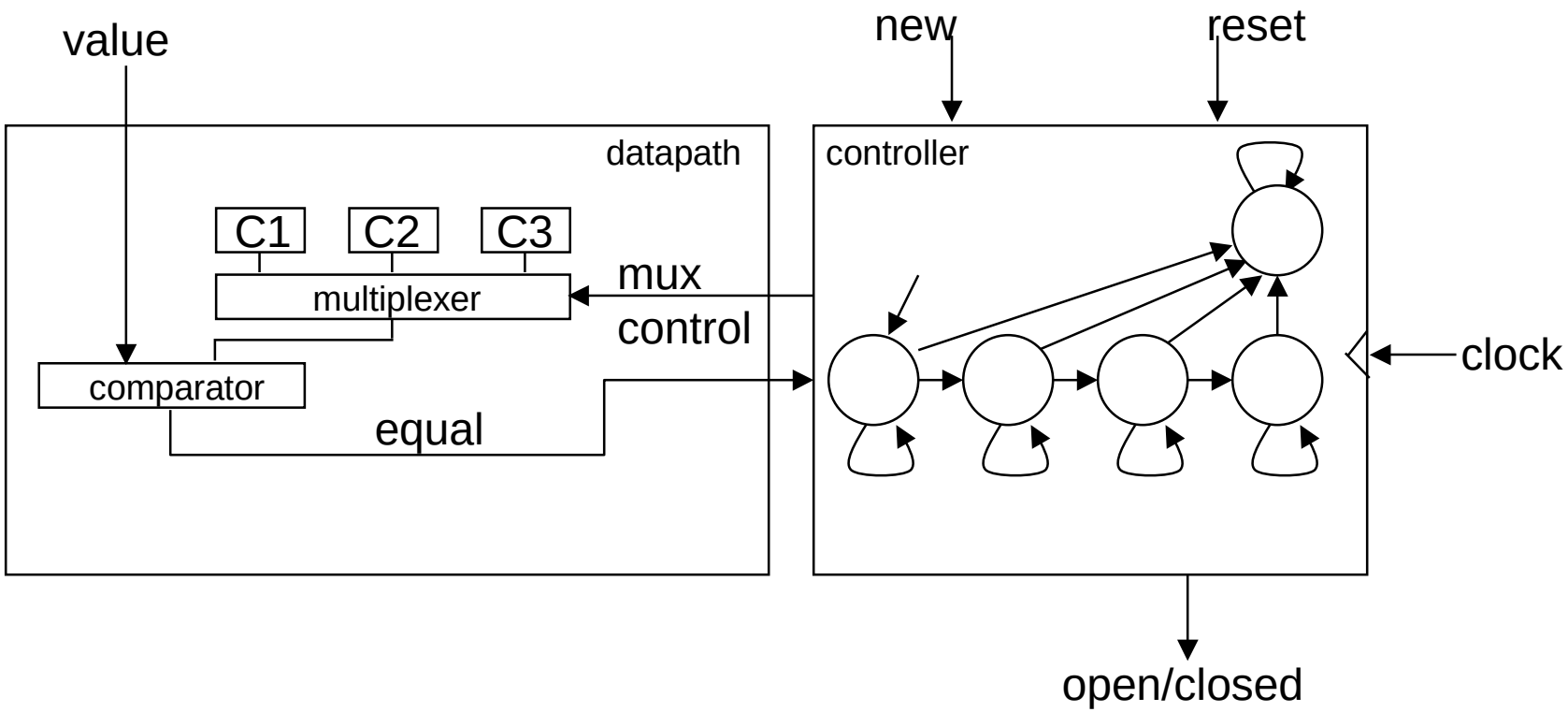
Sequential example (cont'd): abstract control

- Finite-state diagram
  - states: 5 states: S1, S2, S3, Open, Err
    - represents synchronization points in execution of machine
    - each state has outputs
  - transitions: 6 from state to state, 5 self transitions, 1 global
    - changes of state occur when clock says it's ok
    - based on value of inputs and previous state
  - inputs: reset, new, V
  - output: open/closed



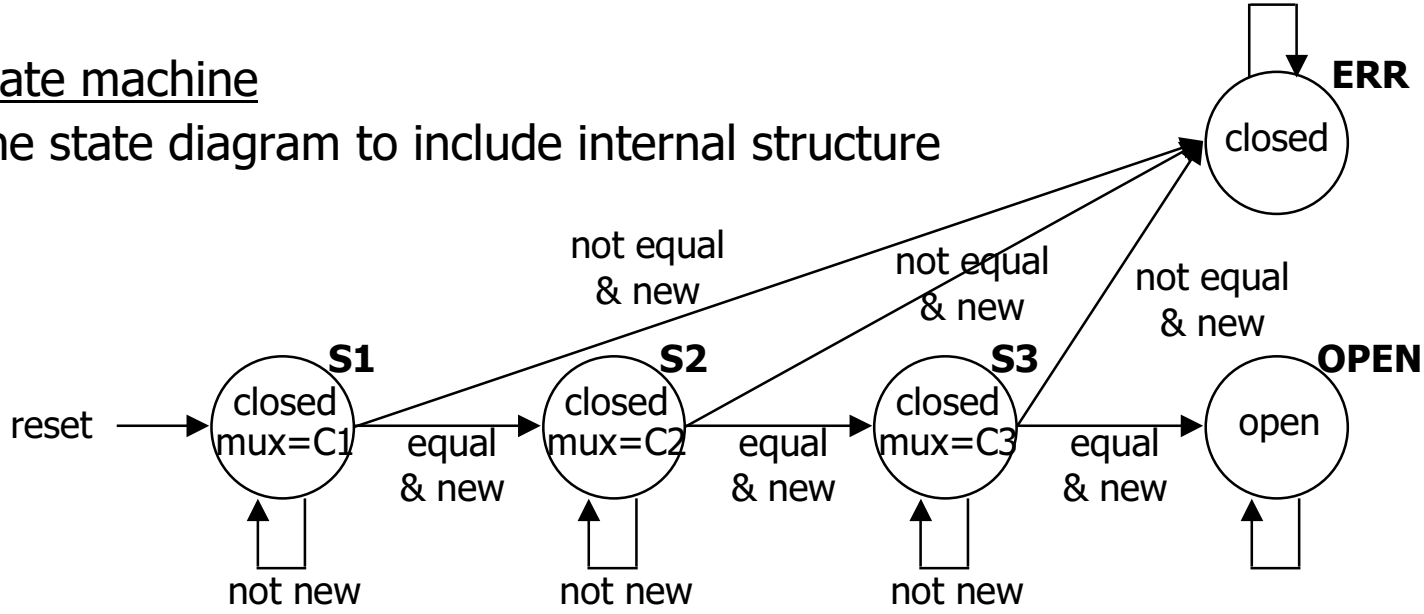
Sequential example implementation

- Internal structure: Datapath and Control
  - data-path (combinational logic for multi-bit numbers)
    - storing and operating on n-bit numbers
  - control
    - decide what to do next based on datapath results, state, and inputs



Implementing the Controller

- Finite-state machine
  - refine state diagram to include internal structure



  - generate state table (much like a truth-table)

reset	new	equal	state	next state	mux	open/closed
1	–	–	–	S1	C1	closed
0	0	–	S1	S1	C1	closed
0	1	0	S1	ERR	–	closed
0	1	1	S1	S2	C2	closed
...						
0	1	1	S3	OPEN	–	open
...						

Design of Controller

- Encode state table
  - state can be: S1, S2, S3, OPEN, or ERR
    - needs at least 3 bits to encode: 000, 001, 010, 011, 100
    - and as many as 5: 00001, 00010, 00100, 01000, 10000
    - choose 4 bits: 0001, 0010, 0100, 1000, 0000
  - output mux can be: C1, C2, or C3
    - needs 2 to 3 bits to encode
    - choose 3 bits: 001, 010, 100
  - output open/closed can be: open or closed
    - needs 1 or 2 bits to encode
    - choose 1 bits: 1, 0

reset	new	equal	state	next state	mux	open/closed
1	–	–	–	0001	001	0
0	0	–	0001	0001	001	0
0	1	0	0001	0000	–	0
0	1	1	0001	0010	010	0
…						
0	1	1	0100	1000	–	1
…						

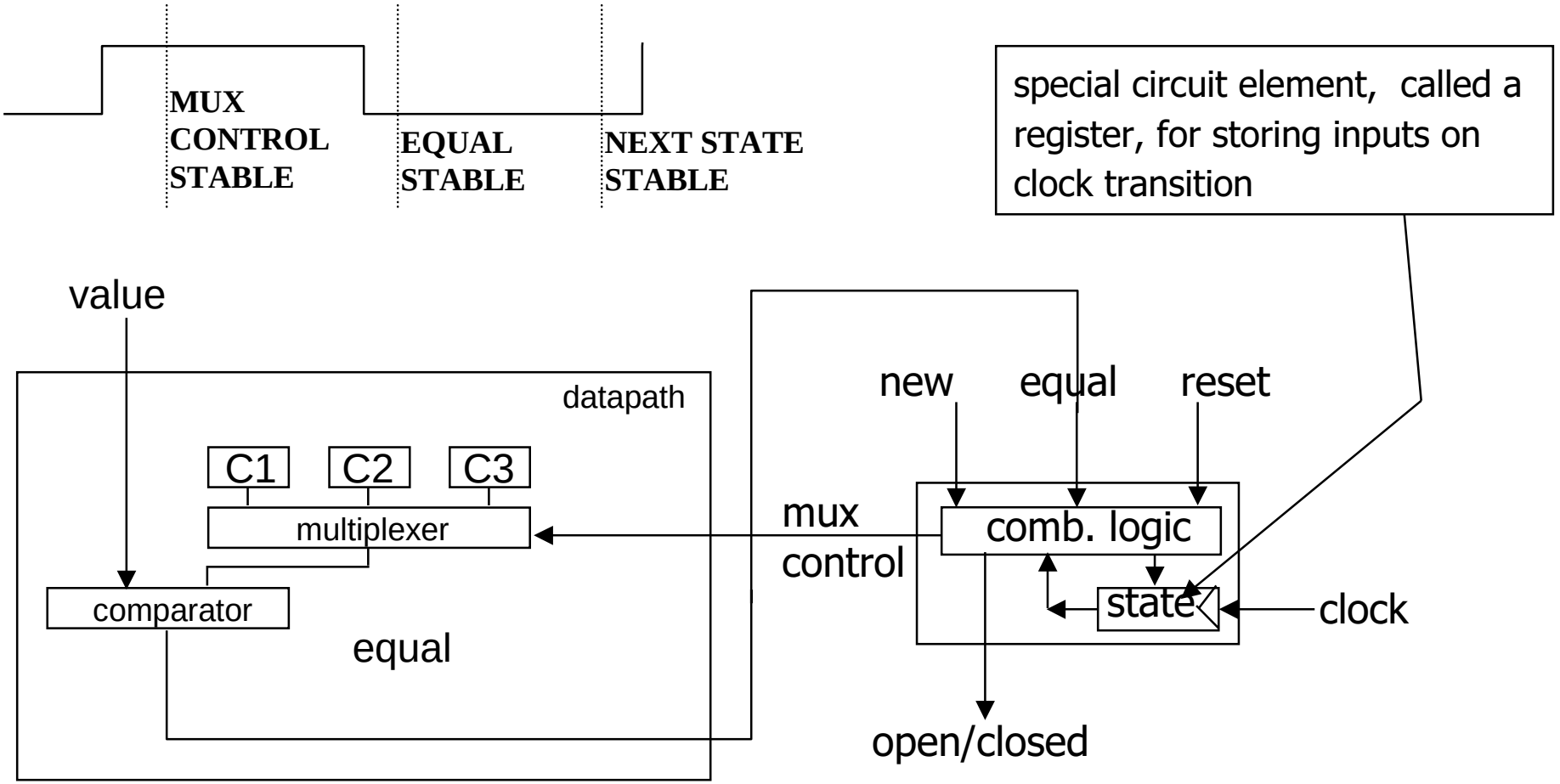
good choice of encoding!

mux is identical to last 3 bits of state

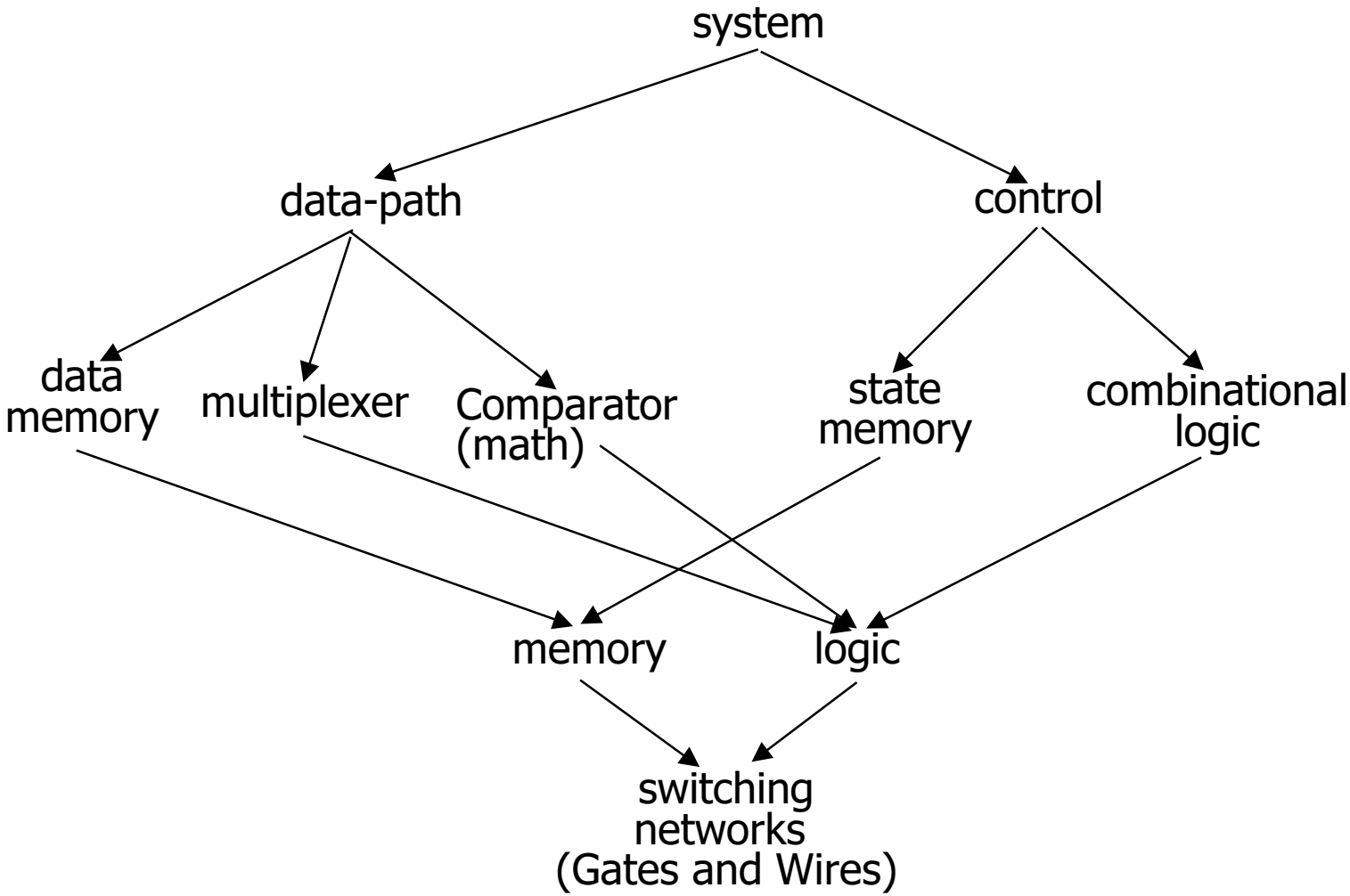
open/closed is identical to first bit of state

Implementation of Controller w/ Datapath

- Implementation of the controller



Design hierarchy



Summary

- That was what the entire course is about (Almost)
  - converting solutions to problems into combinational and sequential networks effectively organizing the design hierarchically
  - doing so with a modern set of design tools that lets us handle large designs effectively
  - taking advantage of optimization opportunities

- Learn where software and hardware come together
  - How does a software program get executed on a digital hardware system?

What is happening now in digital design?

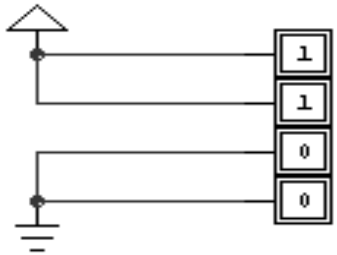
- Big change in the way industry does hardware design over last few years
  - larger and larger designs
  - shorter and shorter time to market
  - cheaper and cheaper products
- Scale
  - pervasive use of computer-aided design tools over hand methods
  - multiple levels of design representation
- Time
  - emphasis on abstract design representations
  - programmable rather than fixed function components
  - automatic synthesis techniques
  - importance of sound design methodologies
- Cost
  - higher levels of integration
  - use of simulation to debug designs

Some Basics. Lecture Overview

- Today
  - The basics: Electronics, binary numbers, base conversion
  - Number systems
    - 2's complement numbers
  - Combinational logic
    - Logic functions and truth tables

The basics→ binary numbers

- Binary (base 2) Number Representation
  - low V → 0
  - hi V → 1



- Base conversion (binary, octal, decimal, hexadecimal)
  - Positional number system
    - $101_2 = 1x2^2+0x2^1+1x2^0 = 5_{10}$
    - $101_8 = 1x8^2+0x8^1+1x8^0 = 65_{10}$
    - $101_{16} = 1x16^2+0x16^1+1x16^0 = 257_{10}$
  - Conversion between binary/octal/hex
    - Binary: 10011110001
    - Octal: 10 | 011 | 110 | 001=2361_8
    - Hex: 100 | 1111 | 0001=$4F1_{16}$
- Addition and subtraction are trivial, but worth practicing
  - See Katz, appendix A

Number systems

- How do we write negative binary numbers?
- Historically: Three approaches
  - Sign and magnitude
  - One's complement
  - Two's complement
- Two's complement makes addition and subtraction easy
  - Used almost universally in present-day systems
- Fractional Numbers
- Floating Point Represenations

Sign-magnitude Representation

- The most-significant bit (msb) is the sign digit
  - 0 ≡ positive
  - 1 ≡ negative
- The remaining bits are the number's magnitude
- Benefit: easy to perform negation: just flip MSB
- Problem 1: Two representations for zero
  - 0 = 0000 *and also* –0 = 1000
- Problem 2: Arithmetic is cumbersome

1001
+ 0011
≠ 1100

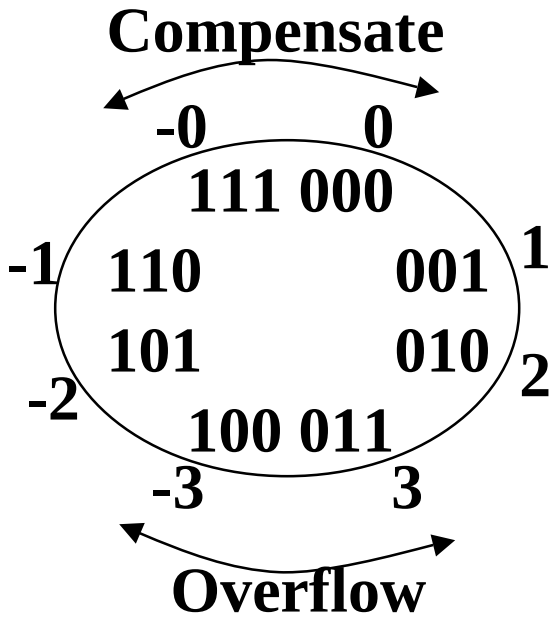
Instead, must change order and perform substraction

011
- 001
= 0010

One's complement

- Negative number: Bitwise complement of positive number
  - $0011 \equiv 3_{10}$
  - $1100 \equiv -3_{10}$
- Almost solves the arithmetic problem

$$\begin{array}{rll} & 110 & (-1) \\ + & 010 & (+2) \\ \hline \neq & 000 & \\ + & 1 & \\ \hline = & 1 & \end{array}$$

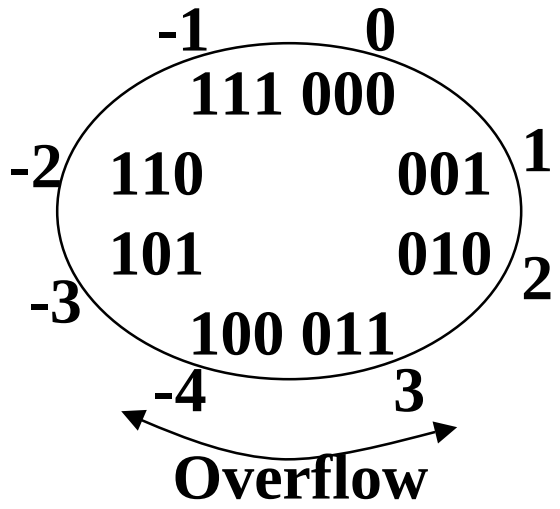


- Remaining problem: Must compensate for two zeros
  - Add (or subtract) carry when (MSB carry & opposite sign)
  - Overflow when (MSB carry & same sign)

Two's complement

- Negative number: Bitwise complement plus one
  - $0011 \equiv 3_{10}$
  - $1101 \equiv -3_{10}$
- Shift the 1's complement number wheel to eliminate -0

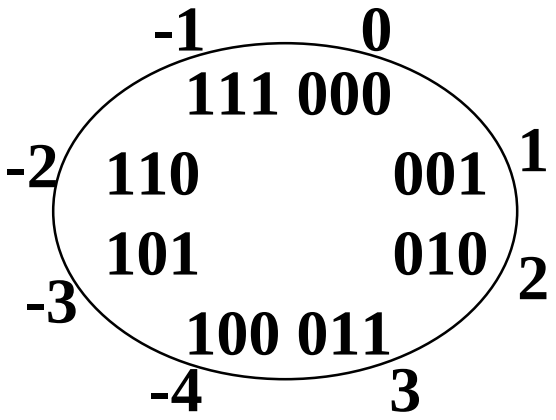
- msb is still the sign digit
  - $0 \equiv$ positive
  - $1 \equiv$ negative
- No need to compensate for +/-



Twos complement (con't)

- Complementing a complement restores the original number
- Arithmetic is easy
  - We ignore the carry
    - Same as a full rotation around the wheel
  - Subtraction = negation and addition
    - Easy to implement in hardware

```
   111   (-1)
+  010   (+2)
=  001
(ignore carry)
```



Where are we now?