

CS1: Introduction to Computation

Day 16: November 19, 2008
Matter Computes



Deep Thoughts

- Deep Thought: a physical computer
 - Computing the answer to the question of life, the universe and everything
 - The answer alone isn't sufficient – need to know the question!
- Deep Thought proposes building a larger computer to figure out the actual question
 - ...called... *THE EARTH!* (ooo)
- *A physical object that embodies computation*

Caltech is about

- Understanding the physical world:
 - physics
 - chemistry
 - biology
- Using this understanding to gain mastery over the physical world.

Computer Science is about

...automated processing of information.

- Big idea 1:
 - Write deterministic instructions which can be executed mechanically and compute most functions.

Computer Science is about

...automated processing of information.

- Big idea 2:
 - Build machines out of the **physical world** to automatically carry out these instructions.

Today

Caltech and CS

- The physical world and information processing are *intimately intertwined*.

Matter Computes

Today

- **Idea: Matter Computes**
 - representation of values
 - physics and computation
 - boolean logic
 - arithmetic
 - finite function
 - variations on theme

Idea: Matter Computes

- Physics is about laws and relationships that govern matter and energy:

$$\Sigma F = 0, F = ma, F = -kx, \dots$$

$$\Sigma I = 0, V = IR, \Delta V = (I \Delta T)/C$$

- You perform **computations** to model how the physical world will behave, which means...

Matter Computes

- The physical world implicitly implements these computations in its behavior:
 - Springs: $F = -Kx$
 - Resistors: $V = IR$
 - Capacitors: $\int (I/c) dt$
- We can exploit these computations to build...

Matter Computers

- *Engineer* the computations we want, by arranging matter's own computations in creative ways
- If the physical world did **not** perform computations, it would be *impossible* to build computing machines!

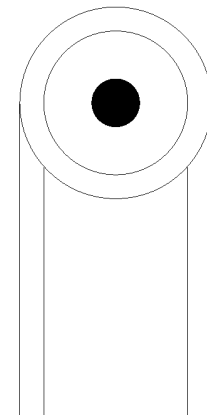
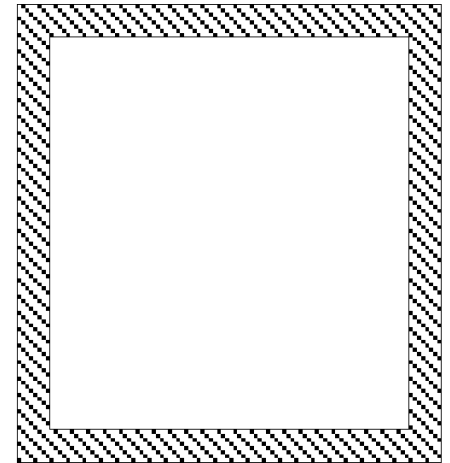
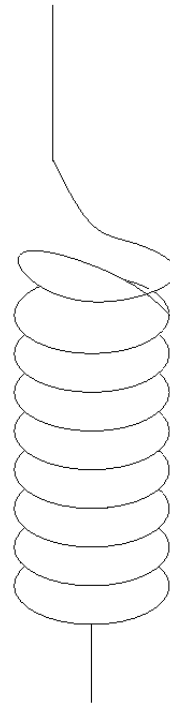
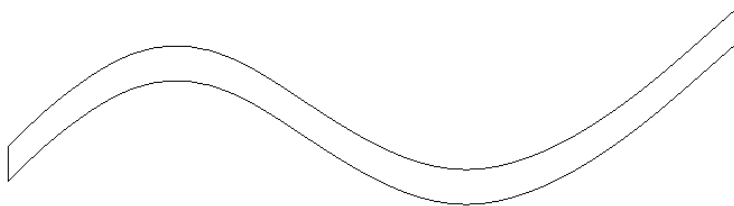
Abstract → Concrete

- Now for a concrete example!
- Build computation out of mechanics:
 - you all understand mechanics
 - emphasize physical nature of computation
 - not just something unique to electronics!

Computational Medium

- Basic building blocks:

- Ropes
- Pulleys
- Springs
- Boxes

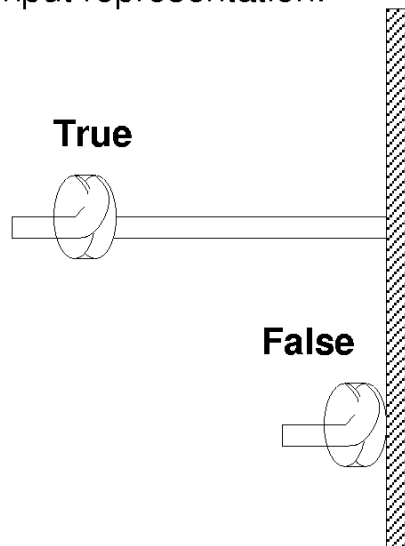


(Example from A.K. Dewdney)

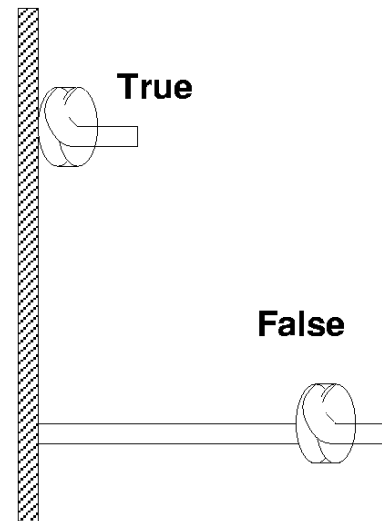
Step 1: Representation

- We need a way to represent logical values (true/false) in our medium.
- We'll use the rope position:

Input representation:



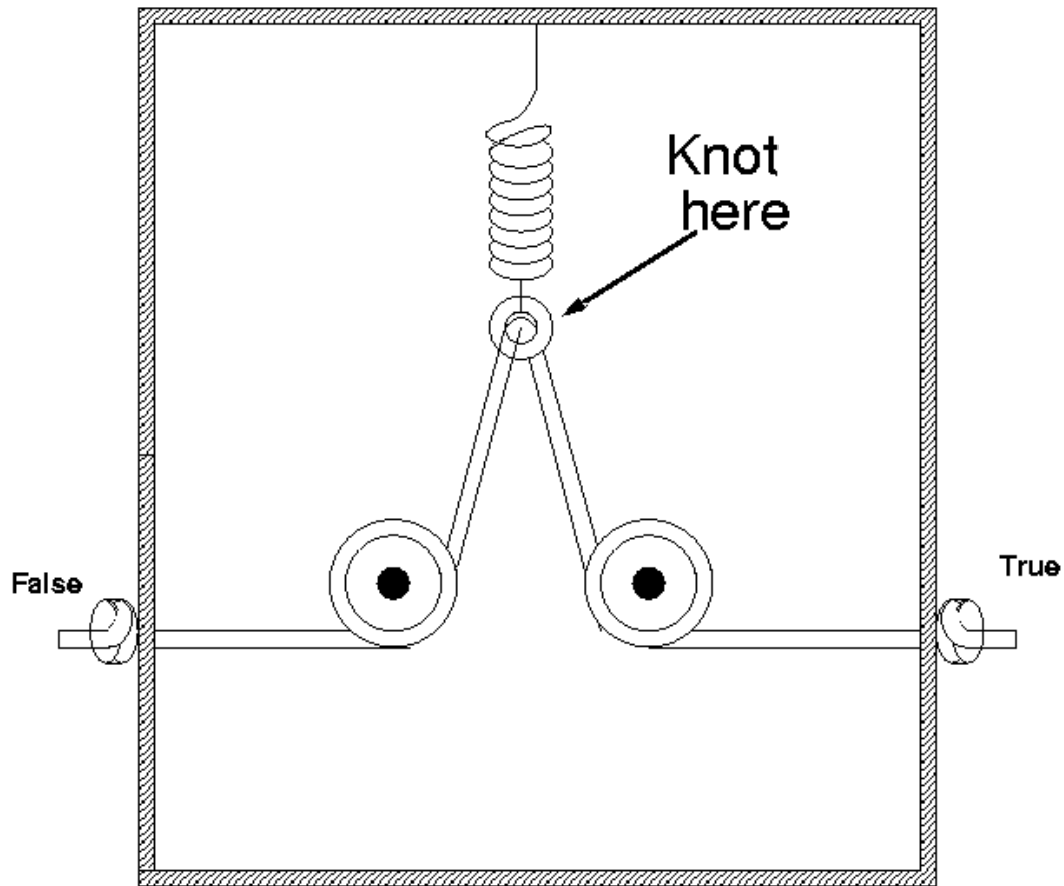
Output Representation:



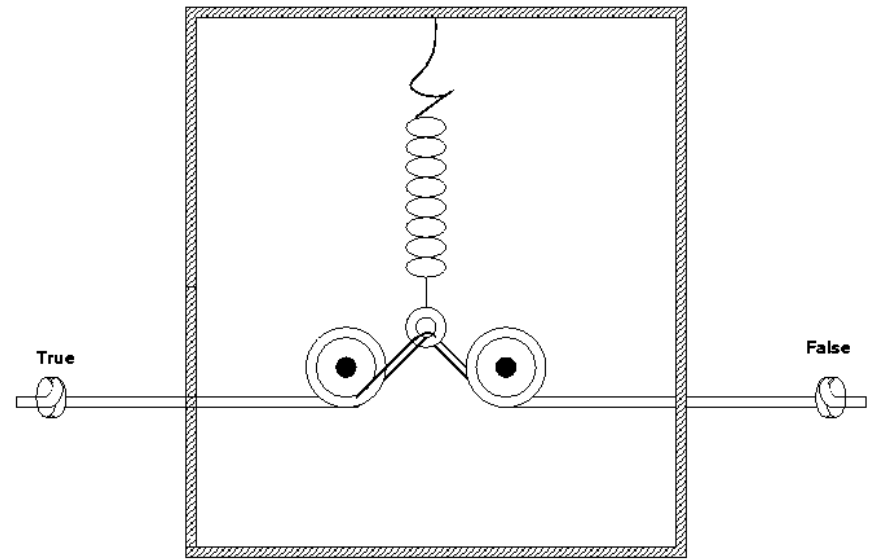
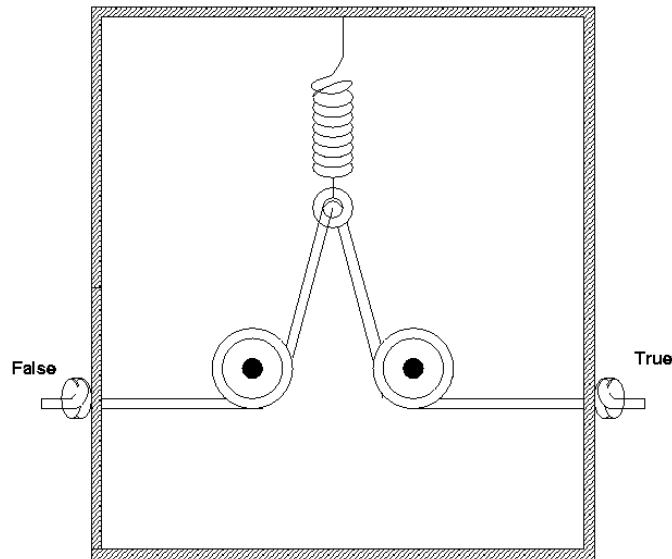
Step 2: Build Primitives

- We need primitive operations.
- This is an *Engineering Problem*:
 - Manage it by creating some building blocks.
- Ropes and pulleys will implement a large number of primitives
 - ...as will any sufficiently powerful computational medium
- Let's implement:
 - inverter
 - 2-input AND gate

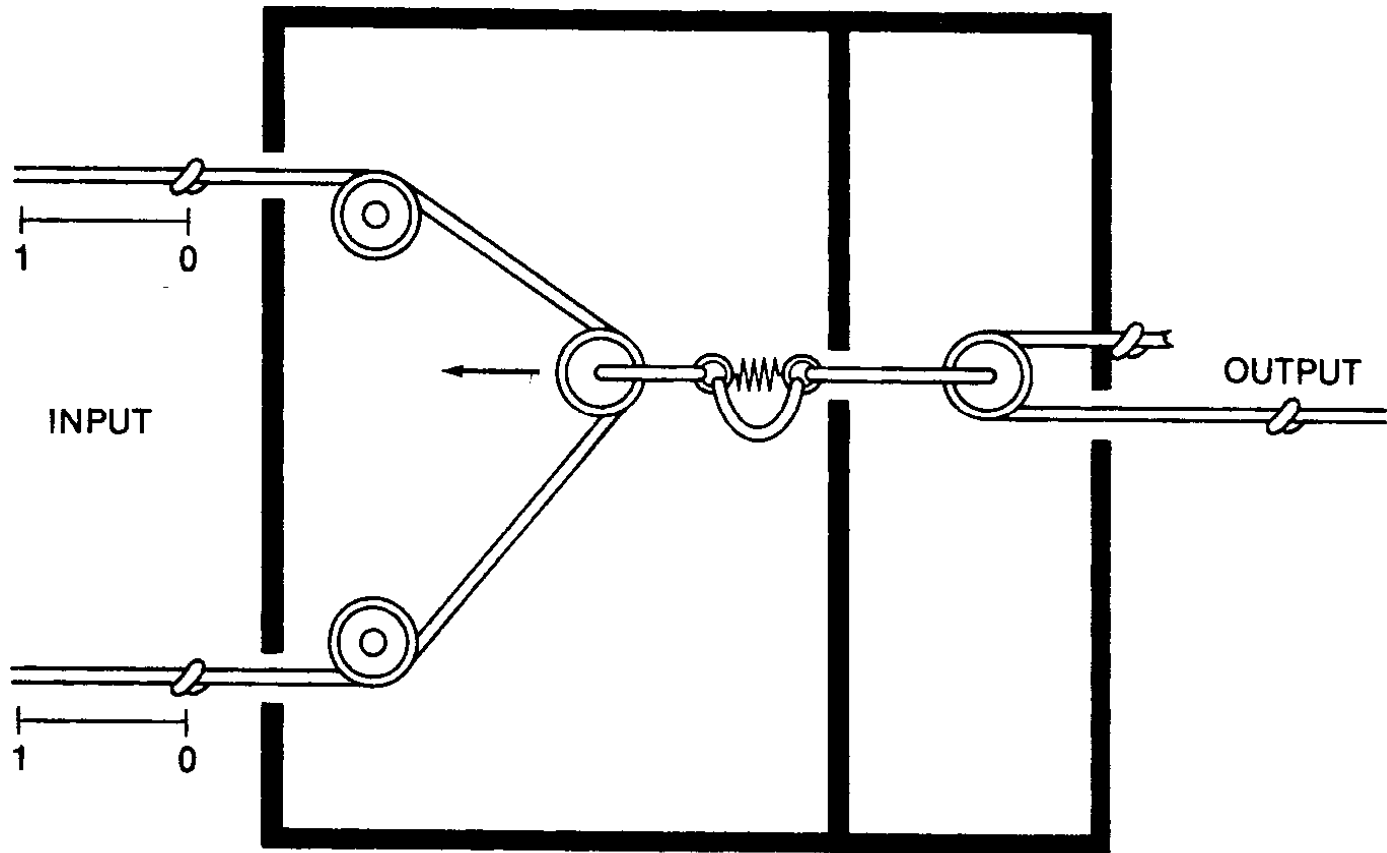
Step 2a: Inverter



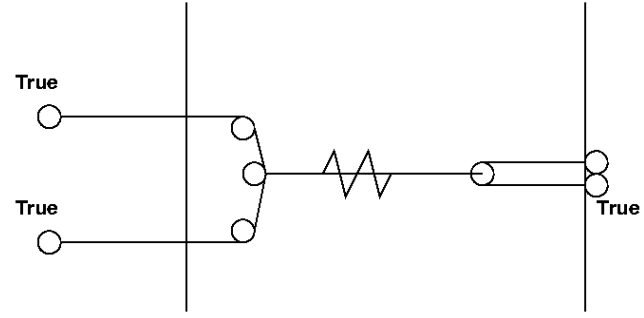
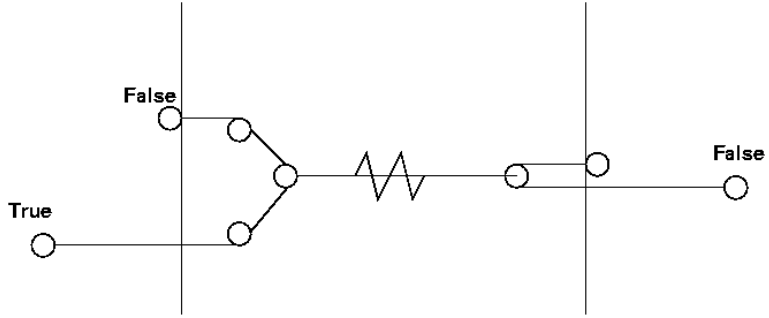
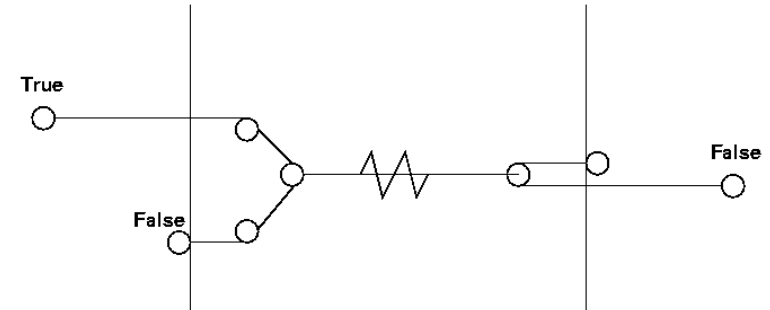
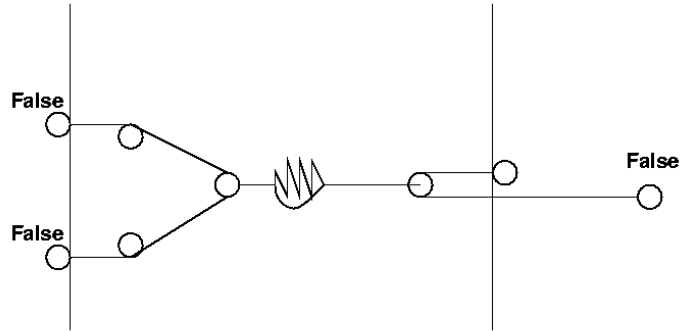
Inverter Truth Table



Step 2b: 2-input AND Gate



AND2 Truth Table



Step 3: Abstract

- Now have two primitive operations:
 - $x = \text{AND2}(a, b)$ (and2 a b)
 - $x = \text{INV}(a)$ (not a)
- Think about the *computations* the ropes, springs, and pulleys perform, entirely in terms of these primitives.
 - Abstract away Hooke's law, conservation laws, friction, geometry...

Step 4: Model Computation

Now what can we compute?

Anything we can build from **not** and **and2**

- $\text{or2} = (\text{not} (\text{and2} (\text{not } a) (\text{not } b)))$
- $\text{and3} = (\text{and2 } a (\text{and2 } b \text{ } c))$
- $\text{xor2} = (\text{not} (\text{or2} (\text{and2 } a \text{ } b) (\text{and2} (\text{not } a) (\text{not } b))))$
- Addition!

Example: Bit Level Addition

- Addition
 - (everyone knows how to do addition base 2, right?)

“New Math”



Tom Lehrer

That Was The Year That Was

Problem

342

-173



169

Problem: Base 8

$$\begin{array}{r} 342 \\ -173 \\ \hline \end{array}$$

$$147$$

Example: Bit Level Addition

- Addition
 - (everyone knows how to do addition base 2, right?)

C: 11011010000

A: 01101101010

B: 01100101100

S: 11010010110

Addition Base 2

- $A = 2^{n-1} \times a_{n-1} + 2^{n-2} \times a_{n-2} + \dots + 2^1 \times a_1 + 2^0 \times a_0$
 $= \sum_i 2^i \times a_i$
 - (Same with B)
- $S = A + B$
 - Add A and B together, bit by bit
- Can break down this problem too.
 - Half-adder: add two binary digits; get sum, carry
 - Use half-adder twice, to incorporate input carry

Half-Adder

- Add two digits; get sum, carry
 - ha-sum = (xor2 a b)
 - ha-carry = (and2 a b)

$a + b = \text{sum, carry}$
$0 + 0 = 0, \text{ carry} = 0$
$0 + 1 = 1, \text{ carry} = 0$
$1 + 0 = 1, \text{ carry} = 0$
$1 + 1 = 0, \text{ carry} = 1$

Full-Adder

- Combine half-adders to get full-adder
 - Sum $s = a + b + \text{carry-in}$
 - $\text{cin} = \text{carry-in}$; $\text{cout} = \text{carry-out}$
 - $s_i = (\text{ha-sum } (\text{ha-sum } a_i b_i) \text{ cin}_i)$
 - $\text{cout}_i = (\text{or2 } (\text{ha-carry } a_i b_i) (\text{ha-carry } (\text{ha-sum } a_i b_i) \text{ cin}_i))$
 - Either $a + b$ could produce a carry-out, or $(a + b) + \text{carry-in}$ (hence the or2)

Full-Adder (2)

- $s_i = (\text{ha-sum } (\text{ha-sum } a_i b_i) \text{ cin}_i)$
- $\text{cout}_i = (\text{or2 } (\text{ha-carry } a_i b_i) (\text{ha-carry } (\text{ha-sum } a_i b_i) \text{ cin}_i))$

$a + b + \text{cin} = \text{sum}, \text{cout}$

$$0 + 0 + 0 = 0, \text{ cout} = 0$$

$$0 + 1 + 0 = 1, \text{ cout} = 0$$

$$1 + 0 + 0 = 1, \text{ cout} = 0$$

$$1 + 1 + 0 = 0, \text{ cout} = 1$$

$a + b + \text{cin} = \text{sum}, \text{cout}$

$$0 + 0 + 1 = 1, \text{ cout} = 0$$

$$0 + 1 + 1 = 0, \text{ cout} = 1$$

$$1 + 0 + 1 = 0, \text{ cout} = 1$$

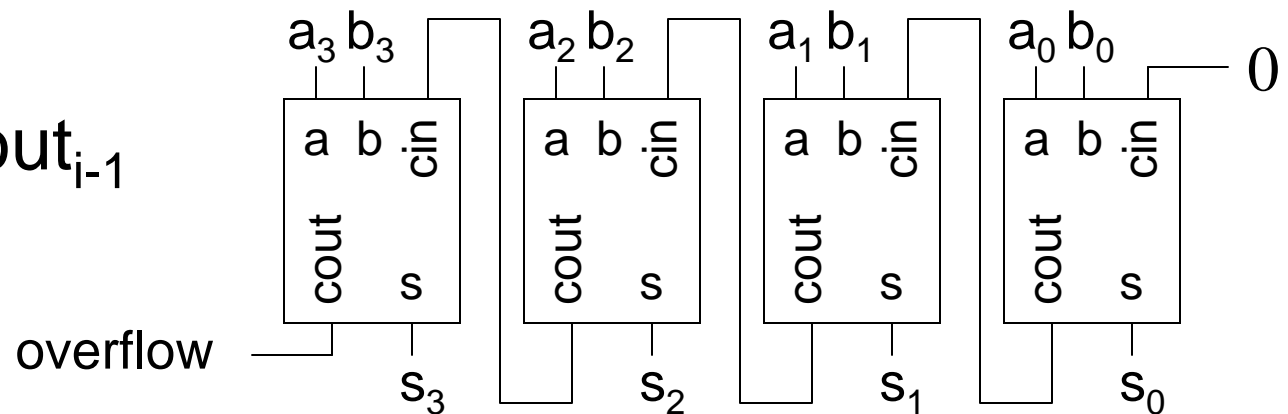
$$1 + 1 + 1 = 1, \text{ cout} = 1$$

The Ripple-Adder

- Now, connect full-adders into a chain

- $cin_0 = 0$

- $cin_i = cout_{i-1}$



- When $cout_{n-1}$ is 1, overflow!

- Now we can add any two integers

- Just need enough rope, pulleys, springs, boxes

Step 4b: Model of Computation

- In fact:
 - *Any* boolean logic expression can be expressed in terms of **not** and **and2**.
- All we have to do is:
 - write boolean equations
 - convert them to **not** and **and2** pulley-gates
 - assemble the gates
- ...and we've implemented the computation

Step 4c: Any Function

- In fact, any *function* can be implemented in Boolean logic
- Function: (mathematical definition)
 - each input “vector”
 - maps to exactly one output “vector”
 - output is deterministic
 - output depends only on the input “vector”

Sketch of “Any Function”

- Consider a binary output function:
 - will have either true or false (1 or 0) for each input vector
 - can pick out each input vector with an **and**
 - *e.g.* a b c d having values 1 0 1 1
 - we would pick this out with the term:
 - (and a (not b) c d)
 - **or** together all such terms for true outputs

Any Function (cont.)

- *e.g.*, we want true when a b c d has values:
 - 1 0 1 1,
 - 0 1 0 0,
 - or 0 0 1 1
- we get:
 - (or (and a (not b) c d)
(and (not a) b (not c) (not d))
(and (not a) (not b) c d))

Any Function Sketch

- Previous example was for a single output bit
- Multiple-output-bit functions are easy
 - just write one equation for each output bit

Review:

- Step 0: picked a computational medium
- Step 1: chose a representation
- Step 2: implemented primitive operators
- Step 3: abstracted up (to logic)
- Step 4: built a model
 - a: gates
 - b: all of Boolean logic
 - c: arithmetic
 - d: any function

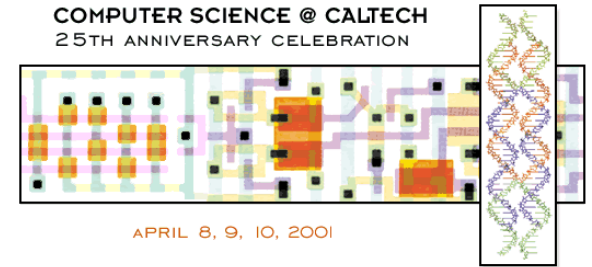
Variations on a Theme

- We can build gates out of:
 - switches (transistors), resistors, and capacitors
 - molecular interactions
 - DNA molecules
 -
- Ultimately, we can build computers out of *any* suitable phenomena!

Value of Rest of Your Classes

- Now you know the *real* reason you study physics, chemistry, biology, ...
 - To understand the physical phenomena from which we build computations

Caltech CS



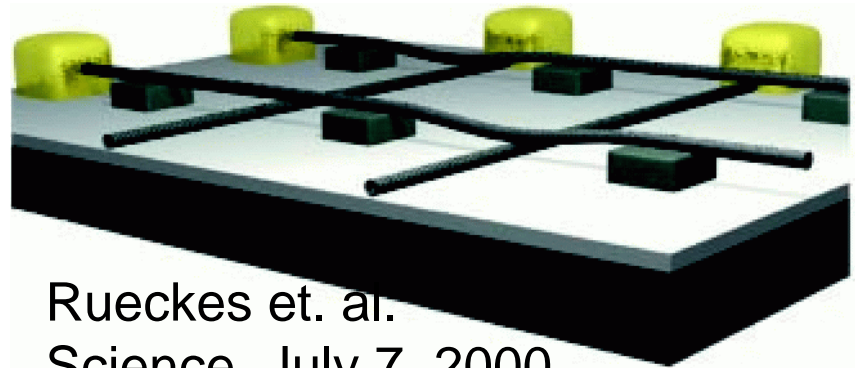
- Founded
 - 32 years ago
 - Ivan Sutherland and Carver Mead
- Radical idea
 - we would build computers out of integrated circuits
 - computer scientists would directly exploit semiconductor device physics

Matter Computes

- A key theme of Caltech CS since its inception
- Continues to be a key theme today
 - look to the future
 - understand the implications of “Matter Computes”
 - limits of physical processes
 - new opportunities to exploit matter’s computation

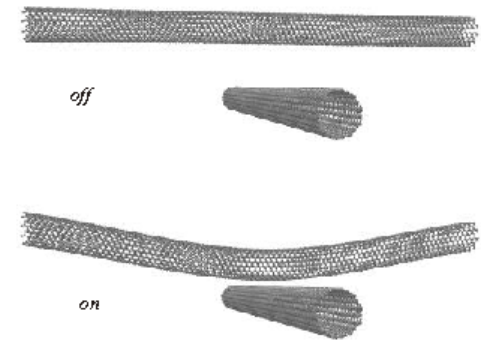
Molecular Electronics

- Can build molecular scale wires (few nm)
 - SWNT, nanowires
 - UCLA/HP, Harvard
- Can build switches at junctions, etc.
 - Molecules
 - electromechanical bistable [on right]
- Assemble regular structure
 - cheap, faulty



Rueckes et. al.
Science, July 7, 2000

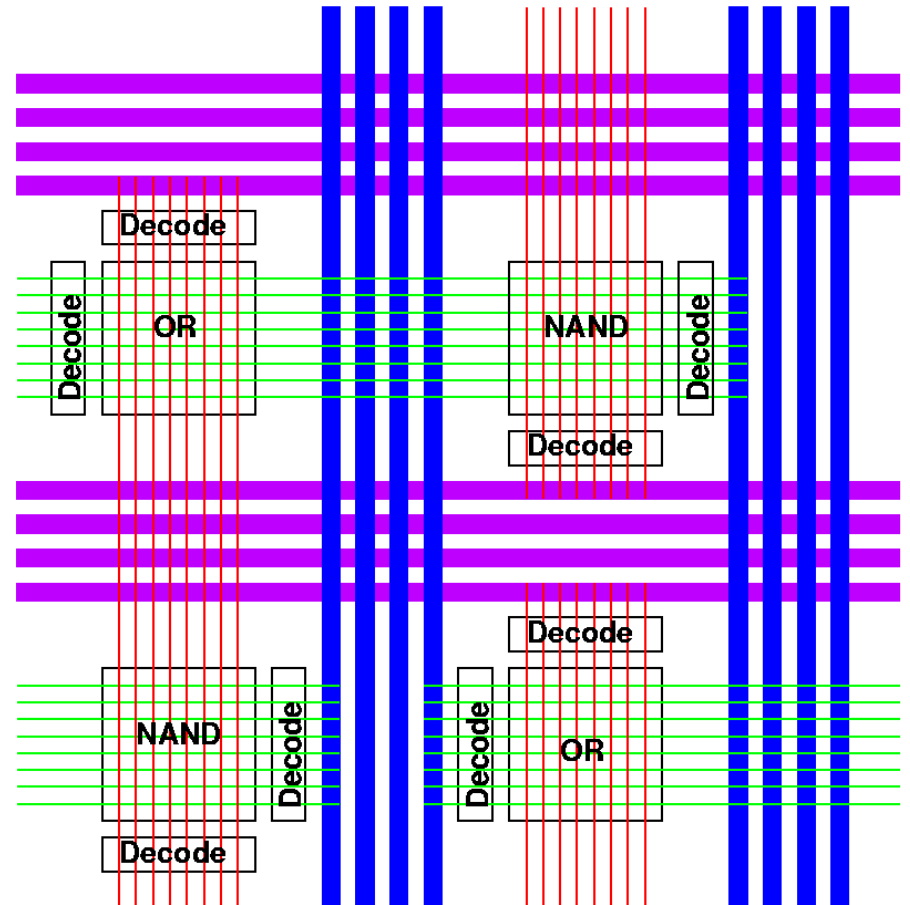
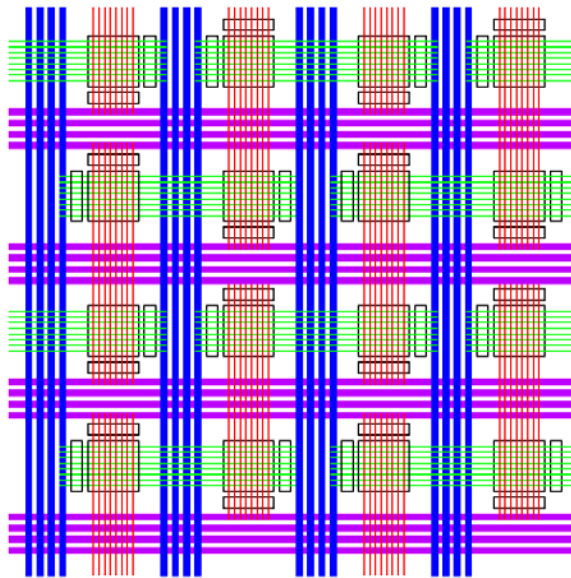
Harvard
SWNT
devices



- How build memories and computation out of this?
- What architectures make sense?

Molecular Architecture (DeHon)

- Crossed Wire nanoarrays
- Implement PLAs, memory, and xbars

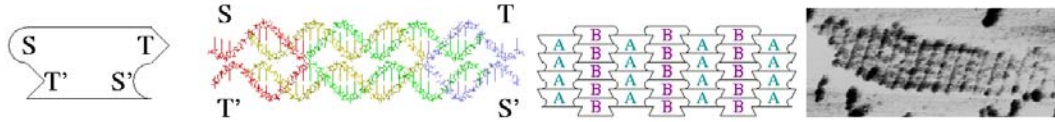


Computing with Biological Substrates

- Protein binding performs very sophisticated computation
 - chemical matching
 - Forms the basis of cellular and biological computation
 - important part of how biological systems work!
 - important for understanding biological systems
- Need to understand the computations they perform

Biomolecular Computation (Winfree)

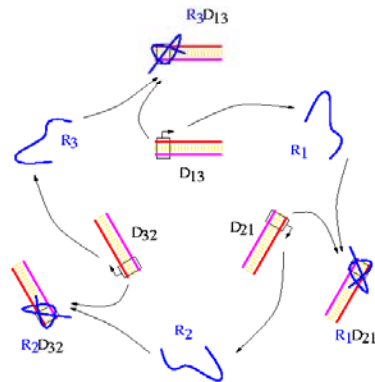
Algorithmic Self-Assembly



- Turing-universal model of computation by self-assembly.

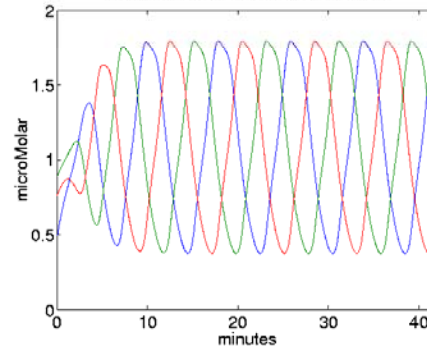
- Experimental realization of programmable self-assembly using DNA.

In Vitro Transcriptional Networks



- Simplified model of genetic regulatory circuits equivalent to digital circuits and neural networks.

Simulation:
Oscillator circuit: RNA concentration



- Experimental realization using DNA, RNA, RNA polymerase, ribonuclease (in progress).

Impact: Brings algorithms and embedded control to chemical devices and nanotechnology \Rightarrow biomolecular computers with chemical I/O.

Quantum Computation

- Quantum mechanics has a unique set of governing equations / computation

$$i\hbar \frac{d}{dt} |\Psi\rangle = \mathbf{H} |\Psi\rangle$$

- Led Feynman (and others) to consider use as substrate
- Peter Shor showed:
 - QC may allow *much* more efficient computation than classical substrates
 - Shor's Algorithm: factors large numbers in polynomial time

QC at Caltech

- Theory (Schulman, Kitaev)
 - How powerful is a quantum computer?
 - How to manage errors in quantum computations?
- Physics/Eng. (Mabuchi, Preskill, Kimble)
 - How to represent information?
 - How to build quantum gates?
 - How to assemble in a controlled way?

Summary

- The physical world performs computations
- As we master understanding and control of the physical world:
 - we can *engineer* the computation it performs to serve our purposes.
- This exploitation has had an enormous impact on our lives and capabilities.
- ...and we've just scratched the tip of the iceberg.

Where you can learn more?

- CS/EE 181 – VLSI (dominant medium)
- CS 185 – Asynchronous VLSI
- CS/Bi 191 – Biomolecular Computation
- Ph/CS 219 – Quantum Computation

References

- *The Tinker Toy Computer*, A. K. Dewdney
 - W. H. Freeman and Company, 1993.
 - Rope-and-Pulley Wonder
 - *Scientific American*, April 1988
- *The Pattern on the Stone*, W. Daniel Hillis
 - tinker-toy gates, hydraulic gates
- Other examples online
 - Lego logic-gates, etc.

Big Idea

- **Matter Computes:**
 - Understand the basic properties of the physical world.
 - Exploit these properties to build machines that can perform computations for us, automatically.