
CS231: Computer Architecture I

María J. Garzarán

Spring 2008

Course Organization

- Course website:
<http://www.cs.uiuc.edu/class/sp08/cs231/>
 - Office hours, policies, schedule, etc. posted
 - A tentative set of lecture notes will be posted
Modifications may take place right up to, or during lecture.
Therefore you should download the class notes after a lecture
- Instructor:
Maria J. Garzaran
Office: 4308
Email: garzaran@cs.uiuc.edu
Office Hours: Wednesdays 11:00 am to noon
Web page: <http://polaris.cs.uiuc.edu/~garzaran>
- TAs:
Office: 0212 SC
Email: ta231@cs.uiuc.edu
Josh Smith (Office hours: TBA)
Rajhans Samdani (Office hours: TBA)
Jin Tae Kwak

Course Objectives

After taking this course, you will:

- Learn how to design *digital* (i.e. boolean) *circuits*
- Have a high-level understanding of how to design a general-purpose computer:
 - Its hardware components
 - What they are built from
 - How to design them
 - Also, how to design *digital* circuits other than computers
- Understand some of the important ideas for designing more complex computers.

Course Organization

- Lectures Monday and Wednesday 10:00 to 10:50 am.
Review Session Friday 10:00 to 10:50 am
- Textbook

Logic and Computer Design Fundamentals, 4rd Edition

by M. Morris Mano and Charles R. Kime.

Published by Prentice-Hall, 2008.

ISBN: 0-13-600158-0

We will mark which section in the book corresponds to the material covered in each lecture

Lecture notes are enough to do the homeworks and the exams, but reading the book is highly recommended

Course Organization

- Daily Quizzes:

There will be a quizz after every lecture. You can try each quiz multiple times until the due date.

The quizz will be due:

- 10 am Wed for the quizz on Monday class
- 10 am Friday for the quizz on Wed class

There is a quizz 0 after this class. It is a fake quizz, just to test that things will work fine for the real quizzes.

Course Organization

- Weekly Homeworks:

Will be posted on Mondays and due the following Monday.

Homeworks will be accepted up to two days late, with a 10% penalty for each late day.

You may make only one submission per problem set (i.e., you may not turn in most of the problems on time and then a few more the next day).

To submit the homeworks:

- Hand it to the TA
- Slide it under the TA office door (0212) when no one is there -- DO NOT place in the bins outside the door

Course Organization

- Exams:

There will be two midterms and a final.

Midterms will cover the material since the previous midterm.

However, the final exam will cover the material from the beginning, with special emphasis on the material covered after the second midterm.

No calculators, books or notes will be allowed in the exams.

- Evaluation:

- Daily quizzes: 5%
- Homeworks: 30%
- Midterms: 38%
- Finals: 27%

The distribution of final grades will be approximately 30% As, 35% Bs, 30% Cs, 5% other.

Percentage are subject to change.

Course Organization

- Cheating

You can discuss the homeworks with other members of the class, myself, or the TA. However, do not look at or copy anyone else's solutions.

I am not concerned with how you come to understand the problem and how to solve it, but once you have the background necessary to solve it, you must provide your own solution.

The penalty for cheating ranges from a failing grade for an assignment to dismissal from the university. You can read the gory details of the University's cheating policy in Rule 33 of the Code of Policies and Regulations Applying to All Students.

Questions?

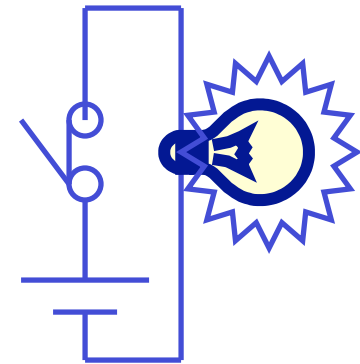
- For questions regarding homeworks, clarification of the material, etc, you should use the newsgroup `class.cs231` on the department's server.
 - You need to SET UP the newsgroup
 - Check if your question has already been asked before posting
 - A message to the newsgroup is the preferred method because:
 - It is faster
 - Other students can see the answer
- You can also send an email to ta231@cs.uiuc.edu

Today's lecture

- A grand overview
- How have we been able to make a "Machine" that can do complex things
 - Add and multiply *really* fast
 - Weather forecast, design of medicinal drugs
 - Speech recognition, Robotics, Artificial Intelligence..
 - Web browsers, internet communication protocols
- Starting at (almost) the lowest level
 - Gates to Gates

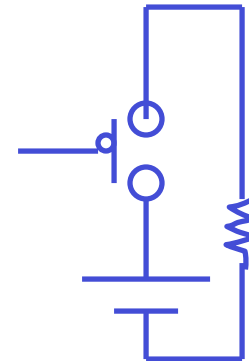
The Modest Switch

- All these capabilities are built from an extremely simple component:
 - A controllable switch
- The usual Electrical switch we use every day
 - The electric switch we use turns current on and off
 - But we need to turn it on and off by hand
 - The result of turning the switch on?
 - The "top end" in the figure becomes
 - raised to a high voltage
 - Which makes the current flow through the bulb

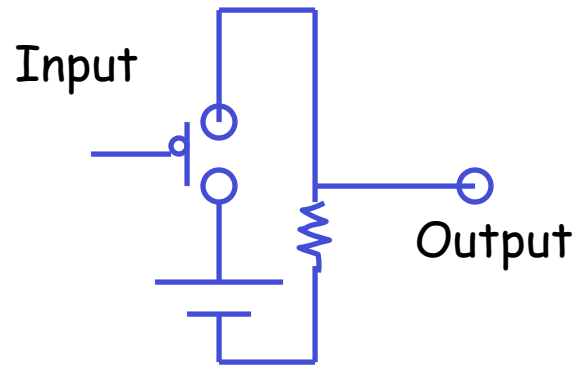


•The Controllable Switch

- No hands
- Voltage controls if the switch is on or off
- High voltage at input: switch on
 - Otherwise it is off

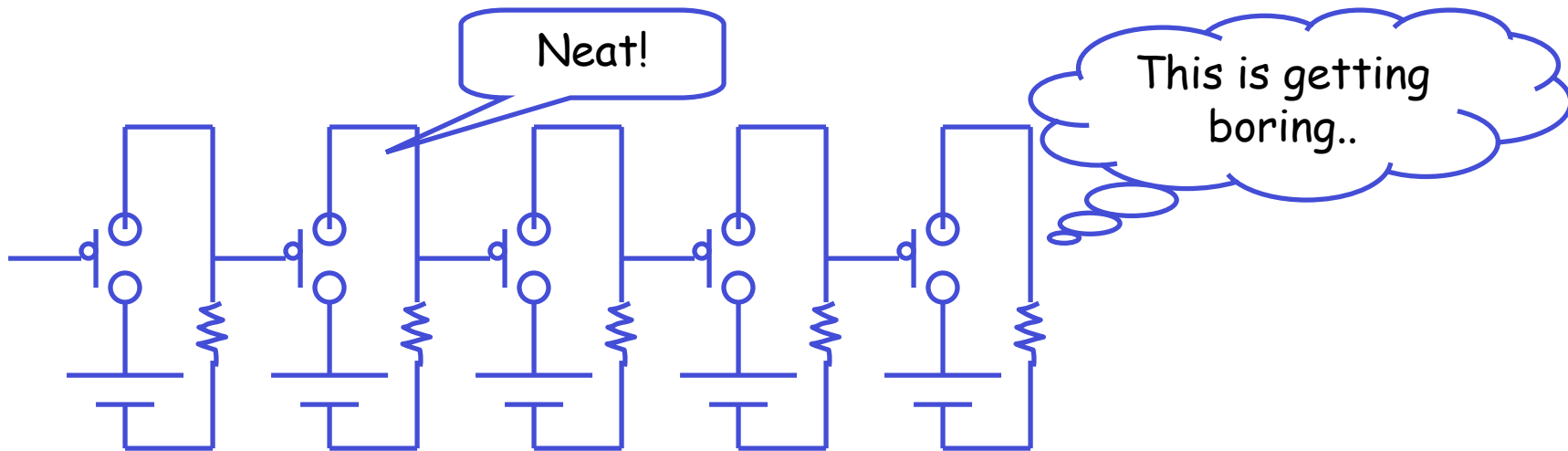


Using the switch

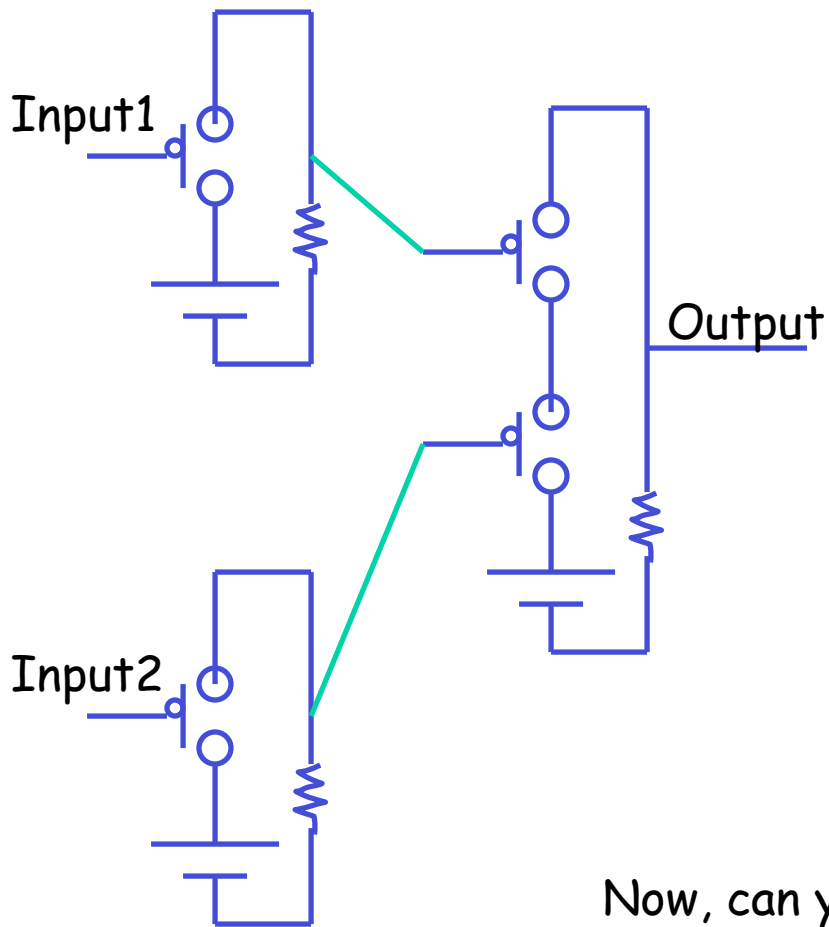


Output is high (voltage) if and only if the input is high

Now we can make one circuit control another switch...



Lets use them creatively



Output is high if both the inputs input1 **AND** input2 are high

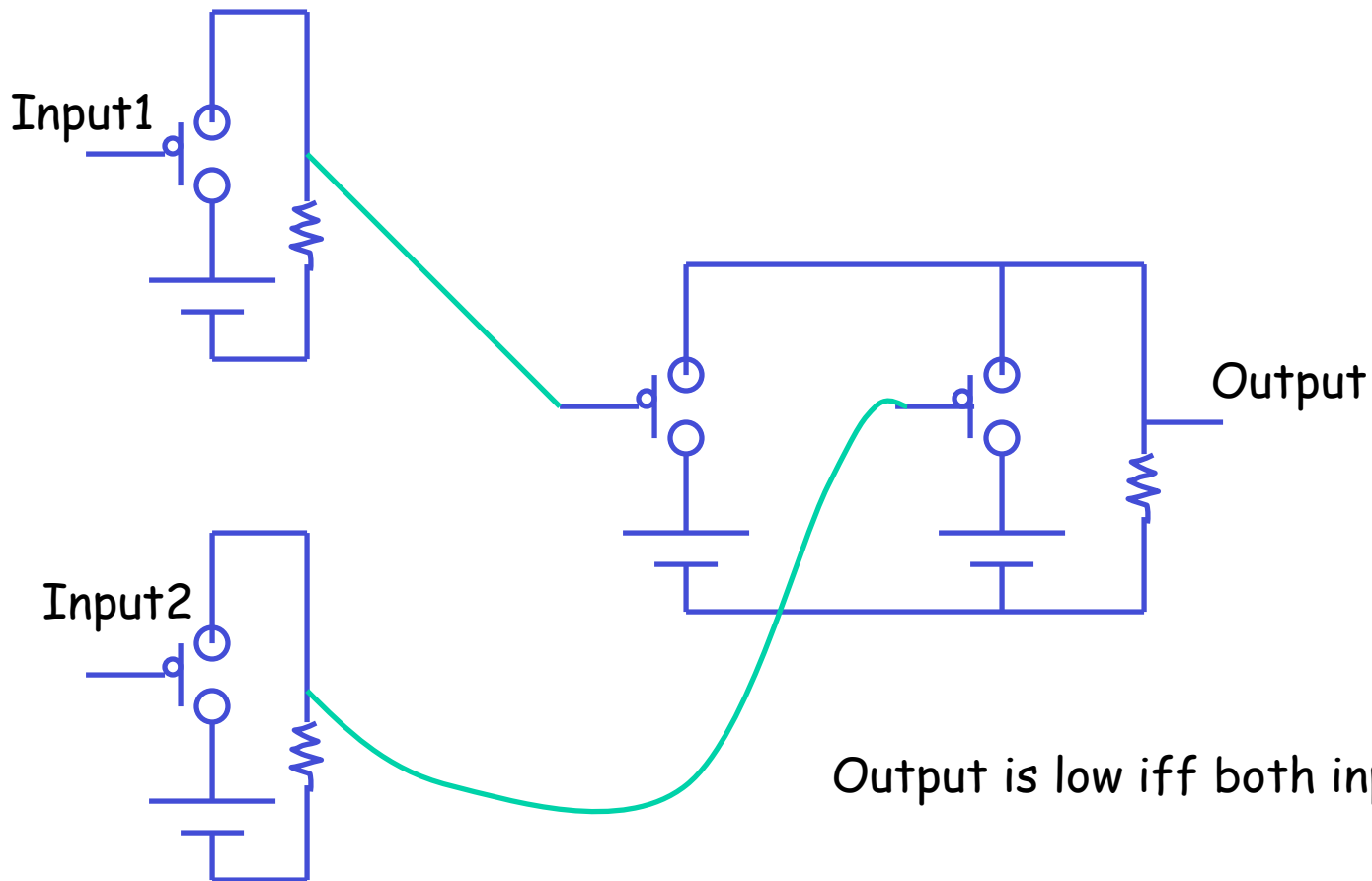
If either of the inputs is low, the output is low.

This is called an **AND** gate



Now, can you make an OR gate with switches?

OR Gate



Output is low iff both inputs are low

I.e. Output is high if either of the inputs (or both) are high (input1 OR input2)

Basic Gates

- There are three basic kinds of logic gates

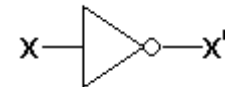
Operation:

AND
of two inputs

OR of two
inputs

NOT
(complement)
on one input

Logic gate:



• Two Questions:

- How can we implement such switches?
- What can we build with Gates? And How?

How to make switches?

- Use mechanical power
- Use hydrolic pressure
- Use electromechanical switches (electromagnet turns the switch on)
- Current technology:
 - Semiconductor transistors
 - A transistor can be made to conduct electricity depending on the input on the 3rd input
 - CMOS "gates" (actually, switches)
- We can now manufacture millions of transistors on a single silicon chip!

So, switches and Gates are no magic.

We believe they can be built

Two properties of Switches and Gates:

Size

Switching and Propagation delay

A little bit about technology

- Two properties of Switches and Gates:
 - Size
 - Switching and propagation delay
- Smaller the size, smaller the propagation delay (typically)!
- Smaller the size, cheaper the processor!
 - Silicon is sand anyway
 - But you can put more logic on a single chip
- This nice positive feedback cycle has
 - Made processors faster and cheaper
 - Over the last 30 years! (1972: Intel 4004)
 - Before that: A processor was built with **MANY** chips

What can we do with Gates?

- What do you want to do?
- Let us say we want to add numbers automatically
- What are numbers? How are they represented
 - Roman XVII
 - Decimal: 17
- How to add them, depends on how they are represented
 - One representation may be better than other for adding
 - Try adding two long roman numbers
 - Decimal is better but
 - We have only two "values", high and low, in our gates
 - So,
 - Let us think about why decimal is better
 - And can we design a representation that allows us to use the binary (hi/low) gates that we have.

Decimal review

- Numbers consist of a bunch of digits, each with a *weight*

1	6	2	.	3	7	5	Digits
100	10	1		1/10	1/100	1/1000	Weights

- These weights are all powers of the base, which is 10. We can rewrite this:

1	6	2	.	3	7	5	Digits
10^2	10^1	10^0		10^{-1}	10^{-2}	10^{-3}	Weights

- To find the decimal value of a number, multiply each digit by its weight and sum the products.

$$(1 \times 10^2) + (6 \times 10^1) + (2 \times 10^0) + (3 \times 10^{-1}) + (7 \times 10^{-2}) + (5 \times 10^{-3}) = 162.375$$

Now we can see why addition is easier with decimal system than the roman system. The idea of positional weights and carry!

Nothing special about 10!

- Decimal system (and the idea of "0") was invented in India around 100-500AD
- Why did they use 10? Anything special about it?
 - Not really.
 - Probably the fact that we have 10 fingers influenced this
- Will a base other than 10 work?
 - Sure: 345 in base 9 = $5 + 9 \cdot 4 + 9^2 \cdot 3 = 284$ in base 10
 - Base 9 has only 9 symbols: 1, 2, 3, 4, 5, 6, 7, 8, 0
 - What about base 2? (1 and 0)
 - 1101 in base 2: $1 + 2 \cdot 0 + 4 \cdot 1 + 8 \cdot 1 = 13$
- Base 2 system will work for our gates!
 - Base 2 Addition:
 - Compare this with decimal addition

+

1	0	0	1	1
0	0	1	1	0
1	1	0	0	1

Converting binary to decimal

- We can use the same trick to convert **binary**, or base 2, numbers to decimal. This time, the weights are powers of **2**.
 - Example: **1101.01** in binary

1	1	0	1	.	0	1	Binary digits, or bits
2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	Weights (in base 10)

- The decimal value is:

$$\begin{array}{cccccccc} (1 \times 2^3) & + & (1 \times 2^2) & + & (0 \times 2^1) & + & (1 \times 2^0) & + & (0 \times 2^{-1}) & + & (1 \times 2^{-2}) & = \\ 8 & + & 4 & + & 0 & + & 1 & + & 0 & + & 0.25 & = 13.25 \end{array}$$

Powers of 2:

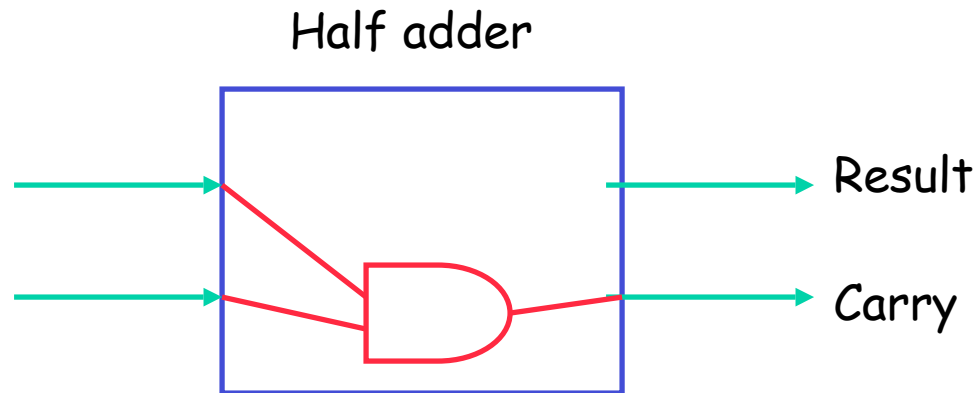
$2^0 = 1$	$2^4 = 16$	$2^8 = 256$
$2^1 = 2$	$2^5 = 32$	$2^9 = 512$
$2^2 = 4$	$2^6 = 64$	$2^{10} = 1024$
$2^3 = 8$	$2^7 = 128$	

Useful abbreviations:

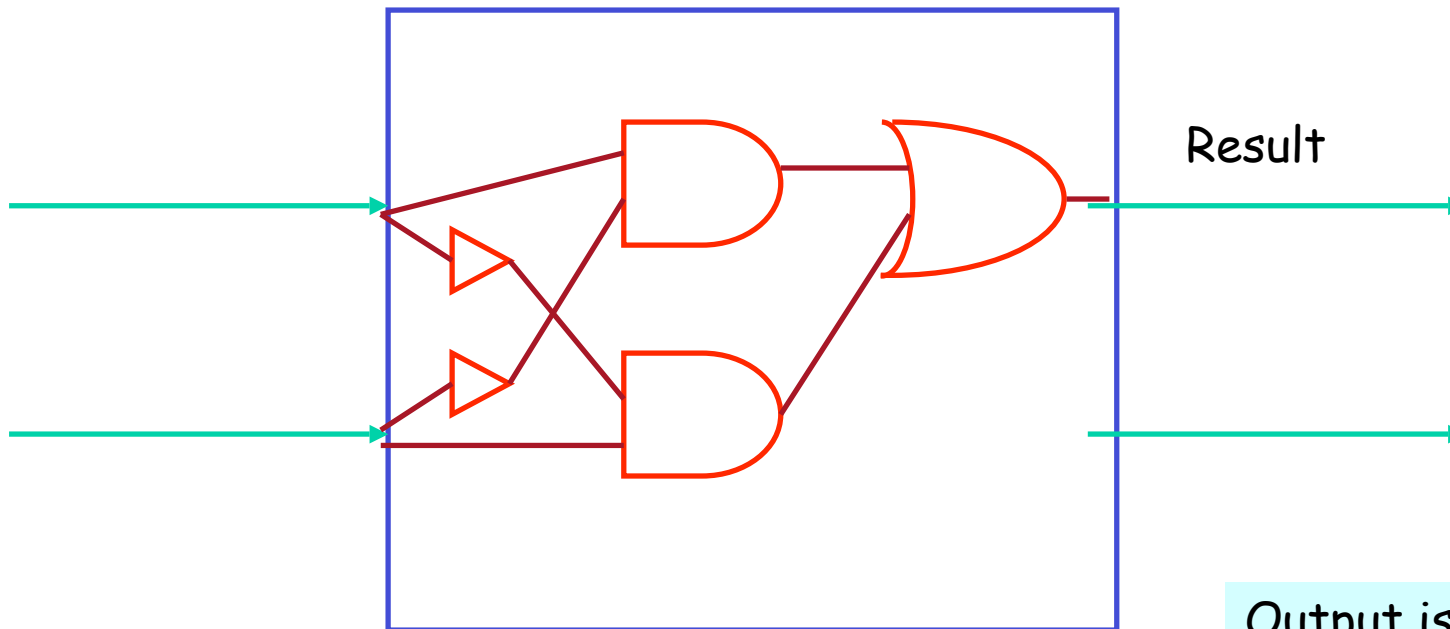
$K = 2^{10} = 1,024$
$M = 2^{20} = 1,048,576$
$G = 2^{30} = 1,073,741,824$

Doing addition with gates

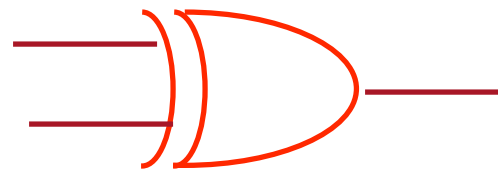
- Lets do simple stuff first:
 - Can we add two numbers each with just 1 bit?
 - Bit: binary digit
 - $0+0 = 0$, $0+1 = 1$, $1+0 = 1$, and $1+1 = ???$
 - 2. But 2 is not a symbol.
 - 10 (just as $5 + 5$ is 10 in decimal)
 - Result is 0 with 1 carried over to the next bit..
 - Whats 1 and 0? High and low voltage respectively.



Half adder: result



This circuit is so common, that it has a name and symbol as a gate by itself: Exclusive OR



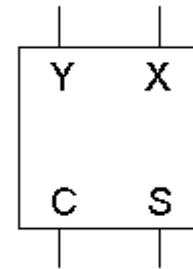
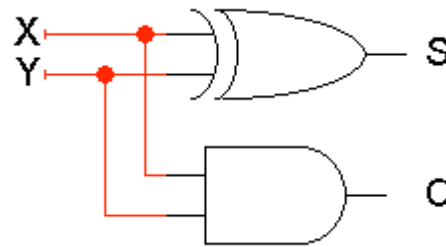
Exclusive OR

Output is 1 iff exactly one of the 2 inputs is 1

Adding two bits

- A **half adder** is used to add two bits.
- The result consists of two bits: a **sum** (the right bit) and a **carry out** (the left bit)
- Here is the circuit and its block symbol

$0 + 0 = 00$
 $0 + 1 = 01$
 $1 + 0 = 01$
 $1 + 1 = 10$



Adding three bits

- But what we really need to do is add *three* bits: the augend and addend, and the *carry in* from the right.

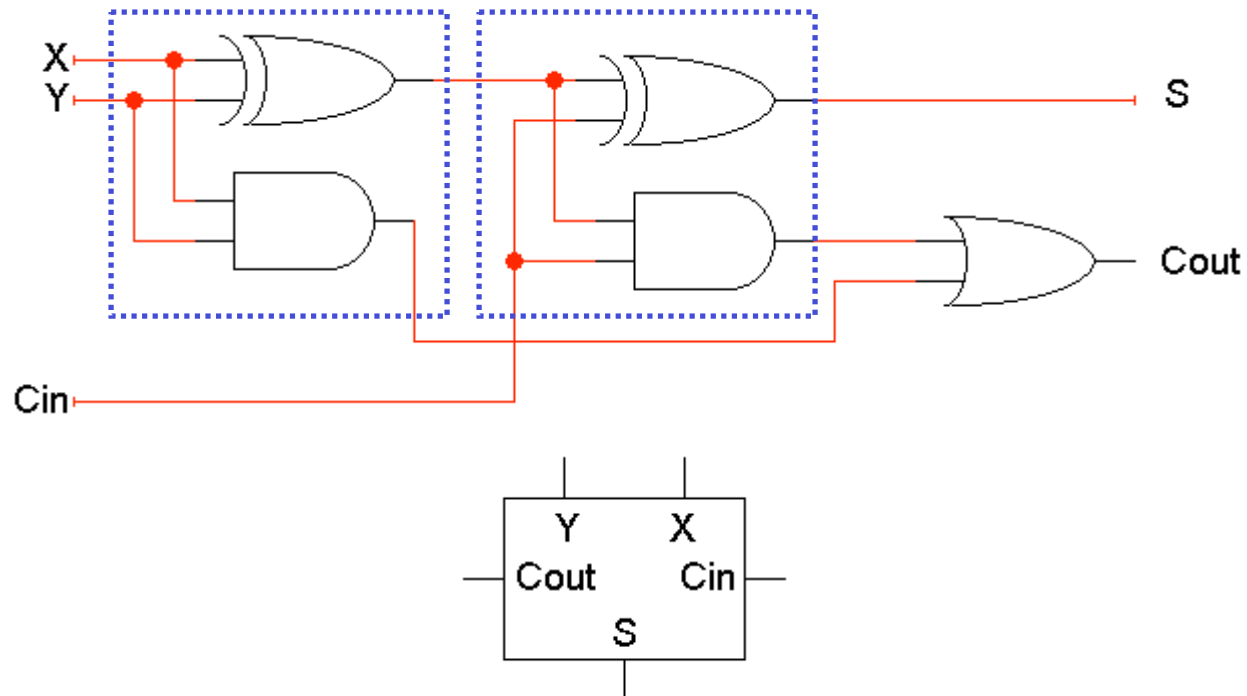
$$\begin{array}{r}
 \\
 \\
 + \\
 \hline
 1 \\

 \end{array}$$

X	Y	C _{in}	C _{out}	S	
0	0	0	0	0	0 + 0 + 0 = 00
0	0	1	0	1	0 + 0 + 1 = 01
0	1	0	0	1	0 + 1 + 0 = 01
0	1	1	1	0	0 + 1 + 1 = 10
1	0	0	0	1	1 + 0 + 0 = 01
1	0	1	1	0	1 + 0 + 1 = 10
1	1	0	1	0	1 + 1 + 0 = 10
1	1	1	1	1	1 + 1 + 1 = 11

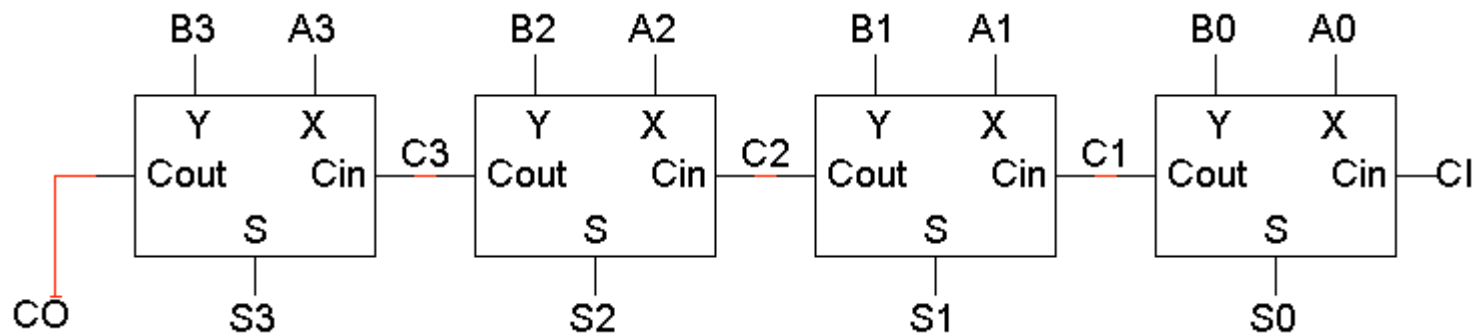
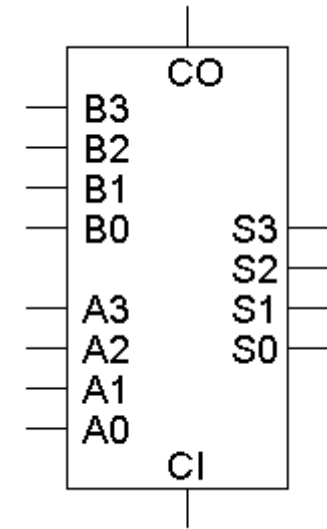
Full adder circuit

- Why are these things called half adders and full adders?
- You can build a full adder by putting together two half adders.



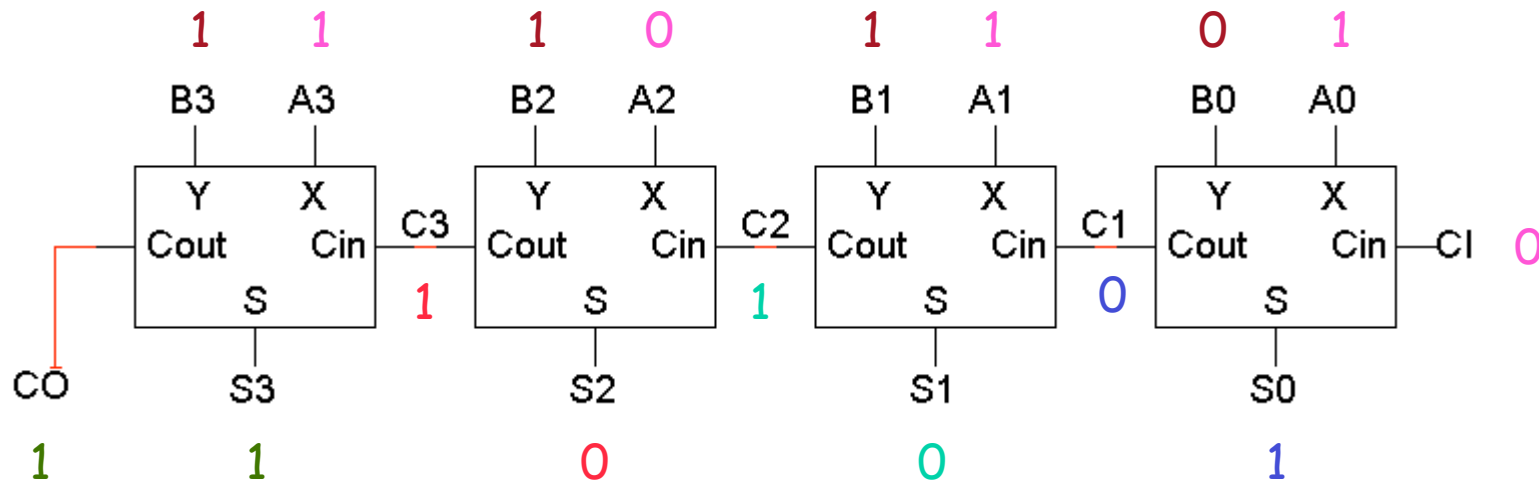
A 4-bit adder

- Four full adders together can make a 4-bit adder
- There are *nine* total inputs to the 4-bit adder:
 - two 4-bit numbers, $A_3 A_2 A_1 A_0$ and $B_3 B_2 B_1 B_0$
 - an initial carry in, CI
- The *five* outputs are:
 - a 4-bit sum, $S_3 S_2 S_1 S_0$
 - a carry out, CO



An example of 4-bit addition

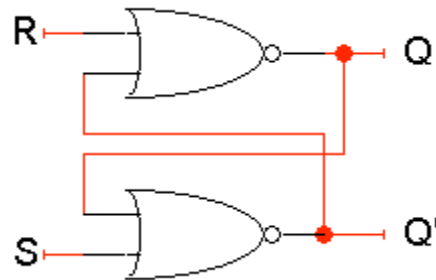
- Let's put our initial example into this circuit: $A=1011$, $B=1110$



- Step 1: Fill in all the inputs, including $CI=0$
- Step 2: The circuit produces $C1$ and $S0$ ($1 + 0 + 0 = 01$)
- Step 3: Use $C1$ to find $C2$ and $S1$ ($1 + 1 + 0 = 10$)
- Step 4: Use $C2$ to compute $C3$ and $S2$ ($0 + 1 + 1 = 10$)
- Step 5: Use $C3$ to compute CO and $S3$ ($1 + 1 + 1 = 11$)
- The final answer is 11001

Now that we can add, how about some memory?

- We want to save results computed before, and recall them in a later calculation, for example
- Gates help us build memory
- How can a circuit "remember" anything on its own?
 - After all, the values on the wires are always changing, as outputs are generated in response to inputs.
- The basic idea is feedback: we make a "loop" in the circuit, so the circuit outputs are inputs as well



Set and Reset inputs...

When S and R are 0, Q is "stable": whatever it was, it stays in that state. Ergo : memory.

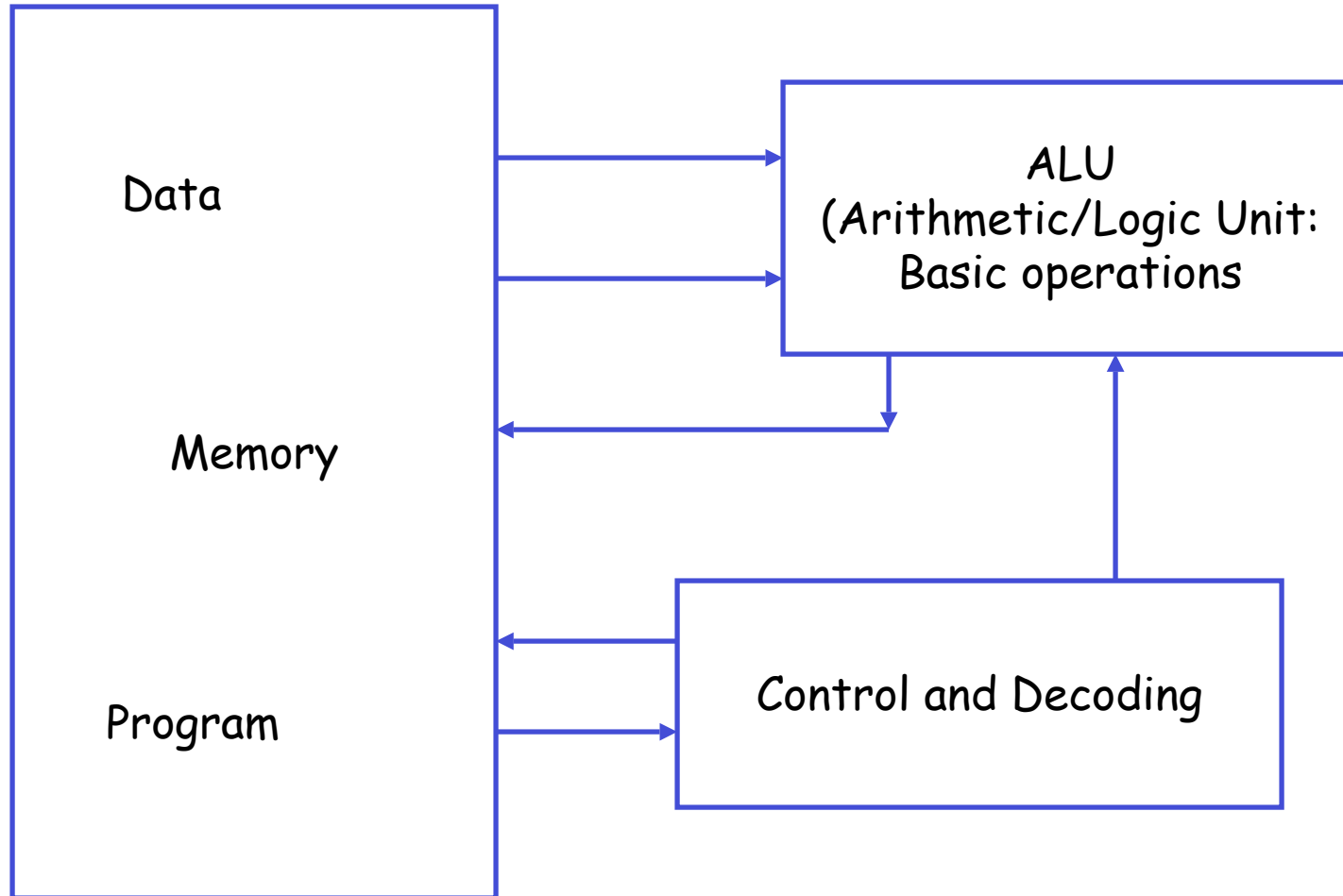
When S is 1 and R is 0, Q becomes 1

When R is 1 and S is 0, Q becomes 0

So, we have built a calculator

- It is not a computer yet...
 - We have to type each step into a calculator
 - We'd like to "program" standard steps
 - E.g. Add 57 numbers sitting in memory in specific places
 - Also, support other operations (subtract..)
- Two new ideas and components are needed for this:
 - Addressable memory
 - Stored Program
- Addressable memory
 - Memory organized in a bunch of locations, such that contents of specified location can be brought back to the adder when needed.
 - Each memory location has an "address" (binary, of course)
- Stored Program:
 - The instructions for which numbers to operate on, what operation to do (add/subtract, ..) and where to store the result
 - The instructions themselves can be represented in binary and stored in the memory!
 - The processor must have circuits to decode and "interpret" these instructions

Components of a basic computer



Summary

- Controllable Switches are easy to make
- These switches can be used to put together "Logic Gates"
- Logic Gates can be put together to make half adder, full adders and multi-bit adders
 - So we can see they can be used for other such circuits as well
- Logic Gates can be used to make circuits that "remember" or store data
- A Computer includes, at its heart :
 - An ALU (Arithmetic Logic Unit)
 - Instruction Decoding and associated circuits
 - Memory
 - Stored Program