Gates and Circuits

9/13/16

You’re going to want scratch paper today ... borrow some if needed.
The system stack

This class

Starting this week

hardware

software

software

c program
compiler
shell
operating system
memory
CPU
circuits
transistors
wires
electrical engineering
# How a Computer Runs a Program

<table>
<thead>
<tr>
<th>C Program</th>
<th>How C program is run on System:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Program</td>
<td>How instructions &amp; data are encoded</td>
</tr>
<tr>
<td>Operating System</td>
<td>OS Abstractions, Resource management</td>
</tr>
<tr>
<td>Computer Hardware</td>
<td>How underlying HW organized &amp; works</td>
</tr>
</tbody>
</table>

## What we know so far:

- Much of the C programming language
  - types, operators, arrays, parameter passing, some structs
- Binary encodings & sizes for different C types
  - char, unsigned char, int, unsigned int, ...
- How to perform binary operations (Add, Sub)
Von Neumann Architecture

• A computer is a generic computing machine:
  • Based on Alan Turing’s Universal Turing Machine
  • Stored program model: computer stores program rather than encoding it (feed in data and instructions)
  • No distinction between data and instructions memory

• 5 parts connected by buses (wires):
  • Memory, Control, Processing, Input, Output
Memory

• Stores instructions and data.
• Addressable, like array indices.
  • addr 0, 1, 2, ...

• Memory Address Register: address to read/write
• Memory Data Register: value to read/write
Central Processing Unit (CPU)

• **Processing Unit**: executes instructions selected by the control unit
  • ALU (arithmetic logic unit): simple functional units: ADD, SUB, AND...
  • Registers: temporary storage directly accessible by instructions

• **Control unit**: determines the order in which instructions execute
  • PC: program counter: address of next instruction
  • IR: instruction register: holds current instruction
  • clock-based control: clock signal+IR trigger state changes
Input/Output

• Keyboard
• Files on the hard drive
• Network communication
First Goal: Build a model of the CPU

Three main classifications of HW circuits:

1. **ALU:** implement arithmetic & logic functionality
   (ex) adder to add two values together

2. **Storage:** to store binary values
   (ex) Register File: set of CPU registers, Also: main memory (RAM)

3. **Control:** support/coordinate instruction execution
   (ex) fetch the next instruction to execute
Abstraction

- User / Programmer
  Wants low complexity

- Applications
  Specific functionality

- Software library
  Reusable functionality

- Operating system
  Manage resources

- Complex devices
  Compute & I/O
Abstraction

Complex devices
Compute & I/O

Hardware Circuits
Logic Gates
Transistors

Here be dragons.
(Electrical Engineering)
(Physics)
Logic Gates

Input: Boolean value(s) (high and low voltages for 1 and 0)
Output: Boolean value, the result of a Boolean function

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A &amp; B</th>
<th>A</th>
<th>B</th>
<th>~A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
More Logic Gates

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A NAND B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Note the circle on the output. This means “negate it.”
Combinational Logic Circuits

- Build up higher level processor functionality from basic gates.

Outputs are Boolean functions of inputs.
Outputs continuously respond to changes to inputs.
What does this circuit output?

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Out_A</th>
<th>Out_B</th>
<th>Out_C</th>
<th>Out_D</th>
<th>Out_E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Build new gates

- Build-up XOR from basic gates (AND, OR, NOT)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A ^ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Q: When is $A ^ B = 1$?

$A \ ^ \ B \ = \ (\neg A \ & \ B) \ | \ (A \ & \ \neg B)$
Which of these is an XOR circuit?

Draw an XOR circuit using AND, OR, and NOT gates.

I’ll show you the clicker options after you’ve had some time.
Which of these is an XOR circuit?

A:

B:

C:

D:

E: None of these is an XOR.
Checking the XOR circuit

\[ A \oplus B = (\neg A \land B) \lor (A \land \neg B) \]

\[
\begin{array}{c|c|c}
A & B & A \oplus B \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]
Abstracting the XOR circuit

\[ A^\wedge B \equiv (\neg A \& B) \mid (A \& \neg B) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A ^ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
First Goal: Build a model of the CPU

Three main classifications of HW circuits:

1. **ALU**: implement arithmetic & logic functionality
   (ex) adder to add two values together

2. **Storage**: to store binary values
   (ex) Register File: set of CPU registers

3. **Control**: support/coordinate instruction execution
   (ex) fetch the next instruction to execute
Building an ALU via abstraction

Step 1: zoom in
- Build circuits for each operation the ALU must perform
  - Arithmetic
  - Integer addition, subtraction, multiplication ...
  - Floating point addition, subtraction, multiplication ...
  - Logic
  - Bitwise operations: AND, OR, ...
  - Shifts: left, right, arithmetic

Step 2: zoom out
- Take each component circuit as given.
- Connect the components to memory and control circuits.
Addition Circuits via abstraction

- We want to build an N-bit (e.g. 32-bit) adder.

- Step 1: design a 1-bit adder.

- Step 2: string N 1-bit adders together.
1-bit adder

Inputs: A, B
Outputs: sum, cout

Let’s fill in the truth table.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Sum (A+B)</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Which of these circuits is a one-bit adder?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Sum (A+B)</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

A:  

![Diagram A](image1)

B:  

![Diagram B](image2)

C:  

![Diagram C](image3)

D:  

![Diagram D](image4)

E: None of these
What’s missing?

• This circuit is called a half-adder.

  A  B  Sum  Cout
  0  0   0    0
  0  1   1    0
  1  0   1    0
  1  1   0    1

• A one-bit full-adder takes a third input: cin.

  0011010
+  0001111
Which of these is a full-adder?

Hint: use abstraction. Start with two half-adders and connect them appropriately.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Half-Adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>Sum</th>
<th>Cout</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>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Full-Adder
Which of these is a full-adder?

A: 

B: 

C: 

D: None of these.
N-bit adder (ripple-carry adder)
3-bit ripple-carry adder

010 (2) + 011 (3) = 011 (3)
Arithmetic Logic Unit (ALU)

• One component that knows how to manipulate bits in multiple ways
  • Addition
  • Subtraction
  • Multiplication / Division
  • Bitwise AND, OR, NOT, etc.

• Built by combining components
  • Take advantage of sharing HW when possible (e.g., subtraction using adder)
Simple 3-bit ALU: Add and bitwise OR

3-bit inputs A and B:

At any given time, we only want the output from ONE of these!
Simple 3-bit ALU: Add and bitwise OR

3-bit inputs A and B:

- $A_0$
- $A_1$
- $A_2$
- $B_0$
- $B_1$
- $B_2$

Extra input: control signal to select Sum vs. OR

3-bit adder outputs $Sum_0$, $Sum_1$, $Sum_2$

Circuit that takes in $Sum_{0-2}$ / $Or_{0-2}$ and only outputs one of them, based on control signal.

$Or_0$

$Or_1$

$Or_2$
Which of these circuits lets us select between two inputs?

A:

B:

C:
Multiplexor: Chooses an input value

Inputs: \(2^N\) data inputs, N signal bits

Output: is one of the \(2^N\) input values

- Control signal \(s\), chooses the input for output
  - When \(s\) is 1: choose \(a\), when \(s\) is 0: choose \(b\)

\[
\text{out} = (s \& a) \lor (\sim s \& b)
\]
N-Way Multiplexor

Choose one of N inputs, need $\log_2 N$ select bits

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>$s_0$</th>
<th>choose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>D0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>D1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>D2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>D3</td>
</tr>
</tbody>
</table>

4-Way Multiplexor

MUX4

S Input to choose D0
Simple 3-bit ALU: Add and bitwise OR

3-bit inputs
A and B:

\[ A_0 \quad A_1 \quad A_2 \]
\[ B_0 \quad B_1 \quad B_2 \]

Extra input: control signal to select Sum vs. OR

3-bit adder

\[ \text{Sum}_0 \quad \text{Sum}_1 \quad \text{Sum}_2 \]

Multiplexer!

\[ \text{Or}_0 \quad \text{Or}_1 \quad \text{Or}_2 \]
1. Build a subtraction circuit

• Start with a 4-bit addition circuit.
• Create a 4-bit subtraction circuit.

2. Build an ALU that does + and -

• Use one 4-bit adder circuit.
• This adder should be used to perform addition and subtraction.
• Add control circuitry (a multiplexor) to determine which operation gets performed.