Sit at the same table as Wednesday

Stack machines

$x + y \times z + u$

no architectural registers

many architectures (real and virtual) have been designed this way

push x
push y
push z
multiply
add
push u
add

Administrivia

- HW3 due today (9/29), Lab 4 due Monday
- Review on Wed 10/4 in class for Midterm 10/6
Where we are going

Instruction set architecture (e.g., MIPS)

Compiler

translating source code (C or Java)
Programs to assembly language
And linking your code to
Library code

Memory system

I/O system

Processor

How the software talks
To the hardware

Datapath & Control

How a processor runs MIPS
Programs!

I/O system

Digital logic

How switches (1 or 0) can be used
to build Interesting functions:
from integer arithmetic to
programmable computers
“Synchronous digital systems”

- We’ll be designing circuits using two very convenient **abstractions** of electrical circuits
  - **digital**: voltage is quantized into two values: 0 and 1, allowing us to combine components without increasing complexity
  
  ![NAND gate diagram](image)

  ![NAND gate diagram](image)

- **synchronous**: time is discrete, that is, time advances in distinct steps and the values on all the wires changes at these steps
  
  ![Clock and D-Q waveforms](image)
Digital abstraction

• Compose circuits without them affecting each other
• Why? outputs are “perfect” high and low voltages
Synchronous abstraction

- time is discrete
- time advances in distinct steps and the values on all the wires changes at these steps
“Synchronous digital systems”

• We’ll be designing circuits using two very convenient abstractions of electrical circuits
  • digital: voltage is quantized into two values: 0 and 1, allowing us to combine components without increasing complexity
  • synchronous: time is discrete, that is, time advances in distinct steps and the values on all the wires changes at these steps

• Two parts to synchronous digital systems
  1. combinational logic
     • ignore time and just think about inputs and outputs
     • e.g., bitwise operations and arithmetic on binary numbers
  2. synchronous logic
     • we’ll incorporate state, which will change as time advances
     • e.g., memory and registers
Combinational logic

represent an input using a boolean variable

- **boolean**: domain is \{false,true\}, or think of it as \{0,1\}

three basic operations: AND, OR, NOT

notation:

\(A \cdot B\) means “true iff A is true and B is true”
\(A + B\) means “true iff A is true or B is true”
\(\bar{A}\) means “true if A is not true”

Combine operations together to build more complex expressions

for example, read this expression: 
\((A \cdot B) + (\bar{A} \cdot C)\)
## Truth tables

### $C = A \cdot B$

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

### $C = \overline{A}$

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

### $D = (A \cdot B) + (A \cdot C)$

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

defines a logical function by enumerating all inputs and the corresponding output
Peer instruction

• Which logic equation describes this truth table?

\[ a) \quad C = A \cdot B \]
\[ b) \quad C = \overline{A} \cdot \overline{B} \]
\[ c) \quad C = \overline{A + B} \]
\[ d) \quad C = \overline{A} \cdot B \]
The basic component: a switch

Current flows from the battery when there is a complete circuit.

When the switch is closed (connects the circuit), the current flows to the light.

When the switch is open (disconnects the circuit), the current does not flow to the light.
Building logic from switches

Two switches connected in parallel. At least one must be closed for the bulb to turn on.

Two switches connected in series. Both must be closed for the bulb to turn on.
Logic gates

The state of the switch is the input to the circuit.
Peer instruction

What is the truth table for this circuit?
Controlling the switches

• To compose two gates, we need to be able to change the state of the switches of the second gate with the output of the first gate.

![Diagram of a combined OR-AND gate with switches and inputs/outputs labeled.]