Many ways to build logic out of MOSFETs

pass transistor logic (most similar to the first switch logic we saw)

dynamic CMOS logic
  • Clock=0 *precharges* X through the pMOS
  • clock=1 gives the two nMOS time to drain charge on X to ground (if A and B turn them on)
Switches other than transistors

- Relays (electro-mechanical)
- Diodes

OR gate

[Image of relays]
[Image of diodes]

Wikimedia commons
CS 2630
Computer Organization

Combinational logic for arithmetic
Brandon Myers
University of Iowa
Peer instruction

• Draw the circuit for the following truth table (this one has 2 inputs, 2 outputs)

<table>
<thead>
<tr>
<th>In0</th>
<th>In1</th>
<th>Out0</th>
<th>Out1</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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Answer the number of logic gates used when finished; work in pairs
Where we are going

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

Instruction set architecture (e.g., MIPS)
- How the software talks To the hardware

Memory system

Processor
- How a processor runs MIPS Programs!

I/O system
- How switches (1 or 0) can be used to build Interesting functions:
  - from integer arithmetic to programmable computers

Datapath & Control

Digital logic
Ingredients for a processor

• Combinational logic
  • arithmetic and bitwise operations
  • multiplexors (pick 1 out of N inputs)
  • control logic
  • branch calculation
  • address calculation

• Synchronous logic (we’ll get to these later)
  • registers
  • program counter, or PC
  • data memory
  • instruction memory

start learning how to build these combinational logic circuits
How do we add two numbers

addu $t0, $t1, $t2

We know: The numbers are physically stored in registers as bits

The sum of two numbers is a function we can implement using combinational logic
Start with a 1-bit adder

Carry in

A  B

+  

Sum  Carry out
Peer instruction: 2-bit adder

Number of rows of truth table for Sum1 output of 2-bit adder?
How many rows in this truth table?

Current state (the input)

4GB memory x 8bit/byte = 32 Gbits

32 registers x 4 Byte x 8bit/byte = 1Kbit

PC register x 4 Byte x 8bit/byte = 32 bits

New state (the output)

Processor takes one step from current state to new state. How many rows in the truth table of bit 0?

https://commons.wikimedia.org/wiki/File:Performance_PIPER_die.JPG
Administrivia

- 2 Midterm practices are on ICON
  - take them like a real exam and *then* check your answers with the solutions

- Debug Your Brain next Wednesday will be dedicated to review
  - extra office hour 4-5pm on Monday 2/27, to be dedicated to HW3

- Reminder: Midterm 3/6
  - open notes open book, no devices
Field-programmable gate array (FPGA)
an alternative to manufacturing your own silicon chip

Look up table (LUT) – a **programmable truth table** in hardware!

“3-LUT” means 3 inputs (so must program 8 rows)

https://en.wikipedia.org/wiki/Field-programmable_gate_array
CS 2630
Computer Organization

Adders and timing
Brandon Myers
University of Iowa
How humans add two numbers

- compute $652_{10} + 367_{10}$ on paper

- compute $11010_2 + 01011_2$ on paper
Building larger circuits out of *components*: 2-bit adder built from 1-bit adders

Individual components have their own small truth tables; then we just combine the components.
32-bit adder

Reflect: We now have a circuit for A+B. What is a small change we can make to get A-B?
Under the rug of combinational logic: computation time
Propagation delay through logic gates

Resistor-capacitor (RC) model of NOT gate when given input 0

- Input changes:
  - $1 \rightarrow 0$
  - $0 \rightarrow 1$

- Circuit diagram with RC model:
  - Input $0 \rightarrow 1$
  - Output $1 \rightarrow 0$

- NOT gate symbol with input and output transitions:
  - Input $0 \rightarrow 1$
  - Output $1 \rightarrow 0$
Charging a capacitor in an RC circuit

Propagation delay through logic gates

Resistor-capacitor (RC) model of NOT gate when given input 0

When the input changes from 1 to 0, there is a delay before the output of the NOT gate reaches 1 draining the capacitor when input goes 0→1
Back up to the level of logic gates

• In CS2630, when we think about delay, **we’ll use the following simplified model**
  • propagation delay is higher for more complex gates
  • propagation delay increases with the number of inputs to a gate
    • e.g., 2-input OR gate is faster than a 3-input OR gate
  • propagation delay of a sequence of gates is additive
  • all inputs must be available before logic gate begins computing

Example: assume delay of inverter is 0.5 ns

\[
\begin{array}{c}
\text{T=0} & \text{T=0.5} & \text{T=1} & \text{T=1.5} \\
\text{NOT} & \text{NOT} & \text{NOT} & \\
\end{array}
\]
Delay of adder circuit

1-bit adder delay: sum=2
1-bit adder delay: carry=1

Peer instruction: delay to get Sum31?
Delay of adder circuit

This adder is called “ripple carry adder”
Next ...

• We now know how one kind of adder works and how long it takes to compute the result
• Can we build faster adders? Yes! Use hierarchy/trees
Administrivia

• Reminders:
  • extra office hour 4-5pm today for HW3
  • DYB this week dedicated to review
  • Midterm on Monday 3/6: open note open book, no devices
Carry-select adder
Peer instruction

What is the delay from the time A and B are available to when Sum is available?

delays:
sum: 1
carry: 1
mux: 1
Administrivia

• Reminders:
  • DYB tonight 6-7pm dedicated to review
    • bring your prioritized list of specific problems or questions that you’d like to review
    • it will be most useful if you take the practice exam(s) first
  • Midterm on Monday 3/6: open note open book, no devices
Peer instruction

delays:
sum: 1
carry: 1
mux: 1

time signal is available shown in red/bold

What is the delay from the time A and B are available to when Sum is available?
Best case delay of adders? $O(\log n)$

compute carries

use carries to finish the sums

$p$ means “propagate a carry”

$g$ means “generate a carry”

important: $p$ and $g$ are not functions of $c$

Best case delay of adders? O(log n)

compute carries

critical path

use carries to finish the sums

Multiplication

• Evaluate $501_{10} \times 422_{10}$ on paper

• Evaluate $1100_2 \times 1001_2$ on paper
Aside: useful logic circuit notation

**Splitter (2-bit to 2 1-bit)**

**Same wire used twice**

**Splitter (2 1-bit to 2-bit)**

**Cannot attach 2 inputs!**
Shifting by a constant

A

110000001

B = A << 1

B

100000010
Variable shifter: first attempt with 1-bit muxes

...5 more muxes, one to calculate each bit of the output

$B = A << C$
Variable shifter, similar design with 8-bit mux

Can you think of a design that avoids using a huge 8-bit x 8-choice mux?

e.g., using only 8-bit x 2-choice muxes and hardwired shifters?
Multi-stage shifter using the magic of binary decision-making

• Hint: Recall the envelopes from HW1

another solution (but uses more 2-input MUXs) is to build the 8-input MUX on the previous slide using 2-input MUXs, like a soccer tournament with 8 teams and one champion
Peer instruction

• Build a 2-bit multiplier (4-bit output)
• you may use any number of

• example test cases:
  • 01 * 01 = 0001
  • 10 * 01 = 0010
  • 10 * 11 = 0110
  • 11 * 11 = 1001