Numbers & Arithmetic

Hakim Weatherspoon
CS 3410, Spring 2012
Computer Science
Cornell University

See: P&H Chapter 2.4 - 2.6, 3.2, C.5 – C.6
Example: Big Picture

- Computer System Organization and Programming platform from 10 years ago
Goals for today

Today

- Review Logic Minimization
- Build a circuit (e.g. voting machine)
- Number representations
- Building blocks (encoders, decoders, multiplexors)

Binary Operations

- One-bit and four-bit adders
- Negative numbers and two’s compliment
- Addition (two’s compliment)
- Subtraction (two’s compliment)
- Performance
## Logic Minimization

- How to implement a desired function?

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\text{out} = \overline{a} \overline{b} c + \overline{a} \overline{b} \overline{c} + \overline{a} b \overline{c} + ab \overline{c}
\]
Logic Minimization

• How to implement a desired function?

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>out</th>
<th>minterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\overline{a} \overline{b} \overline{c}$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$\overline{a} \overline{b} c$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$\overline{a} b \overline{c}$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$\overline{a} b c$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$a \overline{b} \overline{c}$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$a \overline{b} c$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$a b \overline{c}$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$a b c$</td>
</tr>
</tbody>
</table>

sum of products:
• OR of all minterms where out=1

corollary: *any* combinational circuit *can be* implemented in two levels of logic (ignoring inverters)
Karnaugh Maps

How does one find the most efficient equation?
– Manipulate algebraically until...?
– Use Karnaugh maps (optimize visually)
– Use a software optimizer

For large circuits
– Decomposition & reuse of building blocks
Minimization with Karnaugh maps (1)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>0</td>
</tr>
<tr>
<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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Out = \overline{abc} + \overline{a}bc + a\overline{bc} + a\overline{t}c
Minimization with Karnaugh maps (1)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>0</td>
</tr>
<tr>
<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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Sum of minterms yields
- $\overline{abc} + \overline{abc} + \overline{abc} + \overline{abc}$
Minimization with Karnaugh maps (2)

Sum of minterms yields
\[
\overline{abc} + \overline{abc} + \overline{abc} + \overline{abc}
\]

Karnaugh maps identify which inputs are (ir)relevant to the output.
Minimization with Karnaugh maps (2)

Sum of minterms yields
- \( \overline{abc} + \overline{abc} + \overline{abc} + \overline{abc} \)

Karnaugh map minimization
- Cover all 1’s
- Group adjacent blocks of \(2^n\) 1’s that yield a rectangular shape
- Encode the common features of the rectangle
  - out = \( \overline{a}b + \overline{ac} \)
Karnaugh Minimization Tricks (1)

\[
\text{out} = a \overline{c} + b \overline{c} + a \overline{b}
\]
Karnaugh Minimization Tricks (1)

Minterms can overlap

- \( \text{out} = b\bar{c} + a\bar{c} + ab \)

Minterms can span 2, 4, 8 or more cells

- \( \text{out} = \bar{c} + ab \)
### Karnaugh Minimization Tricks (2)

The images show two Karnaugh maps with labeled axes `ab` and `cd` and four rows and columns. The maps are filled with binary values:

#### Top Map
- **Top-left** row: 00, 01, 11, 10
- Values: 00, 01, 00, 00

#### Bottom Map
- **Top-left** row: 00, 01, 11, 10
- Values: 10, 00, 00, 10

The maps are used to identify and mark adjacent 1s for simplification. The `out` values for each map are marked as `6d (note: handwritten)`. This suggests that the goal is to simplify the expression to these indicated output values.
Karnaugh Minimization Tricks (2)

- The map wraps around
  - out = $\overline{bd}$

- out = $\overline{bd}$
# Karnaugh Minimization Tricks (3)

## Diagram 1

<table>
<thead>
<tr>
<th>cd</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Out = \( \overline{A} \)

## Diagram 2

<table>
<thead>
<tr>
<th>cd</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Out = \( \overline{B} \)
Karnaugh Minimization Tricks (3)

• “Don’t care” values can be interpreted individually in whatever way is convenient
  – assume all x’s = 1
  – out = d

  – assume middle x’s = 0
  – out = \overline{bd}

  – assume 4^{th} column x = 1
  – out = \overline{bd}
A multiplexer selects between multiple inputs
- out = a, if d = 0
- out = b, if d = 1

- Build truth table
- Minimize diagram
- Derive logic diagram
Multiplexer Implementation

- Build a truth table
  
  \[ = abd + ab\overline{d} + a\overline{b}d + a\overline{b}\overline{d} \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>d</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>
</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>
Multiplexer Implementation

- Build the Karnaugh map

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>d</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>
</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>

<table>
<thead>
<tr>
<th>d&lt;sup&gt;ab&lt;/sup&gt;</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Multiplexer Implementation

- Derive Minimal Logic Equation

\[ \text{out} = ad + bd \]
Multiplexer Implementation

- Derive Minimal Logic Equation

\[ \text{out} = \overline{ad} + bd \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>d</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>
</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>

Truth Table and Circuit Diagram
Question

• How many logic gates and transistors did we save with minimized circuit?
  – (do not count inverters)
  – out = abd + abd + abd + abd
  – out = ad + bd

• (a) 2 gates and 16 transistors
• (b) 2 gates and 8 transistors
• (c) 4 gates and 8 transistors
• (d) 8 gates and 8 transistors
• (e) none of the above
Logic Gates

- One can buy gates separately
  - ex. 74xxx series of integrated circuits
  - cost ~$1 per chip, mostly for packaging and testing

- Cumbersome, but possible to build devices using gates put together manually
Integrated Circuits

- Or one can manufacture a complete design using a custom mask
- Intel Westmere has approximately 1.17 billion transistors
Recap

• We can now implement any logic circuit
  – Can do it efficiently, using Karnaugh maps to find the minimal terms required
  – Can use either NAND or NOR gates to implement the logic circuit
  – Can use P- and N-transistors to implement NAND or NOR gates
Voting machine

• Lets build something interesting

• A voting machine

• Assume:
  – A vote is recorded on a piece of paper,
  – by punching out a hole,
  – there are at most 7 choices
  – we will not worry about “hanging chads” or “invalids”
Voting machine

• For now, let’s just display the numerical identifier to the ballot supervisor
  – we won’t do counting yet, just decoding
  – we can use four photo-sensitive transistors to find out which hole is punched out

• A photo-sensitive transistor detects the presence of light
• Photo-sensitive material triggers the gate
Ballot Reading

– Input: paper with a hole in it
– Output: number the ballot supervisor can record

Ballots

The 3410 optical scan vote counter reader machine
Input

• Photo-sensitive transistor
  • photons replenish gate depletion region
  • can distinguish dark and light spots on paper

• Use array of N sensors for voting machine input
Output

- 7-Segment LED
- photons emitted when electrons fall into holes
Block Diagram

detect

N

enc → bin → dec → 8

8
Encoders

- N might be large
- Routing wires is expensive
- More efficient encoding?

$\log_2(N)$
Number Representations

- Base 10 - Decimal

6 3 7

10^2 10^1 10^0

- Just as easily use other bases
  - Base 2 - Binary
  - Base 8 - Octal
  - Base 16 - Hexadecimal
Counting

- Counting

\[
\begin{array}{c|c}
\text{doc} & \text{oct} \\
\hline
0 & 0 \\
1 & 1 \\
2 & 2 \\
3 & 3 \\
4 & 4 \\
5 & 5 \\
6 & 6 \\
7 & 7 \\
8 & 8 \\
9 & 9 \\
10 & 10 \\
11 & 11 \\
12 & 12 \\
22 & 22 \\
99 & 99 \\
100 & 100 \\
\end{array}
\]
Base Conversion

• Base conversion via repetitive division
  – Divide by base, write remainder, move left with quotient

\[ 637 \div 10 = 63 \text{ rem } 7 \]
\[ 63 \div 10 = 6 \text{ rem } 3 \]
\[ 6 \div 10 = 0 \text{ rem } 6 \]
Base Conversion

- Base conversion via repetitive division
  - Divide by base, write remainder, move left with quotient

\[
\begin{align*}
637 \div 2 &= 318 \text{ rem } 1 \\
318 \div 2 &= 159 \text{ rem } 0 \\
159 \div 2 &= 79 \text{ rem } 1 \\
79 \div 2 &= 39 \text{ rem } 1 \\
39 \div 2 &= 19 \text{ rem } 1 \\
19 \div 2 &= 9 \text{ rem } 1 \\
9 \div 2 &= 4 \text{ rem } 1 \\
4 \div 2 &= 2 \text{ rem } 0 \\
2 \div 2 &= 1 \text{ rem } 0 \\
1 \div 2 &= 0 \text{ rem } 1
\end{align*}
\]

56 = 101101110101101011001011011100111011101110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100111001110011100!}
Base Conversion

• Base conversion via repetitive division
  – Divide by base, write remainder, move left with quotient

\[
\begin{align*}
637 \div 16 &= 39 \text{ r } 13 \\
16 &\equiv 2 \text{ r } 7 \\
16 &\equiv 0 = 2 \\
0 \times 2 &\equiv 2
\end{align*}
\]
Hexadecimal, Binary, Octal Conversions
A 3-bit encoder with 4 inputs for simplicity
A 3-bit encoder with 4 inputs for simplicity

- $o2 = \overline{abcd}$
- $o1 = \overline{abcd} + \overline{abcd}$
- $o0 = \overline{abcd} + \overline{abcd}$

Encoder Truth Table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>o2</th>
<th>o1</th>
<th>o0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<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>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Ballot Reading

- Ok, we built first half of the machine
- Need to display the result

Ballots

The 3410 optical scan vote counter reader machine
7-Segment LED Decoder

- 3 inputs
- Encode 0 – 7 in binary
- 7 outputs
- One for each LED
7 Segment LED Decoder Implementation

<table>
<thead>
<tr>
<th>b2</th>
<th>b1</th>
<th>b0</th>
<th>d6</th>
<th>d5</th>
<th>d4</th>
<th>d3</th>
<th>d2</th>
<th>d1</th>
<th>d0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram of 7 Segment LED Display
## 7 Segment LED Decoder Implementation

<table>
<thead>
<tr>
<th>b2</th>
<th>b1</th>
<th>b0</th>
<th>d6</th>
<th>d5</th>
<th>d4</th>
<th>d3</th>
<th>d2</th>
<th>d1</th>
<th>d0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<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>
<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>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</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>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

![LED Display Diagram]
Ballot Reading and Display

The 3410 optical scan vote counter reader machine
Building Blocks

binary encoder

binary decoder

Multiplexor

$2^N \downarrow$ $N \downarrow 2^N$

$N \downarrow$ $N \downarrow 2^N$

$N \downarrow 2^N$
Make sure you are
• Registered for class, can access CMS
• Have a Section you can go to
• Have project partner in same Lab Section

Lab1 and HW1 are out
• Both due in one week, next Monday, start early
• Work alone
• But, use your resources
  • Lab Section, Piazza.com, Office Hours, Homework Help Session,
  • Class notes, book, Sections, CSUGLab

Homework Help Session
• Wednesday and Friday from 3:30-5:30pm
• Location: 203 Thurston
Administrivia

Check online syllabus/schedule
• http://www.cs.cornell.edu/Courses/CS3410/2012sp/schedule.html
• Slides and Reading for lectures
• Office Hours
• Homework and Programming Assignments
• Prelims (in evenings):
  • Tuesday, February 28th
  • Thursday, March 29th
  • April 26th

Schedule is subject to change
Binary Addition

- Addition works the same way regardless of base
- Add the digits in each position
- Propagate the carry

\[
\begin{array}{c}
183 \\
+ 254 \\
\hline
437
\end{array}
\]

\[
\begin{array}{c}
001110 \\
+ 011100 \\
\hline
101100
\end{array}
\]
1-bit Adder

Half Adder
- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry
1-bit Adder with Carry

Full Adder
- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded
4-bit Adder

4-Bit Full Adder
- Adds two 4-bit numbers and carry in
- Computes 4-bit result and carry out
- Can be cascaded
4-bit Adder
4-bit Adder

- Adds two 4-bit numbers, along with carry-in
- Computes 4-bit result and carry out
Addition with negatives:
- pos + pos → add magnitudes, result positive
- neg + neg → add magnitudes, result negative
- pos + neg → subtract smaller magnitude, keep sign of bigger magnitude
First Attempt: Sign/Magnitude Representation

• First Attempt: Sign/Magnitude Representation
• 1 bit for sign (0=positive, 1=negative)
• N-1 bits for magnitude
Two’s Complement Representation

• Better: Two’s Complement Representation
• Leading 1’s for negative numbers
• To negate any number:
  – complement *all* the bits
  – then add 1
Two’s Complement

• Non-negatives

(usual):

• +0 = 0000
• +1 = 0001
• +2 = 0010
• +3 = 0011
• +4 = 0100
• +5 = 0101
• +6 = 0110
• +7 = 0111
• +8 = 1000

(two’s complement: flip then add 1):

• ~0 = 1111
• ~1 = 1110
• ~2 = 1101
• ~3 = 1100
• ~4 = 1011
• ~5 = 1010
• ~6 = 0111
• ~7 = 0110

• -0 = 0000
• -1 = 1111
• -2 = 1110
• -3 = 1101
• -4 = 1100
• -5 = 1011
• -6 = 1010
• -7 = 1001
• -8 = 1000
Two’s Complement Facts

- Signed two’s complement
- Negative numbers have leading 1’s
- Zero is unique: +0 = -0
- Wraps from largest positive to largest negative
- N bits can be used to represent
  - Unsigned:
    - Eg: 8 bits ⇒
  - Signed (two’s complement):
    - Ex: 8 bits ⇒
Sign Extension & Truncation

• Extending to larger size

• Truncate to smaller size
Two’s Complement Addition

- Addition with two’s complement signed numbers
- Perform addition as usual, regardless of sign (it just works)
Diversion: 10’s Complement

• How does that work?

\[
\begin{align*}
-154 & \\
+283 & \\
\hline
129 & 
\end{align*}
\]
Overflow

• Overflow
  • adding a negative and a positive?
  • adding two positives?
  • adding two negatives?

• Rule of thumb:
  • Overflow happened iff carry into msb != carry out of msb
Two’s Complement Adder

- Two’s Complement Adder with overflow detection
Binary Subtraction

• Two’s Complement Subtraction
• Lazy approach  
  \[
  A - B = A + (-B) = A + (B + 1)
  \]

Q: What if (-B) overflows?
A Calculator

A — 8

B

S

0 = add
1 = sub

decoder
A Calculator

0=add
1=sub
Efficiency and Generality

- Is this design fast enough?
- Can we generalize to 32 bits? 64? more?
Performance

• Speed of a circuit is affected by the number of gates in series (on the critical path or the deepest level of logic)
4-bit Ripple Carry Adder

- First full adder, 2 gate delay
- Second full adder, 2 gate delay
- ...

Carry ripples from lsb to msb
Summary

• We can now implement any combinational (combinatorial) logic circuit
  • Decompose large circuit into manageable blocks
    – Encoders, Decoders, Multiplexors, Adders, ...
  • Design each block
    – Binary encoded numbers for compactness
• Can implement circuits using NAND or NOR gates
• Can implement gates using use P- and N-transistors
• And can add and subtract numbers (in two’s compliment)!
• Next time, state and finite state machines...