From xkcd
New Improved Office Hours

• Mondays 3-4 + Mondays 5-6
• Fridays 3-4 + Fridays 5-6

This week, Friday office hours will end at 5:30, but 6 PM will be the new normal afterwards!
Today’s true story
What do you get when you Google “CS 5 Black”?
That’s wheely funny!
Last time...

We can represent data in binary...

- Integers
- Characters
- Images
- Negative numbers?
- Floating point numbers?
Losing the most significant bit!

Most computers use 8, 16, or more bits to represent a number, but the Ran-O-Matic uses just 3 bits (just to keep the examples nice :^)
Negative numbers!
(Two’s Complement)

000 = 0
001 = 1
010 = 2
011 = 3
100 = 4
101 = 5
110 = 6
111 = 7

We like for $x + -x$ to be 0!
How should -1 be represented to make that happen?
Negative numbers!

000 = 0
001 = 1
010 = 2
011 = 3
100 = 4
101 = 5
110 = 6
111 = 7

000 = 0
001 = 1
010 = 2
011 = 3
______ = -1
______ = -2
______ = -3

What’s weird here?
Negative numbers!

000 = 0
001 = 1
010 = 2
011 = 3
100 = 4
101 = 5
110 = 6
111 = 7

000 = 0
001 = 1
010 = 2
011 = 3
101 = -3
110 = -2
111 = -1

We haven’t used 100. What should that correspond to?
What’s up with this!?

```python
>>> .01*10 == .01/.1
False
```
sinking with floats

Imagine a computer that uses only 4 bits to represent decimals...

In reality, 23 bits or 52 bits will be used to represent the fractional part of a floating-point number

lots of gaps in here...

>>> x = 0.1
The “Power” of Shifting!

“Left Shifting”

\[
\begin{array}{cccc}
10^3 & 10^2 & 10^1 & 10^0 \\
\hline
4 & 2 & & \\
\end{array}
\]

\[
\begin{array}{cccc}
2^3 & 2^2 & 2^1 & 2^0 \\
\hline
1 & 1 & & \\
\end{array}
\]

\[103 \times 10^2 = 3_{10}\]

“Right Shifting”

\[
\begin{array}{cccc}
10^3 & 10^2 & 10^1 & 10^0 \\
\hline
4 & 5 & 7 & \\
\end{array}
\]

\[
\begin{array}{cccc}
2^3 & 2^2 & 2^1 & 2^0 \\
\hline
1 & 0 & 1 & \\
\end{array}
\]

\[2^3 \times 10^2 = 5_{10}\]
Base conversion is similar to `makePoly`

```python
>>> p = makePoly([3, 2, 1])
>>> p
<function makePoly.<locals>.<lambda> at 0x101375d90>

def makePoly1(coeffList):
    if coeffList == []: return lambda X: 0
    else:
        return lambda X: X**(len(coeffList)-1)*coeffList[0]
        +
```

```python
def makePoly2(coeffList):
    if coeffList == []: return lambda X: 0
    else:
        return lambda X: X*makePoly2(coeffList[:-1])(X)
        + coeffList[-1]
```

In your notes
Base conversion is similar to makePoly

```python
>>> p = makePoly([3, 2, 1])
>>> p
<function makePoly.<locals>.<lambda> at 0x101375d90>

def makePoly1(coeffList):
    if coeffList == []: return lambda X: 0
    else:
        return lambda X: X**(len(coeffList)-1)*coeffList[0] + makePoly1(coeffList[1:])(X)

def makePoly2(coeffList):
    if coeffList == []: return lambda X: 0
    else:
        return lambda X: X*makePoly2(coeffList[:-1])(X) + coeffList[-1]
```
Secret Sharing Revisited…

Last time one-time pads used to share a secret between two people so that…

- Either person has no ability to retrieve the secret
- The people *together* can reconstruct the secret

A “better” approach: Shamir’s Algorithm using `makePoly`

Better in the sense that we can share the secret with more than two people!
For your notes
Addition

Base 10 Addition

\[
\begin{array}{ccc}
10^2 & 10^1 & 10^0 \\
\hline
1 & 4 & 3 \\
+ & 8 & 9 \\
\hline
13 & 2 \\
\end{array}
\]

Try it in base 2!

Base 2 Addition

\[
\begin{array}{ccc}
2^2 & 2^1 & 2^0 \\
\hline
1 & 0 & 1 \\
+ & 0 & 1 \\
\hline
0 & 1 & 1 \\
\end{array}
\]
Digital Logic Gates

<table>
<thead>
<tr>
<th>$x$</th>
<th>NOT $x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$x$ AND $y$</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>

Also written $x \cdot y$

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$x$ OR $y$</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>1</td>
</tr>
</tbody>
</table>

Also written $x + y$
From Description to Circuit!

**Abstract**

$f$ is a function of TWO binary (Boolean) variables s.t. the output is 1 if and only if exactly one of the two inputs is 1.

**Concrete**

### Table

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$f(x,y)$</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>

### Formula

$\bar{x}y + xy$

### Circuit

Circuits made of concrete?
Try building circuits for these…

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>“both 0”</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</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>0</td>
</tr>
</tbody>
</table>

Circuit 1  
(In class)

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>“equal”</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</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>

Circuit 2  
(worksheet)

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>“odd # 1’s”</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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Circuit 3  
(Yikes!)
From Description to Circuit: The Minterm Expansion Principle

Words

\( f \) is a function of TWO binary (Boolean) variables s.t. the output is 1 if and only if exactly one of the two inputs is 1

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>XOR</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>

Formula

\( \overline{xy} + xy \)

Circuit
Let’s do this one using the Minterm Expansion Principle

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>“odd # 1’s”</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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</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>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</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>1</td>
</tr>
</tbody>
</table>
Let’s do this one using the Minterm Expansion Principle

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>“odd # 1’s”</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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table $\rightarrow$ Formula $\rightarrow$ Circuit
A Circuit for Adding!

Base 2 Addition

\[
\begin{array}{cccc}
0 & 1 & 0 & 1 \\
+ & 1 & 0 & 0 & 1 \\
\hline
1 & 0 & 0 & 1 \\
\end{array}
\]
A Circuit for Adding!

Base 2 Addition

\[
\begin{array}{cccc}
0 & 1 & 0 & 0 \\
+ & 1 & 0 & 0 \\
\hline
1 & 1 & 0 & 0
\end{array}
\]
A Circuit for Adding!

Base 2 Addition

Cool, but how do we build a FA?
A Circuit for Adding!

Base 2 Addition

\[
\begin{array}{cccc}
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
FA & FA & FA & FA \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
x & y & carry_{in} & sum & carry_{out} \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 \\
\end{array}
\]

Fill this in...

Then, let’s build the circuit!
A Circuit for Adding!

Base 2 Addition

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$\text{carry}_{in}$</th>
<th>$\text{sum}$</th>
<th>$\text{carry}_{out}$</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>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</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>0</td>
<td>1</td>
<td>0</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>0</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>

Now what?
A Circuit for Adding!

Base 2 Addition

\[
\begin{array}{cccc}
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 \\
\hline
+ & & & \\
\hline
FA & FA & FA & FA \\
\end{array}
\]

\[
\begin{array}{cccc}
x & y & \text{carry}_{in} \\
0 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
1 & 1 & 1 \\
\hline
\text{sum} & & & \\
0 & 0 & 0 \\
1 & 1 & 0 \\
1 & 0 & 1 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
0 & 1 & 1 \\
0 & 0 & 1 \\
1 & 1 & 1 \\
\hline
\text{carry}_{out} & & & \\
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 1 \\
0 & 1 & 1 \\
0 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\]
A Circuit for Adding!

Base 2 Addition

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>carry\text{in}</th>
<th>sum</th>
<th>carry\text{out}</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>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</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>0</td>
<td>1</td>
<td>0</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>0</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>

+ 0 1 0 0 1
FA FA FA FA

$x$ $y$ carry\text{in}

$x$ $y$ carry\text{in}

$x$ $y$ carry\text{in}

$0$ $0$ $0$ $0$

$0$ $0$ $1$ $0$

$0$ $1$ $0$ $0$

$0$ $1$ $1$ $1$

$1$ $0$ $0$ $0$

$1$ $0$ $1$ $0$

$1$ $1$ $0$ $1$

$1$ $1$ $1$ $1$
Does the Minterm Expansion Algorithm produce the smallest circuits possible?