Overview

Basics

Addition

Expanded gates
Logic and circuits

Why and/or/not?
Logic and circuits

Why and/or/not?

▶ Universal.
Logic and circuits

Why and/or/not?

- Universal.
- Already central to the study of logic.
Logic and circuits

Why and/or/not?

- Universal.
- Already central to the study of logic.
- Can be implemented in physical hardware.
Logic and circuits

Why and/or/not?

- Universal.
- Already central to the study of logic.
- Can be implemented in physical hardware.

A transistor is roughly a voltage controlled switch and can be used to construct and/or/not.
Logic and circuits

Why and/or/not?

▶ Universal.
▶ Already central to the study of logic.
▶ Can be implemented in physical hardware.

A transistor is roughly a voltage controlled switch and can be used to construct and/or/not. Voltage high: True.
Logic and circuits

Why and/or/not?

- Universal.
- Already central to the study of logic.
- Can be implemented in physical hardware.

A transistor is roughly a voltage controlled switch and can be used to construct and/or/not. Voltage high: True. Voltage low: False.
Logic and circuits

Why and/or/not?

- Universal.
- Already central to the study of logic.
- Can be implemented in physical hardware.

A transistor is roughly a voltage controlled switch and can be used to construct and/or/not. Voltage high: True. Voltage low: False.

To a first approximation, digital circuits are giant logic expressions.
Logic and circuits

Why and/or/not?

- Universal.
- Already central to the study of logic.
- Can be implemented in physical hardware.

A transistor is roughly a voltage controlled switch and can be used to construct and/or/not. Voltage high: True. Voltage low: False.

To a first approximation, digital circuits are giant logic expressions. A computer is a giant logical expression whose outputs at one time step become the inputs at the next time step.
Basic Circuit Components

And gates drawn as rectangles with a semicircular cap.
Basic Circuit Components

And gates drawn as rectangles with a semicircular cap. Or gates similar, but concave rounded bottom and pointy top.
Basic Circuit Components

And gates drawn as rectangles with a semicircular cap. Or gates similar, but concave rounded bottom and pointy top. Not gates are triangles with a circle on the point.
Basic Circuit Components

And gates drawn as rectangles with a semicircular cap. Or gates similar, but concave rounded bottom and pointy top. Not gates are triangles with a circle on the point.
Basic Circuit Components

And gates drawn as rectangles with a semicircular cap. Or gates similar, but concave rounded bottom and pointy top. Not gates are triangles with a circle on the point.

How many and/or/nots?
Circuit Flow

Wires show where values propagate to.
Circuit Flow

Wires show where values propagate to. We can label each wire with an expression relating the inputs of the circuit to the value propagating on that wire.

This circuit outputs \((p \text{ AND } q) \text{ OR } (\text{NOT}(p) \text{ AND } \text{NOT}(q))\).
Circuit Flow

Wires show where values propagate to. We can label each wire with an expression relating the inputs of the circuit to the value propagating on that wire.

This circuit outputs \((p \text{ AND } q) \text{ OR } (\text{NOT}(p) \text{ AND } \text{NOT}(q))\). We’ll call this circuit “equal2” since it is testing the equality of 2 truth values or two bits.
IfThenElse1

Here’s another useful circuit. It outputs either input $t$ or input $e$ depending on the value of condition $c$. 
IfThenElse1

Here’s another useful circuit. It outputs either input $t$ or input $e$ depending on the value of condition $c$. 
Nested circuits

We can use IfThenElse1 to make another circuit.
Nested circuits

We can use IfThenElse1 to make another circuit. What is it?
Nested circuits

We can use IfThenElse1 to make another circuit. What is it?
Nested circuits

We can use IfThenElse1 to make another circuit. What is it?
Nested circuits 2

Here’s another variant of IfThenElse1.
Nested circuits 2

Here’s another variant of IfThenElse1. What is it?
Nested circuits 2

Here’s another variant of IfThenElse1. What is it?

\[
p \quad c \quad \text{IfThenElse1} \quad t \quad e \quad \text{out} \quad \text{out}
\]
Nested circuits 2

Here’s another variant of IfThenElse1. What is it?

\[
\begin{array}{c}
p \quad c \quad \text{IfThenElse1} \quad \text{out} \\
q \quad t \quad e \quad \text{out} \\
\end{array}
\]

\[
\begin{array}{c}
p \quad c \quad \text{IfThenElse1} \quad \text{out} \\
q \quad t \quad e \quad \text{out} \\
\end{array}
\]

\[
\begin{array}{c}
p \quad c \quad \text{IfThenElse1} \quad \text{out} \\
q \quad t \quad e \quad \text{out} \\
\end{array}
\]

\[
\begin{array}{c}
p \quad c \quad \text{IfThenElse1} \quad \text{out} \\
q \quad t \quad e \quad \text{out} \\
\end{array}
\]

\[
\begin{array}{c}
p \quad c \quad \text{IfThenElse1} \quad \text{out} \\
q \quad t \quad e \quad \text{out} \\
\end{array}
\]

\[
\begin{array}{c}
p \quad c \quad \text{IfThenElse1} \quad \text{out} \\
q \quad t \quad e \quad \text{out} \\
\end{array}
\]
Truth and numbers

We can think of $T$ as being 1 and $F$ as being 0 and create circuits that compute on numbers.
Truth and numbers

We can think of $\text{T}$ as being 1 and $\text{F}$ as being 0 and create circuits that compute on numbers. Let’s add two bits.
Truth and numbers

We can think of T as being 1 and F as being 0 and create circuits that compute on numbers. Let’s add two bits.

\[
\begin{array}{c|cc}
\ a \ & b & c & s \\
0 & 0 & 0 & a + b & a + b \\
\end{array}
\]
Truth and numbers

We can think of $\text{T}$ as being 1 and $\text{F}$ as being 0 and create circuits that compute on numbers. Let’s add two bits.

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$a+b$</th>
<th>$a+b$</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>
</tbody>
</table>

\[c = a \text{ AND } b, \quad s = a \text{ XOR } b.\]
Truth and numbers

We can think of \( T \) as being 1 and \( F \) as being 0 and create circuits that compute on numbers. Let’s add two bits.

<table>
<thead>
<tr>
<th></th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( s )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( a + b )</td>
<td>( a + b )</td>
<td></td>
<td></td>
</tr>
<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>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Truth and numbers

We can think of $T$ as being 1 and $F$ as being 0 and create circuits that compute on numbers. Let’s add two bits.

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$a+b$</th>
<th>$a+b$</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>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Truth and numbers

We can think of $T$ as being 1 and $F$ as being 0 and create circuits that compute on numbers. Let’s add two bits.

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$s$</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>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Truth and numbers

We can think of $\text{T}$ as being 1 and $\text{F}$ as being 0 and create circuits that compute on numbers. Let’s add two bits.

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

$c =$
Truth and numbers

We can think of \( T \) as being 1 and \( F \) as being 0 and create circuits that compute on numbers. Let’s add two bits.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>( c )</th>
<th>( s )</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>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\( c = a \ AND \ b \).
Truth and numbers

We can think of $T$ as being 1 and $F$ as being 0 and create circuits that compute on numbers. Let’s add two bits.

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

$c = a \text{ AND } b$. $s =$
Truth and numbers

We can think of \( T \) as being 1 and \( F \) as being 0 and create circuits that compute on numbers. Let’s add two bits.

\[
\begin{array}{c|c|c|c}
 a & b & c & s \\
\hline
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 \\
\end{array}
\]

\( c = a \text{ AND } b. \ s = a \text{ XOR } b. \)
Half adder

Here’s a circuit for adding two bits.
Half adder

Here’s a circuit for adding two bits.

```
HalfAdder1
```

```
a
b

p XOR2
q

out

s

c
```
Adding two two-bit numbers

We can use the classic additional algorithm, modified to base 2.
Adding two two-bit numbers

We can use the classic additional algorithm, modified to base 2. Let’s compute $11 + 01$ (aka $3 + 1$ in binary).
Adding two two-bit numbers

We can use the classic additional algorithm, modified to base 2. Let’s compute $11 + 01$ (aka $3 + 1$ in binary).

\[
\begin{array}{c}
1 & 1 \\
+ & 0 & 1 \\
\hline
1 & 1 & 0
\end{array}
\]
Adding two two-bit numbers

We can use the classic additional algorithm, modified to base 2. Let’s compute $11 + 01$ (aka $3 + 1$ in binary).

\[
\begin{array}{c}
1 & 1 \\
\hline \\
+ & 0 & 1 \\
\hline \\
1 & 1 & 1 \\
\hline \\
+ & 0 & 1 \\
\hline \\
0 & 0 & 0 \\
\end{array}
\]
Adding two two-bit numbers

We can use the classic additional algorithm, modified to base 2. Let’s compute $11 + 01$ (aka $3 + 1$ in binary).

\[
\begin{array}{c}
1 \\
+ 0 \\
\hline
1 \\
\end{array}
\]

\[
\begin{array}{c}
1 \\
1 \\
+ 0 \\
\hline
0 \\
\end{array}
\]

\[
\begin{array}{c}
1 \\
1 \\
+ 0 \\
\hline
1 \\
\end{array}
\]

\[
\begin{array}{c}
1 \\
+ 0 \\
\hline
1 \\
\end{array}
\]

\[
\begin{array}{c}
1 \\
\hline
100
\end{array}
\]
Full adder

To handle this computation, we need to add *three* 1-bit numbers, not just two: $a + b + c$. 
Full adder

To handle this computation, we need to add *three* 1-bit numbers, not just two: $a + b + c$. Can add the first two bits to get a two-bit result.
Full adder

To handle this computation, we need to add three 1-bit numbers, not just two: \( a + b + c \). Can add the first two bits to get a two-bit result.

Now, we need to add a two-bit number and a one bit number.
Partial full adder

We have $c_1 s_1 + c$. 
Partial full adder

We have \( c_1 s_1 + c \). Not quite clear what to do. Could add the low order bit \( s_1 \) to \( c \)…
Partial full adder

We have $c_1 s_1 + c$. Not quite clear what to do. Could add the low order bit $s_1$ to $c$...

The low order bit of the sum $s_2$ is correct now. But, we have two carries, $c_1$ and $c_2$, each worth two...
Towards the full full adder

Need to add the two carries $c_1$ and $c_2$. They are in the two's place.
Towards the full full adder

Need to add the two carries $c_1$ and $c_2$. They are in the two’s place. Could throw on another HalfAdder1.
Towards the full full adder

Need to add the two carries $c_1$ and $c_2$. They are in the two’s place. Could throw on another HalfAdder1. But:

- If $a$ and $b$ are both 1, their sum is 10. Thus, the carry $c_1$ is a one, but the sum $s_1$ is a zero. Adding in $c$ won’t cause a second carry in $c_2$. 
Towards the full full adder

Need to add the two carries $c_1$ and $c_2$. They are in the two’s place. Could throw on another HalfAdder1. But:

- If $a$ and $b$ are both 1, their sum is 10. Thus, the carry $c_1$ is a one, but the sum $s_1$ is a zero. Adding in $c$ won’t cause a second carry in $c_2$.
- On the other hand, if at least one of $a$ and $b$ are less than 1, the bottom carry $c_2$ might be 1, but the top one $c_2$ is zero.
Truth table for adding the carries

<table>
<thead>
<tr>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$c_1 + c_2$</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>—</td>
</tr>
</tbody>
</table>
Truth table for adding the carries

What's the simplest gate that matches this truth table?
Truth table for adding the carries

<table>
<thead>
<tr>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$c_1 + c_2$</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>—</td>
</tr>
</tbody>
</table>

What’s the simplest gate that matches this truth table? Or!
Full full adder

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```

```
 a  a  c
 b  b  s
 c  s
```
5-bit-number adder

Now, we can do the fundamental operation for adding multi-bit values.
5-bit-number adder

Now, we can do the fundamental operation for adding multi-bit values. Can daisy chain them to add multi-bit numbers.
5-bit-number adder

Now, we can do the fundamental operation for adding multi-bit values. Can daisy chain them to add multi-bit numbers.
Deep circuits

Note that are circuits are getting kind of deep now.
Deep circuits

Note that circuits are getting kind of deep now. How deep?
Deep circuits

Note that are circuits are getting kind of deep now. How deep?

- Add5 is 1 HalfAdder1 plus 4 FullAdder1 deep.
Deep circuits

Note that are circuits are getting kind of deep now. How deep?

- Add5 is 1 HalfAdder1 plus 4 FullAdder1 deep.
- FullAdder1 is 2 HalfAdder1 plus 1 OR deep.
Deep circuits

Note that are circuits are getting kind of deep now. How deep?

- Add5 is 1 HalfAdder1 plus 4 FullAdder1 deep.
- FullAdder1 is 2 HalfAdder1 plus 1 OR deep.
- HalfAdder1 is 1 XOR2 deep (and 1 AND in parallel).
Deep circuits

Note that are circuits are getting kind of deep now. How deep?

- Add5 is 1 HalfAdder1 plus 4 FullAdder1 deep.
- FullAdder1 is 2 HalfAdder1 plus 1 OR deep.
- HalfAdder1 is 1 XOR2 deep (and 1 AND in parallel).
- XOR2 is 1 IfThenElse1 plus 2 NOT deep.
Deep circuits

Note that circuits are getting kind of deep now. How deep?

- Add5 is 1 HalfAdder1 plus 4 FullAdder1 deep.
- FullAdder1 is 2 HalfAdder1 plus 1 OR deep.
- HalfAdder1 is 1 XOR2 deep (and 1 AND in parallel).
- XOR2 is 1 IfThenElse1 plus 2 NOT deep.
- IfThenElse1 is 1 NOT, 1 AND, and 1 OR deep.
Deep circuits

Note that are circuits are getting kind of deep now. How deep?

- Add5 is 1 HalfAdder1 plus 4 FullAdder1 deep.
- FullAdder1 is 2 HalfAdder1 plus 1 OR deep.
- HalfAdder1 is 1 XOR2 deep (and 1 AND in parallel).
- XOR2 is 1 IfThenElse1 plus 2 NOT deep.
- IfThenElse1 is 1 NOT, 1 AND, and 1 OR deep.

That’s 49 gates deep, if you backsolve.
Deep circuits

Note that are circuits are getting kind of deep now.

How deep?

- Add5 is 1 HalfAdder1 plus 4 FullAdder1 deep.
- FullAdder1 is 2 HalfAdder1 plus 1 OR deep.
- HalfAdder1 is 1 XOR2 deep (and 1 AND in parallel).
- XOR2 is 1 IfThenElse1 plus 2 NOT deep.
- IfThenElse1 is 1 NOT, 1 AND, and 1 OR deep.

That’s 49 gates deep, if you backsolve.

Time to compute grows with the depth of the circuit.
Deep circuits

Note that are circuits are getting kind of deep now. How deep?

- Add5 is 1 HalfAdder1 plus 4 FullAdder1 deep.
- FullAdder1 is 2 HalfAdder1 plus 1 OR deep.
- HalfAdder1 is 1 XOR2 deep (and 1 AND in parallel).
- XOR2 is 1 IfThenElse1 plus 2 NOT deep.
- IfThenElse1 is 1 NOT, 1 AND, and 1 OR deep.

That’s 49 gates deep, if you backsolve.

Time to compute grows with the depth of the circuit. Fortunately, there are better designs for adding.
Negative numbers

We’ve defined addition in terms of logic gates.
Negative numbers

We’ve defined addition in terms of logic gates. How about subtraction?
Negative numbers

We’ve defined addition in terms of logic gates. How about subtraction? We can compute subtraction via adding a negative number.
Negative numbers

We’ve defined addition in terms of logic gates. How about subtraction? We can compute subtraction via adding a negative number. What’s a negative number in bits...?
Negative numbers

We’ve defined addition in terms of logic gates. How about subtraction? We can compute subtraction via adding a negative number. What’s a negative number in bits...?

$-a$ is a value such that $a + (-a) = 0$. 
Negative numbers

We’ve defined addition in terms of logic gates. How about subtraction? We can compute subtraction via adding a negative number. What’s a negative number in bits...?

\(-a\) is a value such that \(a + (-a) = 0\).

In computers, when we add two 5-bit numbers, we generally store a 5-bit answer, which means throwing away the carry.
Negative numbers

We’ve defined addition in terms of logic gates. How about subtraction? We can compute subtraction via adding a negative number. What’s a negative number in bits...?

$-a$ is a value such that $a + (-a) = 0$.

In computers, when we add two 5-bit numbers, we generally store a 5-bit answer, which means throwing away the carry. Thus, addition is actually being done mod
Negative numbers

We’ve defined addition in terms of logic gates. How about subtraction? We can compute subtraction via adding a negative number. What’s a negative number in bits...?

$-a$ is a value such that $a + (-a) = 0$.

In computers, when we add two 5-bit numbers, we generally store a 5-bit answer, which means throwing away the carry. Thus, addition is actually being done mod 32.
Negative numbers

We’ve defined addition in terms of logic gates. How about subtraction? We can compute subtraction via adding a negative number. What’s a negative number in bits...?

$-a$ is a value such that $a + (-a) = 0$.

In computers, when we add two 5-bit numbers, we generally store a 5-bit answer, which means throwing away the carry. Thus, addition is actually being done mod 32. The additive inverse of $a$ mod 32 is 32 if $a = 0$, otherwise
Negative numbers

We’ve defined addition in terms of logic gates. How about subtraction? We can compute subtraction via adding a negative number. What’s a negative number in bits...?

$-a$ is a value such that $a + (-a) = 0$.

In computers, when we add two 5-bit numbers, we generally store a 5-bit answer, which means throwing away the carry. Thus, addition is actually being done mod 32. The additive inverse of $a$ mod 32 is 32 if $a = 0$, otherwise $32 - a$. 
Negative numbers

We’ve defined addition in terms of logic gates. How about subtraction? We can compute subtraction via adding a negative number. What’s a negative number in bits...

\(-a\) is a value such that \(a + (-a) = 0\).

In computers, when we add two 5-bit numbers, we generally store a 5-bit answer, which means throwing away the carry. Thus, addition is actually being done mod 32. The additive inverse of \(a\) mod 32 is 32 if \(a = 0\), otherwise \(32 - a\).

Example: \(-5 \equiv \)
Negative numbers

We’ve defined addition in terms of logic gates. How about subtraction? We can compute subtraction via adding a negative number. What’s a negative number in bits...?

$-a$ is a value such that $a + (-a) = 0$.

In computers, when we add two 5-bit numbers, we generally store a 5-bit answer, which means throwing away the carry. Thus, addition is actually being done mod 32. The additive inverse of $a$ mod 32 is 32 if $a = 0$, otherwise $32 - a$.

Example: $-5 \equiv 27 \pmod{32}$. 
Negative numbers

We’ve defined addition in terms of logic gates. How about subtraction? We can compute subtraction via adding a negative number. What’s a negative number in bits...?

$-a$ is a value such that $a + (-a) = 0$.

In computers, when we add two 5-bit numbers, we generally store a 5-bit answer, which means throwing away the carry. Thus, addition is actually being done mod 32. The additive inverse of $a$ mod 32 is 32 if $a = 0$, otherwise $32 - a$.

Example: $-5 \equiv 27 \pmod{32}$.

In binary: 00101 $\rightarrow$ 11011.
Negative numbers

We’ve defined addition in terms of logic gates. How about subtraction? We can compute subtraction via adding a negative number. What’s a negative number in bits...?

$-a$ is a value such that $a + (-a) = 0$.

In computers, when we add two 5-bit numbers, we generally store a 5-bit answer, which means throwing away the carry. Thus, addition is actually being done mod 32. The additive inverse of $a$ mod 32 is 32 if $a = 0$, otherwise $32 - a$.

Example: $-5 \equiv 27 \pmod{32}$.

In binary: $00101 \rightarrow 11011$. 
Bit flips

Define $\overline{x}$ to be the number $x$ but with all its bits flipped.
Bit flips

Define $\overline{x}$ to be the number $x$ but with all its bits flipped.

Note: $x + \overline{x} = 2^5 - 1$ since the sum has every bit turned on.
Bit flips

Define $\overline{x}$ to be the number $x$ but with all its bits flipped.

Note: $x + \overline{x} = 2^5 - 1$ since the sum has every bit turned on. Thus, $\overline{x} = 2^5 - 1 - x$. 
Bit flips

Define $\overline{x}$ to be the number $x$ but with all its bits flipped.

Note: $x + \overline{x} = 2^5 - 1$ since the sum has every bit turned on. Thus, $\overline{x} = 2^5 - 1 - x$. If we add one, it's $32 - x$, the additive inverse.
Negation by two’s complement

Intro to Circuits
Addition
IfThenElse5

Now that we add and subtract 5-bit numbers, we might want to perform other operations as well.
IfThenElse5

Now that we add and subtract 5-bit numbers, we might want to perform other operations as well.
IfThenElse5

Now that we add and subtract 5-bit numbers, we might want to perform other operations as well.

Depending on condition bit $c$, produces either 5-bit $t$ or 5-bit $e$. 
Equal5

Takes two 5-bit values \( a_4a_3a_2a_1a_0 \) and \( b_4b_3b_2b_1b_0 \). Returns \( T \) if and only if \( a_i = b_i \) for all \( i \).
Equal5

Takes two 5-bit values $a_5a_4a_3a_2a_1a_0$ and $b_5b_4b_3b_2b_1b_0$. 
Equal5

Takes two 5-bit values $a_5 a_4 a_3 a_2 a_1 a_0$ and $b_5 b_4 b_3 b_2 b_1 b_0$. Returns $T$ if and only if $a_i == b_i$ for all $i$. 