Today

- R-C Circuits cont
- Diodes
- Representing information

There are only 10 types of people in the world: Those who understand binary and those who don't.
Representing Information Using Voltage
Representing Information Using Voltage

Analog representation: the precise voltage matters.

• Suppose we observed voltage $v$ on a wire (e.g., an output from an accelerometer)
• The encoded quantity is some function of that voltage:

$$acceleration = f(v)$$
Representing Information Using Voltage

The simplest form assumes a linear relationship:

\[ \text{acceleration} = \alpha v + \beta \]
Representing Information Using Voltage

• Digital representation: the value to be represented is binary – i.e., true or false

• For example, a bit $b$ is:

$$ b = \begin{cases} 
\text{true} & v > 2.5 \text{ Volts} \\
\text{false} & \text{otherwise} 
\end{cases} $$
Representing Information Using Voltage

We typically use the shorthand:

\[ 0 = \text{false} \]
\[ 1 = \text{true} \]
Computing In Binary
(i.e., Logic)
What is the Gate?

- Logical Symbol:

- Algebraic Notation:

- Truth Table:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
The NOT Gate

• Logical Symbol: 

• Algebraic Notation: \( B = \overline{A} \)

• Truth Table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
And This Gate?

• Logical Symbol:

• Algebraic Notation: \( C = ? \)

• Truth Table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
The “AND” Gate

- Logical Symbol:

  ![AND Gate Diagram]

- Algebraic Notation: \( C = A \cdot B = AB \)

- Truth Table:

<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>
And This Gate?

- **Logical Symbol:**

- **Algebraic Notation:** $C = ?$

- **Truth Table:**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
The “OR” Gate

- Logical Symbol:

- Algebraic Notation: \( C = A + B \)

- Truth Table:

<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>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>

Andrew H. Fagg: Embedded Real-Time Systems: Logic
Exclusive OR ("XOR") Gates

• Logical Symbol:

• Algebraic Notation:  \( C = A \oplus B \)

• Truth Table:

<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>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>
Exclusive OR ("XOR") Gates

• Logical Symbol:

A

\[ \rightarrow \]

C

B

• Algebraic Notation: \( C = A \oplus B \)

<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>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>
2-Input Multiplexer

A multiplexer is a device that selects between two input lines

- A & B are the inputs
- S is the selection signal (also an input)
- C is a copy of A if S=0
- C is a copy of B if S=1
2-Input Multiplexer

A multiplexer is a device that selects between two input lines
- A & B are the inputs
- S is the selection signal (also an input)
- C is a copy of A if S=0
- C is a copy of B if S=1
A multiplexer is a device that selects between two input lines

- A & B are the inputs
- S is the selection signal (also an input)
- C is a copy of A if S=0
- C is a copy of B if S=1
N-Input Multiplexer

Suppose we want to select from between $N$ different inputs.

- This requires more than one select line. How many?
N-Input Multiplexer

How many select lines?

- $M = \log_2 N$
- $N = 2^M$

What would the $N=8$ implementation look like?
Back to Binary…

With a binary digit, we can only represent two different values…
Back to Binary…

With a binary digit, we can only represent two different values…

- As in the decimal number system, we introduce multiple digits…
Binary Encoding

How do we convert from binary to decimal in general?

<table>
<thead>
<tr>
<th>B2</th>
<th>B1</th>
<th>B0</th>
<th>decimal</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>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>
Binary to Decimal Conversion

\[ \text{value} = B_0 + B_1 \cdot 2^1 + B_2 \cdot 2^2 + B_3 \cdot 2^3 + \ldots \]

\[ \text{value} = \sum_{i=0}^{N-1} B_i \cdot 2^i \]

How do we convert from decimal to binary?
Decimal to Binary Conversion

```c
int value;

For each i: B[i] = 0

for(i = 0; value > 0; ++i) {
    B[i] = remainder of: value/2;
    value = value/2;
}
```
• Circuit example...
Binary Addition

Consider the following binary numbers:

0 0 1 0 0 1 1 0
0 0 1 0 1 0 1 1

How do we add these numbers?
Binary Addition

0 0 1 0 0 1 1 0
0 0 1 0 1 0 1 1

1
Binary Addition

\[ \begin{array}{cccccccc}
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
\hline
0 & 1 & 0 & 1 & 1 & 0 \\
\end{array} \]

And we have a carry now!
Binary Addition

0 0 1 0 0 1 1 0
0 0 1 0 1 0 1 1

And we have a carry again!
Binary Addition

0 0 1 0 0 1 1 0
0 0 1 0 1 0 1 1

0 0 0 1

and again!
Binary Addition

0 0 1 0 0 1 1 0
0 0 1 0 1 0 1 1
↓
1 0 0 0 1
Binary Addition

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

\[010001\]

One more carry!
Binary Addition

\[
\begin{array}{cccccccc}
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
\end{array}
\]
Binary Addition

Behaves just like addition in decimal, but:

• We carry to the next digit any time the sum of the digits is 2 (decimal) or greater
Negative Numbers

So far we have only talked about representing non-negative integers

• What can we add to our binary representation that will allow this?
Representing Negative Numbers

One possibility:

• Add an extra bit that indicates the sign of the number
• We call this the “sign-magnitude” representation
Sign Magnitude Representation

+12  0  0  0  0  1  1  0  0
Sign Magnitude Representation

+12: 0 0 0 0 1 1 0 0

-12: 1 0 0 0 1 1 0 0
Sign Magnitude Representation

+12  0 0 0 0 1 1 0 0

-12  1 0 0 0 1 1 0 0

What is the problem with this approach?
Sign Magnitude Representation

What is the problem with this approach?

• Some of the arithmetic operators that we have already developed do not do the right thing
Sign Magnitude Representation

Operator problems:
• For example, we have already discussed a counter (that implements an ‘increment’ operation)

\[
\begin{align*}
-12 & \quad 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0
\end{align*}
\]
Sign Magnitude Representation

Operator problems:

-12 1 0 0 0 1 1 0 0

Increment
Sign Magnitude Representation

Operator problems:

-12

\[\begin{array}{c}
1
\end{array}\]

Increment

\[\begin{array}{c}
1
\end{array}\]

1 0 0 0 1 1 0 0

1 0 0 0 1 1 0 1
Sign Magnitude Representation

Operator problems:

-12  1 0 0 0 1 1 0 0

Increment

-13  1 0 0 0 1 1 0 1

!!!!
Representing Negative Numbers

An alternative:

• When taking the additive inverse of a number, invert all of the individual bits
• The leftmost bit still determines the sign of the number
One’s Complement Representation

12

-12

Invert

0 0 0 0 1 1 0 0

1 1 1 1 0 0 1 1
One’s Complement Representation

12
-12

0 0 0 0 1 1 0 0

Invert

1 1 1 1 0 0 1 1

Increment

1 1 1 1 0 1 0 0

Andrew H. Fagg: Embedded Real-Time Systems: Logic 48
One’s Complement Representation

12 → 0 0 0 0 1 1 0 0 (Invert)

-12 → 1 1 1 1 0 0 1 1 (Invert)

-11 → 1 1 1 1 0 1 0 0 (Increment)
One’s Complement Representation

What problems still exist?
One’s Complement Representation

What problems still exist?
• We have two distinct representations of ‘zero’:

  0 0 0 0 0 0 0 0

  1 1 1 1 1 1 1 1
One’s Complement Representation

What problems still exist?

• We can’t directly add a positive and a negative number:

\[
\begin{array}{c}
12 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
+ & + \\
-5 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0
\end{array}
\]
One’s Complement Representation

12
+
-5

0 0 0 0 1 1 0 0
+
1 1 1 1 1 0 1 0

0 0 0 0 0 1 1 0
One’s Complement Representation

12 0 0 0 0 1 1 0 0
+ +
-5 1 1 1 1 1 0 1 0
6 0 0 0 0 1 1 0

!!!!
Representing Negative Numbers

An alternative:
(a little intuition first)

0 0 0 0 0 0 0 0 0

Decrement
Representing Negative Numbers

An alternative:
(a little intuition first)

0 0 0 0 0 0 0 0

Decrement

1 1 1 1 1 1 1 1
Representing Negative Numbers

An alternative:
(a little intuition first)

Define this as

0 0 0 0 0 0 0 0 0
-1 1 1 1 1 1 1 1 1
Representing Negative Numbers

A few more numbers:

<table>
<thead>
<tr>
<th>Number</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0 0 0 0 0 0 1 1</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>-1</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>-2</td>
<td>1 1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>-3</td>
<td>1 1 1 1 1 1 0 1</td>
</tr>
</tbody>
</table>
Two’s Complement Representation

In general, how do we take the additive inverse of a binary number?
Two’s Complement Representation

In general, how do we take the additive inverse of a binary number?

• Invert each bit and then add ‘1’
Two’s Complement Representation

Invert each bit and then add ‘1’

\[
\begin{align*}
5 & \rightarrow 00000101 \\
-5 & \rightarrow 11111011
\end{align*}
\]
Two’s Complement Representation

Now: let’s try adding a positive and a negative number:

\[
\begin{array}{c}
12 \\
+ \\
-5
\end{array}
\quad\begin{array}{c}
00001100 \\
+ \\
11111011
\end{array}
\]
Two’s Complement Representation

Now: let’s try adding a positive and a negative number:

12 + 0 0 0 0 1 1 0 0
+ +
-5 1 1 1 1 1 0 1 1

0 0 0 0 0 1 1 1
Two’s Complement Representation

Now: let’s try adding a positive and a negative number:

\[
\begin{array}{c}
12 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
+ & + \\
-5 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
\hline
7 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
\end{array}
\]
Two’s Complement Representation

Two’s complement is used for integer representation in today’s processors.
Two’s Complement Representation

Two’s complement is used for integer representation in today’s processors.

One oddity: we can represent one more negative number than we can positive numbers.
Implementing Subtraction

How do we implement a ‘subtraction’ operator?
(e.g., A – B)
Implementing Subtraction

How do we implement a ‘subtraction’ operator? (e.g., A – B)

- Take the 2s complement of B
- Then add this number to A
Other Useful Number Systems

You already know:

• Decimal – base 10
• Binary – base 2
Other Useful Number Systems

You already know:
• Decimal – base 10
• Binary – base 2

But it is common to also see:
• Octal – base 8
• Hexadecimal – base 16
# Other Number Systems

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Octal</th>
<th>Hex</th>
<th>Decimal</th>
<th>Binary</th>
<th>Octal</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1000</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>1001</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>1010</td>
<td>12</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
<td>3</td>
<td>11</td>
<td>1011</td>
<td>13</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
<td>4</td>
<td>12</td>
<td>1100</td>
<td>14</td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
<td>5</td>
<td>13</td>
<td>1101</td>
<td>15</td>
<td>D</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
<td>6</td>
<td>14</td>
<td>1110</td>
<td>16</td>
<td>E</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
<td>7</td>
<td>15</td>
<td>1111</td>
<td>17</td>
<td>F</td>
</tr>
</tbody>
</table>
Binary to Hex Conversion

What is the hex equivalent of:

0 1 1 0 0 0 1 1 1 0 0 1 0 0 1
Binary to Hex Conversion

What is the hex equivalent of:

0 1 1 0 0 0 1 1 1 0 0 1 0 0 1

Partition the binary digits into groups of four – **starting from the right-hand-side**
Binary to Hex Conversion

What is the hex equivalent of:

```
0 1 1|0 0 0 1|1 1 0 0|1 0 0 1
```

```
3 1 C 9
```

Convert the individual groups
Binary to Hex Conversion

In C notation (the programming language), we will write:

0x31C9
Binary to Octal Conversion

What is the octal equivalent of:

01100011110010011
Binary to Octal Conversion

What is the octal equivalent of:

0 1 1 0 0 0 1 1 1 0 0 1 0 0 1

Partition the binary digits into groups of three – starting from the right-hand-side
Binary to Octal Conversion

What is the octal equivalent of:

0 1 1 0 0 0 1 1 1 0 0 1 0 0 1

Convert the individual groups:

3 0 7 1 1
Binary to Octal Conversion

In C notation (the programming language), we will write:

030711
Octal or Hex to Binary

How do we perform this type of conversion?
Octal or Hex to Binary

How do we perform this type of conversion?

• For each octal or hex digit, convert to the binary equivalent (3 or 4 binary digits, respectively)

• Append the binary digits together
Binary Notation in C

How would we write a binary constant in C?
Binary Notation in C

How would we write a binary constant in C?

0b011000111001001