Unit 3

Number Systems

Boolean Algebra Part 1
ANALOG VS. DIGITAL
Analog vs. Digital

• The analog world is based on continuous events. Observations can take on (real) any value.

• The digital world is based on discrete events. Observations can only take on a finite number of discrete values.
Analog vs. Digital

• Q. Which is better?
• A. Depends on what you are trying to do.

• Some tasks are better handled with analog data, others with digital data.
  – Analog means continuous/real valued signals with an infinite number of possible values
  – Digital signals are discrete [i.e. 1 of n values]
Analog vs. Digital

• How much money is in my checking account?
  – Analog: Oh, some, but not too much.
  – Digital: $243.67
Analog vs. Digital

• How much do you love me?
  – Analog: I love you with all my heart!!!!
  – Digital: $3.2 \times 10^3$ MegaHearts
The Real (Analog) World

• The real world is inherently analog.
• To interface with it, our digital systems need to:
  – Convert analog signals to digital values (numbers) at the input.
  – Convert digital values to analog signals at the output.
• Analog signals can come in many forms
  – Voltage, current, light, color, magnetic fields, pressure, temperature, acceleration, orientation
Digital is About Numbers

• In a digital world, numbers are used to represent all the possible discrete events
  – Numerical values
  – Computer instructions (ADD, SUB, BLE, ...)
  – Characters ('a', 'b', 'c', ...)
  – Conditions (on, off, ready, paper jam, ...)

• Numbers allow for easy manipulation
  – Add, multiply, compare, store, ...

• Results are repeatable
  – Each time we add the same two numbers we get the same result
DIGITAL REPRESENTATION
Interpreting Binary Strings

• Given a string of 1’s and 0’s, you need to know the *representation system* being used, before you can understand the value of those 1’s and 0’s.
• Information (value) = Bits + Context (System)

```
01000001 = ?
```

- Unsigned Binary system: $65_{10}$
- BCD System: $41_{BCD}$
- ASCII system: ‘A’$_{ASCII}$
Binary Representation Systems

- Integer Systems
  - Unsigned
    - Unsigned (Normal) binary
  - Signed
    - Signed Magnitude
    - 2’s complement
    - Excess-\(N\)*
    - 1’s complement*

- Floating Point
  - For very large and small (fractional) numbers

- Codes
  - Text
    - ASCII / Unicode
  - Decimal Codes
    - BCD (Binary Coded Decimal) / (8421 Code)

* = Not fully covered in this class
Number Systems

- Number systems consist of
  1. A base (radix) \( r \)
  2. \( r \) coefficients [0 to \( r-1 \)]

- Human System: Decimal (Base 10): 
  0,1,2,3,4,5,6,7,8,9

- Computer System: Binary (Base 2): 0,1

- Human systems for working with computer systems (shorthand for human to read/write binary)
  - Octal (Base 8): 0,1,2,3,4,5,6,7
  - Hexadecimal (Base 16): 0-9,A,B,C,D,E,F (A thru F = 10 thru 15)
Anatomy of a Decimal Number

- A number consists of a string of explicit coefficients (digits).
- Each coefficient has an implicit place value which is a power of the base.
- The value of a decimal number (a string of decimal coefficients) is the sum of each coefficient times its place value.

\[(934)_{10} = 9 \times 10^2 + 3 \times 10^1 + 4 \times 10^0 = 934\]

\[(3.52)_{10} = 3 \times 10^0 + 5 \times 10^{-1} + 2 \times 10^{-2} = 3.52\]
Anatomy of a Binary Number

• Same as decimal but now the coefficients are 1 and 0 and the place values are the powers of 2

\[(1011)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0\]
General Conversion From Base r to Decimal

• A number in base r has place values/weights that are the powers of the base

• Denote the coefficients as: \( a_i \)

\[
(a_3a_2a_1a_0.a_{-1}a_{-2})_r = a_3 \cdot r^3 + a_2 \cdot r^2 + a_1 \cdot r^1 + a_0 \cdot r^0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2}
\]

Left-most digit = Most Significant Digit (MSD)  
Right-most digit = Least Significant Digit (LSD)

\[
N_r \Rightarrow \Sigma_i(a_i \cdot r^i) \Rightarrow D_{10}
\]

Number in base \( r \)  
Decimal Equivalent
Examples

\[(746)_8 = 7 \times 8^2 + 4 \times 8^1 + 6 \times 8^0\]
\[= 448 + 32 + 16 = 486_{10}\]

\[(1A5)_{16} = 1 \times 16^2 + 10 \times 16^1 + 5 \times 16^0\]
\[= 256 + 160 + 5 = 421_{10}\]

\[(AD2)_{16} = 10 \times 16^2 + 13 \times 16^1 + 2 \times 16^0\]
\[= 2560 + 208 + 2 = (2770)_{10}\]
Binary Examples

\[
(1001.1)_2 = 8 + 1 + 0.5 = 9.5_{10}
\]

\[
(10110001)_2 = 128 + 32 + 16 + 1 = 177_{10}
\]
# Powers of 2

<table>
<thead>
<tr>
<th>$2^0$</th>
<th>$2^1$</th>
<th>$2^2$</th>
<th>$2^3$</th>
<th>$2^4$</th>
<th>$2^5$</th>
<th>$2^6$</th>
<th>$2^7$</th>
<th>$2^8$</th>
<th>$2^9$</th>
<th>$2^{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>1024</td>
</tr>
</tbody>
</table>

| 1024  | 512   | 256   | 128   | 64    | 32    | 16    | 8     | 4     | 2     | 1        |
Unique Combinations

- Given \( n \) digits of base \( r \), how many unique numbers can be formed? \( r^n \)
  - What is the range? \([0 \text{ to } r^n-1]\)

2-digit, decimal numbers (\( r=10, n=2 \))

3-digit, decimal numbers (\( r=10, n=3 \))

4-bit, binary numbers (\( r=2, n=4 \))

6-bit, binary numbers (\( r=2, n=6 \))

Main Point: Given \( n \) digits of base \( r \), \( r^n \) unique numbers can be made with the range \([0 \text{ to } (r^n)-1]\)
Approximating Large Powers of 2

• Often need to find decimal approximation of a large powers of 2 like $2^{16}$, $2^{32}$, etc.

• Use following approximations:
  – $2^{10} \approx 10^3$ (1 thousand) = 1 Kilo-
  – $2^{20} \approx 10^6$ (1 million) = 1 Mega-
  – $2^{30} \approx 10^9$ (1 billion) = 1 Giga-
  – $2^{40} \approx 10^{12}$ (1 trillion) = 1 Tera-

• For other powers of 2, decompose into product of $2^{10}$ or $2^{20}$ or $2^{30}$ and a power of 2 that is less than $2^{10}$
  – 16-bit half word: 64K numbers
  – 32-bit word: 4G numbers
  – 64-bit dword: 16 million trillion numbers

\[
\begin{align*}
2^{16} &= 2^6 \times 2^{10} \\
&\approx 64 \times 10^3 = 64,000 \\
2^{24} &= 2^4 \times 2^{20} \\
&\approx 16 \times 10^6 = 16,000,000 \\
2^{28} &= 2^8 \times 2^{20} \\
&\approx 256 \times 10^6 = 256,000,000 \\
2^{32} &= 2^2 \times 2^{30} \\
&\approx 4 \times 10^9 = 4,000,000,000
\end{align*}
\]
Decimal to Unsigned Binary

• To convert a decimal number, \( x \), to binary:
  
  – Only coefficients of 1 or 0. So simply find place values that add up to the desired values, starting with larger place values and proceeding to smaller values and place a 1 in those place values and 0 in all others

\[
25_{10} = \underline{0} \quad \underline{1} \quad \underline{1} \quad \underline{0} \quad \underline{0} \quad \underline{1}
\]

For \( 25_{10} \) the place value 32 is too large to include so we include 16. Including 16 means we have to make 9 left over. Include 8 and 1.
Decimal to Unsigned Binary

73_{10} = \begin{array}{cccccccc}
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\
\end{array}

87_{10} = \begin{array}{cccccccc}
0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\
\end{array}

145_{10} = \begin{array}{cccccccc}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\end{array}

0.625_{10} = \begin{array}{cccc}
1 & 0 & 1 & 0 \\
.5 & .25 & .125 & .0625 \\
\end{array}

Decimal to Another Base

• To convert a decimal number, $x$, to base $r$:
  
  – Use the place values of base $r$ (powers of $r$). Starting with largest place values, fill in coefficients that sum up to desired decimal value without going over.

$$75_{10} = \begin{array}{cccc}
0 & 4 & B & \text{hex} \\
256 & 16 & 1 & \\
\end{array}$$
Hexadecimal and Octal

SHORTHAND FOR BINARY
Binary, Octal, and Hexadecimal

- Octal (base $8 = 2^3$)
  - 1 Octal digit $(?)_8$ can represent: $0 - 7$
  - 3 bits of binary $(? ? ?)_2$ can represent: $000-111 = 0 - 7$
  - Conclusion...
    1 Octal digit = 3 bits

- Hex (base $16 = 2^4$)
  - 1 Hex digit $(?)_16$ can represent: $0-F$ (0-15)
  - 4 bits of binary $(? ? ? ?)_2$ can represent: $0000-1111 = 0-15$
  - Conclusion...
    1 Hex digit = 4 bits
Binary to Octal or Hex

- Make groups of 3 bits starting from radix point and working outward
- Add 0’s where necessary
- Convert each group of 3 to an octal digit

\[
\begin{align*}
101001110.11 & \quad 0 \\
5 & 1 & 6 & 6 \\
516.6_8 &
\end{align*}
\]

- Make groups of 4 bits starting from radix point and working outward
- Add 0’s where necessary
- Convert each group of 4 to an octal digit

\[
\begin{align*}
0000101001110.11 & \quad 00 \\
1 & 4 & E & C \\
14E.C_{16} &
\end{align*}
\]
Octal or Hex to Binary

• Expand each octal digit to a group of 3 bits

\[ 317.2_8 \]
\[ \overline{011001111.010}_2 \]
\[ 11001111.01_2 \]

• Expand each hex digit to a group of 4 bits

\[ D93.8_{16} \]
\[ \overline{110110010011.1000}_2 \]
\[ 110110010011.1_2 \]
Hexadecimal Representation

• Since values in modern computers are many bits, we use hexadecimal as a shorthand notation (4 bits = 1 hex digit)
  – 11010010 = D2 hex or 0xD2 if you write it in C/C++
  – 0111011011001011 = 76CB hex or 0x76CB if you write it in C/C++
ASCII & Unicode

BINARY CODES
Binary Representation Systems

• Integer Systems
  – Unsigned
    • Unsigned (Normal) binary
  – Signed
    • Signed Magnitude
    • 2’s complement
    • 1’s complement*
    • Excess-\(N^*\)

• Floating Point
  – For very large and small (fractional) numbers

• Codes
  – Text
    • ASCII / Unicode
  – Decimal Codes
    • BCD (Binary Coded Decimal) / (8421 Code)

* = Not covered in this class
Binary Codes

- Using binary we can represent any kind of information by coming up with a code
- Using $n$ bits we can represent $2^n$ distinct items

Colors of the rainbow:
- Red = 000
- Orange = 001
- Yellow = 010
- Green = 100
- Blue = 101
- Purple = 111

Letters:
- ‘A’ = 00000
- ‘B’ = 00001
- ‘C’ = 00010
- ‘Z’ = 11001
BCD

- Rather than convert a decimal number to binary which may lose some precision (i.e. $0.1_{10} = \text{infinite binary fraction}$), BCD represents each decimal digit as a separate group of bits (exact decimal precision)
  - Each digits is represented as a separate 4-bit number (using place values 8, 4, 2, 1 for each dec. digit)
  - Often used in financial and other applications where decimal precision is needed

\[
(439)_{10}
\]

BCD Representation: \[0100\ 0011\ 1001\]

Unsigned Binary Rep.: \[110110111\_2\]

Important: Some processors have specific instructions to operate on #’s represented in BCD
ASCII Code

• Used for representing text characters

• Originally 7-bits but usually stored as 8-bits = 1-byte in a computer

• Example:
  - "Hello\n"
  - Each character is converted to ASCII equivalent
    • ‘H’ = 0x48, ‘e’ = 0x65, ...
    • \n = newline character is represented by either one or two ASCII character
      - LF (0x0A) = line feed (moves cursor down a line)
      - CR (0x0D) = carriage return character (moves cursor to start of current line)
      - Newline for Unix / Mac = LF only
      - Newline for Windows = CR + LF
### ASCII Table

<table>
<thead>
<tr>
<th>LSD/MSD</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NULL</td>
<td>DLW</td>
<td>SPACE</td>
<td>0</td>
<td>@</td>
<td>P</td>
<td>`</td>
<td>p</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>q</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>“</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>s</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>ETB</td>
<td>‘</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>TAB</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
</tr>
<tr>
<td>A</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>:</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
</tr>
<tr>
<td>B</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>;</td>
<td>K</td>
<td>[</td>
<td>k</td>
<td>{</td>
</tr>
<tr>
<td>C</td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>&lt;</td>
<td>L</td>
<td>\</td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>=</td>
<td>M</td>
<td>]</td>
<td>m</td>
<td>}</td>
</tr>
<tr>
<td>E</td>
<td>SO</td>
<td>RS</td>
<td>.</td>
<td>&gt;</td>
<td>N</td>
<td>^</td>
<td>n</td>
<td>~</td>
</tr>
<tr>
<td>F</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>?</td>
<td>O</td>
<td>_</td>
<td>o</td>
<td>DEL</td>
</tr>
</tbody>
</table>
UniCode

- ASCII can represent only the English alphabet, decimal digits, and punctuation
  - 7-bit code => $2^7 = 128$ characters
  - It would be nice to have one code that represented more alphabets/characters for common languages used around the world
- Unicode
  - 16-bit Code => 65,536 characters
  - Represents many languages alphabets and characters
  - Used by Java as standard character code

Unicode hex value
(i.e. FB52 => 1111101101010010)
BOOLEAN ALGEBRA INTRO
Boolean Algebra

• A set of theorems to help us manipulate logical expressions/equations
• Axioms = Basis / assumptions used
• Theorems = manipulations that we can use
Axioms

• Axioms are the basis for Boolean Algebra
• Notice that these axioms are simply restating our definition of digital/binary logic
  – A1/A1’ = Binary variables (only 2 values possible)
  – A2/A2’ = NOT operation
  – A3,A4,A5 = AND operation
  – A3’,A4’,A5’ = OR operation

<table>
<thead>
<tr>
<th>(A1)</th>
<th>X = 0 if X \neq 1</th>
<th>(A1’)</th>
<th>X = 1 if X \neq 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A2)</td>
<td>If X = 0, then \overline{X}’ = 1</td>
<td>(A2’)</td>
<td>If X = 1, then \overline{X}’ = 0</td>
</tr>
<tr>
<td>(A3)</td>
<td>0 \cdot 0 = 0</td>
<td>(A3’)</td>
<td>1 + 1 = 1</td>
</tr>
<tr>
<td>(A4)</td>
<td>1 \cdot 1 = 1</td>
<td>(A4’)</td>
<td>0 + 0 = 0</td>
</tr>
<tr>
<td>(A5)</td>
<td>1 \cdot 0 = 0 \cdot 1 = 0</td>
<td>(A5’)</td>
<td>0 + 1 = 1 + 0 = 1</td>
</tr>
</tbody>
</table>
Duality

• Every truth statement can yield another truth statement
  – I exercise if I have time and energy (original statement)
  – I don’t exercise if I don’t have time or don’t have energy (dual statement)

• To express the dual, swap...

\[ 1\text{'s} \iff 0\text{'s} \]

\[ \iff \text{or} \]

Duality

• The “dual” of an expression is not equal to the original

\[ 1 + 0 \neq 0 \cdot 1 \]

Original expression \hspace{2cm} Dual

• Taking the “dual” of both sides of an equation yields a new equation

\[ X + 1 = 1 \hspace{2cm} X \cdot 0 = 0 \]

Original equation \hspace{2cm} Dual
### Single Variable Theorems

- Provide some simplifications for expressions containing:
  - a single variable
  - a single variable and a constant bit
- Each theorem has a dual (another true statement)
- Each theorem can be proved by writing a truth table for both sides (i.e. proving the theorem holds for all possible values of X)

<table>
<thead>
<tr>
<th></th>
<th>Expression 1</th>
<th>Expression 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>$X + 0 = X$</td>
<td>$T1'$</td>
</tr>
<tr>
<td>T2</td>
<td>$X + 1 = 1$</td>
<td>$T2'$</td>
</tr>
<tr>
<td>T3</td>
<td>$X + X = X$</td>
<td>$T3'$</td>
</tr>
<tr>
<td>T4</td>
<td>$(X')' = X$</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>$X + X' = 1$</td>
<td>$T5'$</td>
</tr>
</tbody>
</table>
Single Variable Theorem (T1)

\[ X + 0 = X \quad \text{(T1)} \]

\[ X \cdot 1 = X \quad \text{(T1')} \]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</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>

OR

Hold Y constant

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</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

Whenever a variable is OR’ed with 0, the output will be the same as the variable…

“0 OR Anything equals that anything”

Whenever a variable is AND’ed with 1, the output will be the same as the variable…

“1 AND Anything equals that anything”
Single Variable Theorem (T2)

\[ X + 1 = 1 \quad (T2) \]

\[ X \cdot 0 = 0 \quad (T2') \]

\[
\begin{array}{ccc}
X & Y & Z \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

OR

\[
\begin{array}{ccc}
X & Y & Z \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

AND

Hold Y constant

Whenever a variable is OR'ed with 1, the output will be 1…
“1 OR anything equals 1”

Whenever a variable is AND'ed with 0, the output will be 0…
“0 AND anything equals 0”
Single Variable Theorem (T3)

X+X = X (T3)

X•X = X (T3’)

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</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>

OR

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</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

Whenever a variable is OR’ed with itself, the result is just the value of the variable.

Whenever a variable is AND’ed with itself, the result is just the value of the variable.

This theorem can be used to reduce two identical terms into one OR to replicate one term into two.
Single Variable Theorem (T4)

\[(X')' = X \ (T4)\]

\[(\overline{X}) = X \ (T4)\]

Anything inverted twice yields its original value.
Single Variable Theorem (T5)

\[ X + \overline{X} = 1 \quad (T5) \]

\[ X \cdot \overline{X} = 0 \quad (T5') \]

Whenever a variable is OR’ed with its complement, one value has to be 1 and thus the result is 1.

Whenever a variable is AND’ed with its complement, one value has to be 0 and thus the result is 0.

This theorem can be used to simplify variables into a constant or to expand a constant into a variable.
Application: Channel Selector

- Given 4 input, digital music/sound channels and 4 output channels
- Given individual “select” inputs that select 1 input channel to be routed to 1 output channel
Application: Steering Logic

• 4-input music channels (ICHx)
  – Select one input channel (use ISELx inputs)
  – Route to one output channel (use OSELx inputs)
Application: Steering Logic

- 1\textsuperscript{st} Level of AND gates act as barriers only passing 1 channel
- OR gates combines 3 streams of 0’s with the 1 channel that got passed (i.e. ICH1)
- 2\textsuperscript{nd} Level of AND gates passes the channel to only the selected output

\textbf{AND:}
\begin{align*}
1 \text{ AND } \text{ICH}x &= \text{ICH}x \\
0 \text{ AND } \text{ICH}x &= 0
\end{align*}

\textbf{OR:}
\begin{align*}
0 + \text{ICH1} + 0 + 0 &= \text{ICH1}
\end{align*}

\textbf{AND:}
\begin{align*}
1 \text{ AND } \text{ICH1} &= \text{ICH1} \\
0 \text{ AND } \text{ICH1} &= 0
\end{align*}
Your Turn

• Build a circuit that takes 3 inputs: S, IN0, IN1 and outputs a single bit Y.

• It’s functions should be:
  – If S = 0, Y = IN0 (IN0 passes to Y)
  – If S = 1, Y = IN1 (IN1 passes to Y)