Orange Coast College
Business Division
Computer Science Department

CS 116- Computer Architecture

Logic Design: Part 2
Where are we?

- Number systems
  - Decimal
  - Binary (and related Octal and Hexadecimal)
- Binary encodings
  - BCD (8421 and XS3)
  - Gray
  - ASCII, EBCDIC, Unicode
What's next?

• Binary Arithmetic
  – Addition (nefarious) and Subtraction
  – Multiplication and Division

• Boolean Algebra
  – Not your mom's high school algebra
Binary Addition

• Simply adding two single digit numbers
  – Only 4 possible combinations

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>Carry</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>
<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>
</tbody>
</table>
Binary Addition

- What happens when we consider a carry-in
  - Essentially adding three single digits
  - 8 possible combinations

<table>
<thead>
<tr>
<th>Carry-In</th>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>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>
Binary Addition- Bring it together

• When two 1's are added together, their sum is 10 in base 2
• For the LSBs of the numbers to be added, there are only two digits to consider
• For all the following digits, they may have a carry digit from the previous column
  – As we see later, this can be expressed as a single boolean function
Binary Subtraction

- Subtracting two digits
- 4 possible combinations

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Difference</th>
<th>Borrow</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>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>
</tbody>
</table>
Binary Subtraction

- Subtracting 3 digits
  - Considering the borrow-in
- Self-exercise

<table>
<thead>
<tr>
<th>Borrow-In</th>
<th>A</th>
<th>B</th>
<th>Difference</th>
<th>Borrow</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>1</td>
</tr>
<tr>
<td>0</td>
<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>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<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>
<td>1</td>
</tr>
</tbody>
</table>
Is there a better way?

• Subtraction with borrow is not efficient for digital systems
  – Why? Trust me it is.
  – Hint: Extra logic required

• What is the better way?
  – Complements
Subtraction through Complements

- Problem: Find X-Y

  \( X = (21)_{10} = (10101)_2 \)

  \( Y = (19)_{10} = (10011)_2 \)

- Step 1: Convert Y to 2's complement form

  2's complement of Y = (01101)_2

- Step 2: Add the result to X

  \[
  \begin{align*}
  10101 \\
  + 01101 \\
  \hline
  00010 = (2)_{10}
  \end{align*}
  \]
Binary Multiplication

- The product of two binary numbers is 1
  - Only if both numbers are 1
- The product of two binary numbers is 0
  - If any of the number is 0
- Size of result is the combined size of the operands
- Details
  - Can be done as a group of shifting and additions.
  - Can also be carried out with fractional numbers.
Binary Multiplication:

Example:

(27)\text{$_{10}$} = 11011
(21)\text{$_{10}$} = \times 10101

\[
\begin{array}{c}
\text{11011} \\
\text{00000} \\
\text{11011} \\
\text{00000} \\
\text{11011}
\end{array}
\]

\[
\begin{array}{c}
\text{11011} \\
\hline
\text{1000110111}
\end{array}
\]

\[= 1 + 2 + 4 + 16 + 32 + 512 = (567)\text{$_{10}$}\]
Binary Division

• Similar to decimal number divisions.
  – Converted to a group of shifts, subtractions, & comparisons.
  – Can also be applied to fractional numbers.
  – Causes overflow if the divisor is zero or the quotient needs more bits to express than the computer word (or register) size.
  – There are special techniques for performing division directly on 2's complement numbers.
Binary Division Example

\[(50)_{10} = 110010\]
\[(5)_{10} = 101\]
\[(50)_{10} / (5)_{10} = (10)_{10}\]

\[
\begin{array}{c|c}
  & \text{quotient} \\
\hline
1010 & \\
101 & \text{dividend} \\
101 & \text{shifted divisor} \\
00101 & \text{reduced dividend} \\
101 & \text{shifted divisor} \\
000 & \text{remainder} \\
\end{array}
\]
Boolean Algebra

- The algebra of logic
- Express logic functions with logic equations.
  - Used in electronic circuits to build computers and other logic circuits
- Analyze and describe the behavior of logic circuits
- Amaze and astound your friends
Hmm, what is an algebra?

• A formal mathematical system
  – Consists of a set of objects
  – Operations on those objects.

• Examples
  – Numerical algebra
  – Set algebra
  – Matrix algebra
  – Boolean algebra
Boolean Algebra

- Variables
- Values
- Operations
  - AND, OR, NOT
- Laws
  - \( a + 1 = 1 \)
  - \( A + 0 = 0 \)
  - \( \text{Not(Not}(a) = a \)
  - \ldots \)
Variables and Values

• Variables
  – Named
    • a, b, c, . . .
  – Should be of Boolean type
  – Represent a changing value
    • i.e. Can represent any value from the input domain

• Values
  – Domain: 0 and 1 (True & False)
Operators

• Not
  – Negation/Inversion
  – Symbols: ', ! (in C), \( \overline{A} \)
  – Unary

• Or
  – Logical Sum
  – Symbols: +, || (in C)

• And
  – Logical Product
  – Symbols: *, && (in C)
Operator Precedence

- Determines the order to evaluate
- From highest to lowest
  - ( )- Parentheses
  - Not
  - And
  - Or

- What does this mean?
  - Consider: A+'B*C
    - Negate B,
    - And the result with C
    - Or with A
Boolean Functions

• Formula that has variables and operators
  – The 'output' value is on the left hand side
  – The variables and operators on the right

• Examples:
  • $X = A \cdot B \ (A \text{ AND } B)$
    – $X$ is true if and only if:
      Both $A$ and $B$ are true
  • $X = A + B \cdot 'C \ (A \text{ OR } B \text{ AND } \neg C)$
    – $X$ is true if and only if:
      Either $A$ is true or $B$ is true and $C$ is false
A bit more formal...

- **Literal**
  - A variable or the complement of the variable
  - A, B or A', B'

- **Product Term**
  - A Single literals
  - AND of two or more literals

- **Sum Term**
  - A single literal
  - OR of two or more literals
Sample Terms

• Product Term
  – A'
  – A * B' * C

• Sum Term
  – B
  – A + B + D
Combining the previous

• Product of sums
  – Logical AND of Sum Terms
  – $A' \cdot (A + B + C) \cdot (A + B' + C) \cdot (A' + B' + C)$

• Sum of products
  – Logical OR of Product Terms
  – $A' \cdot (A \cdot B \cdot C) + (A \cdot B' \cdot C) + (A' \cdot B' \cdot C)$
More formal descriptions

• Normal term
  – An expression where no variable appears more than once

• N-Variable Minterm
  – A Normal Product Term of N-Literals
    • Every variable appears exactly once

• N-Variable Maxterm
  – A Normal Sum Term of N-Literals
    • Every variable appears exactly once
Let's look at an example

- 4-variable Minterm’s:
  - $A \cdot B \cdot C \cdot D$
  - $A \cdot B' \cdot C \cdot D$
  - $A \cdot B \cdot C' \cdot D'$
  - $A' \cdot B' \cdot C' \cdot D'$

- 4-variable Maxterm’s:
  - $A + B + C + D$
  - $A' + B' + C + D$
  - $A + B + C' + D'$
  - $A' + B' + C' + D'$
What can we do with these?

• We have all elements of an algebra
  – Values
  – Variables
  – Operators

• Use these elements to derive a series of laws

• Laws are useful because
  – Allow us to find equivalent functions
    • May provide for more efficient circuits
  – Simplify a given function
    • May provide for cheaper circuits
Laws of Boolean Algebra

• Identity
  – $A + 0 = A$
  – $A \times 1 = A$

• Dominance
  – $A + 1 = 1$
  – $A \times 0 = 0$

• Idempotent
  – $A \times A = A$
  – $A + A = A$
Laws of Boolean Algebra

- **Inverse (Complement)**
  - $A + A' = 1$
  - $A \times A' = 0$

- **Involution (Law of the double complement)**
  - $(A')' = A$
    - Two negatives actually make a positive!
Laws of Boolean Algebra

- **Commutative**
  - $A + B = B + A$
  - $A \cdot B = B \cdot A$

- **Associative**
  - $A + (B + C) = (A + B) + C$
  - $A \cdot (B \cdot C) = (A \cdot B) \cdot C$

- **Distributive**
  - $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
  - $A + (B \cdot C) = (A + B) \cdot (A + C)$
Laws of Boolean Algebra

- **Covering (Absorption/Redundance)**
  - $A + A*B = A$
  - $A \cdot (A + B) = A$
  - $A + A' \cdot B = A + B$
  - $A \cdot (A' + B) = A \cdot B$

- **Combining**
  - $A \cdot B + A \cdot B' = A$
  - $(A + B) \cdot (A + B') = A$

- **Consensus**
  - $A \cdot B + A' \cdot C + B \cdot C = A \cdot B + A' \cdot C$
  - $(A + B) \cdot (A' + C) \cdot (B + C) = (A + B) \cdot (A' + C)$
Laws of Boolean Algebra

• Special set of laws
  – DeMorgan's Laws

• DeMorgan's states
  – \((A + B)' = A' \cdot B'\) (NOR)
  – \((A \cdot B)' = A' + B'\) (NAND)

• DeMorgan's theorems apply to more than 2 variables
Examples

• Applying DeMorgan's

\[ X = ((A' + C) \times (B + D'))' \]
\[ = (A' + C)' + (B + D')' \quad \text{By DeMorgan's} \]
\[ = (A'' \times C') + (B' \times D'') \quad \text{By DeMorgan's} \]
\[ = A' \times C' + B' \times D \]

• Another example

\[ X = A' \times B \times C + A' \times B \times C' \]
\[ = A' \times (B \times C + B \times C') \quad \text{By Distributive} \]
\[ = A' \times (B \times (C + C')) \quad \text{By Distributive} \]
\[ = A' \times (B \times 1) \quad \text{By Inverse} \]
\[ = A' \times B \quad \text{By Identity} \]
Interesting tidbits…

• Duality principle:
  – All laws of Boolean algebra are stated in pairs
  – They are obtained by:
    • swapping 0's and 1's, and
    • swapping +'s with *'s
More tidbits...

- The correctness of the laws can be proven by
  - Building their equivalent logic circuits & checking
  - Building the truth table of the function
  - Using Venn diagrams
    - Suitable for up to 3 variables
  - Derivation
    - The successive application of other laws
Truth Table

- Completely describe any combinational logic function
- Describe the output with respect to the input
  - Each entry specifies the value of the output for that particular input combination
  - Each output column represents a different function
- Functions may be directly constructed from the truth table
- For a function with n-inputs, there are $2^n$ entries in the truth table
Truth Table Examples

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Convert a Truth Table

- How do we convert a truth table to (one of) its associated function(s)?
- For each place where output is 1
  - Create an AND expression of the inputs
  - If input is 0, negate its value
  - Else, just use the variable name
  - OR all the expressions together

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Output(C)</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ C = \overline{AB} + AB \]
Now, what does a truth table look like?

- Hint, think addition

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sum</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Surprise, In-class exercise

• Join a different group than you were in before
• Goal: Create the boolean expression for each of the following
  – Simple single digit binary addition table
    • Sum
    • Carry
  – Binary addition with carry table
    • Sum
    • Carry
  – Simple subtraction
    • Difference
    • Borrow
Some Help

1) Simple Addition

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>Carry</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>
<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>
</tbody>
</table>

2) Addition with Carry-in

<table>
<thead>
<tr>
<th>Carry-In</th>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>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>1</td>
<td>0</td>
<td>1</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>

3) Simple Subtraction

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Diff</th>
<th>Borrow</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>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>
</tbody>
</table>