Overview of Computer Science

CSC 101 — Summer 2011

Boolean Logic - Gates and Circuits
The von Neumann Architecture
Lecture 6 — July 13, 2011

Announcements

• Lab #2 Tomorrow
  – Prelab is online
• Lab #1 due @ 5pm today
• Writing Assignment #2 due Tomorrow
  – Post on your CSC101 webpage by 5:00pm
  – Ask if you don’t know how to do this!
• Quiz #1 is Friday
  – Lectures 1-7

Objectives

• Boolean logic
  – True/false and truth tables
  – NOT, AND, OR, XOR, etc.
• Gates and circuits
  – How electricity can implement logic and represent binary values
  – How electrical circuits can perform math and store values
• The von Neumann Architecture
• Absolute vs. Relative Paths
  – Important for tomorrow’s lab
Boolean Logic

- George Boole (1815-1864)
  - The Mathematical Analysis of Logic (1847)
    - Described symbolic logic – a way to use a method like “algebra” to analyze logical relationships
    - Often called Boolean algebra or Boolean logic

- Regular math (algebra) can use any real number
  - A variable has an infinite number of possible values
    - \( y = 5x + 2 \) \( \Leftrightarrow \) \( y \) and \( x \) can take on any numerical values

- The only values in Boolean logic (Boolean algebra) are
  - TRUE and FALSE
    - Since there are only two possible values (two states)
      - TRUE can be represented by a switch that is on (a 1 bit)
      - FALSE can be represented by a switch that is off (a 0 bit)

- Operators are used to manipulate logical values
  - The basic operators for Boolean logic are
    | NOT | XOR (“exclusive OR”) |
    |-----|---------------------|
    | AND | NAND (“NOT AND”)   |
    | OR  | NOR (“NOT OR”)      |

- Each operator can be represented by
  - A Boolean expression
  - A logic diagram symbol
  - A truth table
Boolean Logic

• NOT
  - X = A’
  - If A is FALSE, then A’ is TRUE
  - If A is TRUE, then A’ is FALSE
  - X is always the opposite of A

<table>
<thead>
<tr>
<th>Boolean Expression</th>
<th>Logic Diagram Symbol</th>
<th>Truth Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>X = A’</td>
<td>A → X</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Boolean Logic

• AND
  - X = A • B (or just X = AB)
  - If A is TRUE, AND B is TRUE, then A • B is TRUE; otherwise A • B is FALSE

<table>
<thead>
<tr>
<th>Boolean Expression</th>
<th>Logic Diagram Symbol</th>
<th>Truth Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>X = A • B</td>
<td>A • B</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Boolean Logic

• OR
  - X = A + B
  - If either A OR B is TRUE, then A + B is TRUE; otherwise A + B is FALSE

<table>
<thead>
<tr>
<th>Boolean Expression</th>
<th>Logic Diagram Symbol</th>
<th>Truth Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>X = A + B</td>
<td>A + B</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Boolean Logic

• **XOR** (exclusive OR)
  - \( X = A \oplus B \)
  - If either \( A \) OR \( B \) is TRUE, BUT NOT BOTH, then \( A \oplus B \) is TRUE; otherwise \( A \oplus B \) is FALSE.

![XOR Truth Table](image)

• **NOR** ("not or" ... neither \( A \) NOR \( B \))
  - \( X = (A + B)' \)
  - If neither \( A \) NOR \( B \) is TRUE, then \( (A + B)' \) is TRUE; otherwise \( (A + B)' \) is FALSE.

![NOR Truth Table](image)

• **NAND** (NOT AND)
  - \( X = (A \cdot B)' \)
  - If both \( A \) and \( B \) are TRUE, then \( (A \cdot B)' \) is FALSE; otherwise \( (A \cdot B)' \) is TRUE.

![NAND Truth Table](image)
Boolean Logic

- Electronic circuits that perform Boolean logic are called gates
  - A **NOT** gate inverts its single input
  - An **AND** gate produces 1 only if both inputs are 1
  - An **OR** gate produces 0 only if both inputs are 0
  - An **XOR** gate produces 1 only if the inputs differ
  - A **NAND** gate produces 0 only if both inputs are 1
  - A **NOR** gate produces a 1 only if both inputs are 0
- All of these gates can be created by proper connections of transistors

Transistors

- A transistor has three connections
  - “Source”
  - “Base”
  - “Emitter”
- When electricity is applied to the Base, the switch ‘closes’ and connects the Source to the Emitter
  - If the emitter is ‘grounded’, then when the switch is closed, the source is also grounded
  - When something is grounded, its voltage is zero
  - So, if we make the voltage at the base = 1, then the voltage at the source is 0
  - This is the same behavior as a **NOT** gate

Transistors as Gates

- The simplest gates can be made with just one or two transistors
  - **NOT**, **NAND**, and **NOR** gates
Transistors as Gates

- All six gates can be constructed from one, two or three transistors
  - The Boolean operations that these gates perform can be combined to perform any logical operation, no matter how complex
  - The memory and CPU in your laptop consist of integrated circuits containing billions of these gates

Combinatorial Circuits

- Gates are combined into more complex circuits by using the output of one gate as the input for another
  - Example:
    - By combining one AND gate and one OR gate, we can construct
    - We will see another example of combining logic gates tomorrow

Adders

- The most basic mathematical operation is adding two numbers together
  - Logic circuits represent and manipulate logical values
    - TRUE and FALSE
  - We can represent binary numbers by anything with two values
    - TRUE = 1; FALSE = 0
  - A logic circuit that can add two binary numbers is called an adder
Adders

- An adder is a circuit that can do simple addition
  - The basic building block of a computer’s numerical processor
  - With some cleverness, combinations of adders can perform any kind of arithmetic
- But, a computer also needs memory to provide the numbers to the adder and to store the results
  - Computer memory is also made of logic gates

Basic Components of a Computer

- All computers, large or small, have the same basic parts
  - Central processor
  - Main memory
  - Auxiliary storage devices
  - Input/output (I/O) devices
- The software (instructions for the processor) are stored in the same memory with the data while the program is running

Basic Components of a Computer

- The main parts of a computer are on the motherboard
  - Desktop computer motherboard
Basic Components of a Computer

- The main parts of a computer are on the motherboard
  - Laptop computer motherboard

The von Neumann Architecture

- All modern computers follow the logical model of computing called the von Neumann Architecture
1. Computer organized into four main sections:
  - Central Processing Unit (CPU)
    - All of the computer’s operations and calculations
  - Main Memory
    - Active data and programs
  - Auxiliary Storage
    - Disks and other storage devices
  - I/O Subsystem
    - Input and output “peripherals”

2. Loaded-program design
  - Program instructions are copied from auxiliary storage into main memory for execution
  - Can run any program by simply loading the program’s instructions into main memory
  - No need to reconfigure the hardware for each program
Central Processing Unit

- The CPU has two main jobs
  - Execute program instructions to process data
  - Coordinate the machine’s activities
- Main memory works closely with the CPU
  - Instructions for all running programs are loaded into main memory
  - The data for all running programs also exists in main memory
  - The CPU accesses data and instructions quickly through a high-speed connection called a bus.

Main Memory

- Fast-access, binary storage for active data and program instructions
- Main memory is divided into words
  - Words are a standard, fixed size
    - Usually four bytes per word, depending on the design of the computer
- Each word has a unique address in memory
  - Each word may contain program instructions or data elements

Main Memory

- Main memory generally uses Random Access Memory (RAM)
  - General use
    - For programs and data
  - Very fast
    - Can keep up with fast CPUs
  - Expensive
  - Volatile
    - Data disappears when powered off
How Much RAM?

- Your computer’s RAM is not its hard disk
  - RAM is main memory
  - Disk storage is auxiliary storage
  - These are very different things
- More RAM costs more money
  - More RAM does not make your computer faster
  - But it does let you do more stuff
    - More memory is usually a good investment
- We will cover the differences between main memory and auxiliary storage tomorrow

Absolute vs. Relative Paths

- Two different ways of getting to a location on a computer (or the web!)
- Important for tomorrow’s lab!
- Absolute path:
  - Created whenever your link uses the full URL of an object or page.
Absolute vs. Relative Paths

• Absolute Path:
  - Created whenever your link uses the full URL of an object or page.
  - Example:
    - http://xkcd.com/394/
    - http://www.wfu.edu/~pryoee4/csc101/
  - Accessible anywhere!
  - Best choice when you need to send a user to another site, or are accessing information from another site.

Absolute vs. Relative Paths

• Relative Paths:
  - Used any time you need to send a visitor to another page within your site or include an object from your site (like an image) on one of your pages a relative link will work just fine
  - Example:
    - images/picture1.jpg
    - ../csc101/labs/lab1.html
  - ../ - what is this?
  - Need to know where you are now
    - And then where you want to go.