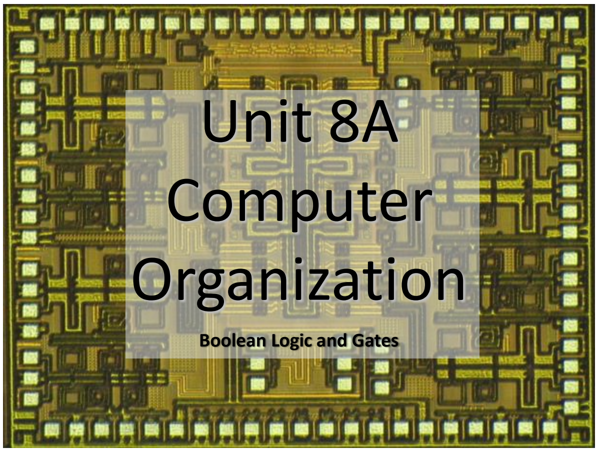


Announcements

- Bring ear buds or headphones to lab!

Representing and Manipulating Data

Last Unit

- How to represent data as a sequence of bits
- How to interpret bit representations
- Use of levels of abstraction in representing more complex information (music, pictures) using simpler building blocks (numbers)

This Unit

- How sequences of bits are implemented using electrical signals, and manipulated by circuits
- Use of levels of abstraction in designing more complex computer components from simpler components

Foundations

• • •

Boolean logic is the logic of digital circuits

Implementing Bits

- Computers compute by manipulating electricity according to specific rules.
- We associate electrical signals inside the machine with bits. Any electrical device with two distinct states (e.g. on/off switch, two distinct voltage or current levels) could implement our bits.
- The rules are implemented by electrical circuits.

Conceptualizing bits and circuits

- **ON** or **1** : **True**
- **OFF** or **0** : **False**
- **circuit behavior:**
  expressed in *Boolean logic* or *Boolean algebra*

Boolean Logic (Algebra)

Computer circuitry works based on **Boolean Logic** (Boolean Algebra) : operations on **True (1)** and **False (0)** values.

A	B	A ∧ B (A AND B) (conjunction)	A ∨ B (A OR B) (disjunction)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

A	¬A (NOT A) (negation)
0	1
1	0

- A and B in the table are Boolean variables,
  AND and OR are operations (also called functions).

●7

Boolean Logic & Truth Tables

- **Example:** You can think of A ∧ B below as
  *15110 is fun **and** 15110 is useful*
  where A stands for the statement *15110 is fun*,
  B stands for the statement *15110 is useful*.

A	B	A ∧ B (A AND B) (conjunction)	A ∨ B (A OR B) (disjunction)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

A	¬A (NOT A) (negation)
0	1
1	0

●9

Logic gates

• • •

the basic elements of digital circuits

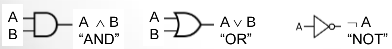
●10

Logic Gates

- A gate is a physical device that implements a Boolean operator by performing basic operations on electrical signals.
- Nowadays, gates are built from transistors.

physical picture of gates

Physical behavior of circuits is beyond the scope of our course.



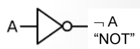
logical picture of gates

11

AND, OR, NOT Gates

A	B	A ∧ B (A AND B) (conjunction)	A ∨ B (A OR B) (disjunction)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

A	¬A (NOT A) (negation)
0	1
1	0



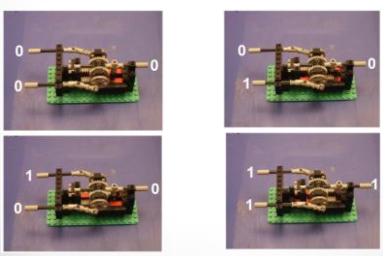
Truth tables define the input - output behavior of logic gates.

A Mechanical Implementation

Push-pull logic AND gate

For an input pushed-in lever represents 1

For an output pushed-in lever represents 0



Source: randomwraith.com by Martin Howard

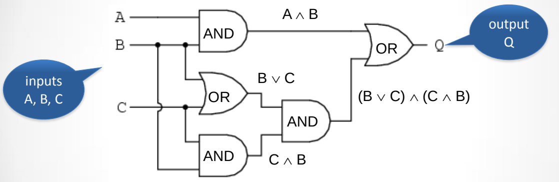
Combinational circuits

• • •

combinations of logic gates

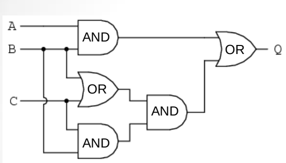
Combinational Circuits

The logic states of inputs at any given time determine the state of the outputs.



What is Q? $(A \wedge B) \vee ((B \vee C) \wedge (C \wedge B))$

Truth Table of a Circuit



$Q = (A \wedge B) \vee ((B \vee C) \wedge (C \wedge B))$

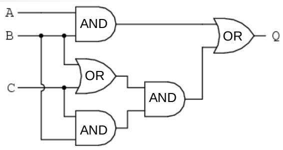
A	B	C	Q
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Describes the relationship between inputs and outputs of a device

How do I know that there should be 8 rows in the truth table?

http://www.allaboutcircuits.com/vol_4/chpt_7/6.html

Truth Table of a Circuit



Q = (A ∧ B) ∨ ((B ∨ C) ∧ (C ∧ B))

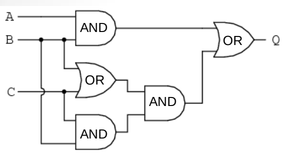
A	B	C	Q
0	0	0	0
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Describes the relationship between inputs and outputs of a device

http://www.allaboutcircuits.com/vol_4/chpt_7/6.html

17

Truth Table of a Circuit



Q = (A ∧ B) ∨ ((B ∨ C) ∧ (C ∧ B))

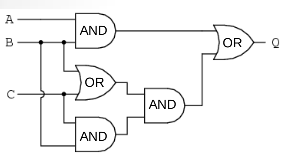
A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Describes the relationship between inputs and outputs of a device

http://www.allaboutcircuits.com/vol_4/chpt_7/6.html

18

Truth Table of a Circuit



Q = (A ∧ B) ∨ ((B ∨ C) ∧ (C ∧ B))

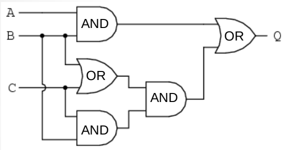
A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Describes the relationship between inputs and outputs of a device

http://www.allaboutcircuits.com/vol_4/chpt_7/6.html

19

Truth Table of a Circuit



Q = (A ∧ B) ∨ ((B ∨ C) ∧ (C ∧ B))

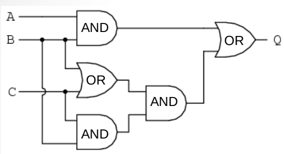
A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Describes the relationship between inputs and outputs of a device

http://www.allaboutcircuits.com/vol_4/chpt_7/6.html

20

Truth Table of a Circuit



Q = (A ∧ B) ∨ ((B ∨ C) ∧ (C ∧ B))

A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	
1	1	0	
1	1	1	

Describes the relationship between inputs and outputs of a device

http://www.allaboutcircuits.com/vol_4/chpt_7/6.html

21

Describing Behavior of Circuits

- Boolean expressions
- Circuit diagrams
- Truth tables

Equivalent notations

22

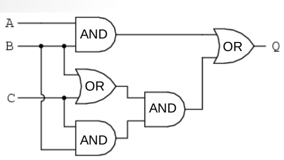
Manipulating circuits

Boolean algebra and logical equivalence

Why manipulate circuits?

- The design process
  - simplify a complex design for easier manufacturing, faster or cooler operation, …
- Boolean algebra helps us find another design guaranteed to have same behavior

Logical Equivalence

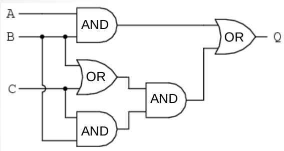


$Q = (A \wedge B) \vee ((B \vee C) \wedge (C \wedge B))$

A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

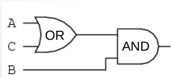
Can we come up with a simpler circuit implementing the same truth table?
Simpler circuits are typically cheaper to produce, consume less energy etc.

Logical Equivalence



$Q = (A \wedge B) \vee ((B \vee C) \wedge (C \wedge B))$

A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



$Q = B \wedge (A \vee C)$

This smaller circuit is logically equivalent to the one above: they have the same truth table. By using laws of Boolean Algebra we convert a circuit to another equivalent circuit.

26

Laws for the Logical Operators $\wedge$ and $\vee$ (Similar to $\times$ and +)

- Commutative: $A \wedge B = B \wedge A$ $\quad A \vee B = B \vee A$
- Associative: $A \wedge B \wedge C = (A \wedge B) \wedge C = A \wedge (B \wedge C)$
  $A \vee B \vee C = (A \vee B) \vee C = A \vee (B \vee C)$
- Distributive: $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$
  $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$
- Identity: $A \wedge 1 = A$ $\quad A \vee 0 = A$
- Dominance: $A \wedge 0 = 0$ $\quad A \vee 1 = 1$
- Idempotence: $A \wedge A = A$ $\quad A \vee A = A$
- Complementation: $A \wedge \neg A = 0$ $\quad A \vee \neg A = 1$
- Double Negation: $\neg\neg A = A$

27

Laws for the Logical Operators $\wedge$ and $\vee$ (Similar to $\times$ and +)

- Commutative: $A \wedge B = B \wedge A$ $\quad A \vee B = B \vee A$
- Associative: $A \wedge B \wedge C = (A \wedge B) \wedge C = A \wedge (B \wedge C)$
  $A \vee B \vee C = (A \vee B) \vee C = A \vee (B \vee C)$
- Distributive: $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$
  $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$ Not true for + and ×
- Identity: $A \wedge 1 = A$ $\quad A \vee 0 = A$

……

The A's and B's here are schematic variables! You can instantiate them with any expression that has a Boolean value:

$$\underbrace{(x \vee y)}_{A} \wedge \underbrace{z}_{B} = \underbrace{z}_{B} \wedge \underbrace{(x \vee y)}_{A} \quad \text{(by commutativity)}$$

28

Applying Properties for ∧ and ∨

Showing $(x \wedge y) \vee ((y \vee z) \wedge (z \wedge y)) = y \wedge (x \text{ or } z)$

Commutativity $A \wedge B = B \wedge A$

$(x \wedge y) \vee ((z \wedge y) \wedge (y \vee z))$

Distributivity $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$

$(x \wedge y) \vee (z \wedge y \wedge y) \vee (z \wedge y \wedge z)$

Associativity, Commutativity, Idempotence

$(x \wedge y) \vee ((z \wedge y) \vee (y \wedge z))$

Commutativity, idempotence $A \wedge A = A$

$(y \wedge x) \vee (y \wedge z)$

Distributivity (backwards) $(A \wedge B) \vee (A \wedge C) = A \wedge (B \vee C)$

$y \wedge (x \vee z)$

Conclusion:

$(x \wedge y) \vee ((y \vee z) \wedge (z \wedge y)) = y \wedge (x \vee z)$

Extending the system

more gates and DeMorgan's laws

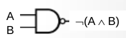
15110 Principles of Computing, Carnegie Mellon University - CORTINA

30

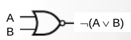
More gates (NAND, NOR, XOR)

A	B	A nand B	A nor B	A xor B
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	0	0	0

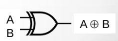
- nand ("not and"): A nand B = not (A and B)



- nor ("not or"): A nor B = not (A or B)



- xor ("exclusive or"):
  A xor B = (A and not B) or (B and not A)



31

A curious fact

- Functional Completeness of NAND and NOR
  - Any logical circuit can be implemented using NAND gates only
- Same applies to NOR

DeMorgan's Law

Nand: $\neg(A \wedge B) = \neg A \vee \neg B$

Nor: $\neg(A \vee B) = \neg A \wedge \neg B$

33

DeMorgan's Law

Nand: $\neg(A \wedge B) = \neg A \vee \neg B$

```
if not (x > 15 and x < 110):  ...
```

is logically equivalent to

```
if (not x > 15) or (not x < 110): ...
```

Nor: $\neg(A \vee B) = \neg A \wedge \neg B$

```
if not (x < 15 or x > 110): ...
```

is logically equivalent to

```
if (not x < 15) and (not x > 110): ...
```

34

A circuit for parity checking

Boolean expressions and circuits

A Boolean expression that checks parity

- 3-bit odd parity checker F: an expression that should be true when the count of 1 bits is odd: when 1 or 3 of the bits are 1s.

P = (¬A ∧ ¬B ∧ C) ∨ (¬A ∧ B ∧ ¬C) ∨ (A ∧ ¬B∧ ¬C) ∨ (A ∧ B ∧ C)

A	B	C	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

There are specific methods for obtaining canonical Boolean expressions from a truth table, such as writing it as a disjunction of conjunctions or as a conjunction of disjunctions. See the bonus slide at the end.

Note we have four subexpressions above each of them corresponding to exactly one row of the truth table where P is 1.

The circuit

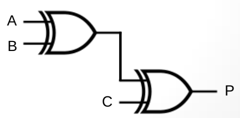
3-bit odd parity checker

P = (¬A ∧ ¬B∧ C) ∨ (¬A ∧ B ∧ ¬C) ∨ (A ∧ ¬B∧ ¬C) ∨ (A ∧ B ∧ C)

logically equivalent

A	B	C	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$P = (A \oplus B) \oplus C$



Summary

You should be able to:

- Identify basic gates
- Describe the behavior of a gate or circuit using Boolean expressions, truth tables, and logic diagrams
- Transform one Boolean expression into another given the laws of Boolean algebra

38

Bonus slide

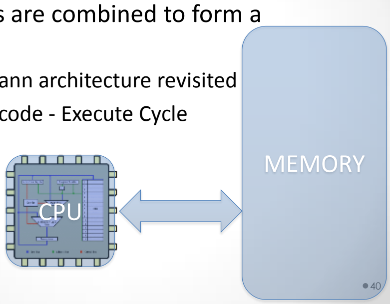
Using Minterms to Construct a Boolean Function from a Truth Table

As presented by Alvarado et. al. in *CS for All*:

1. Write down the truth table for the Boolean function that you are considering
2. Delete all the rows from the truth table where the value of the function is 0
3. For each remaining row create a "minterm" as follows:
   a. For each variable that has a 1 in that row write the name of the variable. If the input variable is 0 in that row, write the variable with a negation symbol.
   b. Take their conjunction (AND them together)
4. Combine all of the minterms using OR (take their disjunction)

Next Time

- How circuits are combined to form a computer
  - Von Neumann architecture revisited
  - Fetch – Decode - Execute Cycle



40