Discrete Structures

Previous Lecture Summery

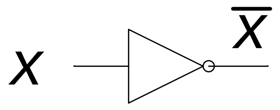
- Basic Logic gates
- Constructing Circuits using logic gates
- Designing Circuits for given Inputs/outputs
- Equivalent Circuits
- Reductions of circuits

Applications of Logic

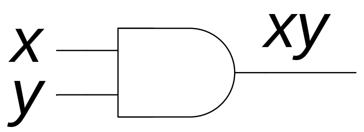
Lectures outline

- Basic Logic gates
- Circuits using logic gates
- Boolean Algebra
- Adders
- Reductions of circuits

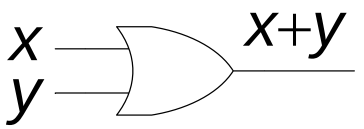
Basic Logic Gates

- Not 

where $\overline{x} = \neg x$

- And 

where $xy = x \wedge y$

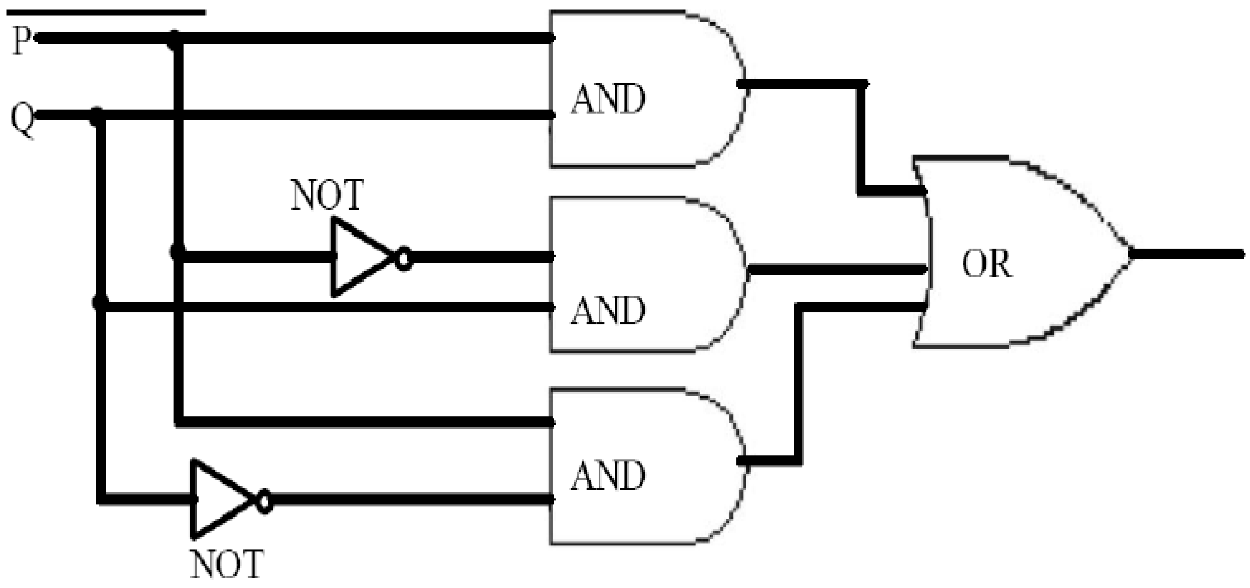
- Or 

where $x+y = x \vee y$

Constructing Circuits

Here is the circuit of the statement

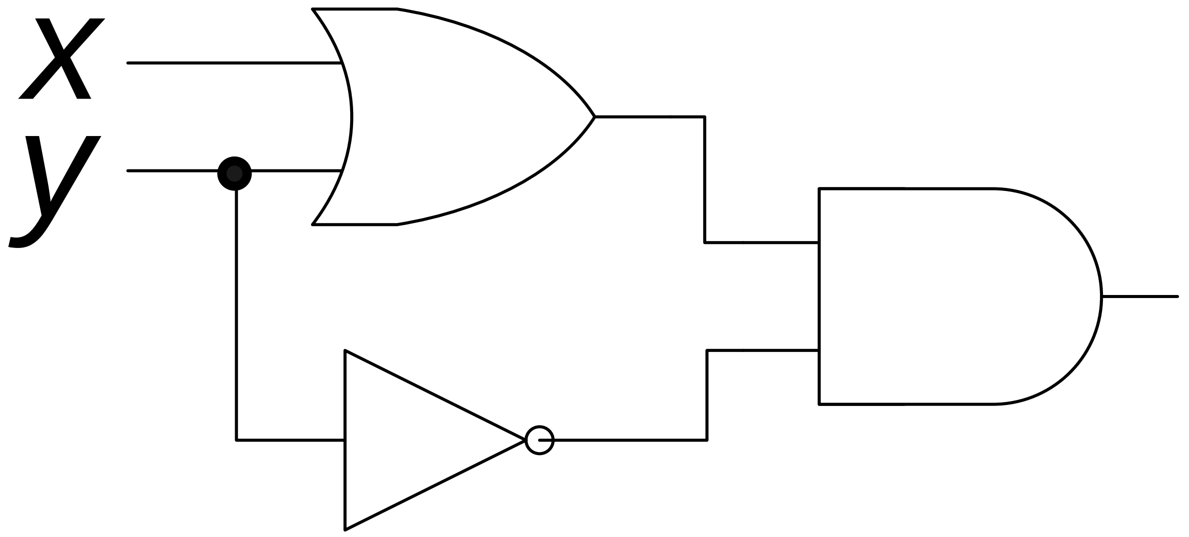
$$(p \wedge q) \vee (\sim p \wedge q) \vee (p \wedge \sim q)$$



Cont...

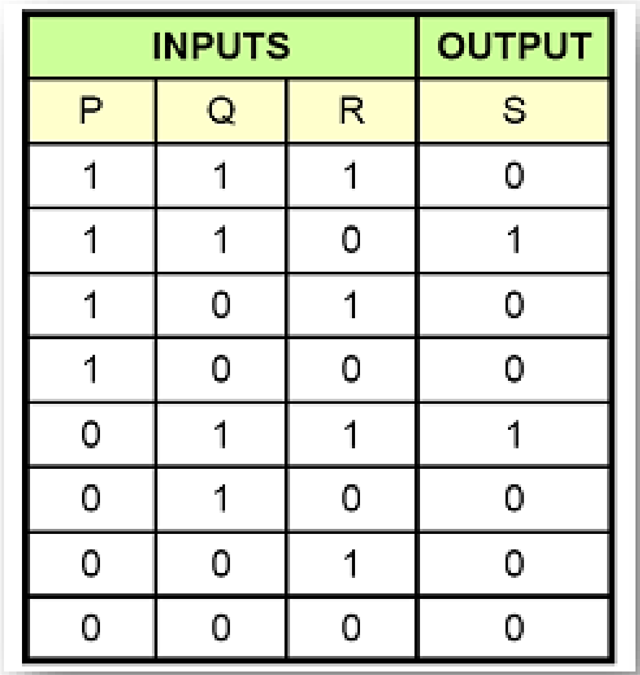
Following is the circuit output of the following statement

$(x + y) \wedge \neg y$



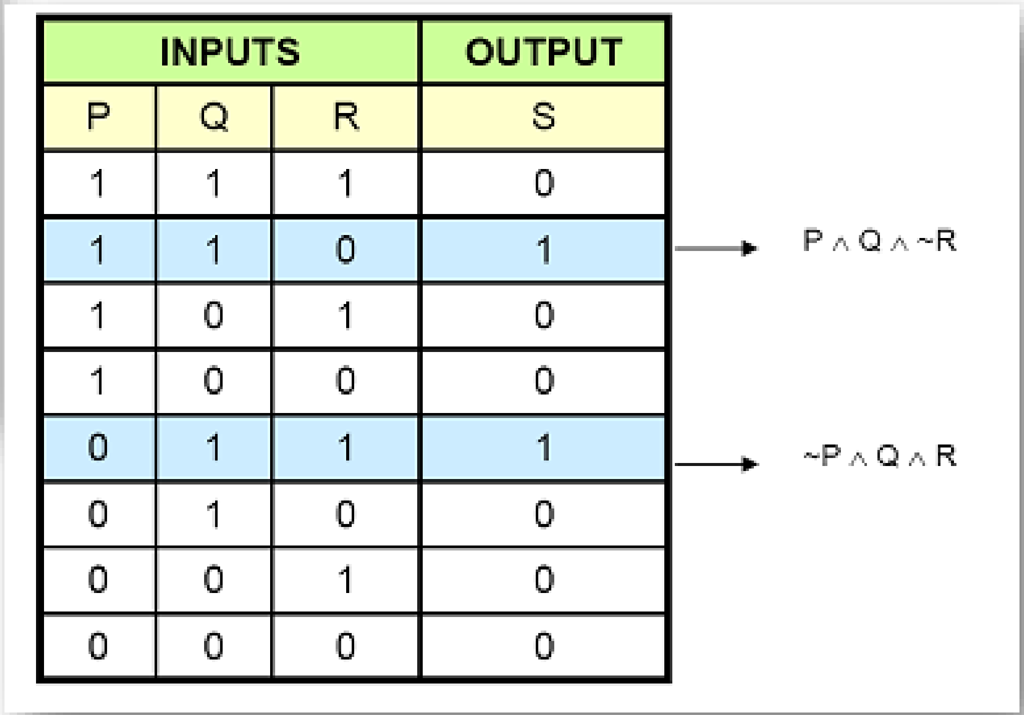
Designing a circuitt for a given input/output

Here is the out put



INPUTS			OUTPUT
P	Q	R	S
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

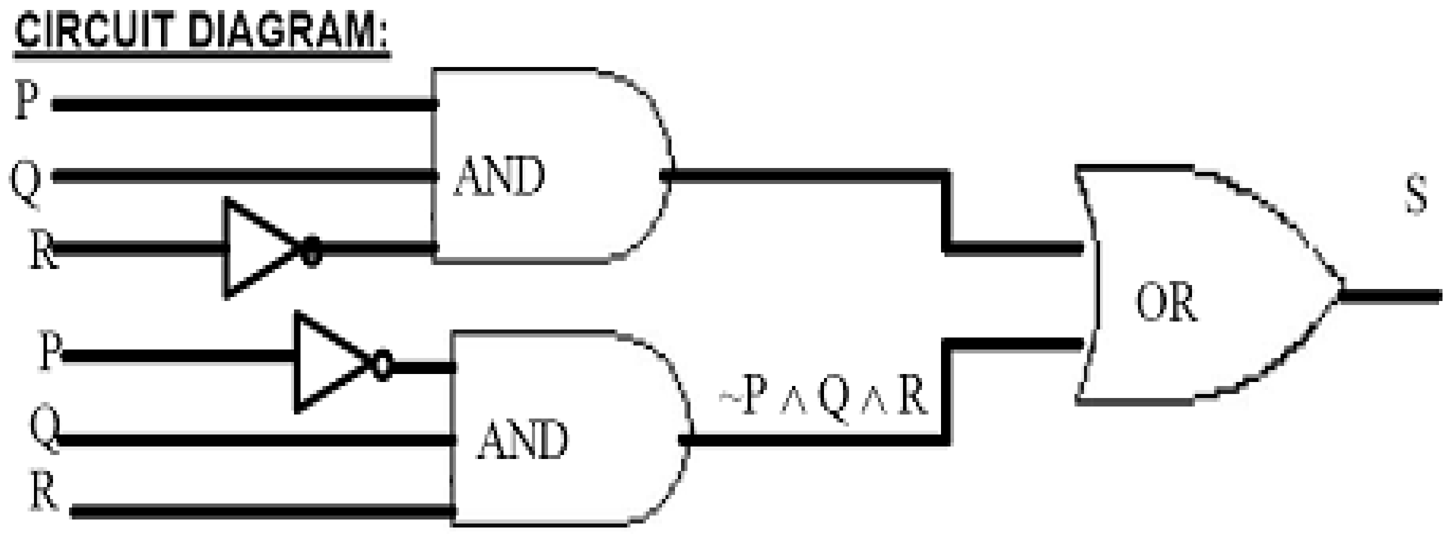
we can write it as following



INPUTS			OUTPUT	
P	Q	R	S	
1	1	1	0	
1	1	0	1	$\rightarrow$ P ∧ Q ∧ ~R
1	0	1	0	
1	0	0	0	
0	1	1	1	$\rightarrow$ ~P ∧ Q ∧ R
0	1	0	0	
0	0	1	0	
0	0	0	0	

Designing a circuitt for a given input/output

Here is the circuit of the previous input/output



Boolean Algebra

- Just like Boolean logic, variables can only be 1 or 0, instead of true/false
- Not

  ~0 = 1

  ~1 = 0

• Or is used as a plus	And is used as a multiplication
0+0 = 0	0 * 0 = 0
0+1=1	0 * 1 = 0
1+0=1	1 * 0 = 0
1+1= ?	1 * 1 = 1

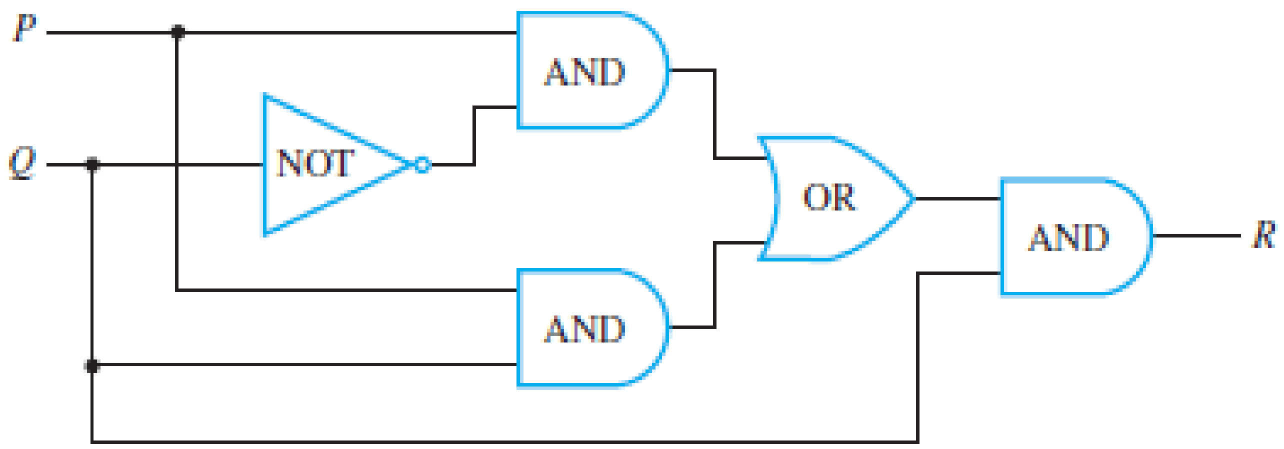
Todays Lecture Outline

- NAND and NOR Gates
- Basics of Boolean Algebra
- Decimal and Binary numbers
- Half Adders
- Circuits using Half adders
- Full adder circuits
- Parallel Adder Circuits

Equivalent Circuits

Following is the circuit representations of the statement

$[(P \wedge \sim Q) \vee (P \wedge Q)] \wedge Q$



Equivalent Circuits

$[(P \wedge \sim Q) \vee (P \wedge Q)] \wedge Q$

$\equiv (P \wedge (\sim Q \vee Q)) \wedge Q$; by the distributive law

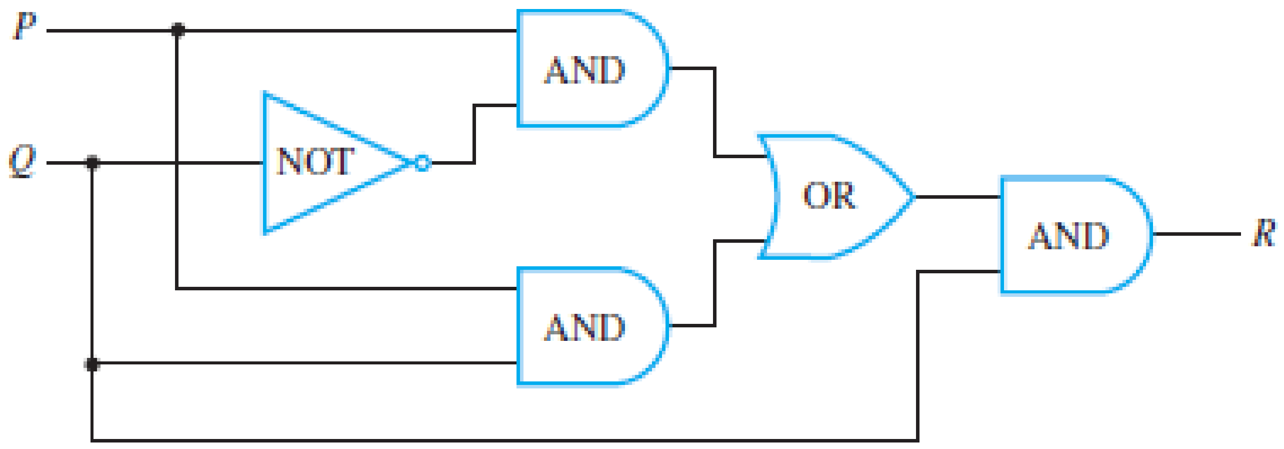
$\equiv (P \wedge (Q \vee \sim Q)) \wedge Q$; by the commutative law for ∨

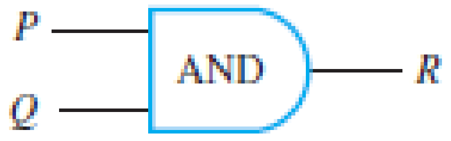
$\equiv (P \wedge \mathbf{t}) \wedge Q$; by the negation law

$\equiv P \wedge Q$; by the identity law.

Equivalent Circuits

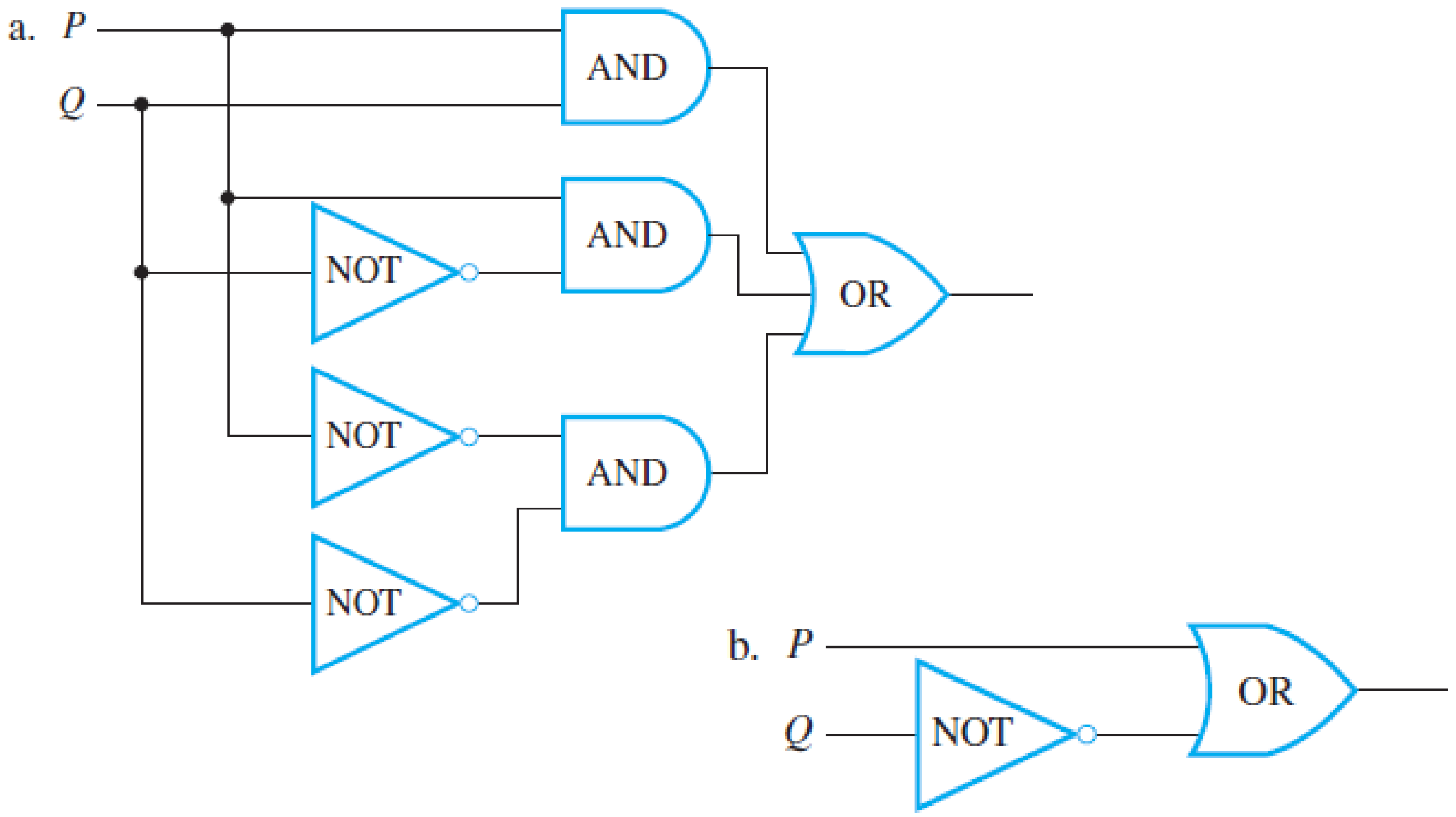
Thus the two circuits are logically equivalent.





Equivalent Circuits

Find the Boolean expressions for the circuits and show that they are logically equivalent



Equivalent Circuits

Find the Boolean expressions for the circuits and show that they are logically equivalent

$(P \wedge Q) \vee (P \wedge {\sim}Q) \vee ({\sim}P \wedge {\sim}Q)$

$\equiv ((P \wedge Q) \vee (P \wedge {\sim}Q)) \vee ({\sim}P \wedge {\sim}Q)$ by inserting parentheses (which is legal by the associative law)

$\equiv (P \wedge (Q \vee {\sim}Q)) \vee ({\sim}P \wedge {\sim}Q)$ by the distributive law

$\equiv (P \wedge \mathbf{t}) \vee ({\sim}P \wedge {\sim}Q)$ by the negation law for $\vee$

$\equiv P \vee ({\sim}P \wedge {\sim}Q)$ by the identity law for $\wedge$

$\equiv (P \vee {\sim}P) \wedge (P \vee {\sim}Q)$ by the distibutive law

$\equiv \mathbf{t} \wedge (P \vee {\sim}Q)$ by the negation law for $\vee$

$\equiv (P \vee {\sim}Q) \wedge \mathbf{t}$ by the commutative law for $\wedge$

$\equiv P \vee {\sim}Q$ by the identity law for $\wedge$

NAND and NOR Gates

Another way to simplify a circuit is to find an equivalent circuit that uses the least number of different kinds of logic gates. Two gates not previously introduced are useful for this: NAND-gate and NOR-gate.

A NAND-gate is a single gate that acts like an AND-gate followed by a NOT-gate.

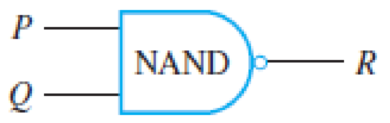
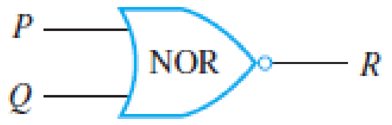
A NOR-gate acts like an OR-gate followed by a NOT-gate.

Thus the output signal of a NAND-gate is 0 when, and only when, both input signals are 1, and the output signal for a NOR-gate is 1 when, and only when, both input signals are 0.

The logical symbols corresponding to these gates are | (for NAND) and ↓ (for NOR), where | is called a **Sheffer stroke** and ↓ is called a **Peirce arrow**. Thus

$$P \mid Q \equiv \sim(P \wedge Q) \quad \textbf{and} \quad P \downarrow Q \equiv \sim(P \vee Q).$$

NAND and NOR Gates

Type of Gate	Symbolic Representation	Action
NAND		<table><tr><th colspan="2">Input</th><th>Output</th></tr><tr><th>P</th><th>Q</th><th>$R = P \mid Q$</th></tr><tr><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td></tr></table>
NOR		<table><tr><th colspan="2">Input</th><th>Output</th></tr><tr><th>P</th><th>Q</th><th>$R = P \downarrow Q$</th></tr><tr><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td></tr></table>

NAND and NOR Gates

It can be shown that any Boolean expression is equivalent to one written entirely with Sheffer strokes or entirely with Peirce arrows. Thus any digital logic circuit is equivalent to one that uses only NAND-gates or only NOR-gates.

Rewriting Expressions Using the Sheffer Stroke

Use the definition of Sheffer stroke to show that

a. $\sim P \equiv P \mid P$ b. $P \vee Q \equiv (P \mid P) \mid (Q \mid Q)$.

$$\begin{aligned}
\text{a. } \sim P &\equiv \sim(P \wedge P) && \text{by the idempotent law for } \wedge \\
&\equiv P \mid P && \text{by definition of } \mid.
\end{aligned}$$

$$\begin{aligned}
\text{b. } P \vee Q &\equiv \sim(\sim(P \vee Q)) && \text{by the double negative law} \\
&\equiv \sim(\sim P \wedge \sim Q) && \text{by De Morgan's laws} \\
&\equiv \sim((P \mid P) \wedge (Q \mid Q)) && \text{by part (a)} \\
&\equiv (P \mid P) \mid (Q \mid Q) && \text{by definition of } \mid.
\end{aligned}$$

Rewriting Expressions Using the Peirce Arrow

Show that the following logical equivalences hold for the Peirce arrow $\downarrow$, where $P \downarrow Q \equiv \sim(P \vee Q)$.

a. $\sim P \equiv P \downarrow P$

b. $P \vee Q \equiv (P \downarrow Q) \downarrow (P \downarrow Q)$

c. $P \wedge Q \equiv (P \downarrow P) \downarrow (Q \downarrow Q)$

b. $(P \downarrow Q) \downarrow (P \downarrow Q)$

$\equiv \sim(P \downarrow Q)$ by part (a)

$\equiv \sim[\sim(P \vee Q)]$ by definition of $\downarrow$

$\equiv P \vee Q$ by the double negative law

Boolean Algebra

Decimal representations

$6152 = 6*1000 + 1*100 + 5*10 + 2*1$

$= 6*10^3 + 1*10^2 + 5*10^1 + 2*10^0.$

More generally, decimal notation is based on the fact that any positive integer can be written uniquely as a sum of products of the form

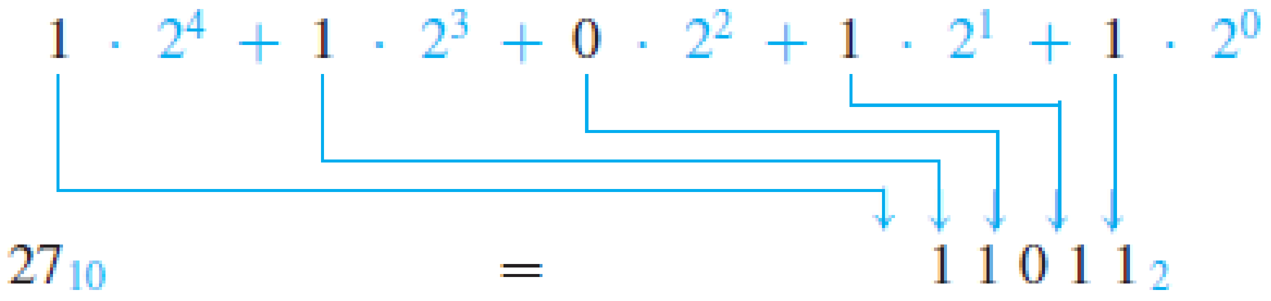
$$d \cdot 10^n$$

where each n is a nonnegative integer and each d is one of the decimal digits 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. The word *decimal* comes from the Latin root *deci,* meaning “ten.”

Boolean Algebra

Converting decimal to binary representations

$$27 = 16 + 8 + 2 + 1$$
$$= 1\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0.$$



$1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$
$27_{10} = 11011_2$

Power of 2	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal Form	1024	512	256	128	64	32	16	8	4	2	1

Boolean Algebra

Any integer can be represented uniquely as a sum of products of the form

$$d \cdot 2^n$$

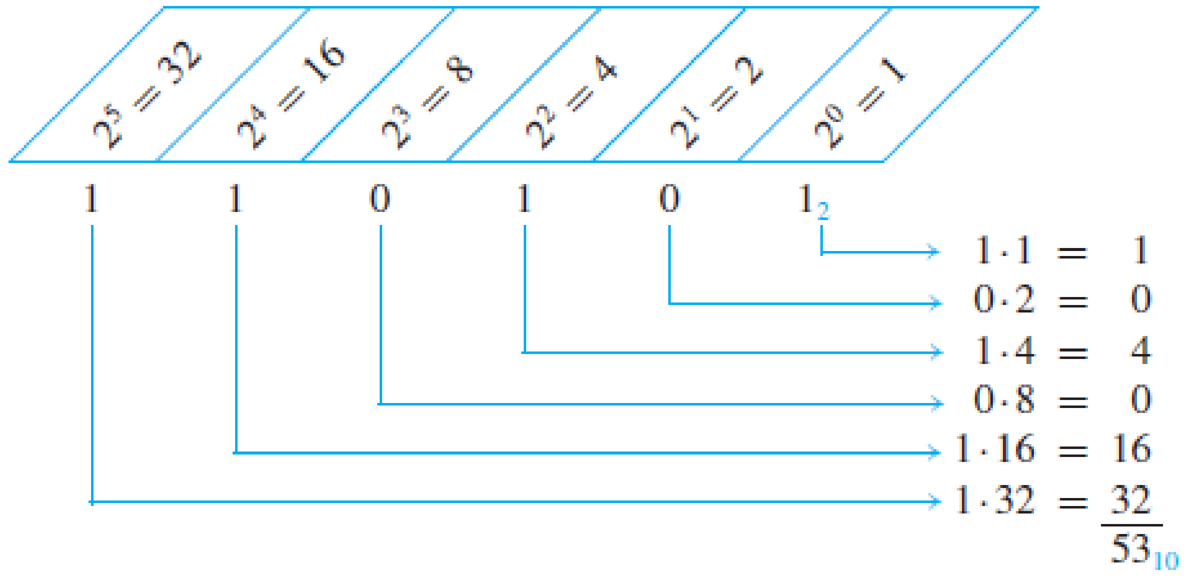
where each *n* is an integer and each *d* is one of the binary digits (or bits) 0 or 1.

$$\begin{aligned}
1_{10} &= & 1\cdot 2^0 &= 1_2 \\
2_{10} &= & 1\cdot 2^1 + 0\cdot 2^0 &= 10_2 \\
3_{10} &= & 1\cdot 2^1 + 1\cdot 2^0 &= 11_2 \\
4_{10} &= & 1\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0 &= 100_2 \\
5_{10} &= & 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 &= 101_2 \\
6_{10} &= & 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 &= 110_2 \\
7_{10} &= & 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 &= 111_2 \\
8_{10} &= & 1\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0 &= 1000_2 \\
9_{10} &= & 1\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 &= 1001_2
\end{aligned}$$

Converting binary to decimal

Represent 110101 in decimal notation.

$$\begin{aligned}110101_2 &= 1\cdot 2^5 + 1\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0\\ &= 32 + 16 + 4 + 1\\ &= 53_{10}\end{aligned}$$



Addition in Binary Notation

Add 11012 and 1112 using binary notation.

Solution: Because $2_{10} = 10_2$ and $1_{10} = 1_2$, the translation of $1_{10} + 1_{10} = 2_{10}$ to binary notation is

$$\begin{array}{r} 1_2 \\ +\ \ 1_2 \\ \hline 10_2 \end{array}$$

It follows that adding two 1's together results in a carry of 1 when binary notation is used. Adding three 1's together also results in a carry of 1 since $3_{10} = 11_2$ ("one one base two").

$$\begin{array}{r} 1_2 \\ +\ \ 1_2 \\ +\ \ 1_2 \\ \hline 11_2 \end{array}$$

Addition in Binary Notation

Thus the addition can be performed as follows:

$$\begin{array}{cccccc}
 & & 1 & 1 & 1 & & \leftarrow \text{carry row} \\
 & & 1 & 1 & 0 & 1_2 & \\
+ & & & 1 & 1 & 1_2 & \\
\hline
 & 1 & 0 & 1 & 0 & 0_2 &
\end{array}$$

Boolean Algebra

- Just like Boolean logic, variables can only be 1 or 0, instead of true/false
- Not

  ~0 = 1

  ~1 = 0

- Or is used as a plus

  0+0 = 0

  0+1=1

  1+0=1

  1+1= ?

And is used as a multiplication

0 * 0 = 0

0 * 1 = 0

1 * 0 = 0

1 * 1 = 1

Half Adder

- Consider adding two 1-bit binary numbers *x* and *y*

  0+0 = 0
  0+1 = 1
  1+0 = 1
  1+1 = 10

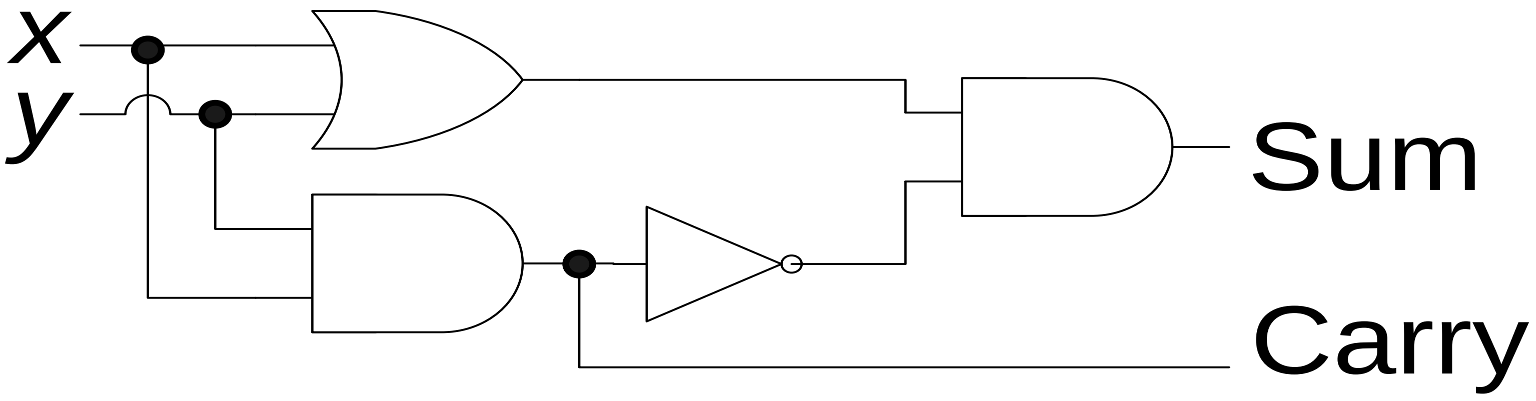
x	*y*	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Carry is *x* AND *y*
- Sum is *x* XOR *y*
- The circuit to compute this is called a half-adder.

Circuit of Half Adder

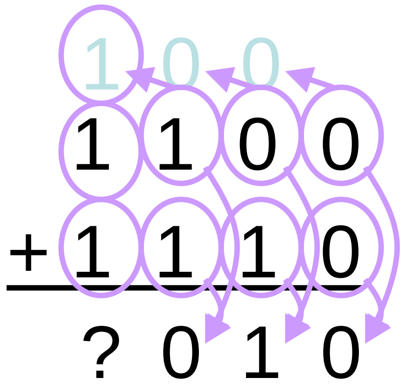
- Sum = *x* XOR *y*
- Carry = *x* AND *y*

x	*y*	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



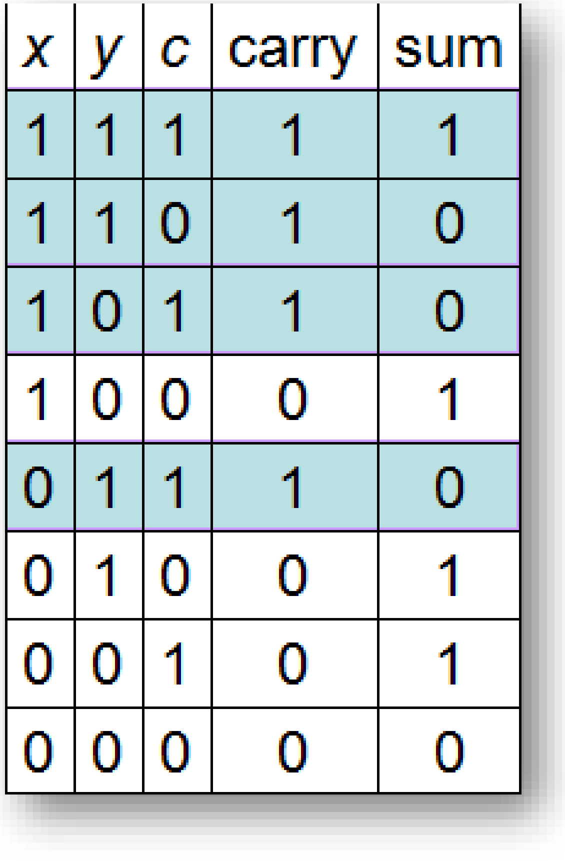
Using Half adders

- We can then use a half-adder to compute the sum of two Boolean numbers



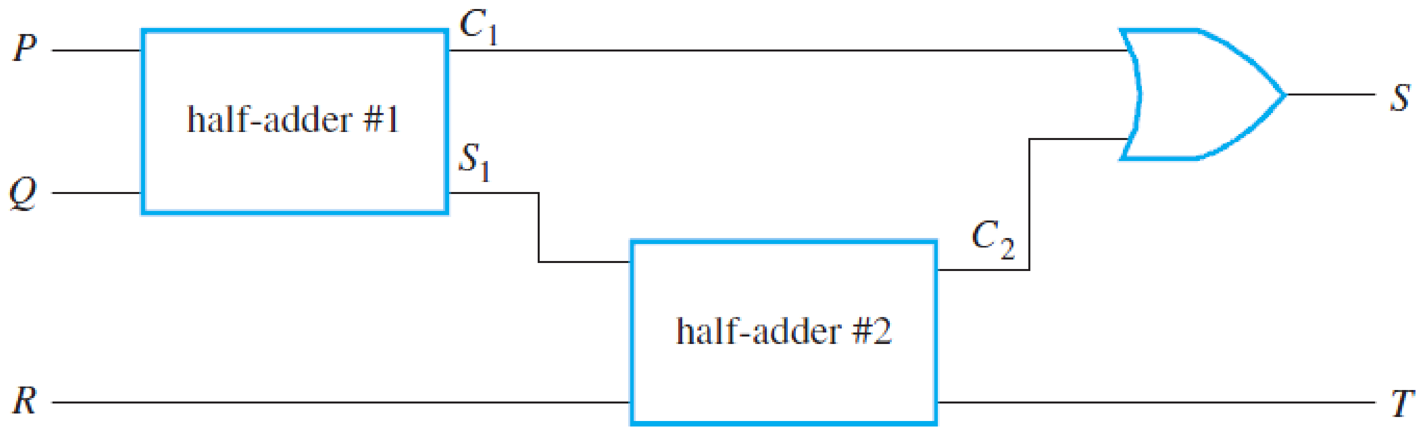
How to fix that

- We need to create an adder that can take a carry bit as an additional input
  - Inputs: *x*, *y*, carry in
  - Outputs: sum, carry out
- This is called a full adder
  - Will add *x* and *y* with a half-adder
  - Will add the sum of that to the carry in
- What about the carry out?
  - It's 1 if either (or both):
  - $x+y = 10$
  - $x+y = 01$ and carry in = 1



x	*y*	*c*	carry	sum
1	1	1	1	1
1	1	0	1	0
1	0	1	1	0
1	0	0	0	1
0	1	1	1	0
0	1	0	0	1
0	0	1	0	1
0	0	0	0	0

The Full adder

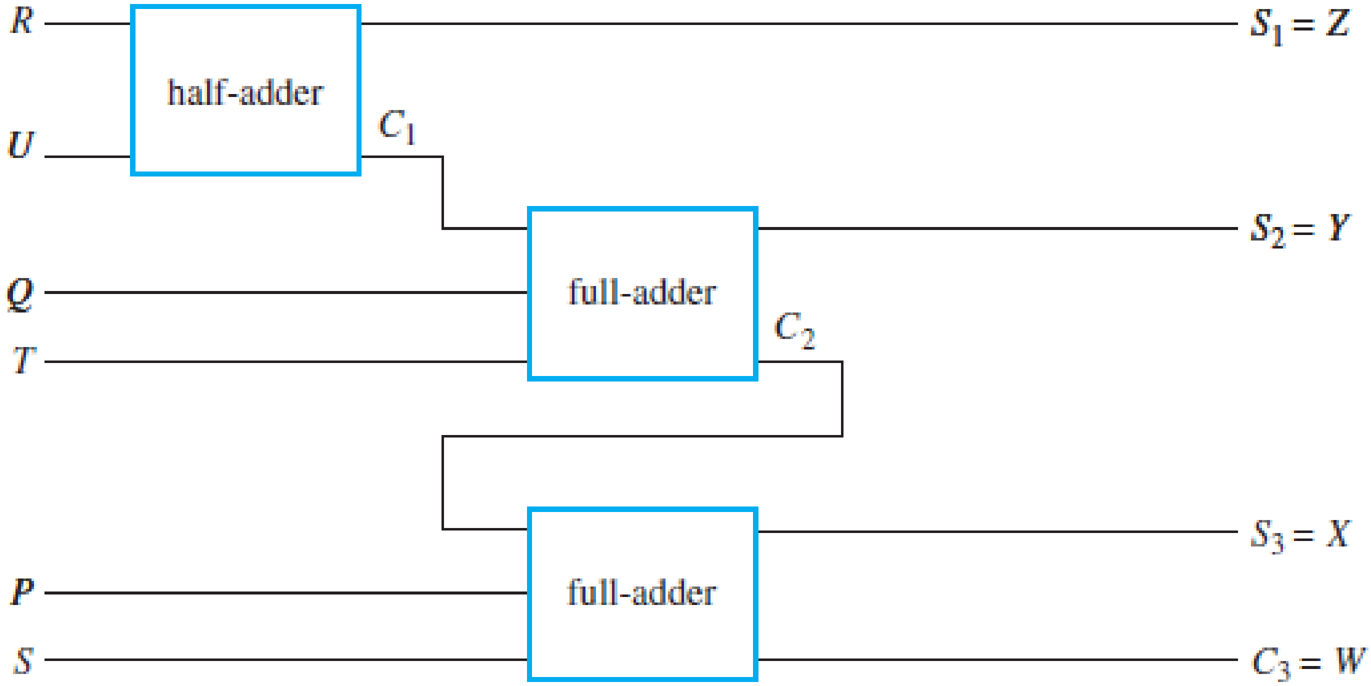


Input/Output Table

P	Q	R	C	S
1	1	1	1	1
1	1	0	1	0
1	0	1	1	0
1	0	0	0	1
0	1	1	1	0
0	1	0	0	1
0	0	1	0	1
0	0	0	0	0

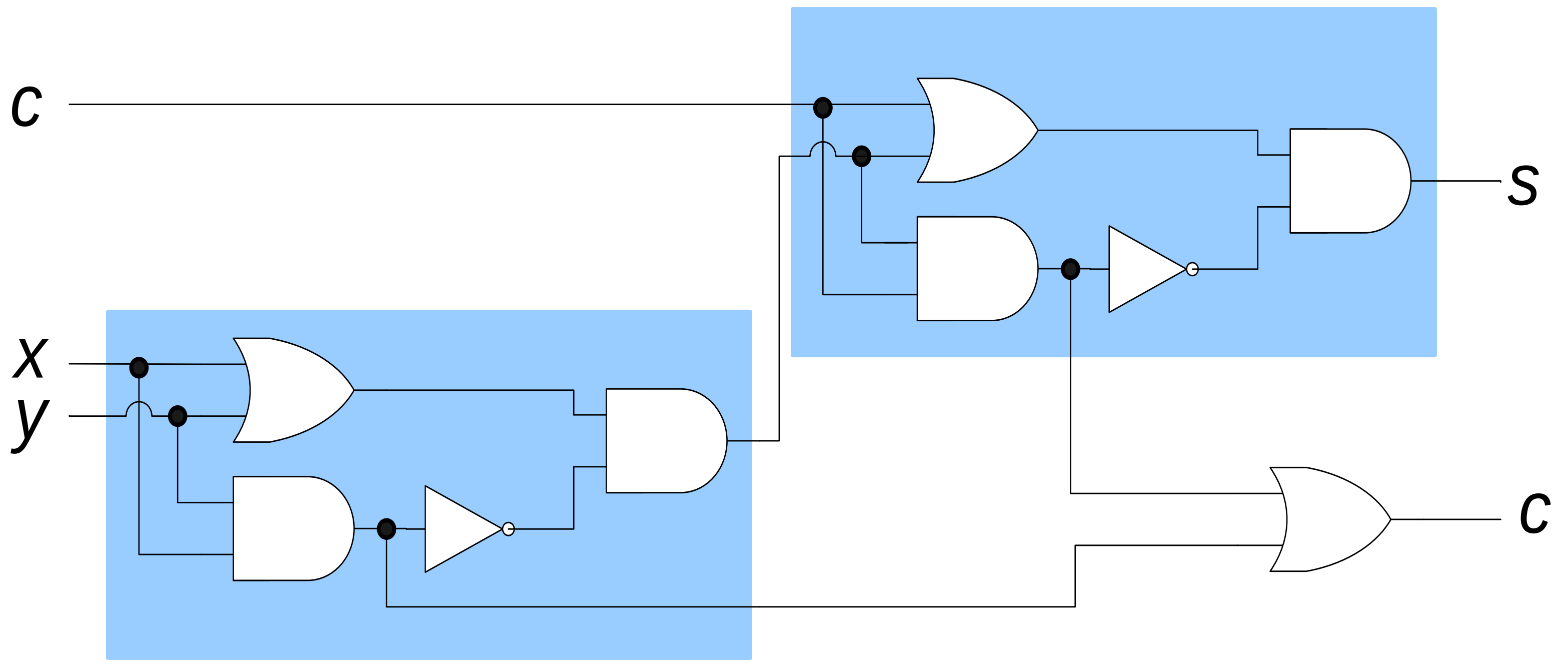
Parallel Adder Circuits

Two full-adders and one half-adder can be used together to build a circuit that will add two three-digit binary numbers *PQR* and *STU* to obtain the sum *WXYZ*. Such a circuit is called a **parallel adder.** Parallel adders can be constructed to add binary numbers of any finite length.



The Full adder

The full circuitry of the full adder



Lecture summary

- Basic Logic gates
- Circuits using logic gates
- Boolean Algebra
- Adders (Half and Full)