CSE332: Computer Architecture & Organization

Lecture 1,2,3: Review of DLD

Tanjila Farah (TnF)

What is Number System?

A number system defines a set of values used to represent quantity.

Quantifying values and items in relation to each other is helpful for us to make sense of our environment.

The Decimal Number System

•The primary number system used is a base ten number system.

•**Base ten number** systems are called *decimal* number systems. - ten numerals, 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

PLACE VALUE

	Millions 10^6	Hundred Thousands 10^5	Ten Thousands 10^4	Thousands 10^3	Hundreds 10^2	Tens 10^1	Ones 10^0	.	Tenths 10^{-1}	Hundredths 10^{-2}	Thousandths 10^{-3}
1,623,051 →	1	6	2	3	0	5	1	.			
0.053 →							0	.	0	5	3
32.4 →						3	2	.	4		

$$9735 = (9 * 10^3) + (7 * 10^2) + (3 * 10^1) + (5 * 10^0).$$

Different Number Systems

- **Octal Number ystem (base 8)**

  Number system with only eight numerals : 0, 1, 2, 3, 4, 5, 6, 7

- **Hexadecimal number system (base 16)**

  Number system with sixteen numerals : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 , A, B, C, D, E, F

- **Binary Number System (base 2)**

  Number system with only two numerals : 0 and 1

Example : Base-*r* to Decimal Conversion

Octal to Decimal

$(127.4)_8 = 1x8^2 + 2x8^1 + 7x8^0 + 4x8^{-1} = (87.5)_{10}$

Hexadecimal to Decimal

$(B65F)_{16} = 11x16^3 + 6x16^2 + 5x16^1 + 15x16^0 = (46687)_{10}$

Binary to Decimal

$(110101)_2 = 1x2^5 + 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = (53)_{10}$

Base-5 to Decimal

$(4021.2)_5 = 4x5^3 + 0x5^2 + 2x5^1 + 1x5^0 + 2x5^{-1} = (511.4)_{10}$

Converting Decimal to Binary

- Simply repeat dividing by 2 and saving the remainder.
- First remainder will be the last digit in the binary number, Second remainder will be the second digit from the right; ….. and so on

Step	Divide	Equals	Remainder	Digits
(1)	105 / 2 =	52	1	1
(2)	52 / 2 =	26	0	01
(3)	26 / 2 =	13	0	001
(4)	13 / 2 =	6	1	1001
(5)	6 / 2 =	3	0	01001
(6)	3 / 2 =	1	1	101001
(7)	1 / 2 =	0	1	1101001

So 105 in decimal is written as 1101001 in binary.

Converting Decimal to Octal

- Simply repeat dividing by 8 and saving the remainder.
- First remainder will be the last digit in the octal number
- Second remainder will be the second digit from the right;

….. and so on.

Step	Divide	Equals	Remainder	Digits
(1)	3034 / 8 =	379	2	2
(3)	379 / 8 =	47	3	32
(5)	47 / 8 =	5	7	732
(6)	5 / 8 =	0	5	5732

So 3034 in decimal is written as 5732 in Octal

Converting Decimal to Hex

- Simply repeat dividing by **16** and saving the remainder.
- First remainder will be the last digit in the octal number
- Second remainder will be the second digit from the right; ….. and so on.

Conversion of 16242 from decimal to hex

Step	**Divide**	**Equals**	**Remainder**	**Digits**
(1)	16242 / 16 =	1015	2 = 2 (hex)	2
(2)	1015 / 16 =	63	7 = 7 (hex)	72
(3)	63 / 16 =	3	15 = F (hex)	F72
(4)	3 / 16 =	0	3 = 3 (hex)	3F72

So 16242 in decimal is written as 3F72 in hex.

Converting Decimal fraction number to Base-*r*

General Rule

- Simply repeat multiplying the number and all successive fractions by **r** and saving the integer parts.

- These integer parts will be the coefficient of the Base-*r* number

- First integer will be the fast digit from left after the radix point, Second integer will be the second digit from the left after radix point   ….. and so on

Converting Decimal fraction number to Base-*r*

Example: convert $(0.6875)_{10}$ to binary Base-2

	Integer		***Fraction***	***Coefficient***
0.6875 x 2 = 1.3750	1	+	0.3750	$a_{-1} = 1$
0.3750 x 2 = 0.7500	0	+	0.7500	$a_{-2} = 0$
0.7500 x 2 = 1.5000	1	+	0.5000	$a_{-3} = 1$
0.5000 x 2 = 1.000	1	+	0.000	$a_{-4} = 1$

$$\mathbf{(0.6875)_{10} = (0.1011)_2}$$

Conversion of Decimal number with radix point to Base-*r*

- Separate the integer and fraction part
- Convert the integer and fraction point to Base-r number separately
- Combining the two answer

Example : *conversion of (105.6875)10 to Binary*

Separate the Integer and Fraction parts

$$(105.6875)_{10} = (105)_{10} + (0.6875)_{10}$$

Convert Integer and Fraction part to Base-r number separately

$(105)_{10} = (1101001)_2$ as discussed before

$(0.6875)_{10} = (1011)2$ as discussed before

Combining the two answers radix point

$$\mathbf{(105.6875)_{10} = (1101001.1011)_2}$$

Converting Between Hex, Octal and Binary

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Converting from Binary to Octal

- Simply by taking the binary digits in groups of three (from right to left) and converting to each group to Octal.

Consider the binary number
10110100111100101101001011

If we take the digits in groups of three from right to left and convert, we get:

10	110	100	111	100	101	101	001	011
2	6	4	7	4	5	5	1	3

10110100111100101101001011 (bin) is 264745513 (oct).

Converting from Octal to Binary

Since each octal digit can be expressed in exactly three binary digits, all we have to do is convert each octal digit to three binary digits.

Converting 7563021 in octal to binary goes as follows:

7	5	6	3	0	2	1
111	101	110	011	000	010	001

So 7563021 (octal) is 111101110011000010001 (binary.)

Converting from Binary to Hex

- Hex is base 16 and 16 is $\mathbf{2^4}$. That is, it takes ***exactly four binary digits to make a hex digit***

- By taking binary digits in **groups of four** (right to left) we can convert binary to hex.

- Consider once more the binary number 1011010011110010111001011. By grouping in fours and converting, we get:

10	1101	0011	1100	1011	0100	1011
2	D	3	C	B	8	B

So 1011010011110010111001011 (binary)

= 2D3CB8B (hex)

= 264745513 (octal)

Converting from Hex to Binary

Simply write each hex digit as four binary digits.

In this way we can convert 6F037C2:

6	F	0	3	7	C	2
0110	1111	0000	0011	0111	1100	0010

Since we can drop the leading zero,

6F037C2 (hex) is 110111100000011011111000010 (binary).

Binary Arithmetic

✓Complements

✓Signed Binary numbers

✓Binary addition, subtraction and Multiplication

Complements

- Used to simplify the subtraction operation and logical manipulations
  - Simplifying operations lead to simpler, less expensive circuits

- **Two types of Complements:**
  - **Radix Complement : (r's complement)**
  - **Diminished radix complement : (r-1)'s complement**

Diminished Radix Complement:

Given a number N having n digits in Base *-r*

the (r-1)'s complement of N is: $(r^n - 1) - N$

Example:

9's complement of 246700 is 999999 – 246700 = 753299

1's complement of 1101100 is 1111111 – 1101100 = 0010011

Radix Complement:

Given a number N having n digits in Base *-r*

the r's complement of N is: $r^n - N$ **for** $N \neq 0$

$= (r^n - 1) - N + 1$

= (r-1)'s complement + 1

Example:

10's complement of 246700 is = 9's complement of 246700 + 1

= (999999 – 246700) + 1

= 753299 + 1

= 753300

2's complement of 1101100 is = 1's complement of 1011000 + 1

= (1111111 – 1101100) +1

= 0010011 + 1

= 0010100

Signed and Unsigned Number

- Binary numbers with only two symbols, 0 and 1.
- No minus sign (-) to represent negative numbers in Binary
- Both signed and unsigned number consist of a string of bits when represented in a computer
- User determine whether the number is signed or unsigned

Unsigned number: The leftmost bit is the most significant bit of the number

Signed number: The leftmost bit is the sign bit and rest represents the number

Sign Bit: bit placed in the leftmost position of the number.

Conventionally

0 : indicate the positive number

1: indicate the negative number

Signed Binary Integers

<u>Three ways to represent negative binary numbers.</u>

✓**Signed-magnitude representation:**

The number consists of magnitude and sign bit

8-bit binary signed magnitude representation of $+(9)_{10}$ is **$(00001001\ )_2$ and $(-9)_{10}$ is $(10001001\ )_2$**

✓**Signed 1's complement representation:**

Taking 1's complement of the positive number including the sign bit

8-bit binary signed 1's complement representation of $-(9)_{10}$ is

$(11110110)_2$

✓**Signed 2's complement representation:**

Taking 2's complement of the positive number including the sign bit

8-bit binary signed 2's complement representation of $-(9)_{10}$ is

$(11110111)_2$

The Binary Number Addition

To add two 1-bit (representations of) **integers:** Count the number of ones in a column and write the result in binary.

The right bit of the result is placed under the column of bits. The left bit is called the "carry out of the column".

				1	1	1	1	0	0	0	0
0	0	1	1	1	0	1	0	1	0	1	0
0	1	0	1	1	1	0	0	1	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---
0	01	01	10	11	10	10	01	10	01	01	00

```
 +6        00000110
+13        00001101
---------  -----------------
+19        00010011
```

Binary Number Subtraction

Similar to decimal system using borrowing:
Subtract one binary number from another by using the standard techniques adapted for decimal numbers (subtraction of each bit pair, right to left, "borrowing" as needed from bits to the left).

Subtraction by adding with two's complement:

Represent negative binary numbers by using the "two's complement" method and a negative place-weight bit. Here, we'll use those negative binary numbers to subtract through addition.

Subtraction: $7_{10} - 5_{10}$
Addition equivalent: $7_{10} + (-5_{10})$

Represent $(+7)_{10}$ and $(-5)_{10}$ in binary 2's complement form

So, in 8-bit binary representation,
$+(7)_{10} = (00000111)_2$ and $(-5)_{10} = (11111011)_2$

Now, let's add them together:

```
 11111111 <--- Carry bits
  00000111
+ 11111011
  ------------
 100000010
 |
 Discard extra bit
```

.
. Answer = $(00000010)_2$

Since we've already defined our number bit field as three bits plus the negative-weight bit, the fifth bit in the answer (1) will be discarded to give us a result of 0010_2, or positive two, which is the correct answer.

Binary Codes

✓An <u>n-bit binary code </u>is a group of n bits that assumes <u>up to 2^n distinct combinations of 1’s and 0’s</u> with each combination representing one element of set that is being coded

✓The bit combination of an n-bit code is determined from the count in binary 0 to 2^n -1

Example:

A set of 8 elements *can be coded with* ***3-bit binary codes*** *(as $8 = 2^3$) with each element assigned one of the following bit combinations:*

000, 001, 010, 011, 100, 101, 110, 111

Nibble: In computers and digital technology, a nibble is four binary digits or half of an eight-bit byte.

Maximum number in a nibble :

$$(1111)_{bin} = (15)_{dec} = (F)_{hex}$$

A nibble can be conveniently represented by one hexadecimal digit.

Byte: A byte is eight binary digits (or bits)

One byte = 8 bits = 2 nibbles

Maximum number in a byte :

$$(11111111)_{bin} = (255)_{dec} = (FF)_{hex}$$

BCD: Binary-coded decimal

A method of using binary digits to represent the decimal digits 0 through 9. A decimal digit is represented by four binary digits, as shown below:

BCD		Decimal
0000	=	0
0001	=	1
0010	=	2
0011	=	3
0100	=	4
0101	=	5
0110	=	6
0111	=	7
1000	=	8
1001	=	9

The binary combinations 1010 to 1111 are invalid and are not used in BCD

BCD Conversion

BCD and binary are not the same.
For example, 49_{10} in binary is 110001_2,

but 49_{10} in BCD is 01001001_{BCD}

$$\begin{array}{cc} 4 & 9 \\ 0100 & 1001 = 01001001_{BCD} \end{array}$$

Each decimal digit is converted to its binary equivalent.

Similarly,

$(185)_{10} = (\ 0001\ 1000\ 0101)_{BCD} = (10111001)_2$

$(212)_{10} = (\ ?\ )_{BCD} = (?)_2$

BCD Addition

Example 1:

4	0100	*Represent in BCD*
+ 5	+0101	*Represent in BCD*
9	1001	***BCD SUM***

$(9)_{10} = (1001)_{BCD} = (1001)_2$

Example 2:

	1 carry	1 carry		
184	0001	1000	0100	*in BCD*
+ 576	+0101	0111	0110	*in BCD*
	0111	10000	1010	*Binary sum for each BCD*
		+0110	+0110	*Add 6 if sum is above 1001*
760	0111	0110	0000	***BCD SUM***

$(760)10 = (0111\ 0110\ 0000)_{BCD} = (1011111000)_2$

Other Decimal Codes

Weighted

Decimal Digit	BCD 8421	2421	8,4,-2,-1	Excess-3
0	0000	0000	0000	0011
1	0001	0001	0111	0100
2	0010	0010	0110	0101
3	0011	0011	0101	0110
4	0100	0100	0100	0111
5	0101	1011	1011	1000
6	0110	1100	1010	1001
7	0111	0111	1001	1010
8	1000	1110	1000	1011
9	1001	1111	1111	1100

Self Complementing

Gray Code: A Gray code is an encoding of numbers so that adjacent numbers have a single digit differing by 1

Decimal Number	Gray Code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

ASCII Code

ASCII, pronounced "ask-key", is the common code for microcomputer equipment. The **standard ASCII character** set consists of 128 decimal numbers ranging from zero through 127 assigned to letters, numbers, punctuation marks, and the most common special characters.

The **Extended ASCII Character Set** also consists of 128 decimal numbers and ranges from 128 through 255 representing additional special, mathematical, graphic, and foreign characters.

Error Detecting Code - Parity Coding

Parity bit: A parity bit is an extra bit included with a message to make the total number of 1's ***either even or odd***

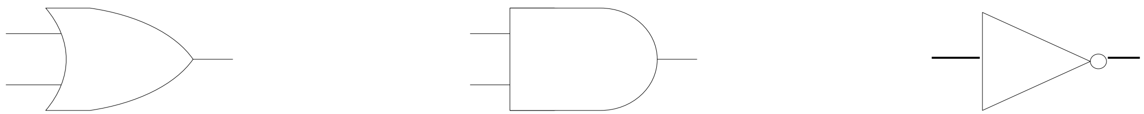
***Or**iginal message bits*	**With even parity**	**with odd parity**
1000001	01000001	11000001
1010100	11010100	01010100

Parity bit helps to detect error during the transmission of information

Boolean Algebra to Logic Gates

- Logic circuits are built from components called logic gates.

- The logic gates correspond to Boolean operations AND (*), OR (+), NOT (’)

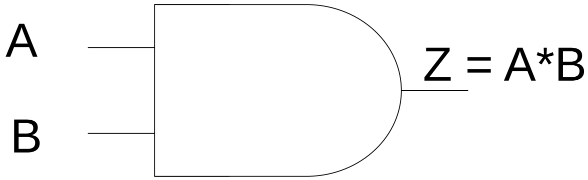


OR **+** **AND** ***** **NOT** **’**

- Binary operations have two inputs
- Unary has one input

AND, OR and NOT operations

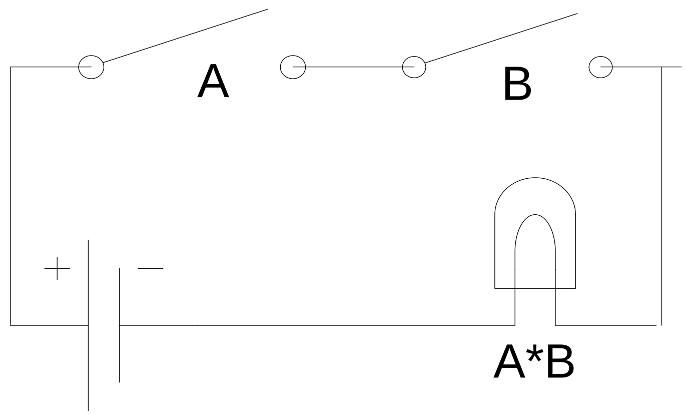
Logic Gate:

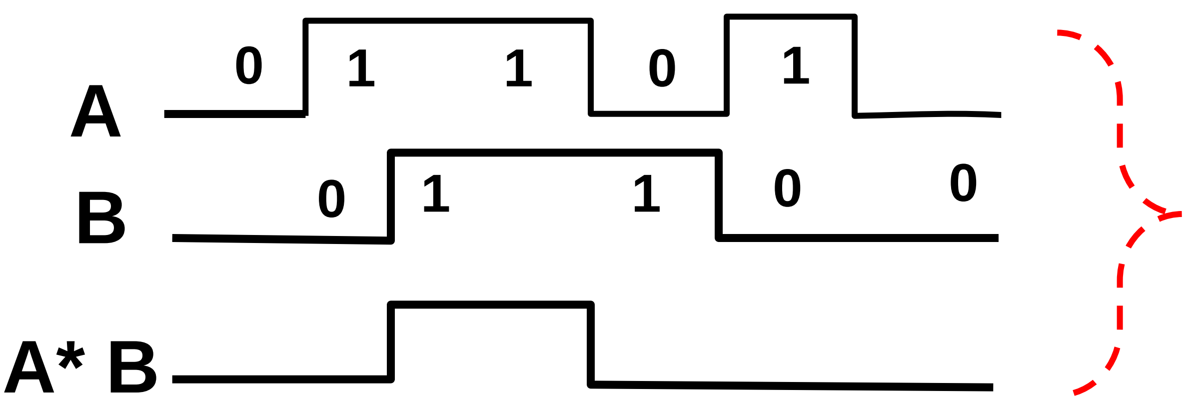


AND Logic Operation

A	B	A*B
0	0	0
0	1	0
1	0	0
1	1	1

Series Circuit:

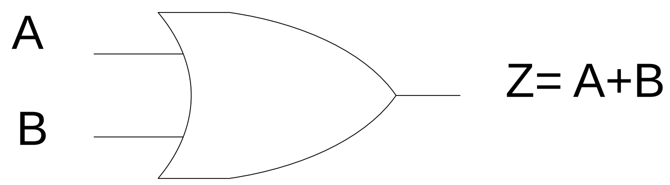




Input- output signals

AND, OR and NOT operations

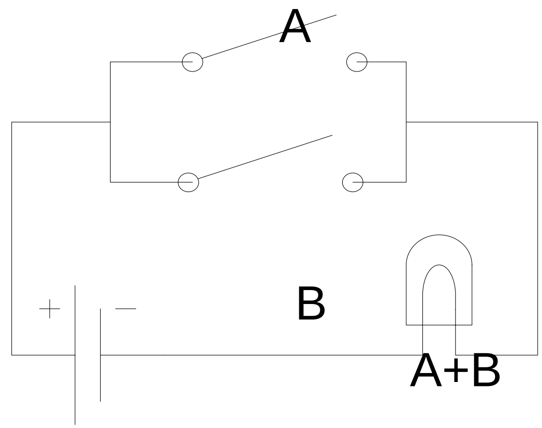
Logic Gate:

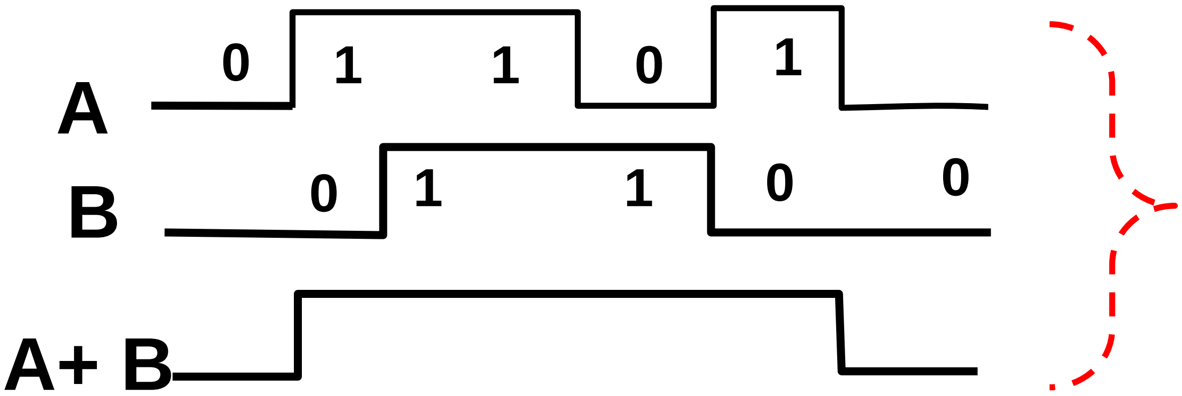


OR Logic Operation

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Parallel Circuit:

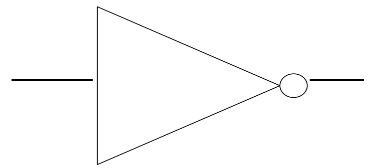




Input- output signals

AND, OR and NOT operations

Logic Gate: A
(also called an inverter)

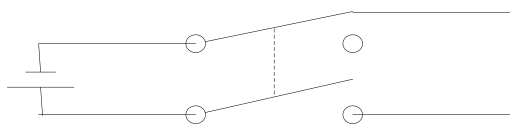


A’ or $\overline{A}$

NOT Logic operation

A	A’
0	1
1	0

Single-throw
Double-pole
Switch:

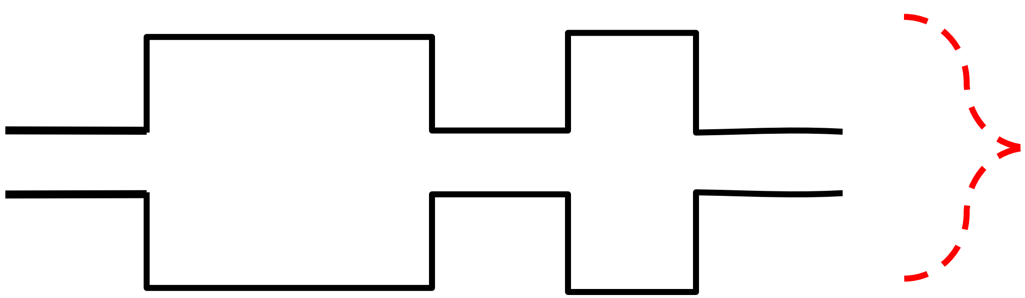


A

A’ or $\overline{A}$

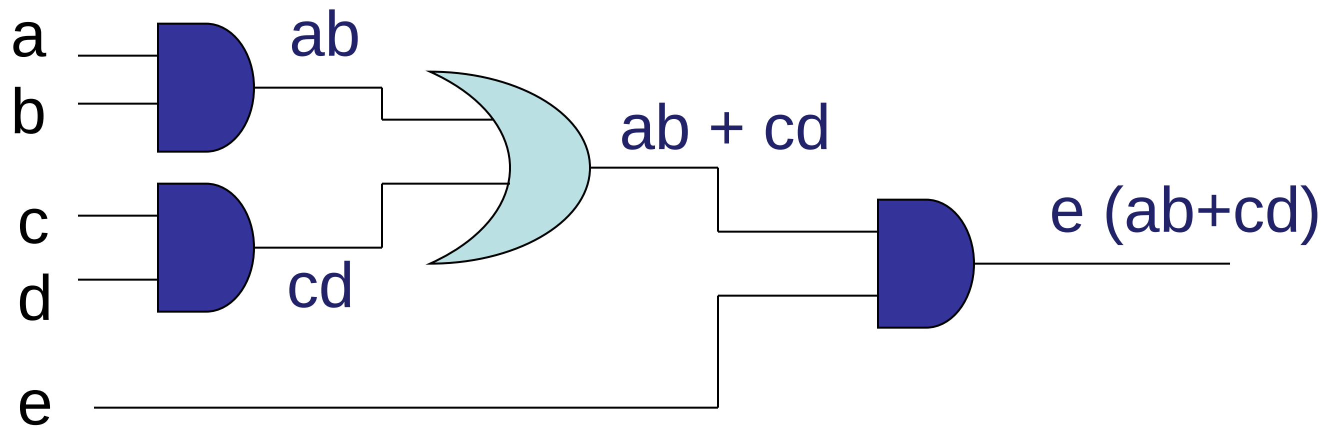
A

A’



Input- output signals

Combinational Logic vs Boolean Algebra



Schematic Diagram:

5 primary inputs

4 components

9 signal nets

12 pins

Boolean Algebra:

5 literals

4 operators

Some Definitions

- Complement: variable with a bar over it

  $\bar{A}$, $\bar{B}$, $\bar{C}$
- Literal: variable or its complement

  A, $\bar{A}$, B, $\bar{B}$, C, $\bar{C}$
- Implicant: product of literals

  $A\bar{B}C$, $\bar{A}C$, BC
- Minterm: product that includes all input variables

  $\bar{A}BC$, $A\bar{B}C$, $\bar{A}BC$
- Maxterm: sum that includes all input variables

  $(A+\bar{B}+C)$, $(\bar{A}+B+C)$, $(\bar{A}+\bar{B}+\bar{C})$

Axioms and Theorem in Boolean Algebra

	Axiom		Dual	Name
A1	$B = 0$ if $B \neq 1$	A1′	$B = 1$ if $B \neq 0$	Binary field
A2	$\overline{0} = 1$	A2′	$\overline{1} = 0$	NOT
A3	$0 \bullet 0 = 0$	A3′	$1 + 1 = 1$	AND/OR
A4	$1 \bullet 1 = 1$	A4′	$0 + 0 = 0$	AND/OR
A5	$0 \bullet 1 = 1 \bullet 0 = 0$	A5′	$1 + 0 = 0 + 1 = 1$	AND/OR

	Theorem		Dual	Name
T1	$B \bullet 1 = B$	T1′	$B + 0 = B$	Identity
T2	$B \bullet 0 = 0$	T2′	$B + 1 = 1$	Null Element
T3	$B \bullet B = B$	T3′	$B + B = B$	Idempotency
T4		$\overline{\overline{B}} = B$		Involution
T5	$B \bullet \overline{B} = 0$	T5′	$B + \overline{B} = 1$	Complements

Boolean Theorems: Summary

	Theorem		Dual	Name
T1	$B \bullet 1 = B$	T1′	$B + 0 = B$	Identity
T2	$B \bullet 0 = 0$	T2′	$B + 1 = 1$	Null Element
T3	$B \bullet B = B$	T3′	$B + B = B$	Idempotency
T4		$\overline{\overline{B}} = B$		Involution
T5	$B \bullet \overline{B} = 0$	T5′	$B + \overline{B} = 1$	Complements

Boolean Theorems of Several Variables

	Theorem		Dual	Name
T6	$B \bullet C = C \bullet B$	T6′	$B + C = C + B$	Commutativity
T7	$(B \bullet C) \bullet D = B \bullet (C \bullet D)$	T7′	$(B + C) + D = B + (C + D)$	Associativity
T8	$(B \bullet C) + B \bullet D = B \bullet (C + D)$	T8′	$(B + C) \bullet (B + D) = B + (C \bullet D)$	Distributivity
T9	$B \bullet (B + C) = B$	T9′	$B + (B \bullet C) = B$	Covering
T10	$(B \bullet C) + (B \bullet \overline{C}) = B$	T10′	$(B + C) \bullet (B + \overline{C}) = B$	Combining
T11	$(B \bullet C) + (\overline{B} \bullet D) + (C \bullet D)$ $= B \bullet C + \overline{B} \bullet D$	T11′	$(B + C) \bullet (\overline{B} + D) \bullet (C + D)$ $= (B + C) \bullet (\overline{B} + D)$	Consensus
T12	$\overline{B_0 \bullet B_1 \bullet B_2 ...}$ $= (\overline{B_0} + \overline{B_1} + \overline{B_2} ...)$	T12′	$\overline{B_0 + B_1 + B_2 ...}$ $= (\overline{B_0} \bullet \overline{B_1} \bullet \overline{B_2})$	De Morgan's Theorem

Minterms

- Minterms are AND terms with every variable present in either true or complemented form.
- Given that each binary variable may appear normal (e.g., x) or complemented (e.g., $\overline{x}$), there are 2^n minterms for n variables.
- Example: Two variables (X and Y) produce 2 x 2 = 4 combinations:

  XY (both normal)
  $X\overline{Y}$ (X normal, Y complemented)
  $\overline{X}Y$ (X complemented, Y normal)
  $\overline{X}\,\overline{Y}$ (both complemented)
- Thus there are four minterms of two variables.

Maxterms

- Maxterms are OR terms with every variable in true or complemented form.
- Given that each binary variable may appear normal (e.g., x) or complemented (e.g., $\overline{x}$), there are 2^n maxterms for n variables.
- Example: Two variables (X and Y) produce 2 x 2 = 4 combinations:

$\mathbf{X + Y}$ (both normal)

$\mathbf{X + \overline{Y}}$ (x normal, y complemented)

$\mathbf{\overline{X} + Y}$ (x complemented, y normal)

$\mathbf{\overline{X} + \overline{Y}}$ (both complemented)

Minterms

- Minterms are AND terms with every variable present in either true or complemented form.
- Given that each binary variable may appear normal (e.g., x) or complemented (e.g., $\overline{\mathbf{x}}$), there are 2^n minterms for n variables.
- Example: Two variables (X and Y) produce 2 x 2 = 4 combinations:

  $\mathbf{XY}$ (both normal)

  $\mathbf{X\overline{Y}}$ (X normal, Y complemented)

  $\mathbf{\overline{X}Y}$ (X complemented, Y normal)

  $\mathbf{\overline{X}\,\overline{Y}}$ (both complemented)

- Thus there are four minterms of two variables.

Maxterms

- Maxterms are OR terms with every variable in true or complemented form.
- Given that each binary variable may appear normal (e.g., x) or complemented (e.g., $\overline{x}$), there are 2^n maxterms for *n* variables.
- Example: Two variables (X and Y) produce 2 x 2 = 4 combinations:

$\mathbf{X} + \mathbf{Y}$ (both normal)

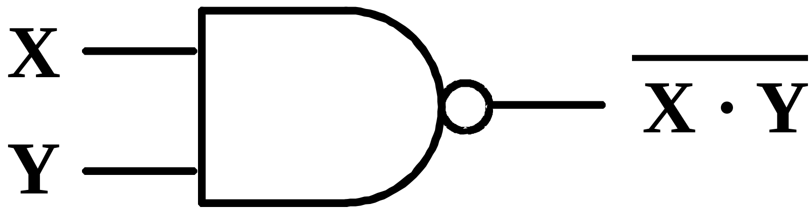
$\mathbf{X} + \overline{\mathbf{Y}}$ (x normal, y complemented)

$\overline{\mathbf{X}} + \mathbf{Y}$ (x complemented, y normal)

$\overline{\mathbf{X}} + \overline{\mathbf{Y}}$ (both complemented)

NAND Gate

- The basic NAND gate has the following symbol and truth table:
  - AND-Invert (NAND) Symbol:

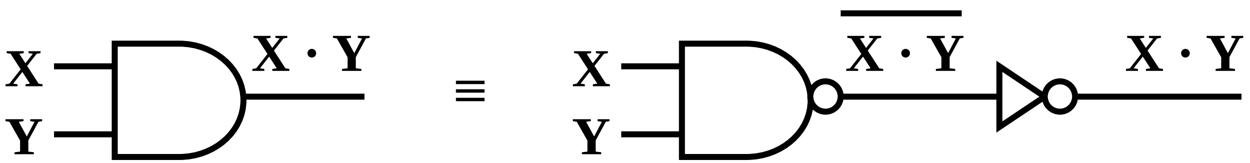


X	Y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

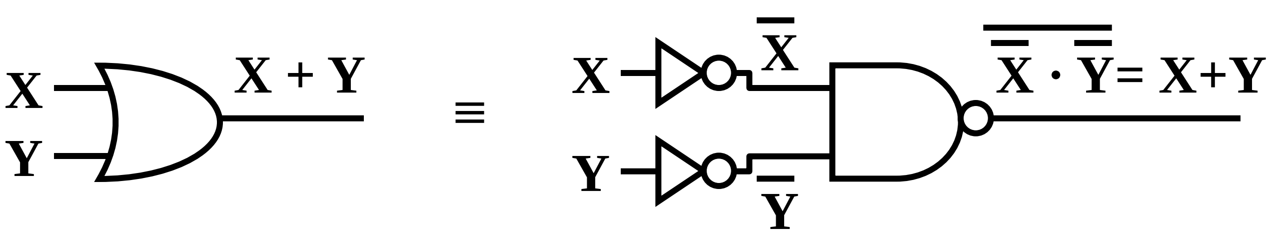
- NAND represents NOT AND. The small “bubble” circle represents the invert function
- The NAND gate is implemented efficiently in CMOS technology in terms of chip area and speed

The NAND Gate is Universal

- NAND gates can implement any Boolean function
- NAND gates can be used as inverters, or to implement AND / OR operations
- A NAND gate with one input is an inverter
- AND is equivalent to NAND with inverted output

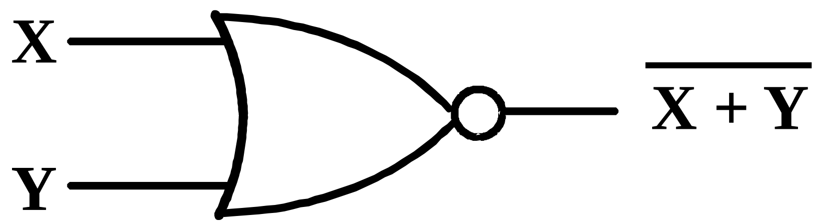


- OR is equivalent to NAND with inverted inputs



NOR Gate

- The basic NOR gate has the following symbol and truth table:
  - OR-Invert (NOR) Symbol:

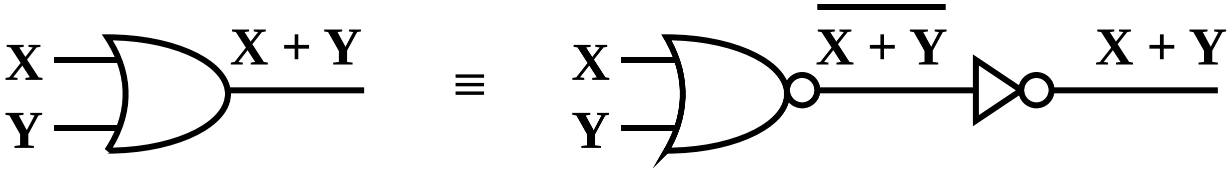


X	Y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

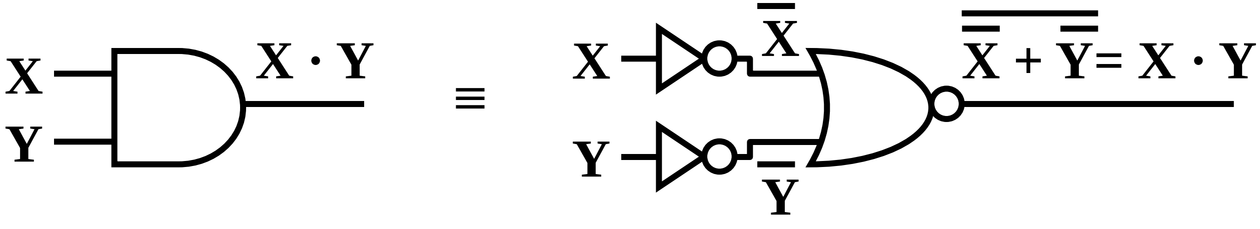
- NOR represents NOT OR. The small "bubble" circle represents the invert function.
- The NOR gate is also implemented efficiently in CMOS technology in terms of chip area and speed

The NOR Gate is also Universal

- NOR gates can implement any Boolean function
- NOR gates can be used as inverters, or to implement AND / OR operations
- A NOR gate with one input is an inverter
- OR is equivalent to NOR with inverted output



- AND is equivalent to NOR with inverted inputs

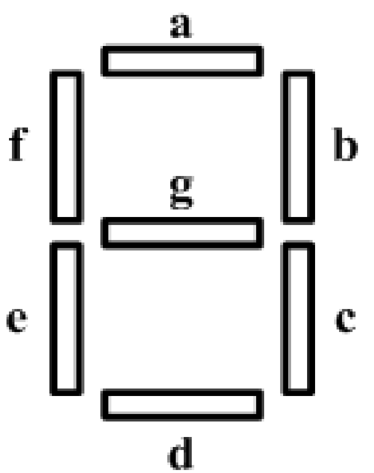


Uses for XOR / XNOR

- SOP Expressions for XOR/XNOR:
  - The XOR function is: $\mathbf{X \oplus Y = X\,\overline{Y} + \overline{X}\,Y}$
  - The eXclusive NOR (XNOR) function, know also as equivalence is: $\mathbf{\overline{X \oplus Y} = X\,Y + \overline{X}\,\overline{Y}}$
- Uses for the XOR and XNORs gate include:
  - Adders/subtractors/multipliers
  - Counters/incrementers/decrementers
  - Parity generators/checkers
- Strictly speaking, XOR and XNOR gates do no exist for more that two inputs. Instead, they are replaced by odd and even functions.

BCD-to-Seven-Segment Decoder

- **Specification**
  - Digital readouts on many digital products often use LED seven-segment displays.
  - Each digit is created by lighting the appropriate segments. The segments are labeled a,b,c,d,e,f,g
  - The decoder takes a BCD input and outputs the correct code for the seven-segment display.



- **Formulation**
  - Input: A 4-bit binary value that is a BCD coded input.
  - Outputs: 7 bits, a through g for each of the segments of the display.
  - Operation: Decode the input to activate the correct segments.

Formulation

- Construct a truth table

Decimal Digit	Input BCD	Seven-Segment Decoder Outputs a b c d e f g
0	0 0 0 0	1 1 1 1 1 1 0
1	0 0 0 1	0 1 1 0 0 0 0
2	0 0 1 0	1 1 0 1 1 0 1
3	0 0 1 1	1 1 1 1 0 0 1
4	0 1 0 0	1 0 1 1 0 1 1
5	0 1 0 1	1 0 1 1 0 1 1
6	0 1 1 0	1 0 1 1 1 1 1
7	0 1 1 1	1 1 1 0 0 0 0
8	1 0 0 0	1 1 1 1 1 1 1
9	1 0 0 1	1 1 1 1 0 1 1
All other inputs		0 0 0 0 0 0 0

Optimization

- Create a K-map for each output and get

  $a = A'C+A'BD+B'C'D'+AB'C'$

  $b = A'B'+A'C'D'+A'CD+AB'C'$

  $c = A'B+A'D+B'C'D'+AB'C'$

  $d = A'CD'+A'B'C+B'C'D'+AB'C'+A'BC'D$

  $e = A'CD'+B'C'D'$

  $f = A'BC'+A'C'D'+A'BD'+AB'C'$

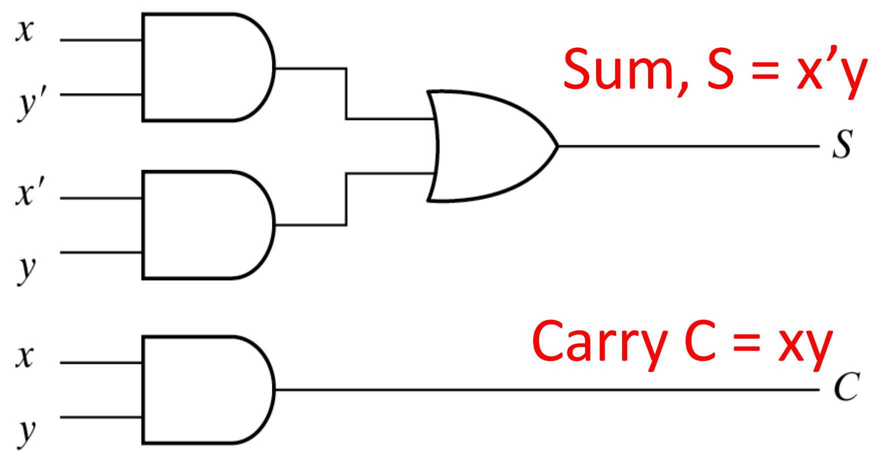
  $g = A'CD'+A'B'C+A'BC'+AB'C'$

Binary Adder-Subtractor

- A combinational circuit that performs the addition of two bits is called a half adder.
- The truth table for the half adder is listed below:

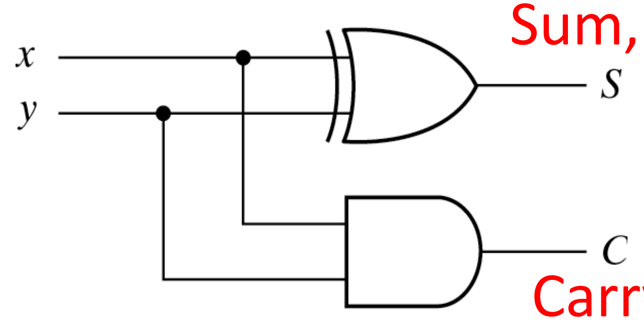
Truth Table – Half Adder

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Sum, S = x’y + xy’

Carry C = xy



Sum, S = x ⊕ y

Carry C = xy

Full-Adder

- Performs the addition of three bits (two significant bits and a previous carry)

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S

X \ YZ	00	01	11	10
0		1		1
1	1		1	

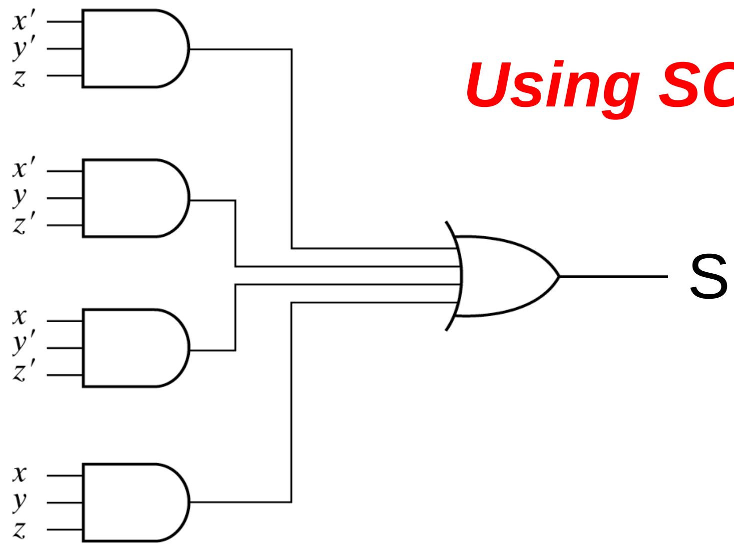
$$S = X'Y'Z + X'YZ' + XY'Z' + XYZ$$
$$= X \oplus Y \oplus Z$$
$$= (X \oplus Y) \oplus Z$$

C

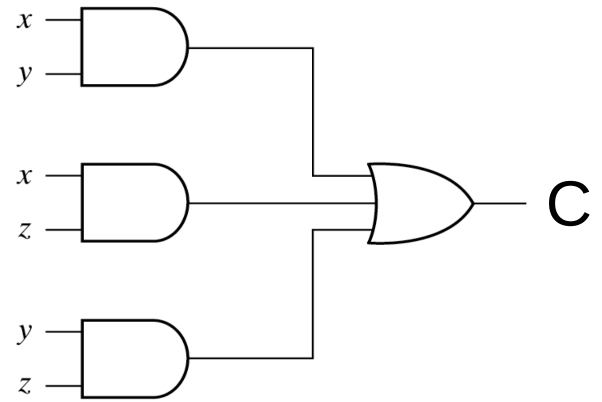
X \ YZ	00	01	11	10
0			1	
1		1	1	1

$$C = XY + XY'Z + X'YZ$$
$$= XY + Z(XY' + X'Y)$$
$$= XY + Z(X \oplus Y)$$

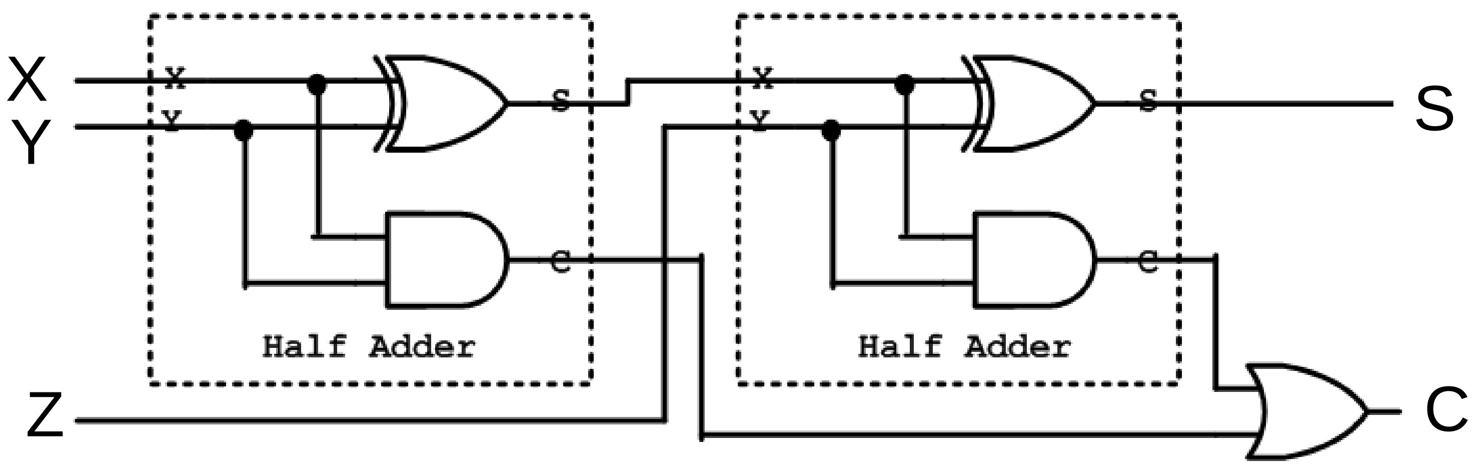
Full adder Implementation



Using SOP

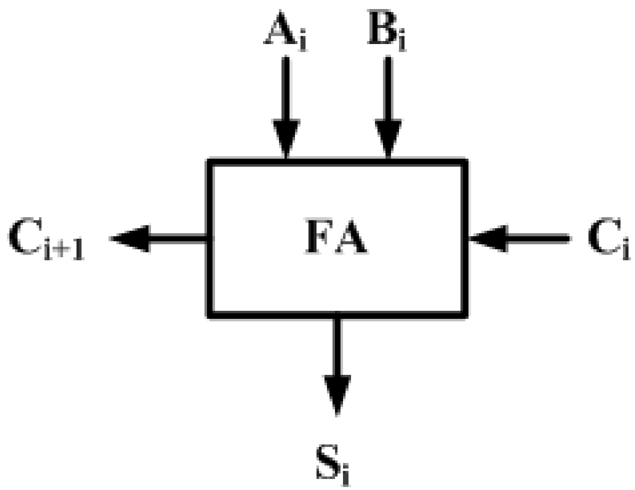


Using two half adders and one OR gate (Carry Look-Ahead a



Full Adder Symbol

- For a multibit implementation need a symbol for the unit. And then can use that symbol in multi-bit or hierarchical representations.



Binary adder

- Ripple Carry Adder (RCA): full adders are connected in cascade.
- All inputs, As, Bs, and C_0 arrive –> C_1 becomes valid –> C_2 becomes valid –> C_3 becomes valid –> C_4 becomes valid

Subscript i:	**3**	**2**	**1**	**0**	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}