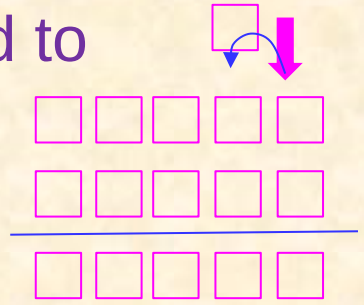
2 Binary Addition and Subtraction

- Binary Addition
- Binary Subtraction
- Adder (Adding Circuit)

Binary Addition

Adding binary numbers:

- Use the same principle as adding decimal numbers;
- Consider the numbers that will be added to bit by bit;
- Adding multiple-bit numbers:
  - start adding from the rightmost (LSB), first;
  - if the carry is occurred, it would be added to the next bit on the left;
  - keep doing this until to the end.
  - The carry of the last bit (if any) will be the leftmost bit of the result.



Binary Addition

In addition of TWO binary numbers,
AUGEND and ADDEND, considered as the inputs,
Which produce the output of SUM and CARRY.
All possible cases of the addition are:

INPUT			OUTPUT	
AUGEND		ADDEND	SUM	CARRY
0	+	0	0	0
0	+	1	1	0
1	+	0	1	0
1	+	1	0	1

Binary Addition

Example: Find the results from adding 11110_2 and 1100_2

CARRY	**1**	**1**	**1**	**0**	**0**	**x**	Check
AUGEND		1	1	1	1	0	30_{10}
ADDEND			1	1	0	0	+ 12_{10}
SUM	**1**	**0**	**1**	**0**	**1**	**0**	**42_{10}**

Binary Addition

From examples above,

there are two types of the binary number addition:

1. Addition between the Augend and Addend at the LSB bit. This type of addition is called **Half Addition**.

2. Addition between the Augend and the Addend, including with the Carry, such as adding from the second digit to the the far left, we call this addition as **Full Addition**.

Binary Addition: Half - Addition

$$\begin{array}{rr} & A \\ + & B \\ \hline C_o & S \end{array}$$

Input		Output	
A (Augend)	B (Addend)	C_O (Carry)	S (Sum)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Binary Addition: Full - Addition

C_{in}

A

\+ B

C_o Sum

Input			Output	
C_{in}	A	B	C_o	Sum
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

Binary Subtraction

- Subtracting binary numbers uses the same principles of decimal digits.
- Subtracting multiple digits would subtract one-by-one, beginning with the **rightmost** digit.
- If the Subtrahend is greater than the Minuend, it is necessary to borrow from the next digit.
- When a borrowing occurred, the borrower must be subtracted from the next Minuend before being subtracted by the next Subtrahend.
- Continue doing this for the rest digits.

Binary Subtraction

In subtraction of TWO binary numbers,

Minuend and Subtrahend, considered as the inputs

And produce the output of Difference and Borrower.

All possible cases of the subtraction are:

INPUT			OUTPUT	
Minuend		Subtrahend	Difference	Borrower
0	-	0	0	0
0	-	1	1	1
1	-	0	1	0
1	-	1	0	0

Binary Subtraction

Example: Subtract 1001_2 from 10011_2

Borrower	1	0	0	0	x
Minuend	~~1~~	0	0	1	1
Subtrahend	0	1	0	0	1
Difference	0	1	0	1	0

Check

19_{10}

$-\ 9_{10}$

10_{10}

Binary Subtraction

From examples above,

there are two types of the binary number subtraction:

1. Subtraction between the Minuend and Subtrahend at the LSB digit. This type of subtraction is called

   Half Subtraction.

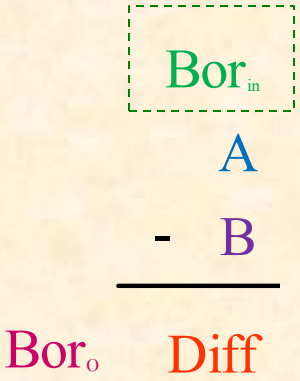
2. Subtraction between the Minuend and the Subtrahend, including with the Borrower, such as subtraction from the second digit to the the far left, we call this subtraction as **Full Subtraction**.

Binary Subtraction: Half - Subtraction

$$\begin{array}{rr} & A \\ - & B \\ \hline B_O & Diff \end{array}$$

Input		Output	
A Minuend	B Subtrahend	B_O Borrower	Diff Difference
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Binary Subtraction: Full- Subtraction



Input			Output	
Bor_i	A	B	Bor_o	Diff
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

Binary Subtraction using Complement

There are two ways to subtract binary numbers.

- **Direct Subtraction:**

Subtracting the Subtrahend directly from the Minuend.

- **Subtraction using a Complement:**

adding the complement of the Subtrahend to the Minuend.

Complements

Complement

A number *N* in base *r* having *n* digits,
can have two types of **Complements** :

- (*r* - 1)'s complement = $(r^n - 1) - N = Max - N$
- *r*'s complement = (*r* - 1) complement + 1

Example: Find the complements of 1234_{10}

For decimal numbers, *r* = 10, *n* = 4 and *r* – 1 = 9,

So, the 9's complement of $1234_{10} = (10^4 - 1) - 1234$
$= 9999 - 1234 = 8765.$

And the 10's complement of $1234_{10} = 8765 + 1$
$= 8766.$

Example: Find the complements of $1101_2 = 13_{10}$

For binary numbers, *r* = 2, *n* = 4 and *r* – 1 = 1,

So, the 1's complement of $1101_2 = (2^4 - 1) - 13 = 15 - 13$
$= 2_{10} = 0010_2$

And the 2's complement of $1101_2 = 0010_2 + 1 = 0011_2$

1's Complement

From the previous Example:

the 1's complement of $1101_2 = 0010_2 = (1101)'$

So, the 1's complement of *N* is the invert of *N*.

And the 2's complement of *N* is the (invert of *N*) + 1 = 0011_2

Example: Find the 1's and 2's complement of 1011101_2

- The 1's complement of 1011101_2
  is $(1011101_2)' = 0100010_2$
- The 2's complement of 1011101_2
  is $0100010_2 + 1 = 0100011_2$

2's Complement
of 1011101
is 0100011

2's Complement

Example: Find the 1's and 2's complement of 10100_2

- The 1's complement of 10100_2

  is $(10100_2)' = 01011_2$

- The 2's complement of 10100_2

  is $01011_2 + 1 = 01100_2$

2's Complement

of 10100

is 01100

2's complement of 101100_2 = ……..

2's complement of 11010_2 = ……..

Binary Subtraction using 1's Complement

Steps for subtracting binary numbers using the 1's complement

1. Find the its 1's complement of subtrahend;

2. Add it to the minuend then consider if the result produces:

2.1 End-around carry (EAC) at the leftmost digit:

➔ Indicates that the answer is **positive**,

➔ The answer by adding EAC to the result.

2.2 No EAC at the leftmost digit:

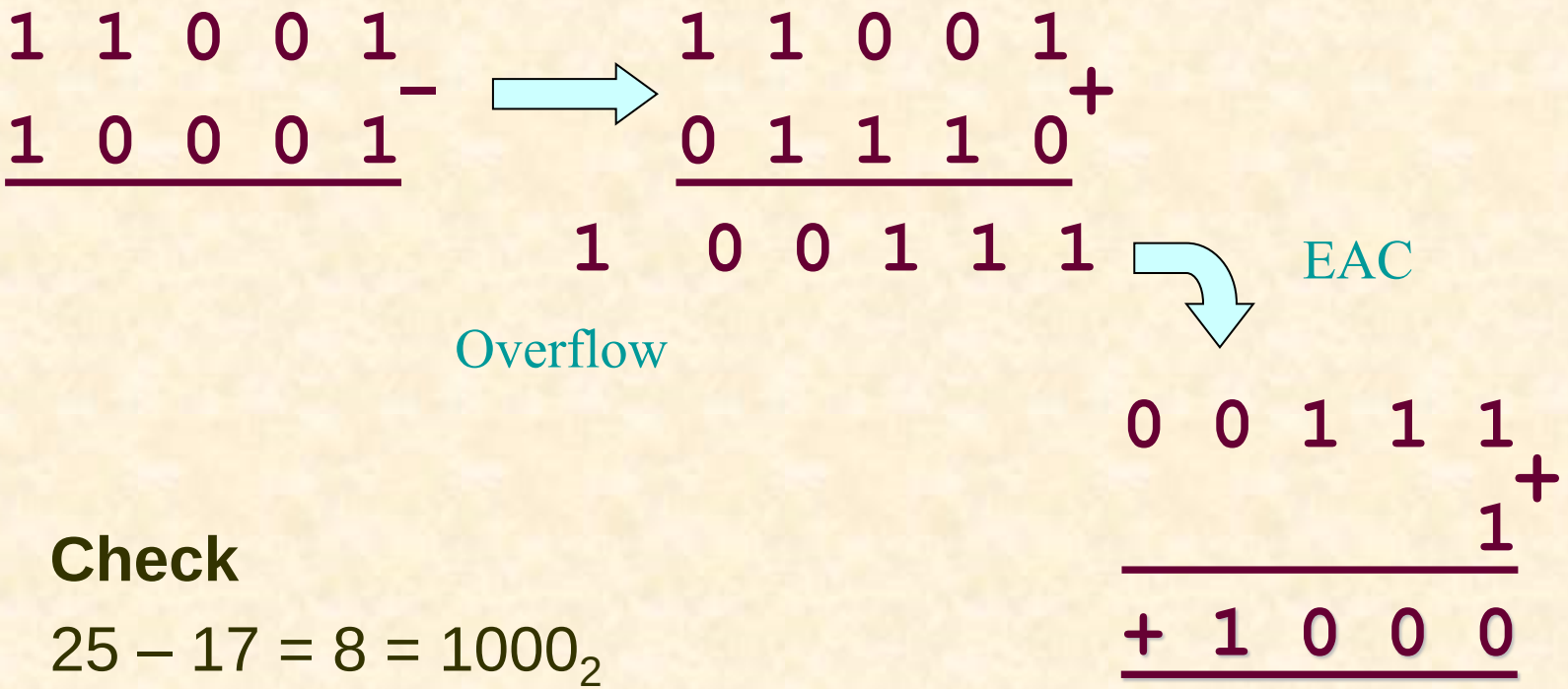
➔ Indicates that the answer is **negative**,

➔ The answer is the 1's complement of the result.

Binary Subtraction using 1's Complement

Example: Find $11001_2 - 10001_2$ using 1's Complement

The 1's complement of $10001_2 = (10001_2)' = 01110_2$



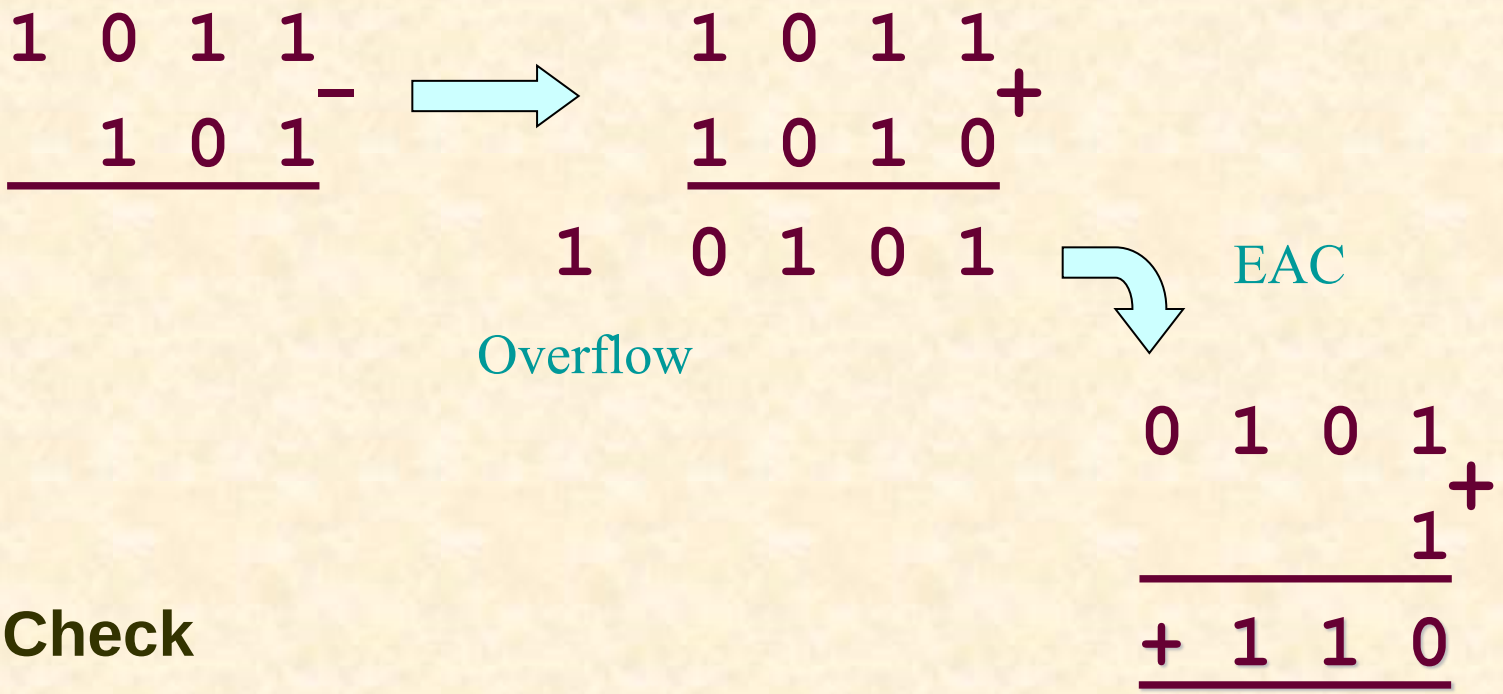
Check

$25 - 17 = 8 = 1000_2$

Binary Subtraction using 1's Complement

Example: Find $1011_2 - 101_2$ using 1's Complement

The 1's complement of $101_2 = (0101_2)' = 1010_2$



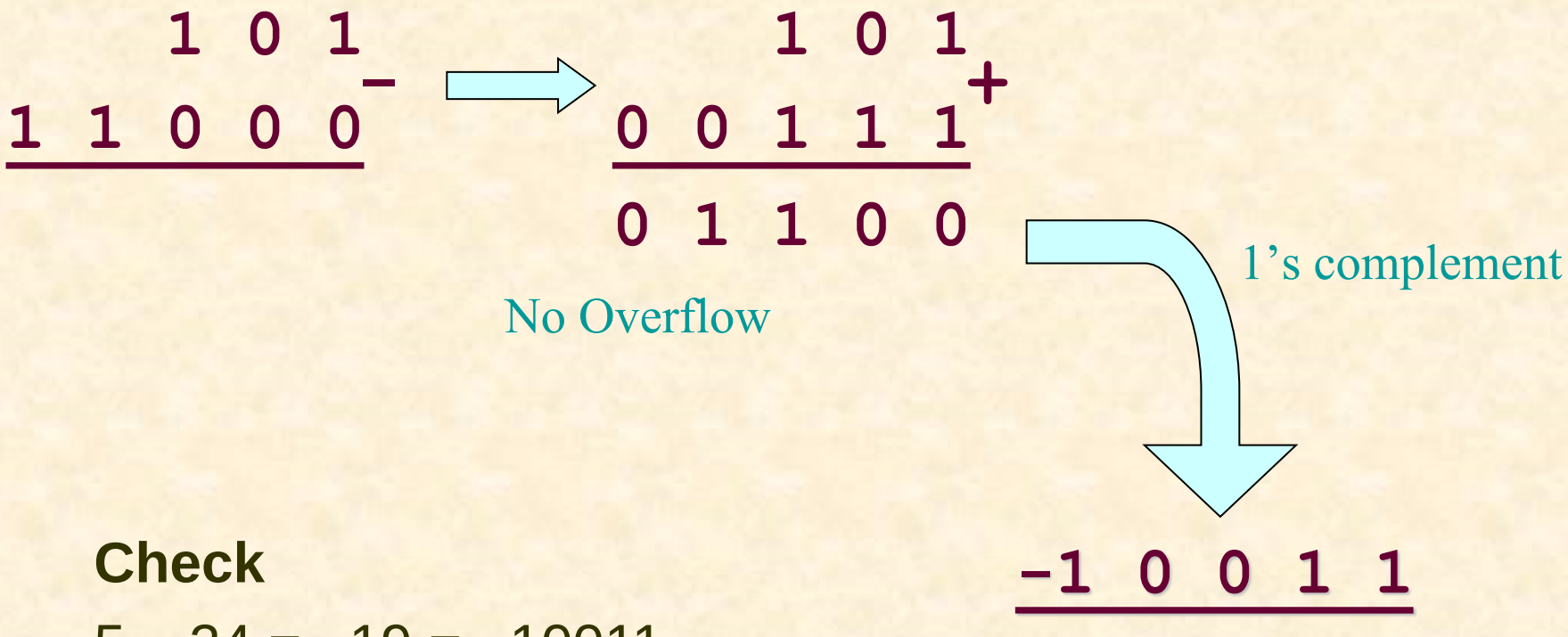
Check

$11 - 5 = 6 = 110_2$

Binary Subtraction using 1's Complement

Example: Find $101_2 - 11000_2$ using 1's Complement

The 1's complement of $11000_2 = (11000_2)' = 00111_2$



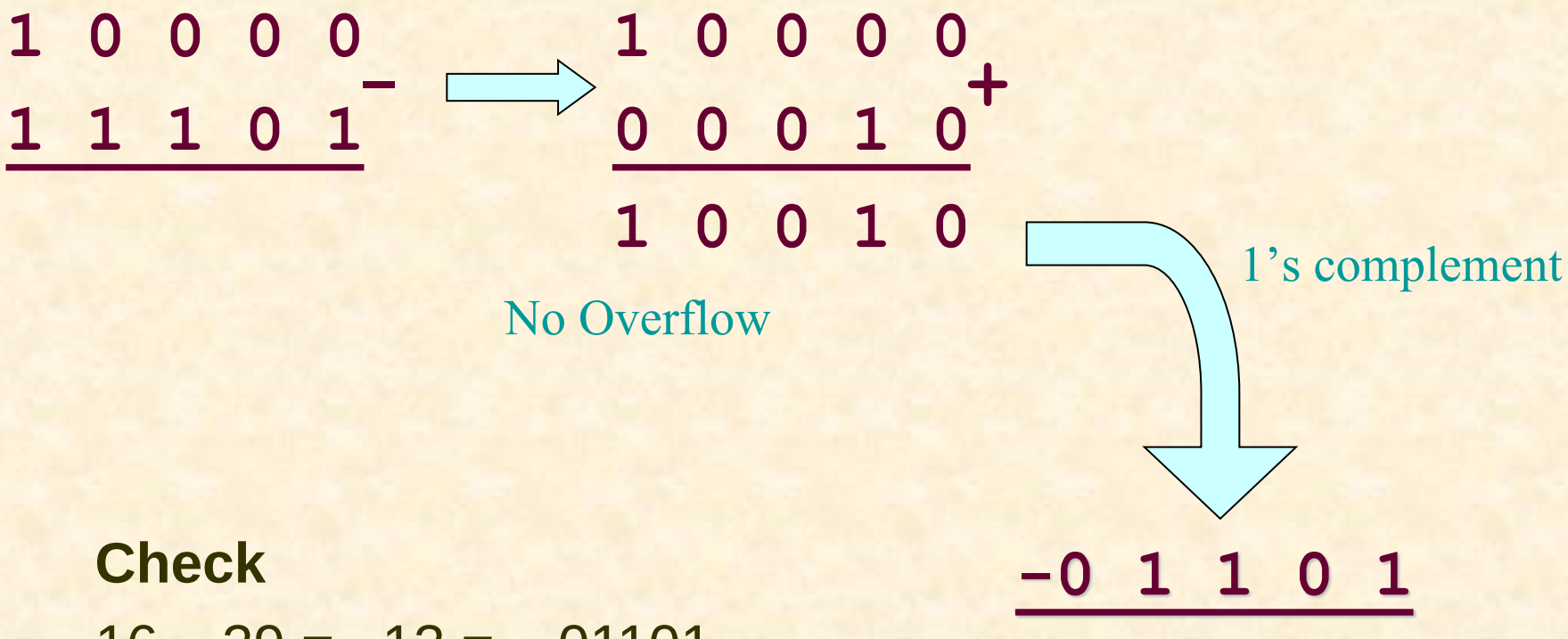
Check

5 – 24 = –19 = –10011

Binary Subtraction using 1's Complement

Example: Find $10000_2 - 11101_2$ using 1's Complement

The 1's complement of $11101_2 = (11101_2)' = 00010_2$



Check

16 – 29 = –13 = – 01101

Binary Subtraction using 2's Complement

Steps for subtracting binary numbers using the 2's complement

1. Find the its 2's complement of subtrahend;

2. Add it to the minuend then consider if the result produces:

2.1 End-around carry (EAC) at the leftmost digit:

➔ Indicates that the answer is **positive**,

➔ The answer is the obtained result.

2.2 No EAC at the leftmost digit:

➔ Indicates that the answer is **negative**,

➔ The answer is the 2's complement of the result.

Binary Subtraction using 2's Complement

Example: Find $1011_2 - 100_2$ using 2's Complement

The 2's complement of 100_2 = $(0100_2)' + 1 = 1100_2$

```
  1 0 1 1          1 0 1 1
-   1 0 0      +   1 1 0 0
-------------    -----------
                   1 0 1 1 1
```

Overflow

Ignore the Overflow

Check

$11 - 4 = 7 = 0111_2$

```
  1 0 1 1
+ 0 1 1 1
---------
```

Binary Subtraction using 2's Complement

Example: Find $10010_2 - 11000_2$ using 2's Complement

The 2's complement of 100_2 = **(**11000_2**)′ + 1** = 01000_2

```
  1 0 0 1 0          1 0 0 1 0
- 1 1 0 0 0        + 0 1 0 0 0
-----------        -----------
                     1 1 0 1 0
```

No Overflow

2's complement

```
  0 0 1 0 1
            +
        1
-----------
-   1 1 0
-----------
```

Check

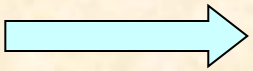
$18 - 24 = -6 = -110_2$

Binary Subtraction using 2's Complement

Example: Find $1001_2 - 10101_2$ using 2's Complement

The 2's complement of 100_2 = $(10101_2)' + 1 = 01011_2$

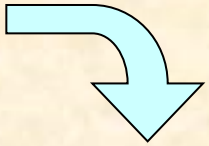
```
  1 0 0 1
- 1 0 1 0 1
-----------
```



```
    1 0 0 1
+   0 1 0 1 1
-------------
  1 0 1 0 0
```

No Overflow

2's complement



```
  0 1 0 1 1
+         1
-----------
- 1 1 0 0
-----------
```

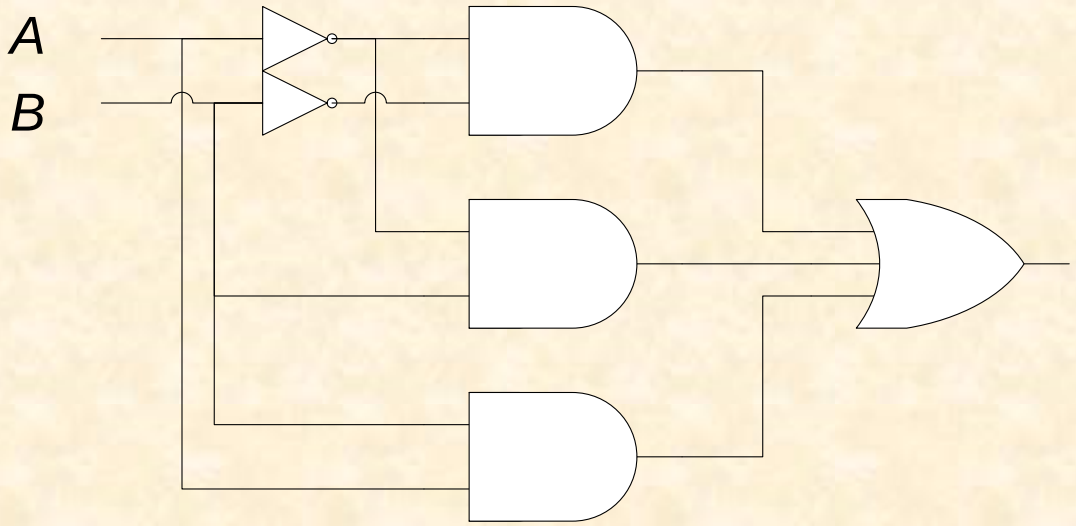
Check

$9 - 21 = -12 = -1100_2$

The Relationship between Boolean Function and Circuit

A logical circuit can be described its operation by a logical equation.

e.g. for the circuit that are made up of multiple gates below,



Analysis

It can be expressed as: $Y = \overline{A}\ \overline{B} + \overline{A}\ B + A\ B$

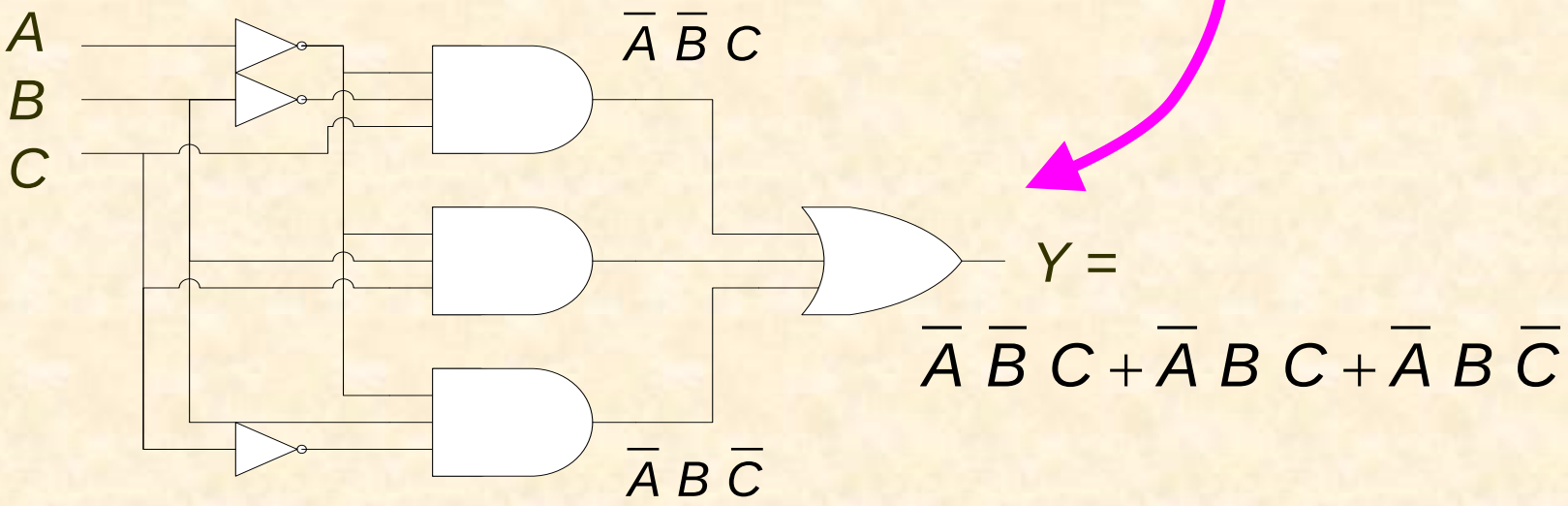
The Relationship between Boolean Function and Circuit

In contrast, a logical equation can be brought to make a circuit.

For example, $Y = \overline{A}\ \overline{B}\ C + \overline{A}\ B\ C + \overline{A}\ B\ \overline{C}$

Can be converted into a circuit as follows:

Design



Logic Circuit Analysis and Design

Two types of electronic circuit study:

- Circuit analysis
- Circuit design

Logic Circuit Analysis

Circuit analysis describes the circuit behavior.

It may be presented by

- the output function / EQUATION
- the TRUTH TABLE of that circuit.

The output equation may be considered by

the output from the gates at the input (on the LEFT),

stage by stage until the last stage of the output (on the RIGHT)

The truth table is described by a table that contains

ALL possible conditions of the inputs, and corresponding outputs.

Logic Circuit Design

Circuit design is to make a logic circuit to work as the function described, or the truth table assigned.

Here are the steps for the circuit design:

1. Defining the truth table;
2. Write the Boolean Equation from the truth table;
3. Minimize the logic terms in the SOP configuration;
4. Then create circuits from the obtained equation.

Truth Table and Logic Equation

- Logic equation
- Truth table

} can be used to represent the operation of a logic circuit.

- Sometimes we may firstly have the logic equation, which can produce the truth tables, logically; [Straight forward]
- Sometimes we may firstly have the truth table, which has to be changed into the equation. [How?]

How to find the logic equation from the truth table ?

Writing Logic Equation from Truth Table

There are **TWO** basic methods for writing the logic equation from the truth table :

- **Sum of Product (SOP):** Collect all conditions in terms of AND that providing the outputs of '1' to the output OR gate;

such as $Y = AB + BC + ABC$

- **Product of Sum (POS):** Collect all conditions in terms of OR that providing the outputs of '0' to the output AND gate;

such as $Y = (A + B)\,(B + C)\,(A + B + C)$

Obviously, **SOP** configuration is more compact and easier, then more popular than **POS**.

Example of Logic Equation from a Truth Table

Sum of Product (SOP)

I/P		O/P
A	***B***	**Min Term**
0	0	$m_0 = \overline{A}\,\overline{B}$
0	1	$m_1 = \overline{A}\,B$
1	0	$m_2 = A\,\overline{B}$
1	1	$m_3 = A\,B$

I/P		O/P
A	*B*	*Y*
0	0	0
0	1	0
1	0	1
1	1	1

$$Y = AB + AB'$$

Product of Sum (POS):

I/P		O/P
A	***B***	**Max Term**
0	0	$M_0 = A + B$
0	1	$M_1 = A + \overline{B}$
1	0	$M_2 = \overline{A} + B$
1	1	$M_3 = \overline{A} + \overline{B}$

$$Y = (A + B)\cdot(A + B')$$
$$= (AA + AB + AB + BB')$$
$$= AB + AB'$$

Sum of Product [1]

Sum of Product is a method for finding equations from truth table.

in the form of OR (Sum) of the AND (Product) terms,

e.g. $Y = AB + \overline{A}CD.$

How to express the equation from the Truth Table:

- Collect all input conditions in terms of AND (Product), which provide the outputs of '1', to the OR (Sum) gate at the output;
- Rewrite the products in term of the logic variables if being '1', and invert (or bar) if being '0'.
- The product of these variables is called **minterms**.
- Finally, the Output can be expressed as the sum (OR) of all **minterms** (Products). For example, $Y = AB + \overline{A}CD.$

Sum of Product [2]

Example 1: Express the Boolean Equation from the truth table.

I/P		O/P
A	*B*	*Y*
0	0	0
0	1	0
1	0	1
1	1	1

$$Y = A\overline{B} + AB$$

Sum of Product [3]

Example 2: Express the Boolean Equation from the truth table.

Input			Output
C	*B*	*A*	*Y*
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

$$Y = \bar{C}BA + CB\bar{A} + CBA$$

Sum of Product [4]

Example 3: Express the Boolean Equation from the truth table.

Input			Output	
C	*B*	*A*	*X*	*Y*
0	0	0	1	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	1
1	0	0	1	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

$$X = \bar{C}\bar{B}\bar{A} + \bar{C}\bar{B}A + \bar{C}B\bar{A} + C\bar{B}\bar{A} + CB\bar{A} + CBA$$

$$Y = \bar{C}B\bar{A} + \bar{C}BA + C\bar{B}A + CB\bar{A} + CBA$$

Product of Sum

Product of Sum is a method for finding equations from truth table.

in the form of AND (Product) of the OR (Sum) terms,

e.g. $Y = (A + B)\cdot(\overline{A} + C + D)$.

How to express the equation from the Truth Table:

- Collect all input conditions in terms of OR (Sum), which provide the outputs of '0', to the AND (Product) gate at the output;
- Rewrite the Sums in term of the logic variables if being '0', and invert (or bar) if being '1'.
- The sum of these variables is called **maxterms**.
- Finally, Output can be expressed as the product (AND) of all maxterms (Sums). For example, $Y = (A + B)\cdot(\overline{A} + C + D)$.

Writing Truth Table from Logic Equation

This method is the opposite to write the equation from the truth table.

It starts by looking at the form of the equation whether it is either SOP or POS.

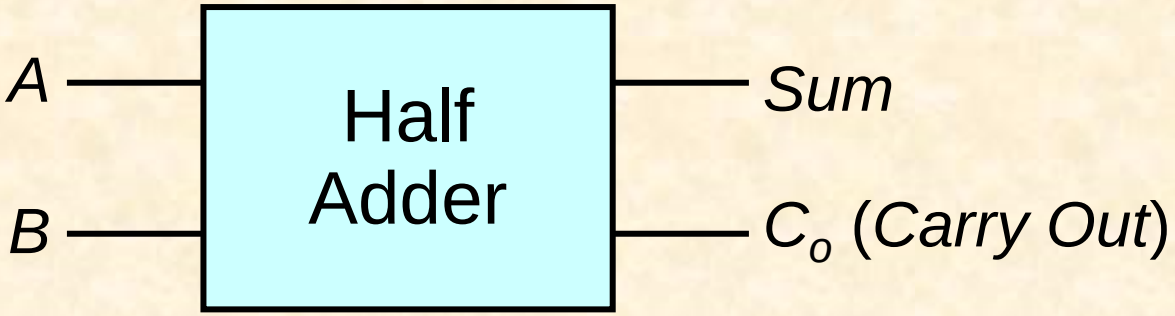
- If the equation is in the form of POS, each term would have an output of 0 in the truth table.
- If the equation is in the form of SOP, each term would have an output of 1 in the truth table.

Creating A Circuit from A Truth Table

- The truth table need to be converted into a Boolean equation,
- By collecting the minterms of the outputs of '1',
- Then write the logical equation in the form of SOP;
- Then draw the circuit corresponding to the final equation,
- Start from the inputs (A, B, C, etc) on the left,
- Put the signals to the individual AND gates to produce the minterm;
- Then combine all the minterms by the OR gate on the rightmost.

Let's try making circuits for Example 1 & 2

Half-Adder Design [1]



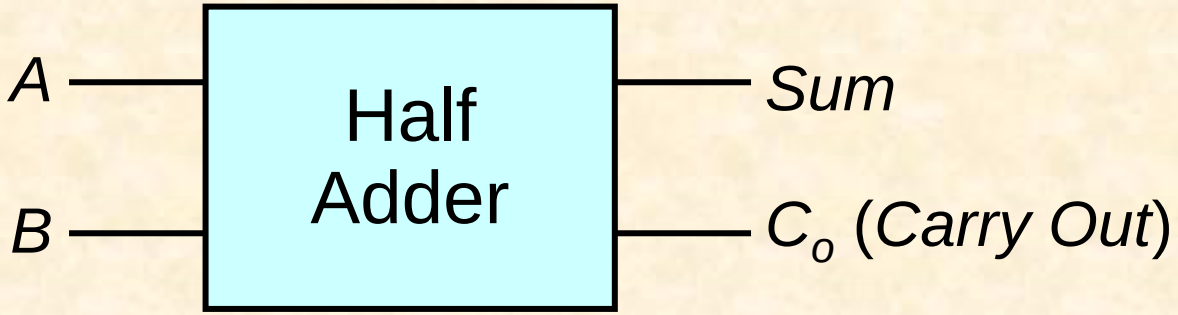
Input		Output	
A	*B*	*Sum*	C_o
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half-Adder Design [2]

- From the truth table obtained, we can consider the Outputs:
  - The Sum has the same function as the EX-OR gate and
  - The Carry has the same function as the AND gate.

- Therefore, we can write both circuits with the same inputs, *A* and *B,* as shown the next page..
- This method is to create a circuit by considering the Output in the truth table and finding the appropriate equations.
- This can be done in the case of a simple circuit.

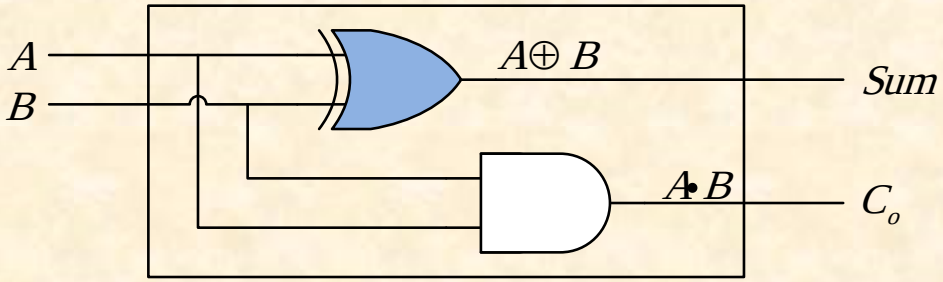
Half-Adder Design [1]

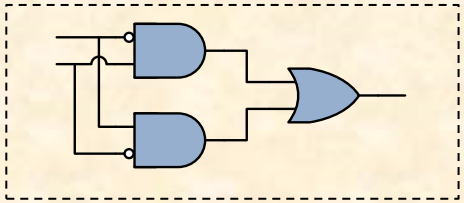


Input		Output	
A	*B*	C_O	*Sum*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

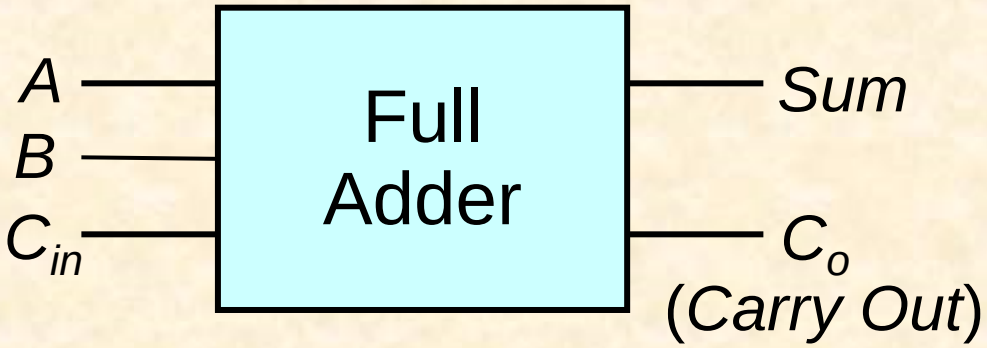
$$Sum = \overline{A}B + A\overline{B} = A \oplus B$$

$$C_o = A \bullet B$$





Full-Adder Design [1]



Input			Output	
C_{in}	A	B	C_o	Sum
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

$$\begin{array}{rr} & C_{in} \\ & A \\ + & B \\ \hline C_o & Sum \end{array}$$

Full-Adder Design [2]

$$Sum = A\bar{B}\overline{C_{in}} + \bar{A}B\overline{C_{in}} + \bar{A}\bar{B}C_{in} + ABC_{in}$$

$$= \left(A\bar{B} + \bar{A}B\right)\overline{C_{in}} + \left(\overline{A\bar{B}} + AB\right)C_{in}$$

$$= \left(A \oplus B\right)\overline{C_{in}} + \left(\overline{A \oplus B}\right)C_{in}$$

$$= (A \oplus B) \oplus C_{in}$$

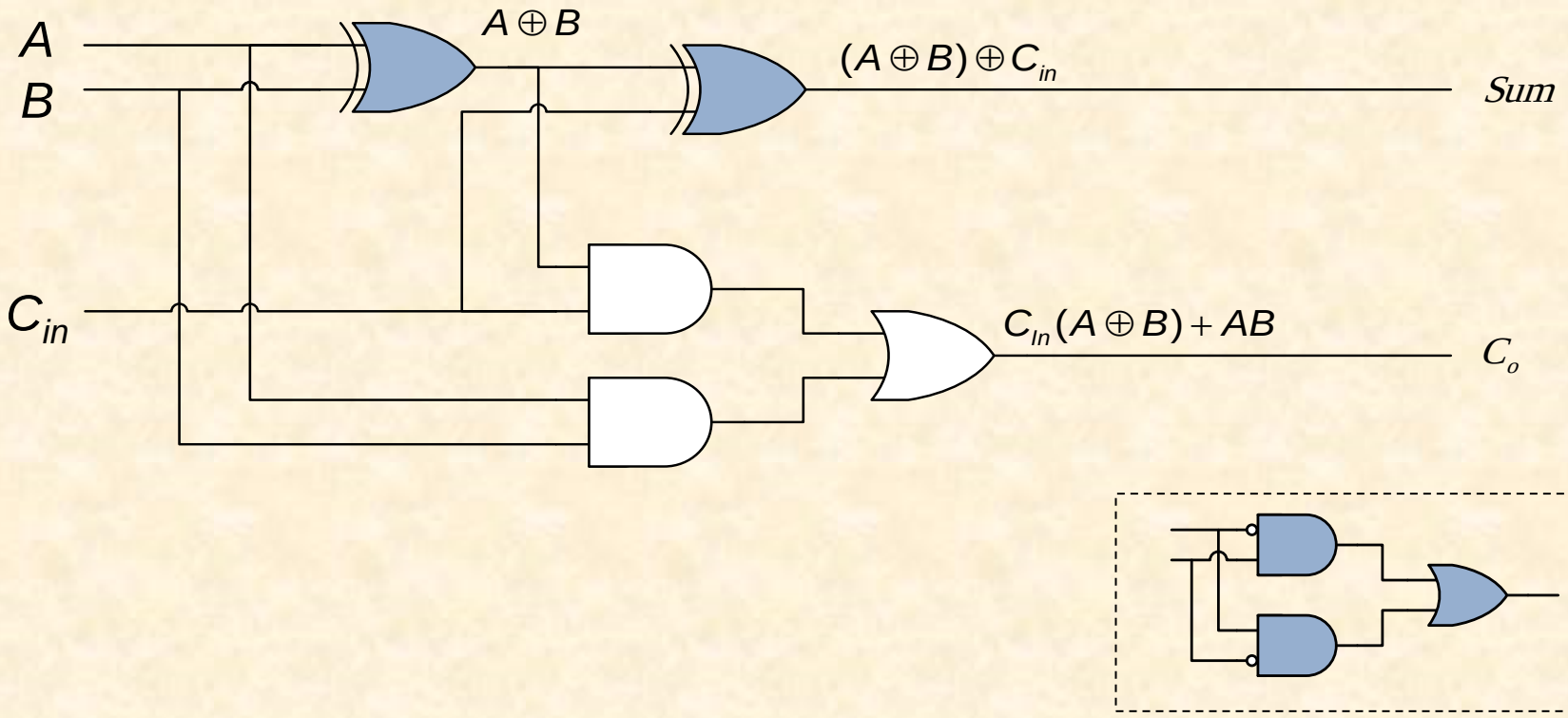
$$C_O = \bar{A}BC_{In} + A\bar{B}C_{In} + AB\overline{C_{In}} + ABC_{In}$$

$$= C_{In}(\bar{A}B + A\bar{B}) + AB(\overline{C_{In}} + C_{In})$$

$$= C_{In}(A \oplus B) + AB$$

Full-Adder Design [3]

$$Sum = (A \oplus B) \oplus C_{in} \qquad C_O = C_{In}(A \oplus B) + AB$$



4-Bit Adder

We can create a 4-bit adding circuit using

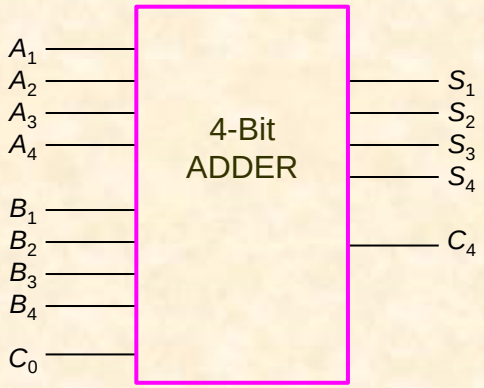
- Available 4 Full Adders:

By connecting 4 Full Adders, one by one.

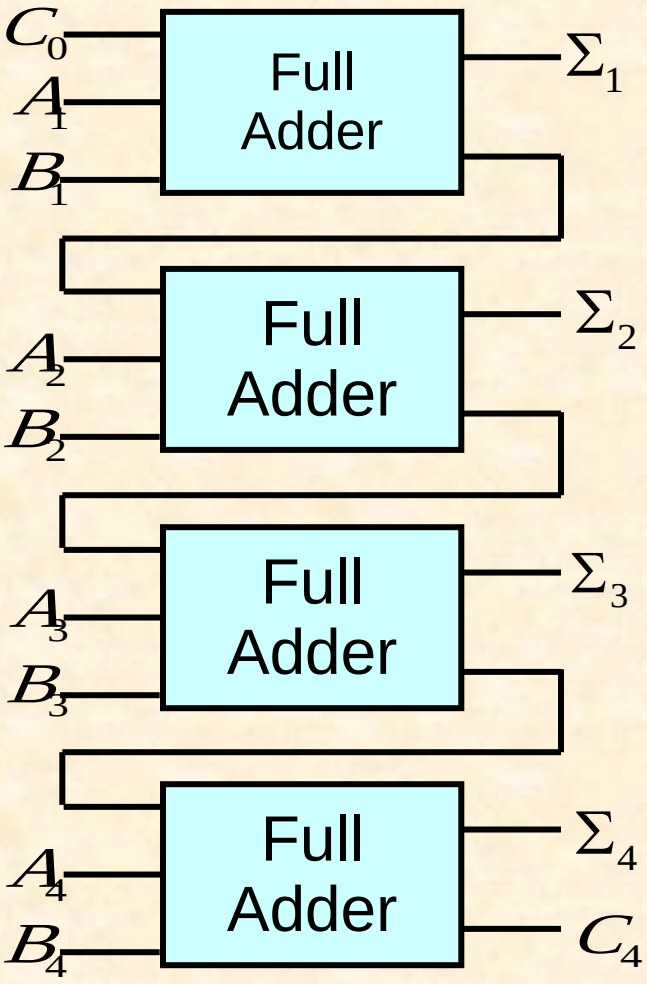
(See the next page).

- Redesign the 4-bit adding circuit:
  - Assign 9 INPUTS: Augend 4 Bits, Addend 4 Bits and Carry In 1 Bit;
  - and 5 OUTPUTS: Sum 4 bits and Carry Out 1 Bit.
  - Then produce the Truth Table/logic function;
  - Minimize the logic function.

(Study in next chapters).



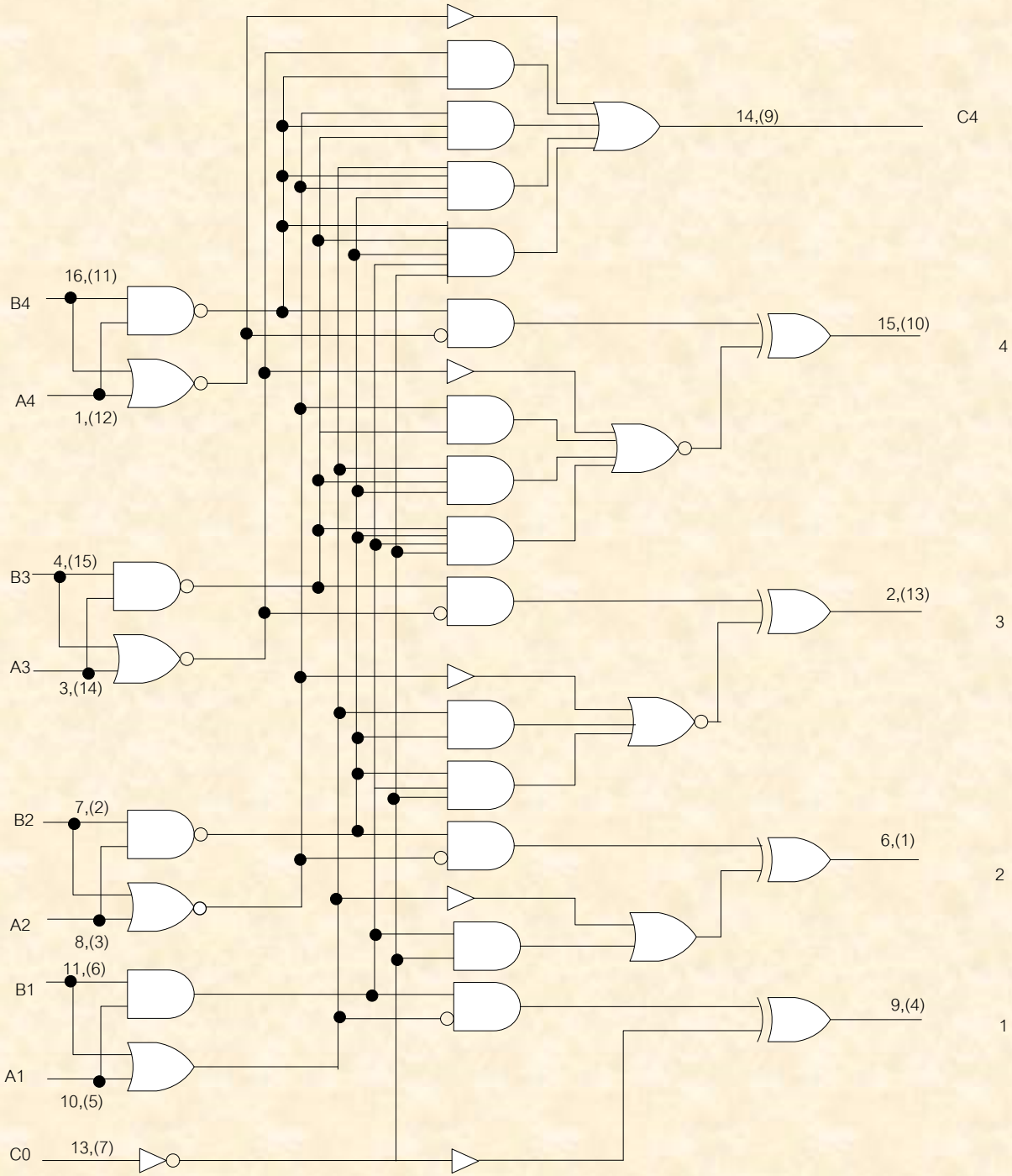
4-Bit Adder: Block Diagram



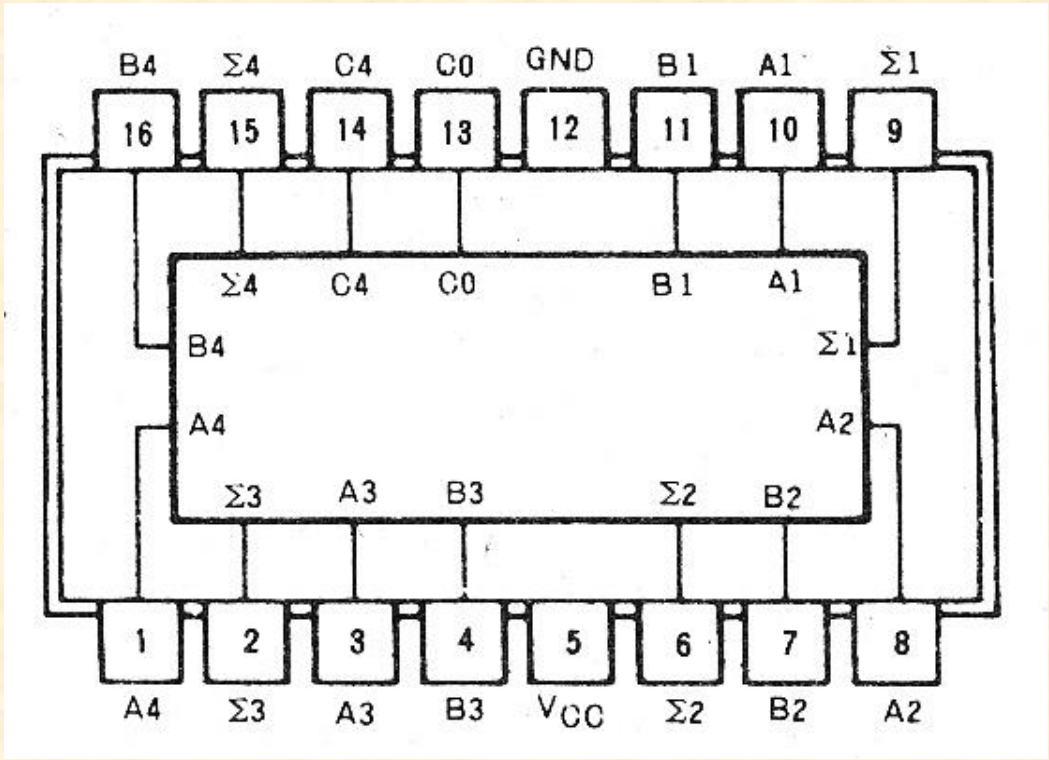
$$\begin{array}{cccccc} & & & & & C_0 \\ & A_4 & A_3 & A_2 & A_1 \\ + & B_4 & B_3 & B_2 & B_1 \\ \hline C_4 & \Sigma_4 & \Sigma_3 & \Sigma_2 & \Sigma_1 \\ \hline \end{array}$$

4-Bit Adder: Commercial ICs

DEVICE NO.	FAMILY	DESCRIPTION
7483	TTL	4-bit binary adder with fast carry
74C83	CMOS	4-bit binary adder with fast carry
4008	CMOS	4-bit full adder with fast carry

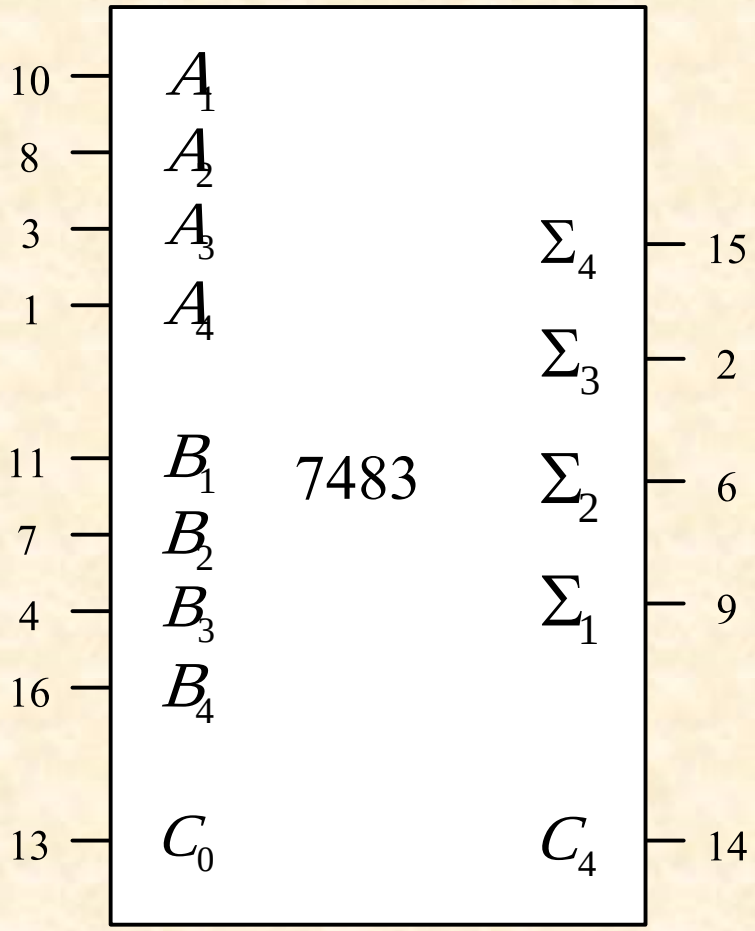

14,(9)
C4
16,(11)
B4
15,(10)
4
A4
1,(12)
4,(15)
B3
2,(13)
3
A3
3,(14)
7,(2)
B2
6,(1)
2
A2
8,(3)
11,(6)
B1
9,(4)
1
A1
10,(5)
C0
13,(7)

4-Bit Adder: Pin layout of IC 7483



$$
\begin{array}{cccccc}
 & & & & & C_0 \\
 & A_4 & A_3 & A_2 & A_1 & \\
+ & B_4 & B_3 & B_2 & B_1 & \\
\hline
C_4 & \Sigma_4 & \Sigma_3 & \Sigma_2 & \Sigma_1 &
\end{array}
$$

4-Bit Adder: Symbol of IC 7483



$$
\begin{array}{cccccc}
 & & & & & C_0 \\
 & A_4 & A_3 & A_2 & A_1 & \\
+ & B_4 & B_3 & B_2 & B_1 & \\
\hline
C_4 & \Sigma_4 & \Sigma_3 & \Sigma_2 & \Sigma_1 &
\end{array}
$$