COMBINATIONAL LOGIC

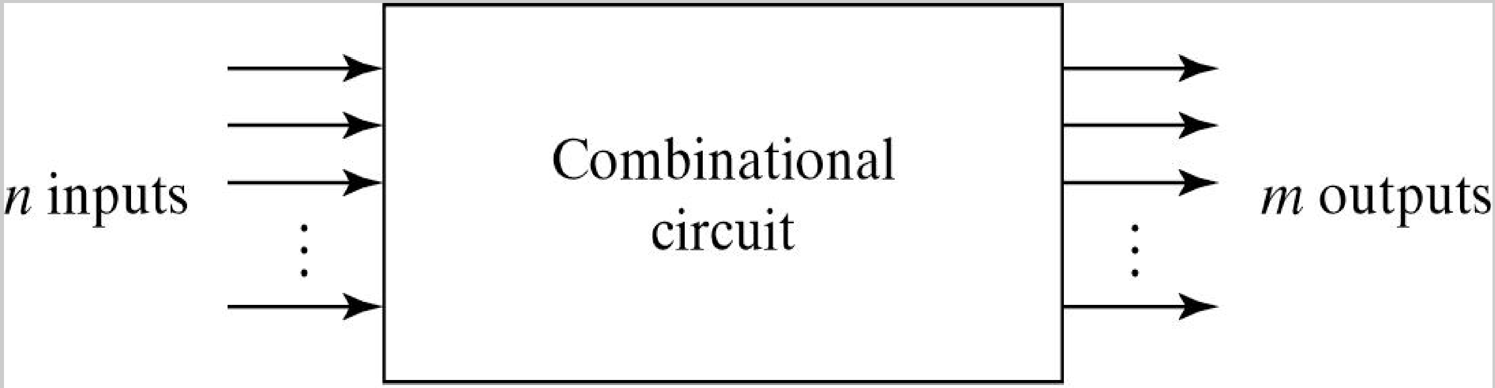
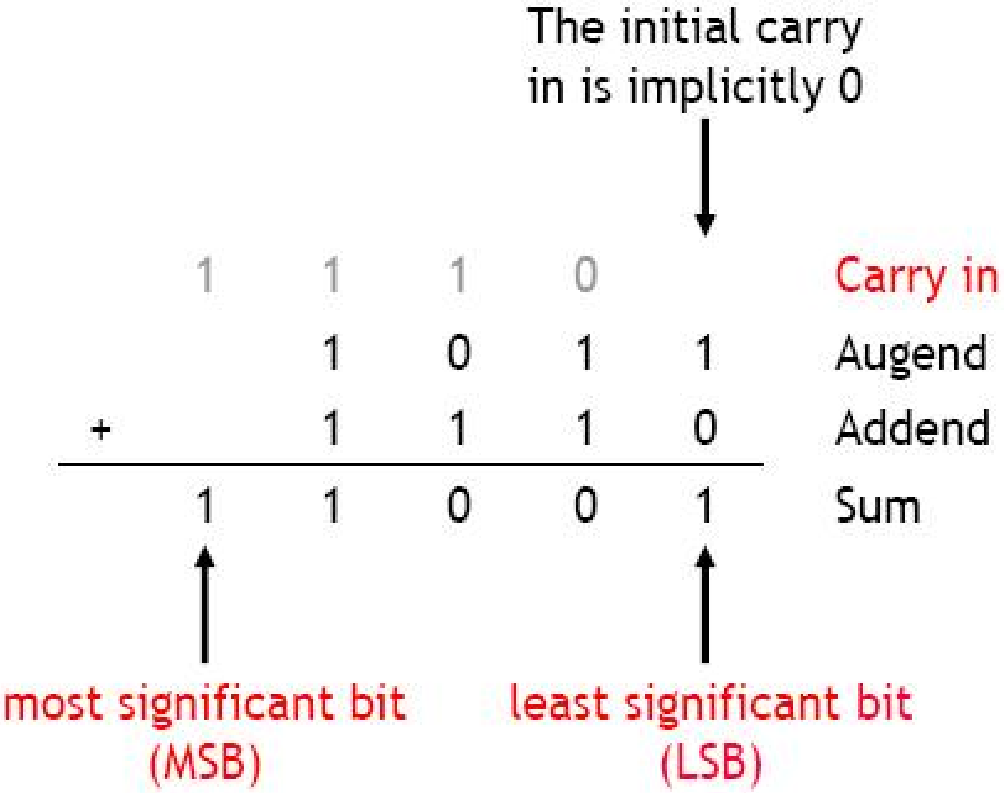


Fig. 4-1 Block Diagram of Combinational Circuit

Binary addition by hand

- You can add two binary numbers one column at a time starting from the right, just like you add two decimal numbers.
- But remember it's binary. For example, 1 + 1 = 10 and you have to carry!



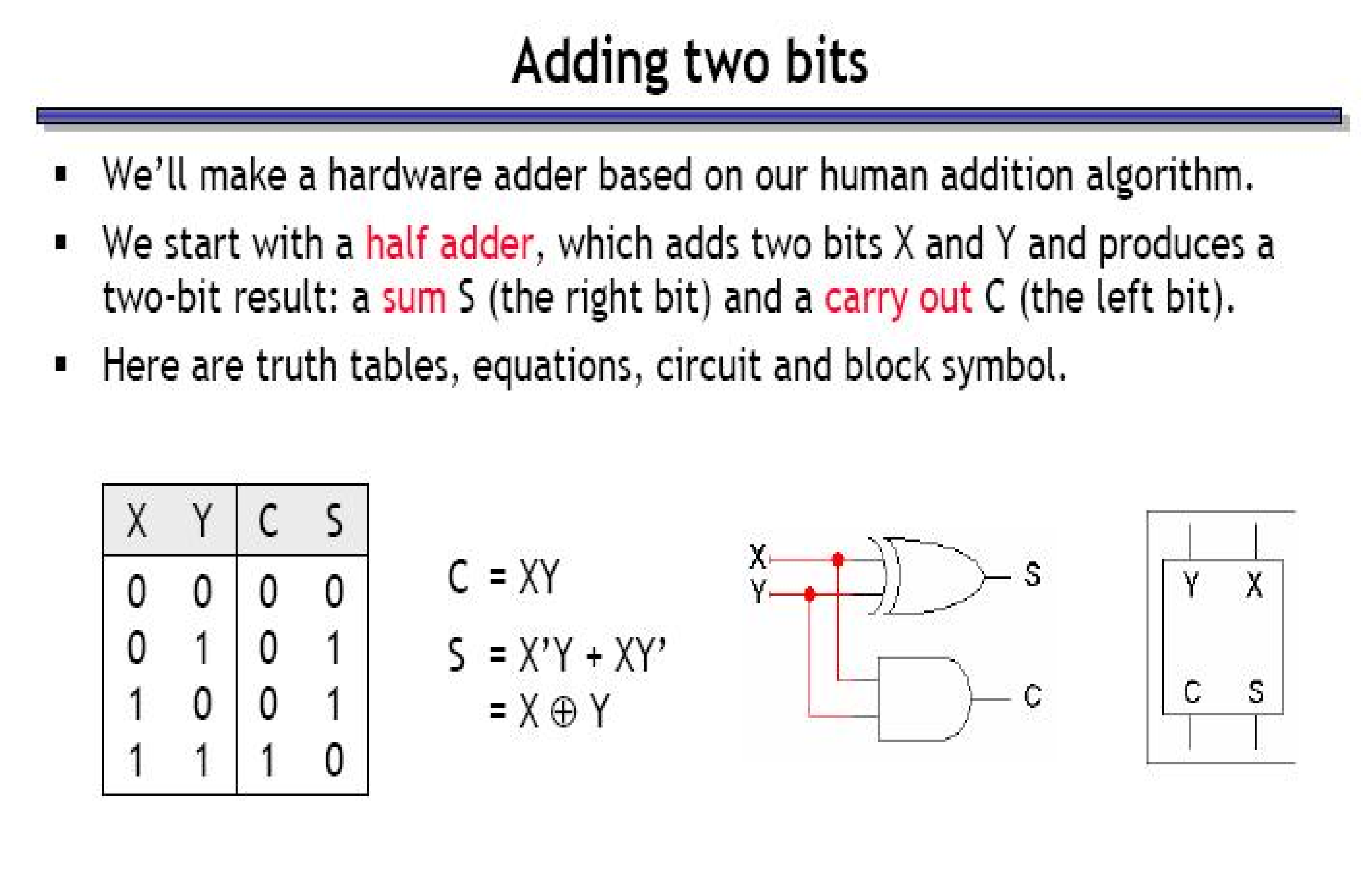
The initial carry in is implicitly 0

	1	1	1	0		Carry in
		1	0	1	1	Augend
+		1	1	1	0	Addend
	1	1	0	0	1	Sum

most significant bit (MSB)

least significant bit (LSB)

HALF ADDER



Adding two bits

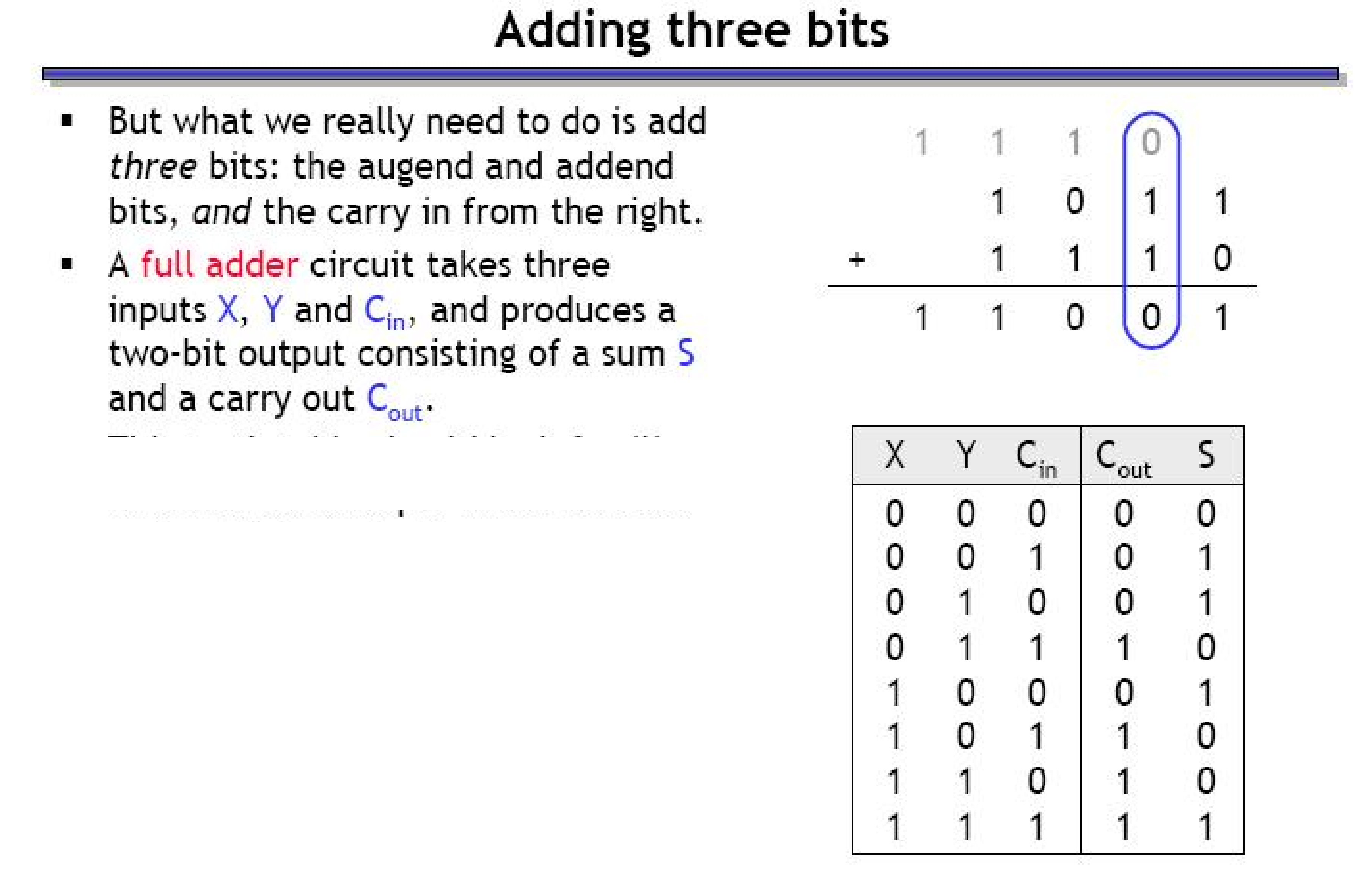
- We’ll make a hardware adder based on our human addition algorithm.
- We start with a half adder, which adds two bits X and Y and produces a two-bit result: a sum S (the right bit) and a carry out C (the left bit).
- Here are truth tables, equations, circuit and block symbol.

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

$$C = XY$$

$$\begin{aligned} S &= X'Y + XY' \\ &= X \oplus Y \end{aligned}$$

FULL ADDER



Adding three bits

- But what we really need to do is add *three* bits: the augend and addend bits, *and* the carry in from the right.
- A full adder circuit takes three inputs X, Y and C_{in}, and produces a two-bit output consisting of a sum S and a carry out C_{out}.

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

X	Y	C_{in}	C_{out}	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

Full adder equations

- Using Boolean algebra, we can simplify S and C_{out} as shown here.

X	Y	C_{in}	C_{out}	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

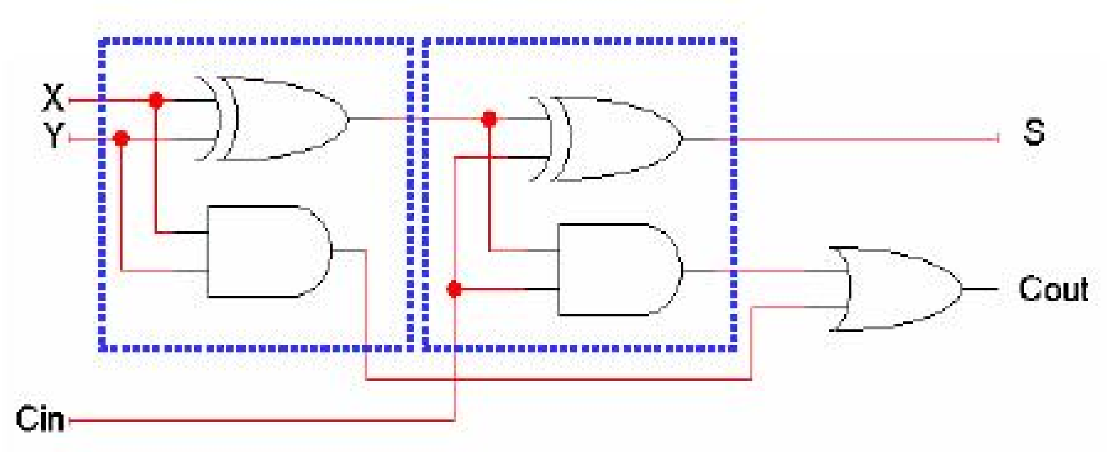
$$\begin{aligned}
S &= \Sigma m(1,2,4,7) \\
&= X'Y'C_{in} + X'YC_{in}' + XY'C_{in}' + XYC_{in} \\
&= X'(Y'C_{in} + YC_{in}') + X(Y'C_{in}' + YC_{in}) \\
&= X'(Y \oplus C_{in}) + X(Y \oplus C_{in})' \\
&= X \oplus Y \oplus C_{in}
\end{aligned}$$

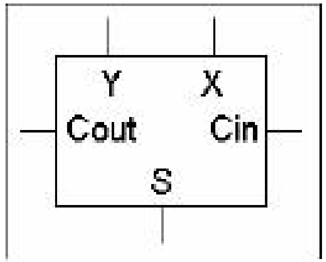
$$\begin{aligned}
C_{out} &= \Sigma m(3,5,6,7) \\
&= X'YC_{in} + XY'C_{in} + XYC_{in}' + XYC_{in} \\
&= (X'Y + XY')C_{in} + XY(C_{in}' + C_{in}) \\
&= (X \oplus Y)C_{in} + XY
\end{aligned}$$

Full adder circuit

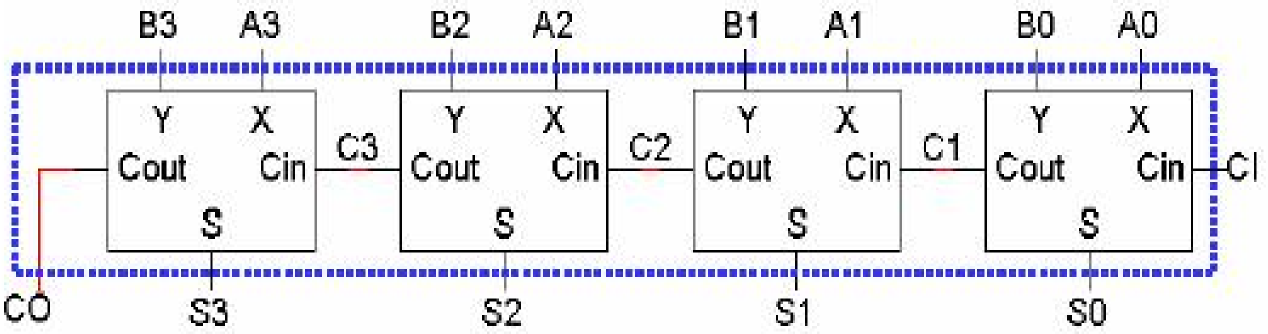
- We write the equations this way to highlight the hierarchical nature of adder circuits—you can build a full adder by combining two half adders!

$$S \quad = X \oplus Y \oplus C_{in}$$
$$C_{out} = (X \oplus Y)\, C_{in} + XY$$

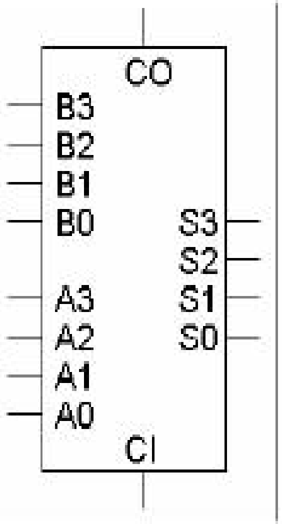




A four-bit adder

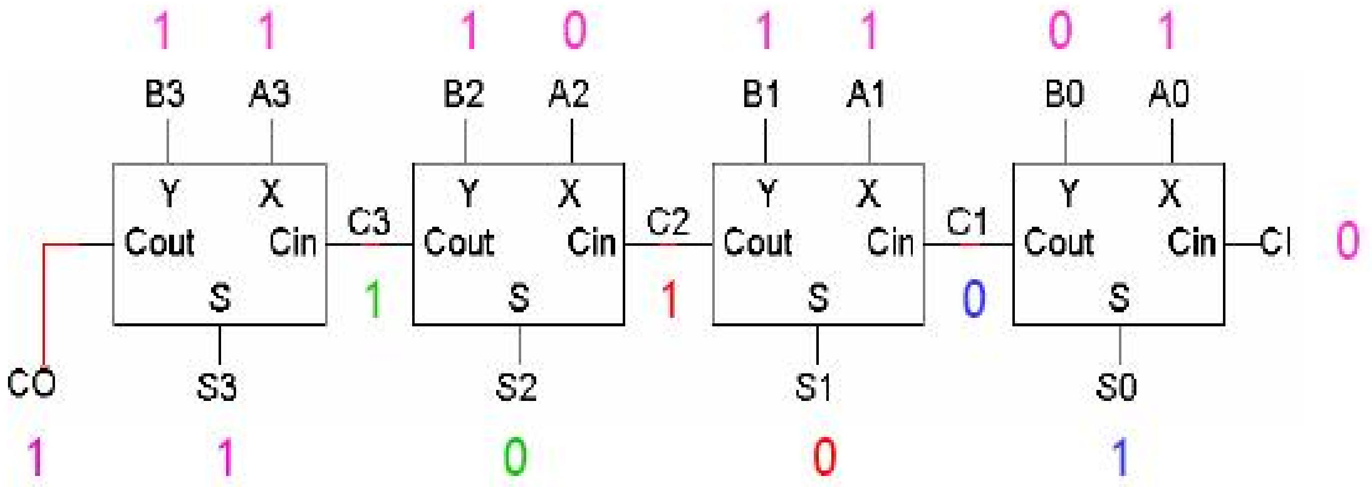


- Similarly, we can cascade four full adders to build a four-bit adder.
  - The inputs are two four-bit numbers (A3A2A1A0 and B3B2B1B0) and a carry in CI.
  - The two outputs are a four-bit sum S3S2S1S0 and the carry out CO.
- If you designed this adder without taking advantage of the hierarchical structure, you'd end up with a 512-row truth table with five outputs!



An example of 4-bit addition

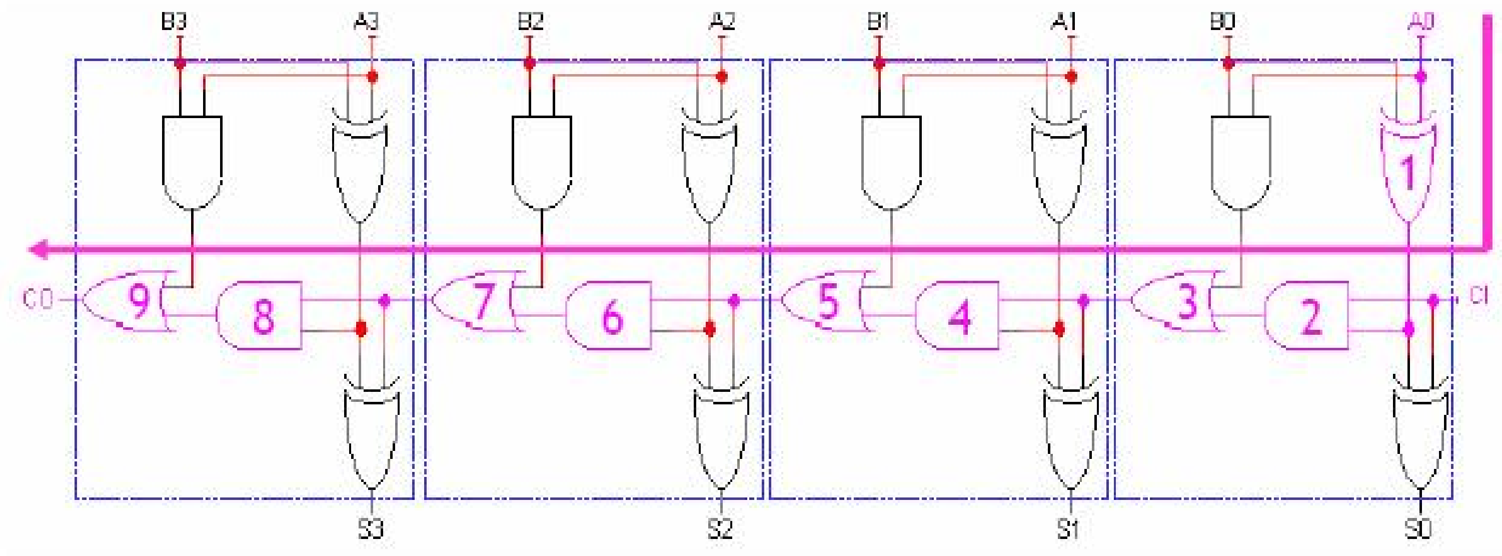
- Let’s put our initial example into this circuit, with A=1011 and B=1110.



1. Fill in all the inputs, including CI=0
2. The circuit produces C1 and S0 (1 + 0 + 0 = 01)
3. Use C1 to find C2 and S1 (1 + 1 + 0 = 10)
4. Use C2 to compute C3 and S2 (0 + 1 + 1 = 10)
5. Use C3 to compute CO and S3 (1 + 1 + 1 = 11)

Ripple carry delays

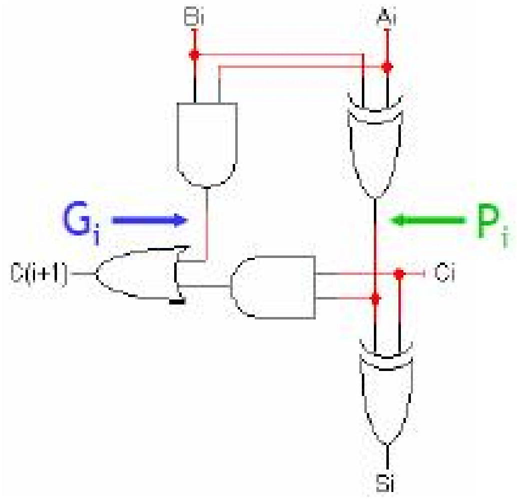
- The diagram below shows our four-bit adder completely drawn out.
- This is called a ripple carry adder, because the inputs A0, B0 and CI "ripple" leftwards until CO and S3 are produced.
- Ripple carry adders are slow!
  - There is a very long path from A0, B0 and CI to CO and S3.
  - For an *n*-bit ripple carry adder, the longest path has 2*n*+1 gates.
  - The longest path in a 64-bit adder would include 129 gates!



A faster way to compute carry outs

- Instead of waiting for the carry out from each previous stage, we can minimize the delay by computing it directly with a two-level circuit.
- First we'll define two functions.
  - The "generate" function G_i produces 1 when there *must* be a carry out from position i (i.e., when A_i and B_i are both 1).

$$G_i = A_iB_i$$



  - The "propagate" function P_i is true when an incoming carry is propagated (i.e, when A_i=1 or B_i=1, but not both).

$$P_i = A_i \oplus B_i$$

- Then we can rewrite the carry out function.

$$C_{i+1} = G_i + P_iC_i$$

A_i	B_i	C_i	C_{i+1}
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	1	0	1
1	1	1	1

- Let’s look at the carry out equations for specific bits, using the general equation from the previous page $C_{i+1} = G_i + P_iC_i$.

$$C_1 = G_0 + P_0C_0$$

$$\begin{aligned} C_2 &= G_1 + P_1C_1 \\ &= G_1 + P_1(G_0 + P_0C_0) \\ &= G_1 + P_1G_0 + P_1P_0C_0 \end{aligned}$$

$$\begin{aligned} C_3 &= G_2 + P_2C_2 \\ &= G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0) \\ &= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \end{aligned}$$

$$\begin{aligned} C_4 &= G_3 + P_3C_3 \\ &= G_3 + P_3(G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0) \\ &= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0 \end{aligned}$$

- These expressions are all sums of products, so we can use them to make a circuit with only a two-level delay.

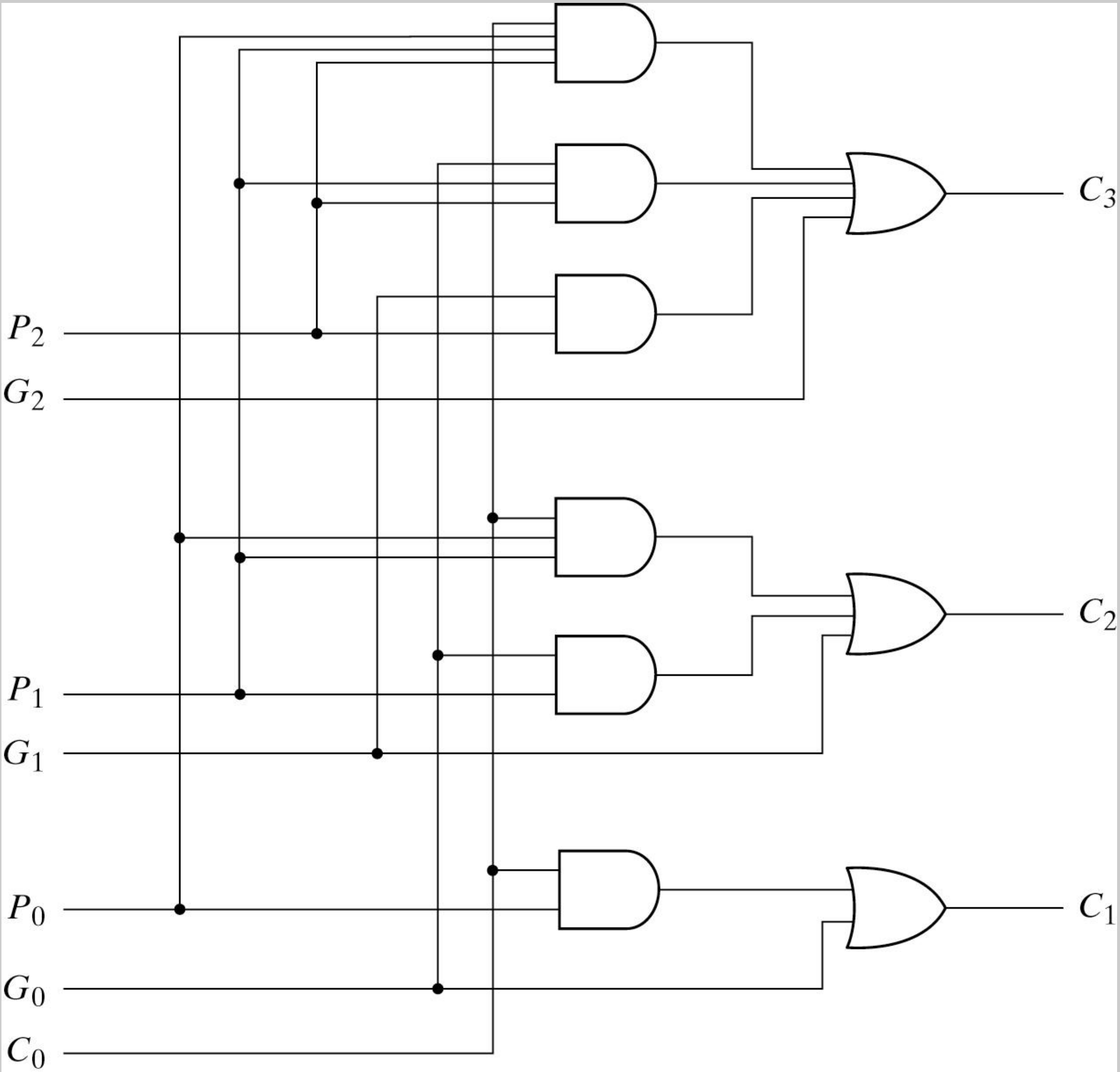


Fig. 4-11 Logic Diagram of Carry Lookahead Generator

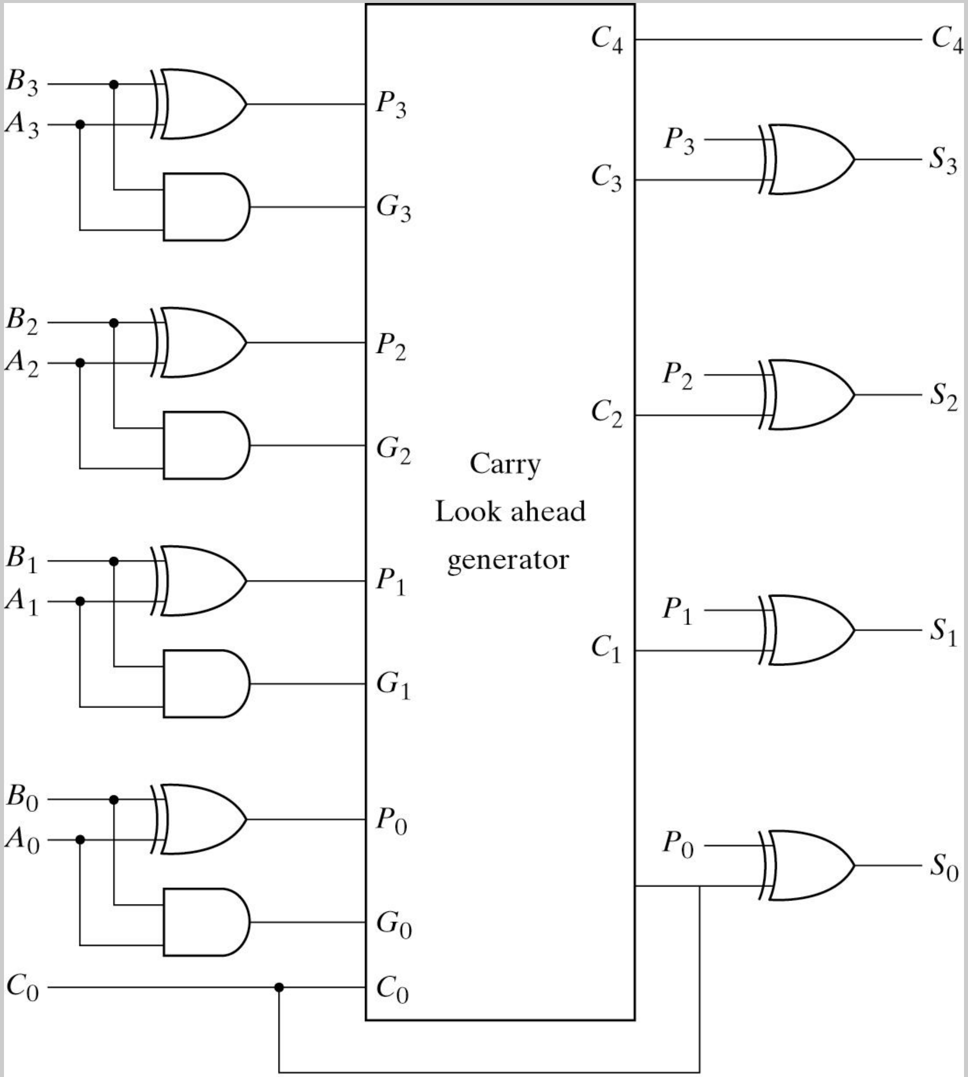
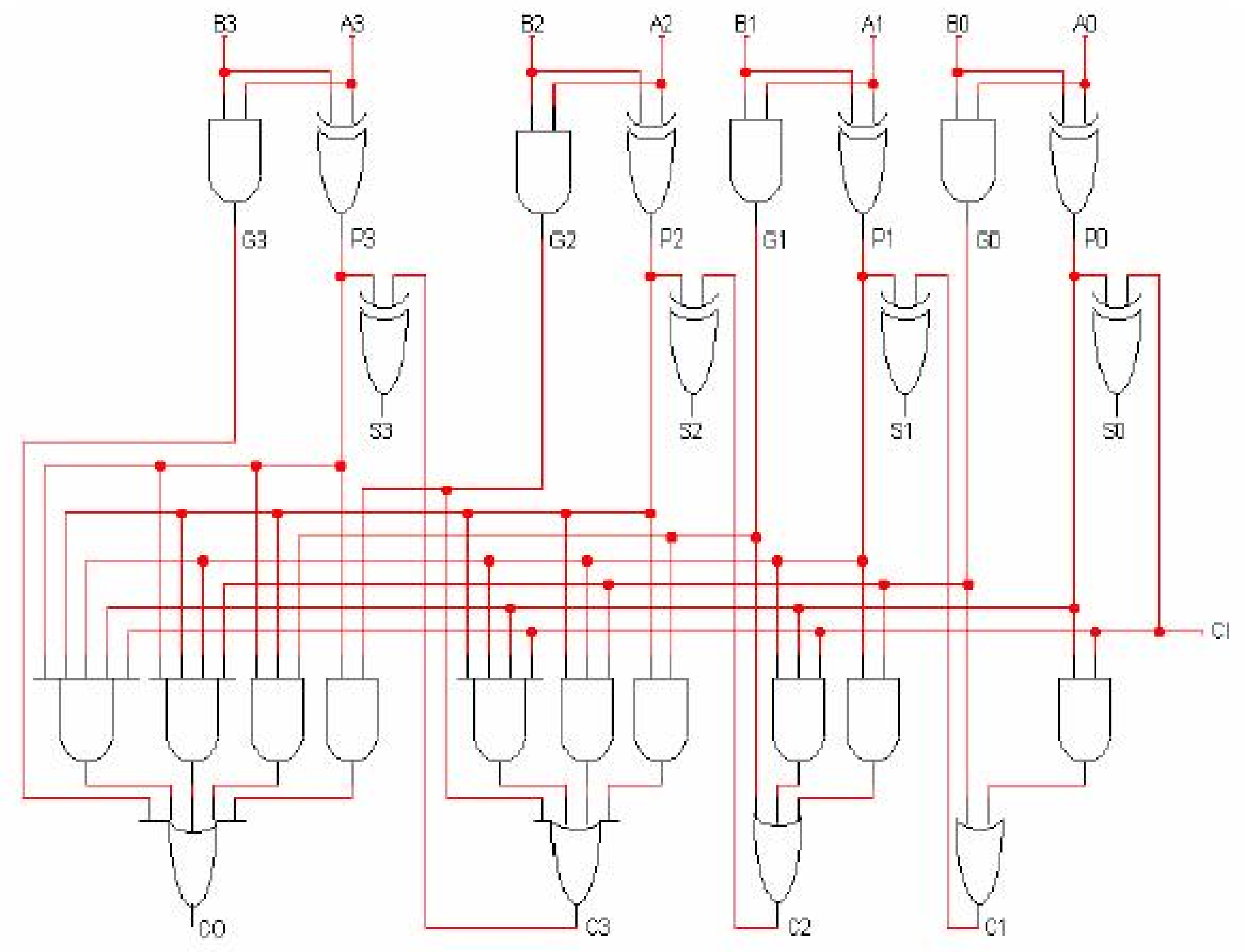
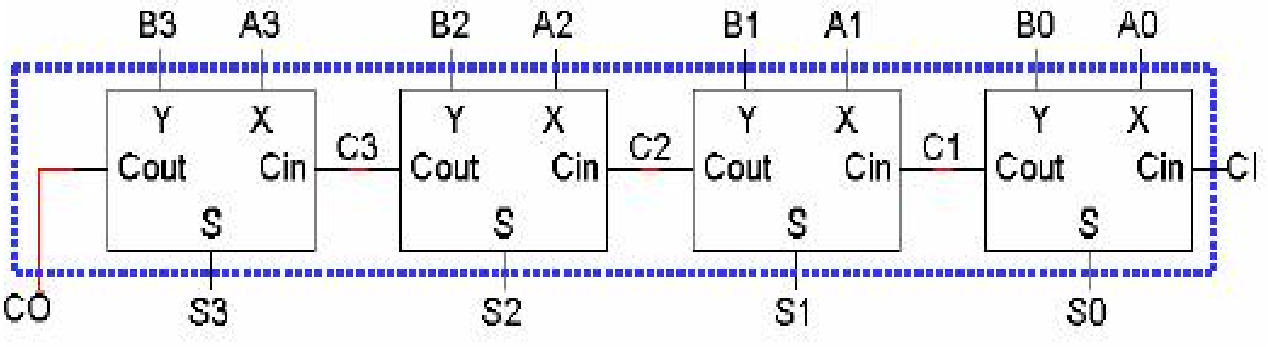


Fig. 4-12 4-Bit Adder with Carry Lookahead

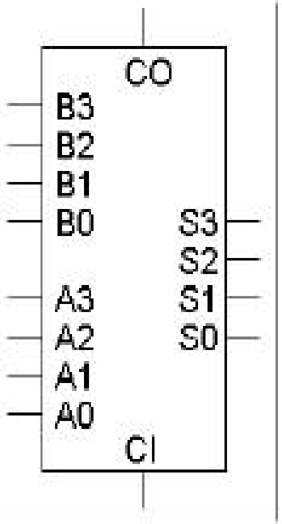
A faster four-bit adder



A four-bit adder

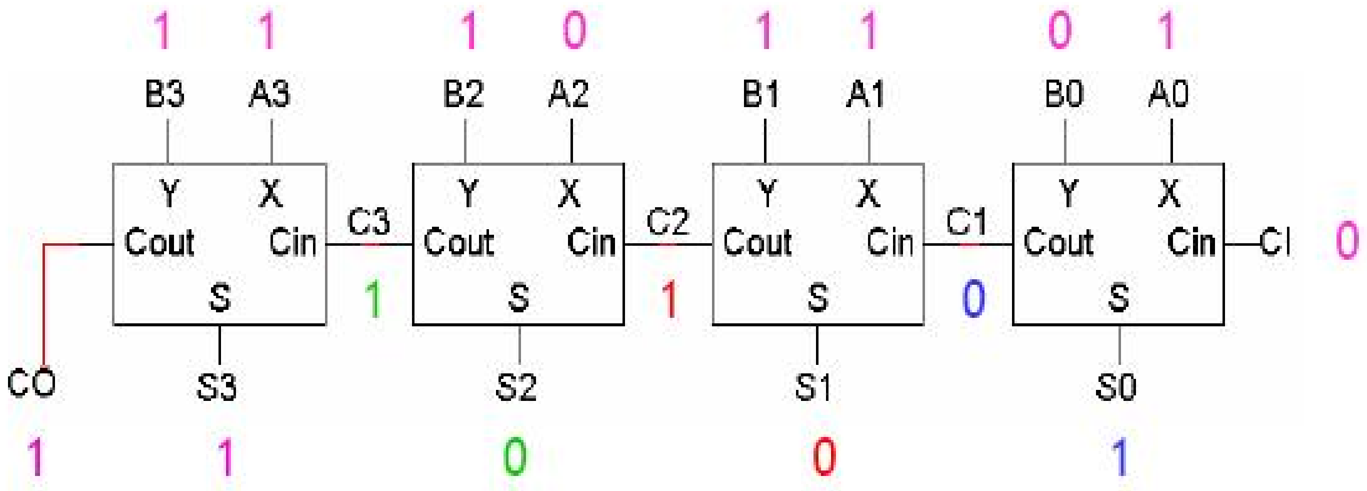


- Similarly, we can cascade four full adders to build a four-bit adder.
  - The inputs are two four-bit numbers (A3A2A1A0 and B3B2B1B0) and a carry in CI.
  - The two outputs are a four-bit sum S3S2S1S0 and the carry out CO.
- If you designed this adder without taking advantage of the hierarchical structure, you'd end up with a 512-row truth table with five outputs!



An example of 4-bit addition

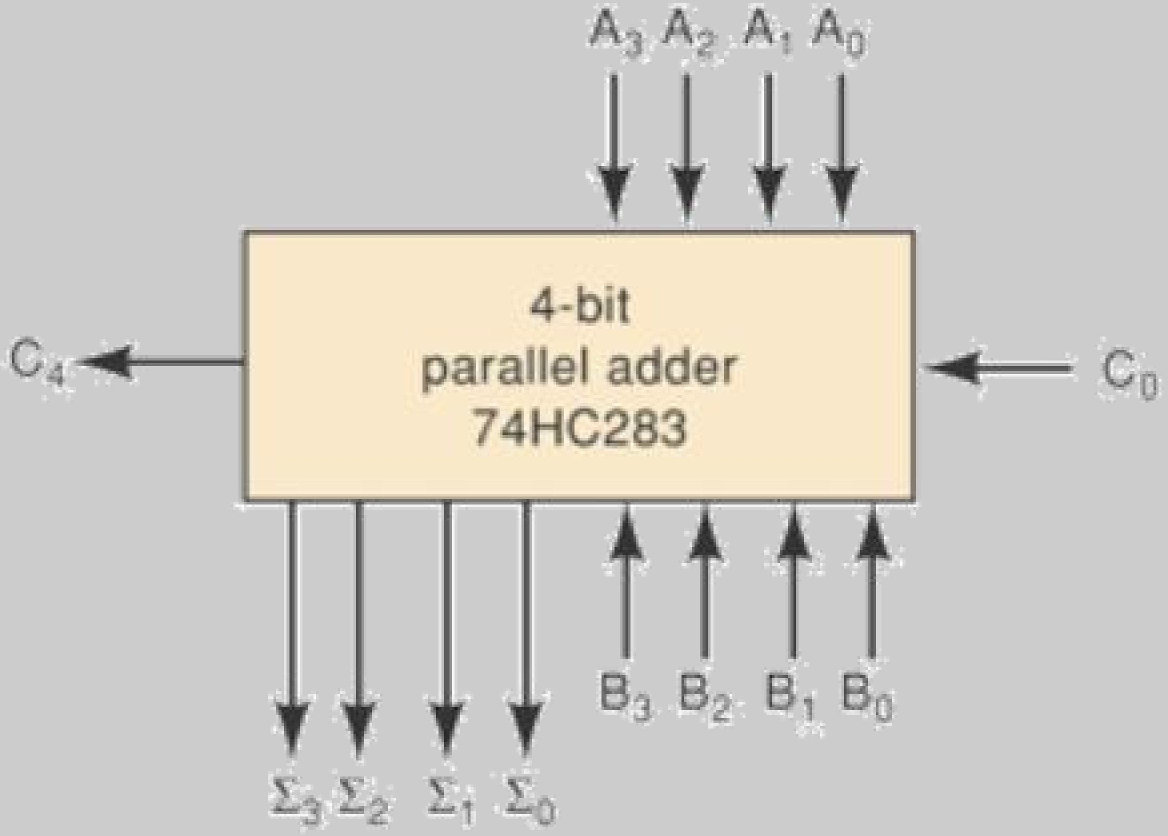
- Let's put our initial example into this circuit, with A=1011 and B=1110.



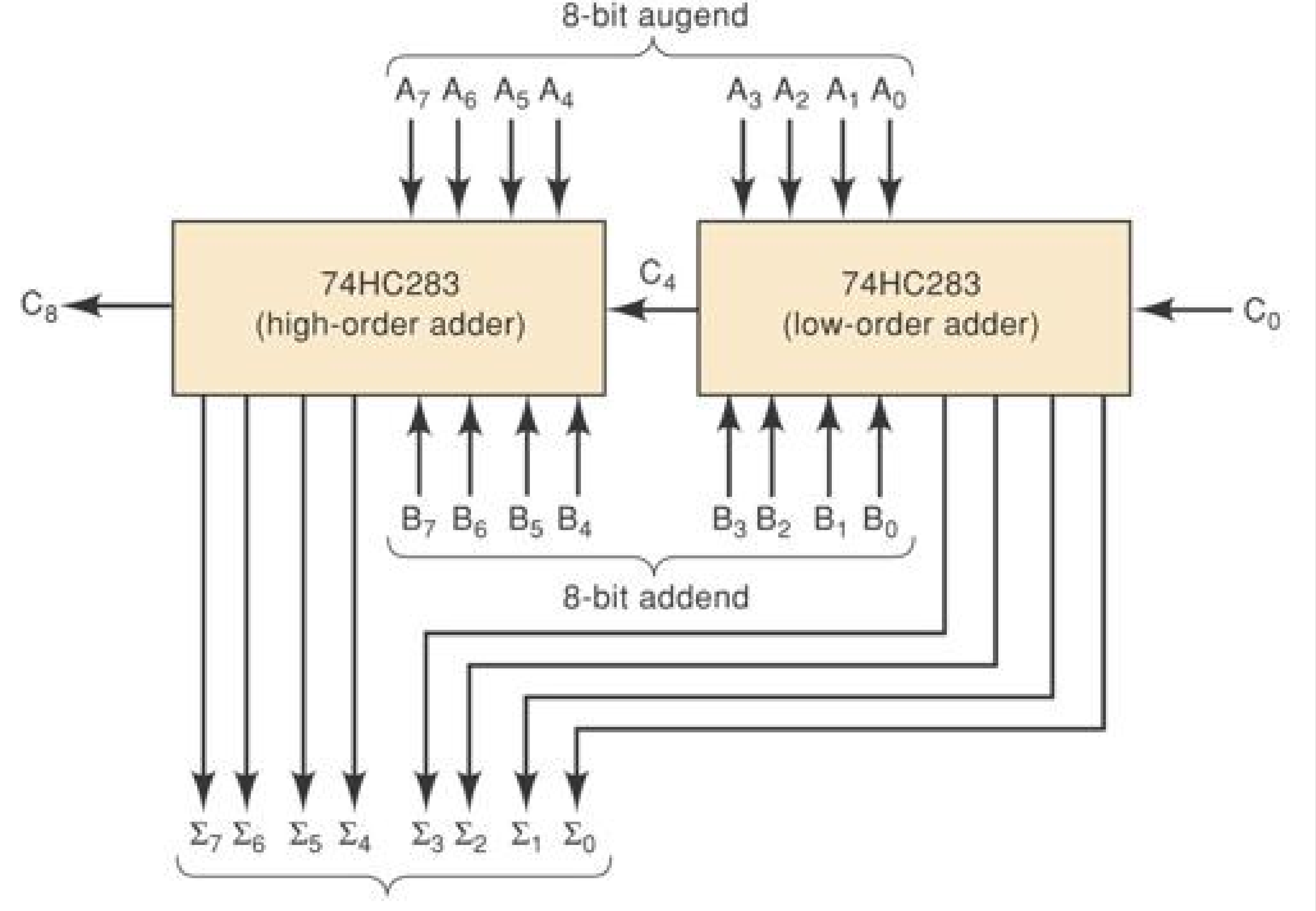
1. Fill in all the inputs, including CI=0
2. The circuit produces C1 and S0 (1 + 0 + 0 = 01)
3. Use C1 to find C2 and S1 (1 + 1 + 0 = 10)
4. Use C2 to compute C3 and S2 (0 + 1 + 1 = 10)
5. Use C3 to compute CO and S3 (1 + 1 + 1 = 11)

Integrated Circuit Parallel Adder
(IC Parallel Adder)

4- Bits adder



8 – Bit IC Parallel Adder



Binary Adder/Subtractors

- **The subtraction *A-B* can be performed by taking the 2's complement of *B* and adding to *A*.**
- **The 2's complement of *B* can be obtained by complementing B and adding one to the result.**

$$
\begin{aligned}
A\text{-}B &= A + 2C(B) \\
&= A + 1C(B) + 1 \\
&= A + B' + 1
\end{aligned}
$$

Binary Adder/Subtractors

Representation of numbers

Representation

- 2’s Complement

  -x: 2^n-x

- 1’s Complement

  -x: 2^n-x-1

Representation

- 2's Complement

  -x: 2^n-x

  e.g. 16-x

- 1's Complement

  -x: 2^n-x-1

  e.g. 16-x-1

Id	2's comp.	1's comp.
0	0	15
-1	15	14
-2	14	13
-3	13	12
-4	12	11
-5	11	10
-6	10	9
-7	9	8
-8	8	

Representation

Id	-Binary	sign mag	2’s comp	1’s comp
0	0000	1000	0000	1111
-1	0001	1001	1111	1110
-2	0010	1010	1110	1101
-3	0011	1011	1101	1100
-4	0100	1100	1100	1011
-5	0101	1101	1011	1010
-6	0110	1110	1010	1001
-7	0111	1111	1001	1000
-8			1000	

Representation

1’s Complement

For a negative number, we take the positive number and complement every bit.

2’s Complement

For a negative number, we do 1’s complement and plus one.

$(b_{n-1}, b_{n-2}, \ldots, b_0)$: $-b_{n-1}2^{n-1} + \mathrm{sum}_{i<n-1}\, b_i 2^i$

Representation

2’s Complement

- $x+y$
- $x-y$: $x+2^n-y= 2^n+x-y$
- $-x+y$: 2^n-x+y
- $-x-y$: 2^n-x+2^n-y

  $= 2^n+2^n-x-y$
- $-(-x)=2^n-(2^n-x)=x$

1’s Complement

- $x+y$
- $x-y$: $x+2^n-y-1= 2^n-1+x-y$
- $-x+y$: $2^n-x-1+y=2^n-1-x+y$
- $-x-y$: $2^n-x-1+2^n-y-1$

  $= 2^n-1+2^n-x-y-1$
- $-(-x)=2^n-(2^n-x-1) -1=x$

Examples

2 + 3 = 5

$$\begin{array}{rcccc} 0 & 0 & 1 & 0 & \\ & 0 & 0 & 1 & 0 \\ + & 0 & 0 & 1 & 1 \\ \hline & 0 & 1 & 0 & 1 \end{array}$$

2 - 3 = -1 (2's)

$$\begin{array}{rcccc} 0 & 0 & 0 & 0 & \\ & 0 & 0 & 1 & 0 \\ + & 1 & 1 & 0 & 1 \\ \hline & 1 & 1 & 1 & 1 \end{array}$$

2 - 3 = -1 (1's)

$$\begin{array}{rcccc} & 0 & 0 & 1 & 0 \\ + & 1 & 1 & 0 & 0 \\ \hline & 1 & 1 & 1 & 0 \end{array}$$

-2 - 3 = -5 (2's)

$$\begin{array}{rcccc} 1 & 1 & 0 & 0 & \\ & 1 & 1 & 1 & 0 \\ + & 1 & 1 & 0 & 1 \\ \hline & 1 & 0 & 1 & 1 \end{array}$$

-2 - 3 = -5 (1's)

$$\begin{array}{rcccc} \mathbf{1} & 1 & 0 & 0 & \\ & 1 & 1 & 0 & 1 \\ + & 1 & 1 & 0 & 0 \\ \hline & 1 & 0 & 0 & 1 \\ & & & & \mathbf{1} \\ \hline & 1 & 0 & 1 & 0 \end{array}$$

Check for overflow (2's)

3 + 5 = 8

$$\begin{array}{rcccc} 0 & 1 & 1 & 1 & \\ & 0 & 0 & 1 & 1 \\ + & 0 & 1 & 0 & 1 \\ \hline & 1 & 0 & 0 & 0 \end{array}$$

C_4C_3

-3 + -5 = -8

$$\begin{array}{rcccc} 1 & 1 & 1 & 1 & \\ & 1 & 1 & 0 & 1 \\ + & 1 & 0 & 1 & 1 \\ \hline & 1 & 0 & 0 & 0 \end{array}$$

C_4C_3

Addition: 2’s Complement Overflow

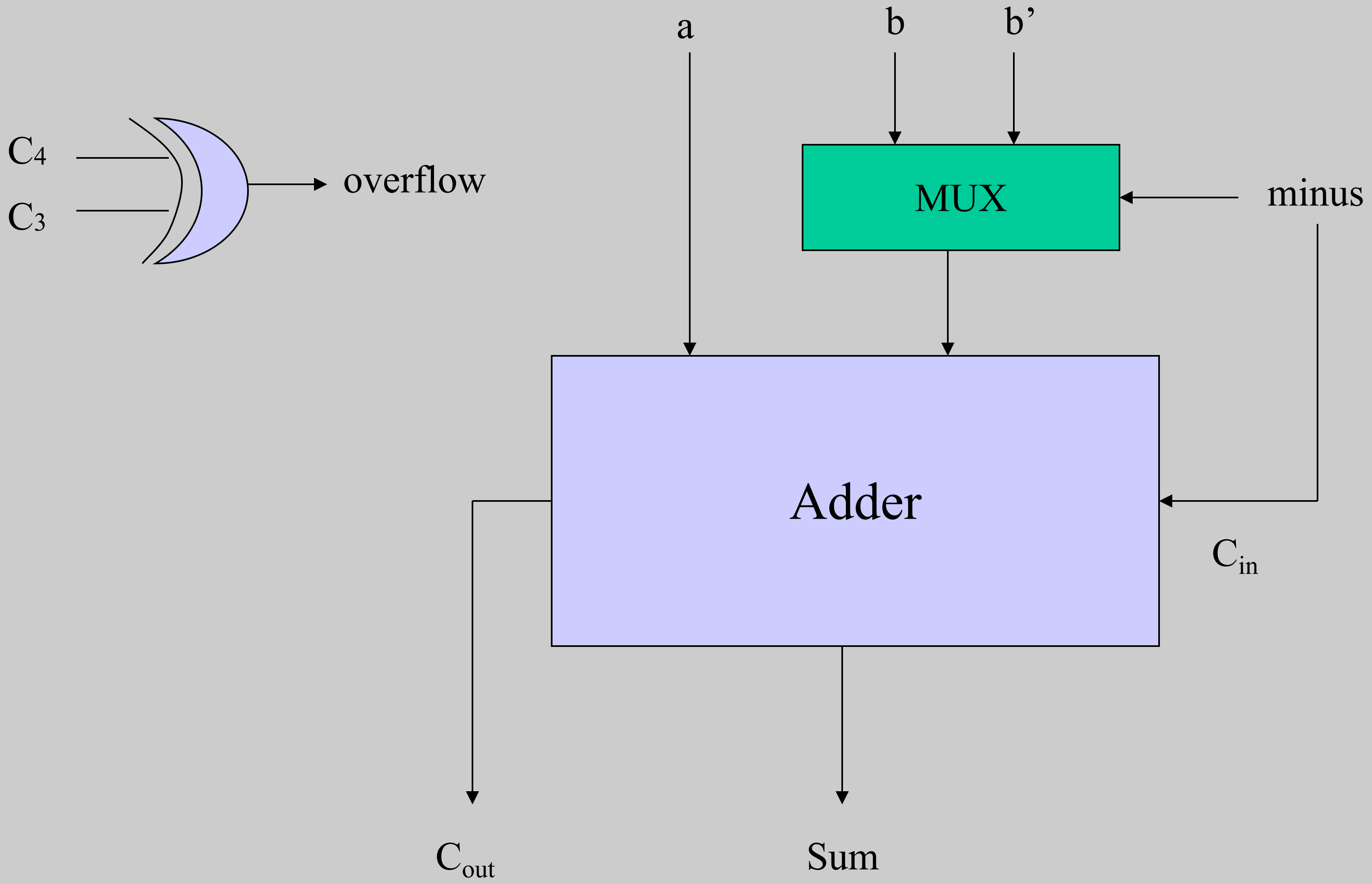
In 2’s complement:

overflow = c_n xor c_{n-1}

Exercise:

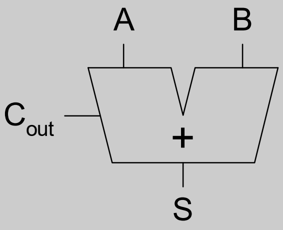
1. Demonstrate the overflow with more examples.
2. Prove the condition.

Addition and Subtraction using 2’s Complement



1-Bit Adders

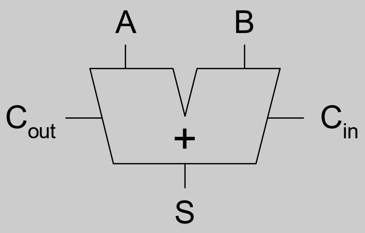
Half Adder



A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$
$$C_{out} = AB$$

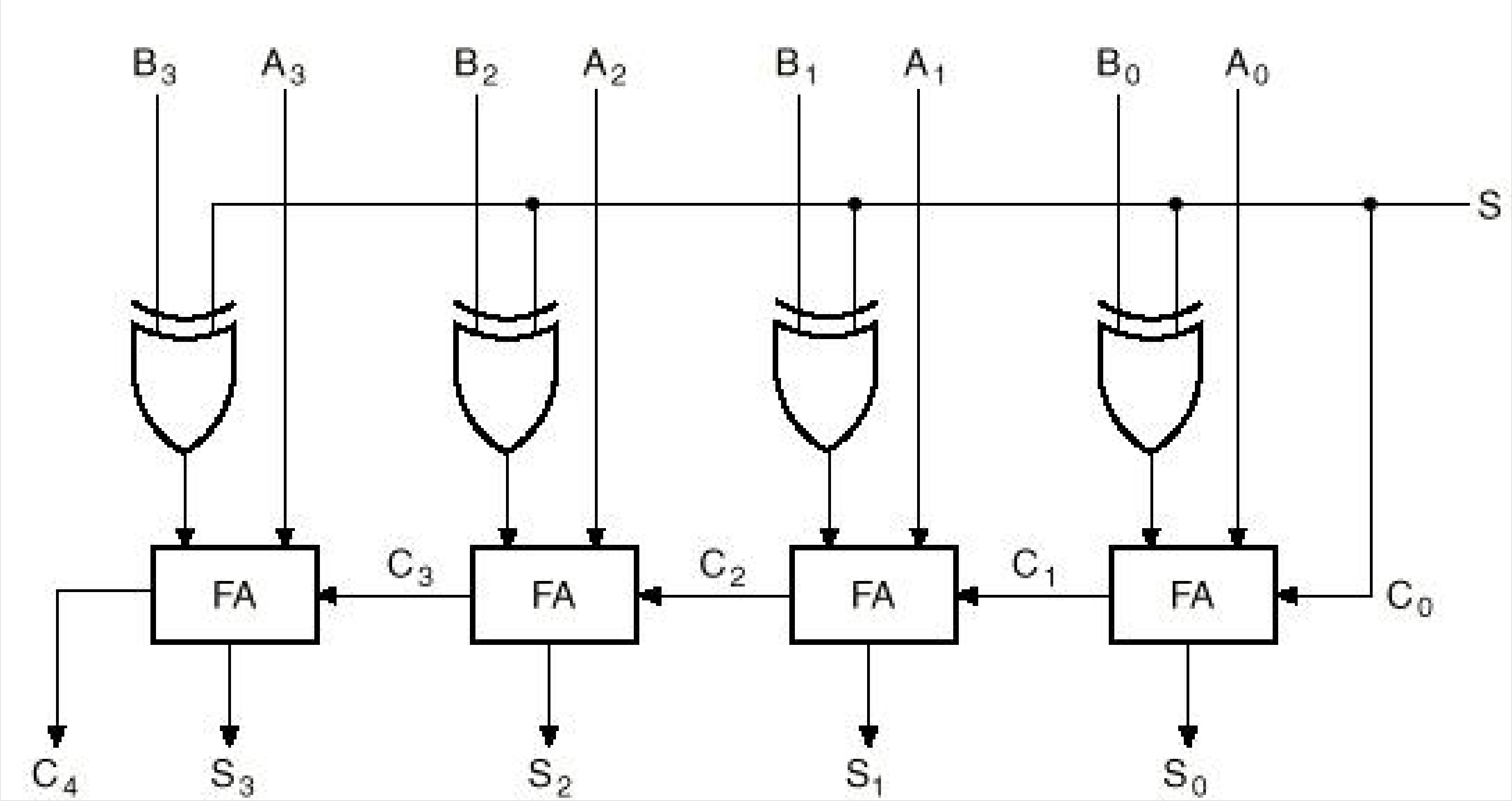
Full Adder



C_{in}	A	B	C_{out}	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

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

4-bit Binary Adder/Subtractor

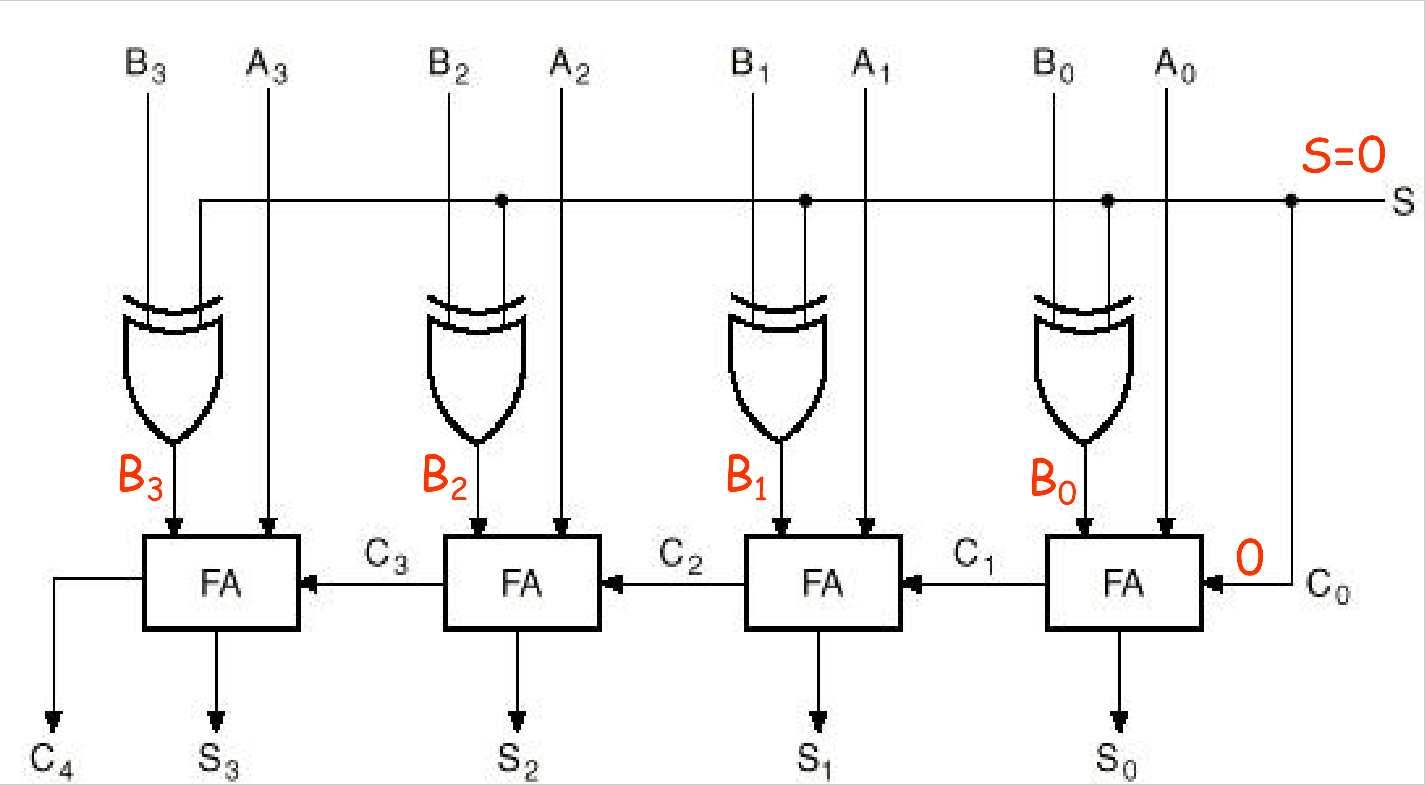


-XOR gates act as programmable inverters

4-bit Binary Adder/Subtractor (cont.)

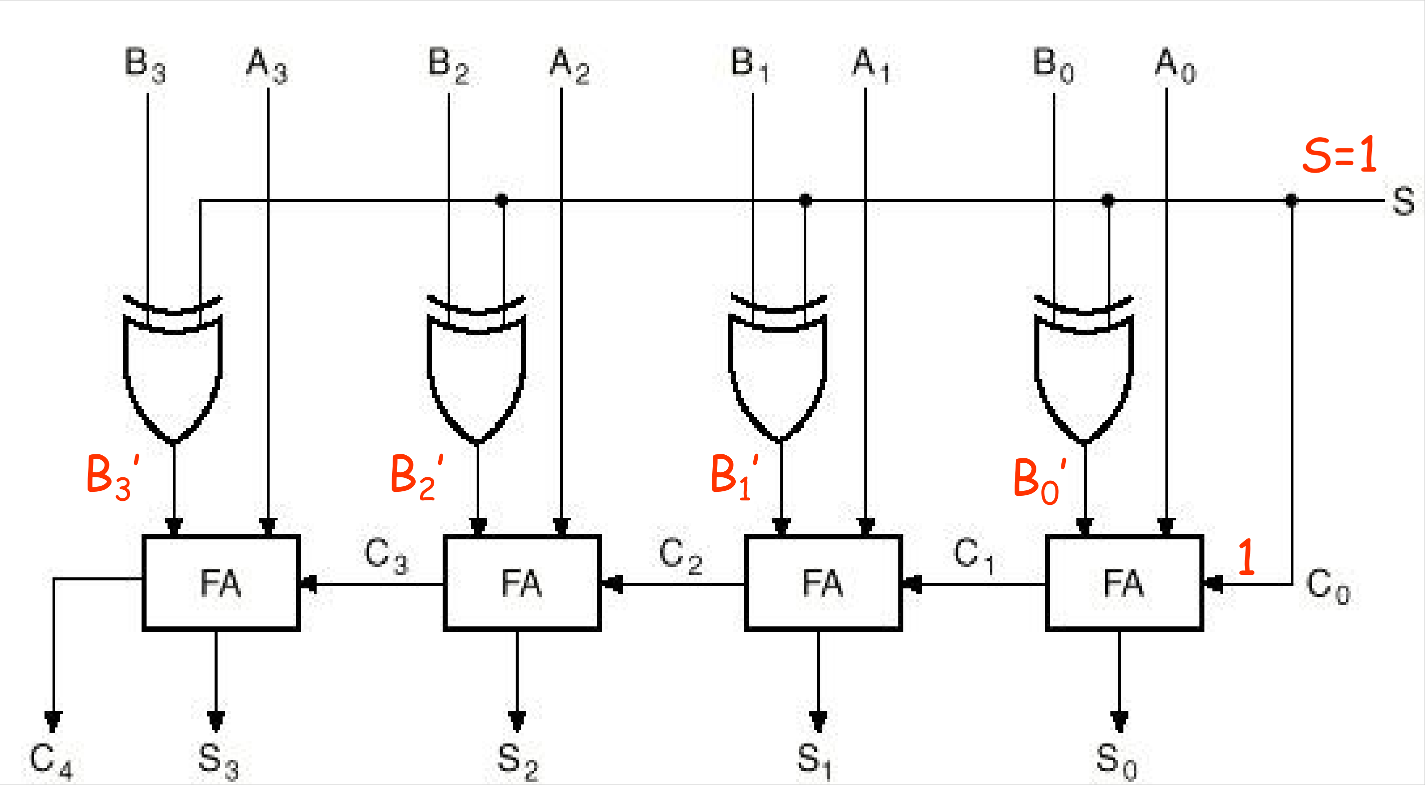
- When *S*=0, the circuit performs *A* + *B*. The carry in is 0, and the XOR gates simply pass *B* untouched.
- When *S*=1, the carry into the least significant bit (LSB) is 1, and *B* is complemented (1's complement) prior to the addition; hence, the circuit adds to A the 1's complement of *B* plus 1 (from the carry into the LSB).

4-bit Binary Adder/Subtractor (cont.)



S=0 selects addition

4-bit Binary Adder/Subtractor (cont.)



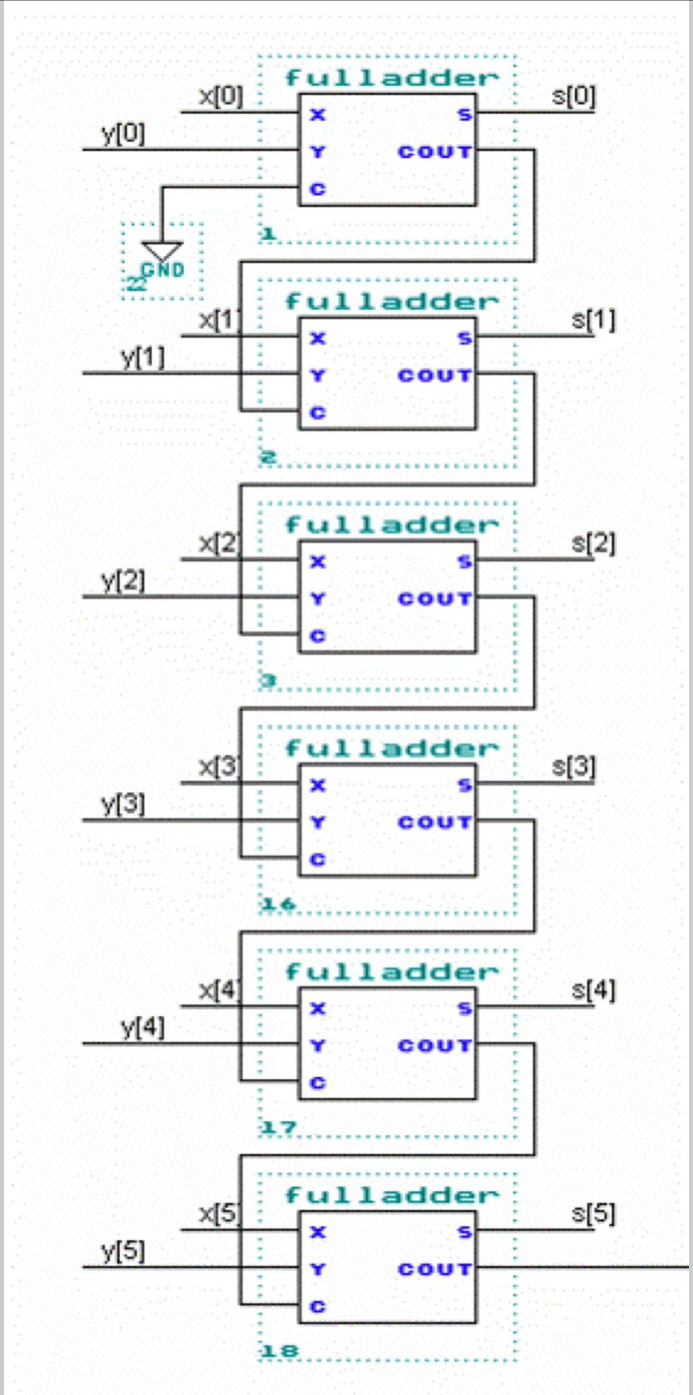
S=1 selects subtraction

Parallel Binary adder

Two Numbers X & Y

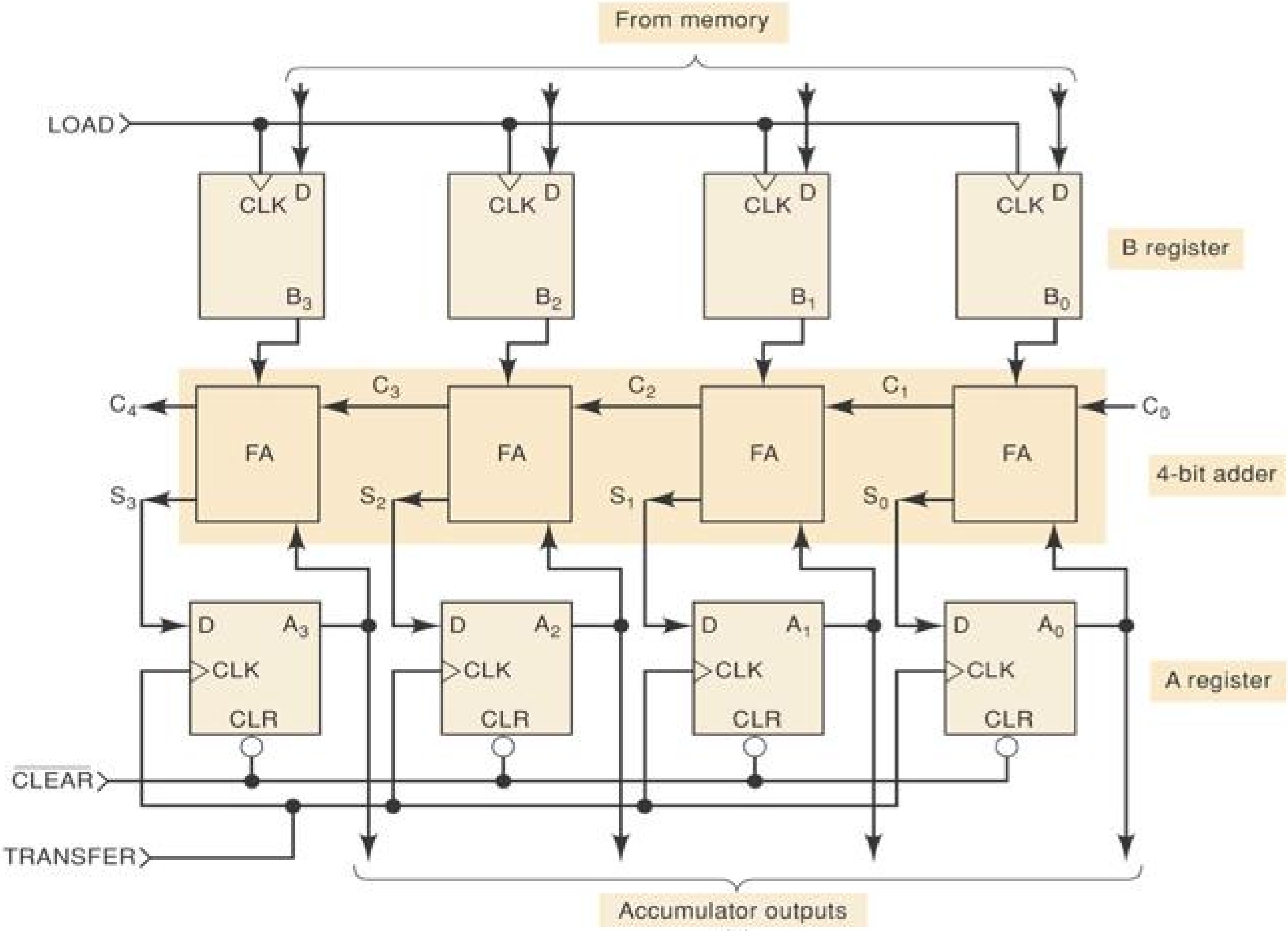
Each number is represented using 6 bits.

$[X]=X_oX_1X_2X_3X_4X_5$
$[Y]=Y_oY_1Y_2Y_3Y_4Y_5$



Complete Parallel Adder With Registers

- D Flip-flops are used to save a single bit
- A number of D flip-flops are connected in a way to form the binary number, where each bit of that number is saved in a single D flip-flop.


From memory
LOAD
CLK
D
B register
B3
B2
B1
B0
C4
C3
C2
C1
C0
FA
4-bit adder
S3
S2
S1
S0
D
A3
A2
A1
A0
CLK
CLR
A register
CLEAR
TRANSFER
Accumulator outputs

(a)

Example: Sequence of operations in Adding 1001 and 0101

1) [A] = 0000. A CLEAR pulse is applied to CLR of each FF on register A at t_1.

2) [M] → [B]. The first binary number 1001 is transferred from memory to B register on the PGT of the LOAD pulse at t_2.

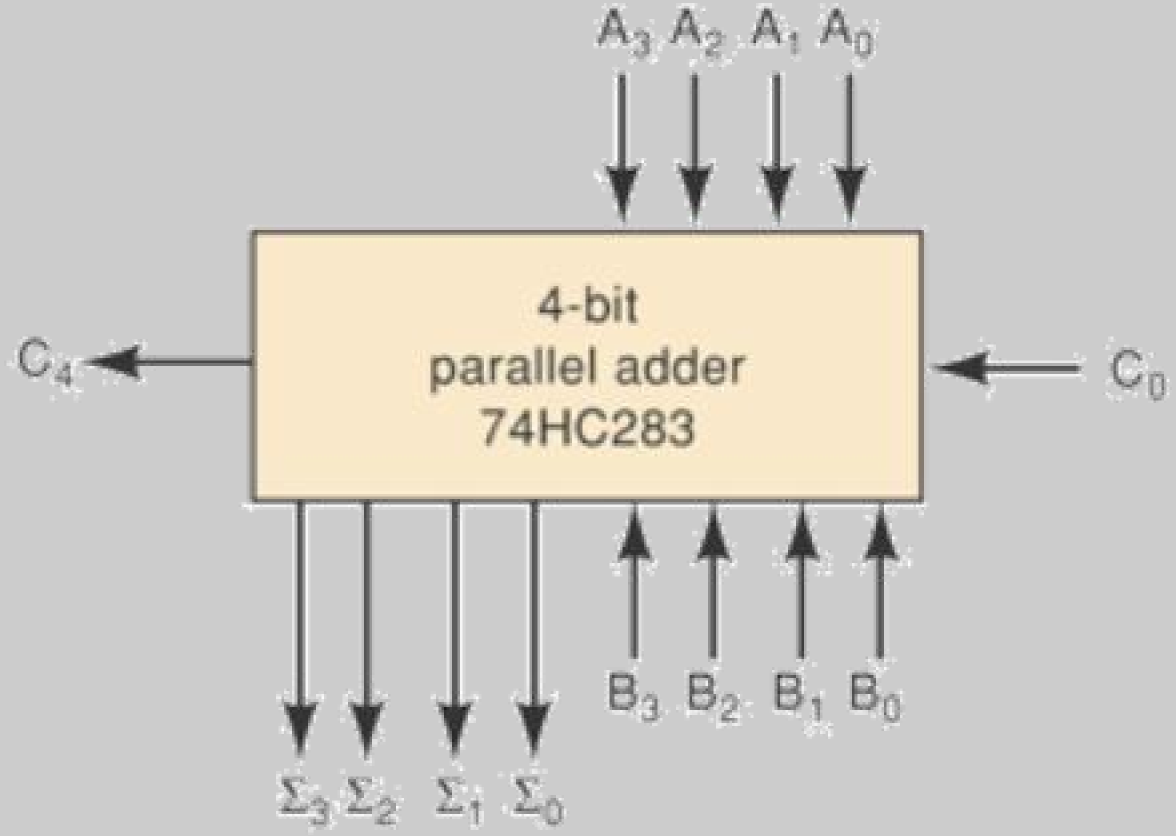
3) [S] → [A]. With [B] = 1001 and [A] = 0000, the full adders produce a sum of 1001 which is transferred to A register on the PGT of the TRANSFER pulse at t_3. This makes [A] = 1001.

4) [M] → [B]. The second binary number 0101 is transferred from memory to B register on the PGT of the second LOAD pulse at t_4. This makes [B] = 0101.

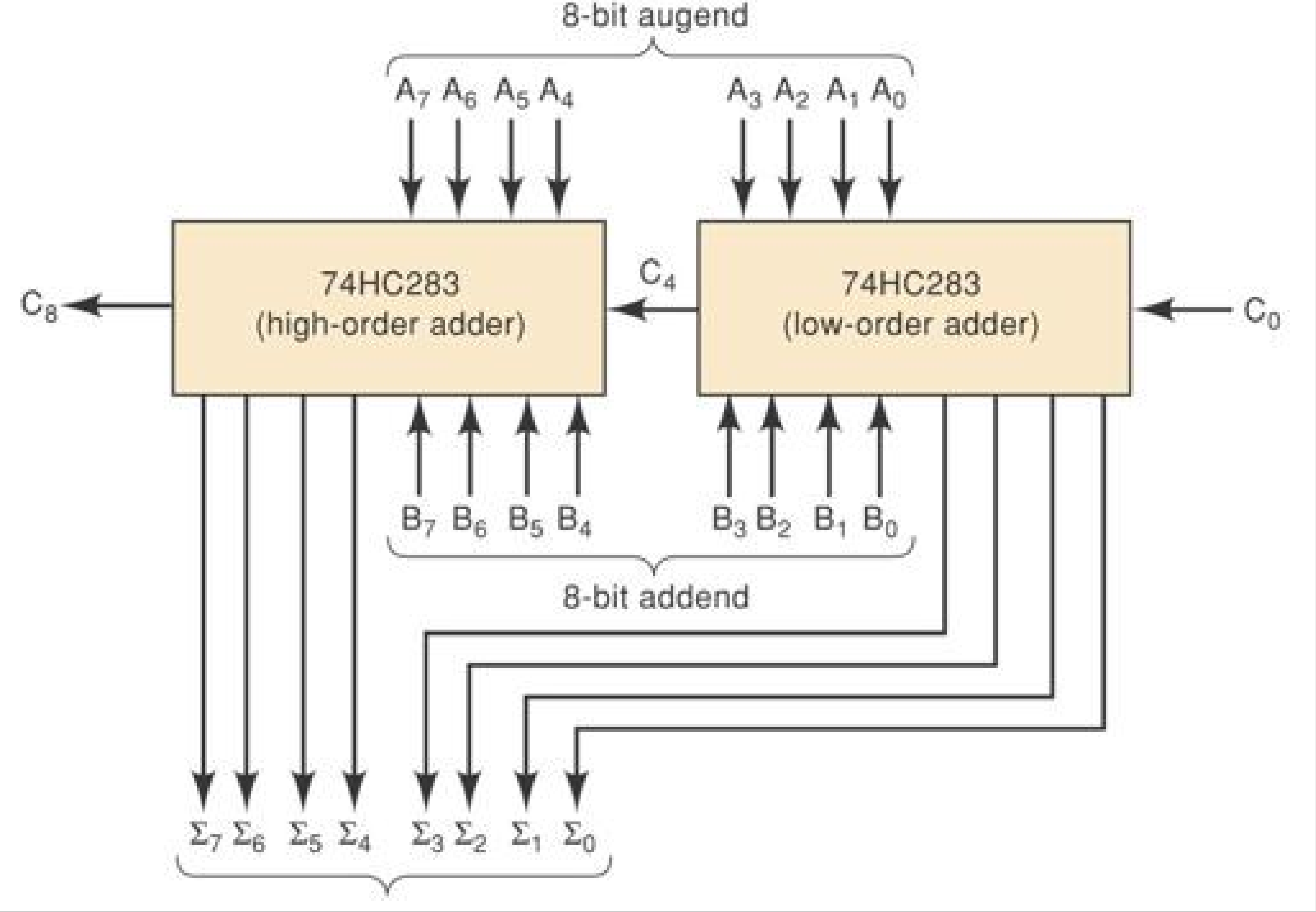
5) [S] → [A]. With [B] = 0101 and [A] = 1001, the full adders produce a sum of 1110 which is transferred to A register on the PGT of the second TRANSFER pulse at t_5. This makes [A] = 1110.

Integrated Circuit Parallel Adder
(IC Parallel Adder)

4- Bits adder



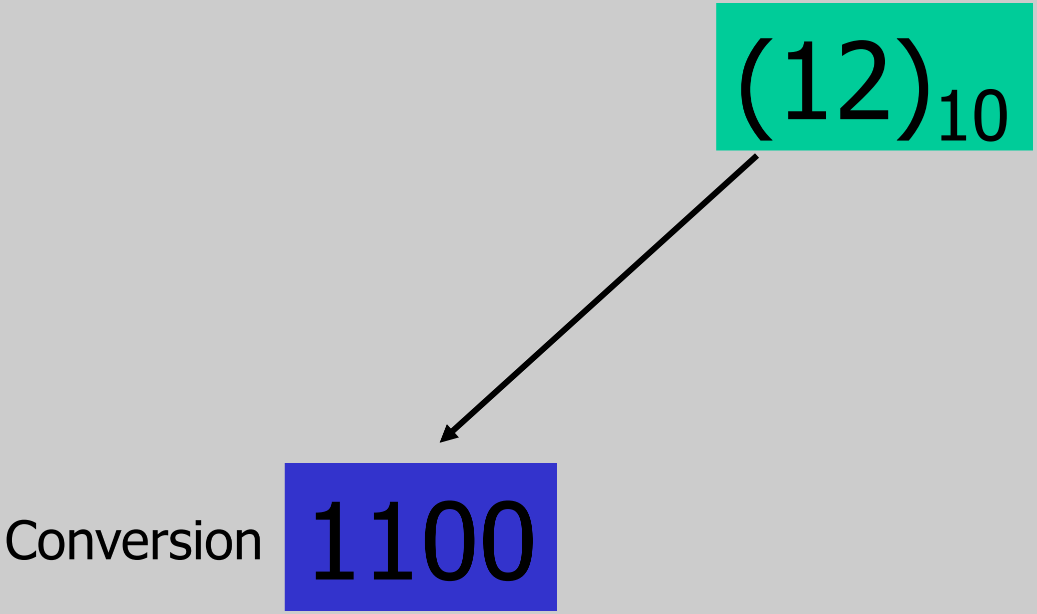
8 – Bit IC Parallel Adder



Conversion and Coding

$(12)_{10}$

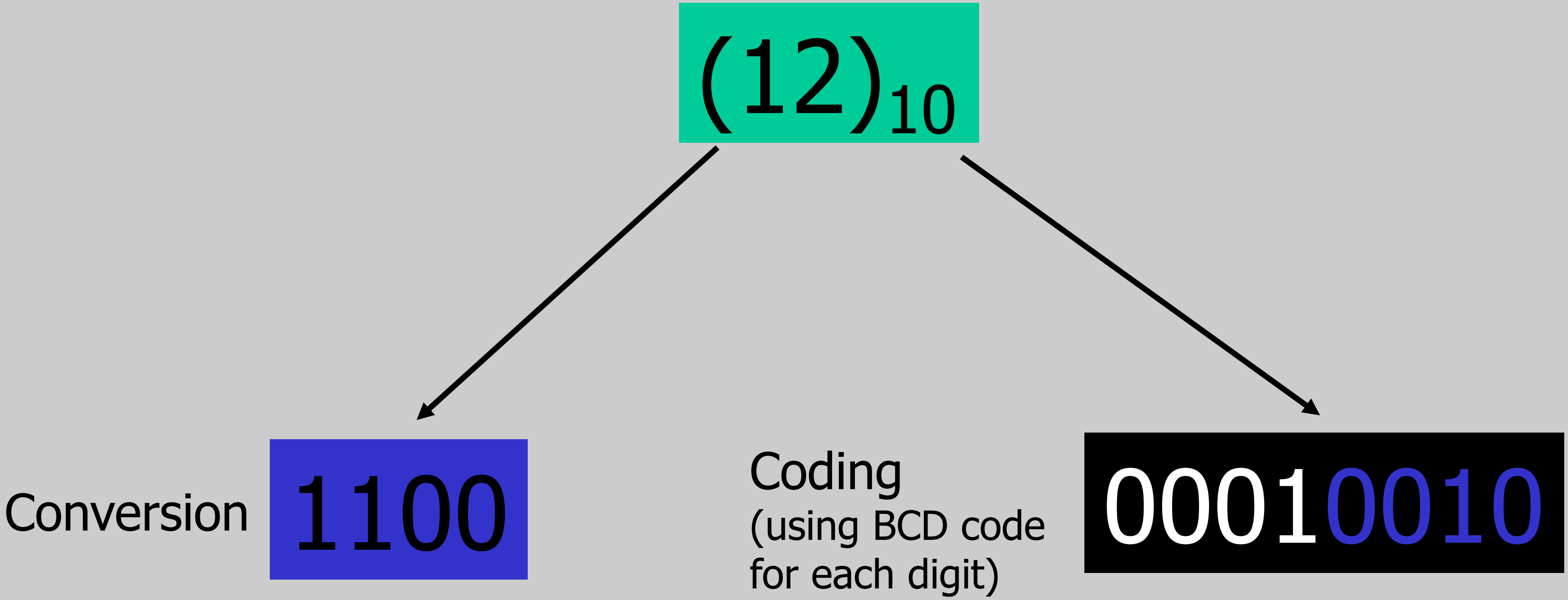
Conversion and Coding



$(12)_{10}$

Conversion 1100

Conversion and Coding



$(12)_{10}$

Conversion 1100

Coding
(using BCD code
for each digit)

00010010

BCD Adder

Design a circuit that calculates the Arithmetic addition of two decimal digits.

$$\begin{array}{ccr} & & 9 \\ & + & 3 \\ \hline 1 & & 2 \end{array}$$

carry

BCD Adder

Design a circuit that calculates the Arithmetic addition of two decimal digits.

$$\begin{array}{r} 6 \\ +\ 3 \\ \hline 9 \end{array}
\qquad
\begin{array}{rl} 0110 & \rightarrow \text{BCD} \\ +\ 0011 & \text{BCD} \\ \hline \boxed{1001} & \text{BCD} \end{array}
\qquad \Bigg| \qquad
\begin{array}{r} 9 \\ +\ 3 \\ \hline 12 \end{array}
\qquad
\begin{array}{rl} 1001 & \text{BCD} \\ +\ 0011 & \text{BCD} \\ \hline \boxed{1100} & \text{not BCD} \end{array}$$

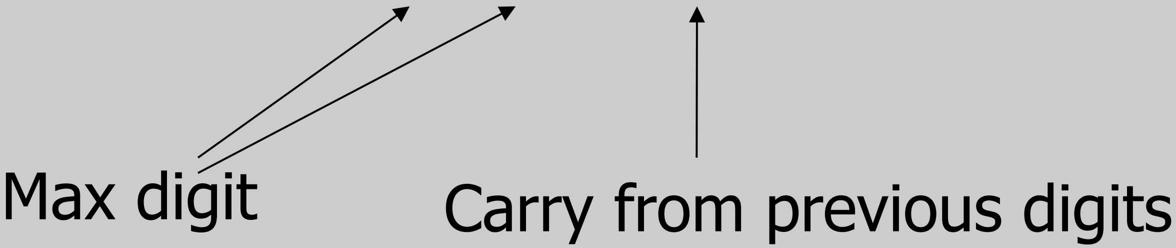
- If sum is up to 9: Use the regular Adder.
- If the sum > 9: Use the regular adder and add 6 to the result

Correction Adder

$$\begin{array}{r} 1 \\ 1100 \\ +\ 0110 \\ \hline 0001\ 0010 \\ 1 \qquad\quad 2 \end{array}$$

BCD Adder

- Maximum sum is 9+9 + 1 = 19



Max digit

Carry from previous digits

BCD adder (sum up to 9)

Number	C	S8	S4	S2	S1
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1

BCD adder (sum up to 9)

Number	C	S8	S4	S2	S1
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1

The sum is the same with BCD

BCD adder (sum is 10 to 19)

Number	C	S8	S4	S2	S1
10	1	0	0	0	0
11	1	0	0	0	1
12	1	0	0	1	0
13	1	0	0	1	1
14	1	0	1	0	0
15	1	0	1	0	1
16	1	0	1	1	0
17	1	0	1	1	1
18	1	1	0	0	0
19	1	1	0	0	1

BCD adder (sum is 10 to 19)

BCD adder sum

Number	C	S8	S4	S2	S1
10	1	0	0	0	0
11	1	0	0	0	1
12	1	0	0	1	0
13	1	0	0	1	1
14	1	0	1	0	0
15	1	0	1	0	1
16	1	0	1	1	0
17	1	0	1	1	1
18	1	1	0	0	0
19	1	1	0	0	1

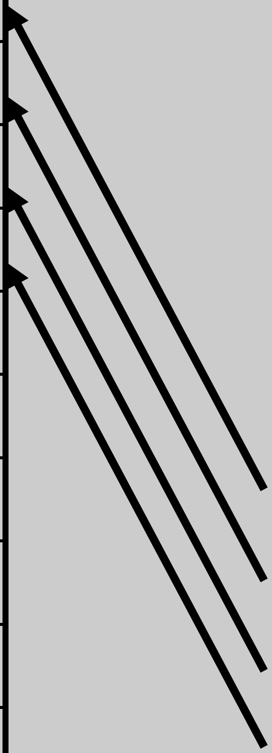
Binary sum

K	Z8	Z4	Z2	Z1
0	1	0	1	0
0	1	0	1	1
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1

BCD adder (sum is 10 to 19)

BCD adder sum

Number	C	S8	S4	S2	S1
10	1	0	0	0	0
11	1	0	0	0	1
12	1	0	0	1	0
13	1	0	0	1	1
14	1	0	1	0	0
15	1	0	1	0	1
16	1	0	1	1	0
17	1	0	1	1	1
18	1	1	0	0	0
19	1	1	0	0	1



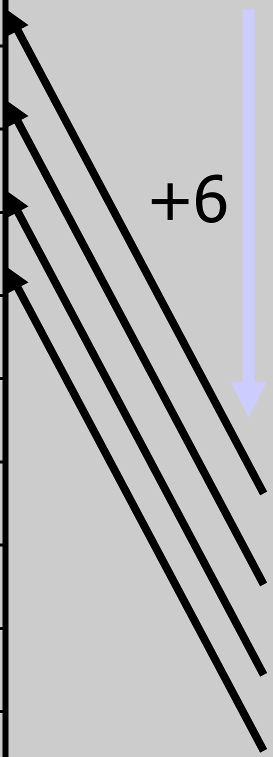
Binary sum

K	Z8	Z4	Z2	Z1
0	1	0	1	0
0	1	0	1	1
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1

BCD adder (sum is 10 to 19)

BCD adder sum

Number	C	S8	S4	S2	S1
10	1	0	0	0	0
11	1	0	0	0	1
12	1	0	0	1	0
13	1	0	0	1	1
14	1	0	1	0	0
15	1	0	1	0	1
16	1	0	1	1	0
17	1	0	1	1	1
18	1	1	0	0	0
19	1	1	0	0	1



Binary sum

K	Z8	Z4	Z2	Z1
0	1	0	1	0
0	1	0	1	1
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1

Algorithm for BCD Adder

- If sum is up to 9
  - Use the regular Adder.

- If the sum > 9
  - Use the regular adder and add 6 to the result

When is the result > 9

Binary sum

Number	K	Z8	Z4	Z2	Z1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1

C = K +

When is the result > 9

Binary sum

Number	K	Z8	Z4	Z2	Z1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1

$C = K + Z8*Z4-$

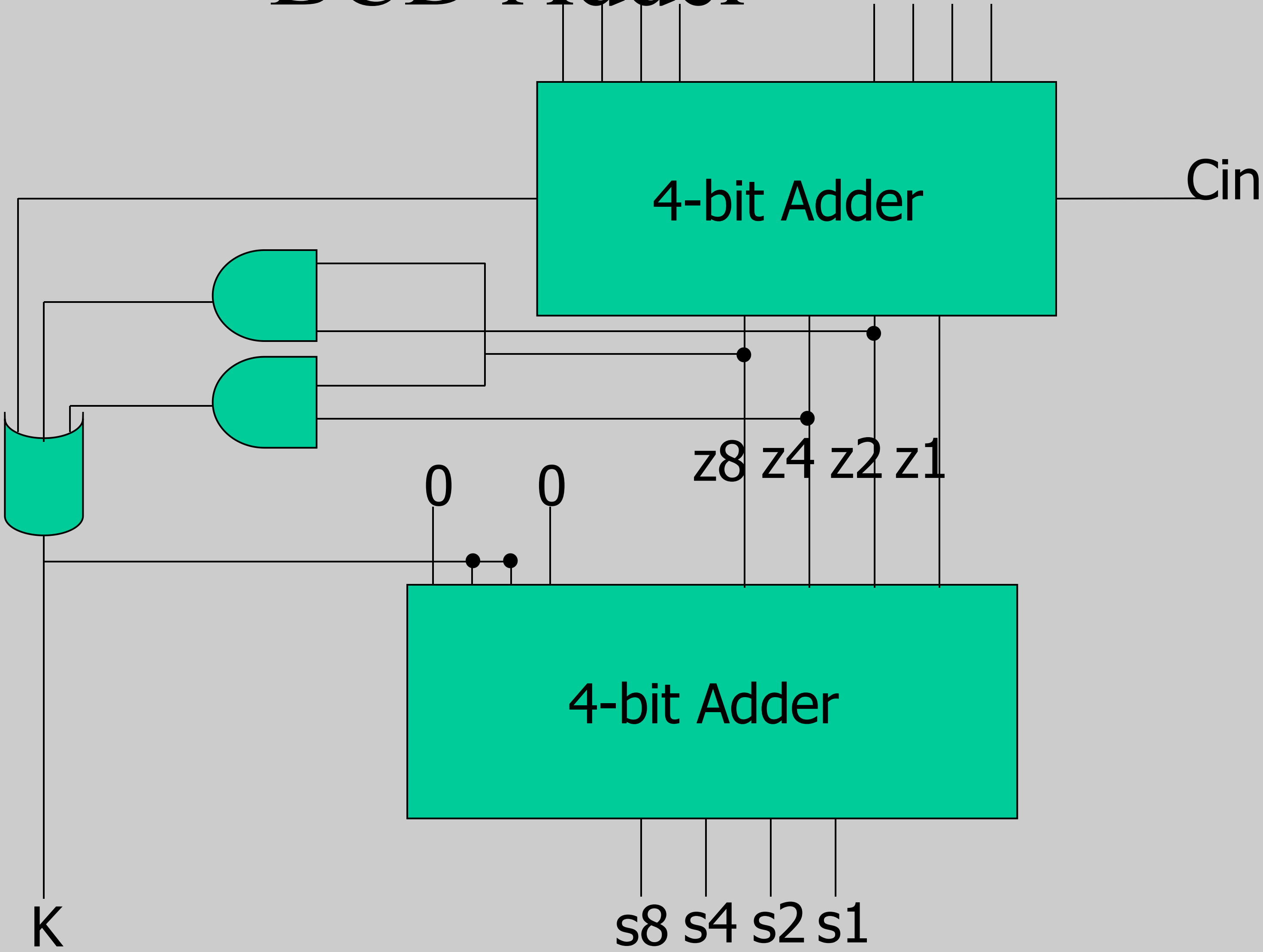
When is the result > 9

Binary sum

Number	K	Z8	Z4	Z2	Z1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1

C = K + Z8*Z4
Z8*Z2

BCD Adder



BCD Adder

BCD Adder

- When the sum of two digits is less than or equal to 9 then the ordinary 4-bit adder can be used
- But if the sum of two digits is greater than 9 then a correction must be added “I.e adding 0110”
- We need to design a circuit that is capable of doing the correct addition

BCD Adder

- The cases where the sum of two 4-bit numbers is greater than 9 are in the following table:

S4	S_3	S_2	S_1	S_0	
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18

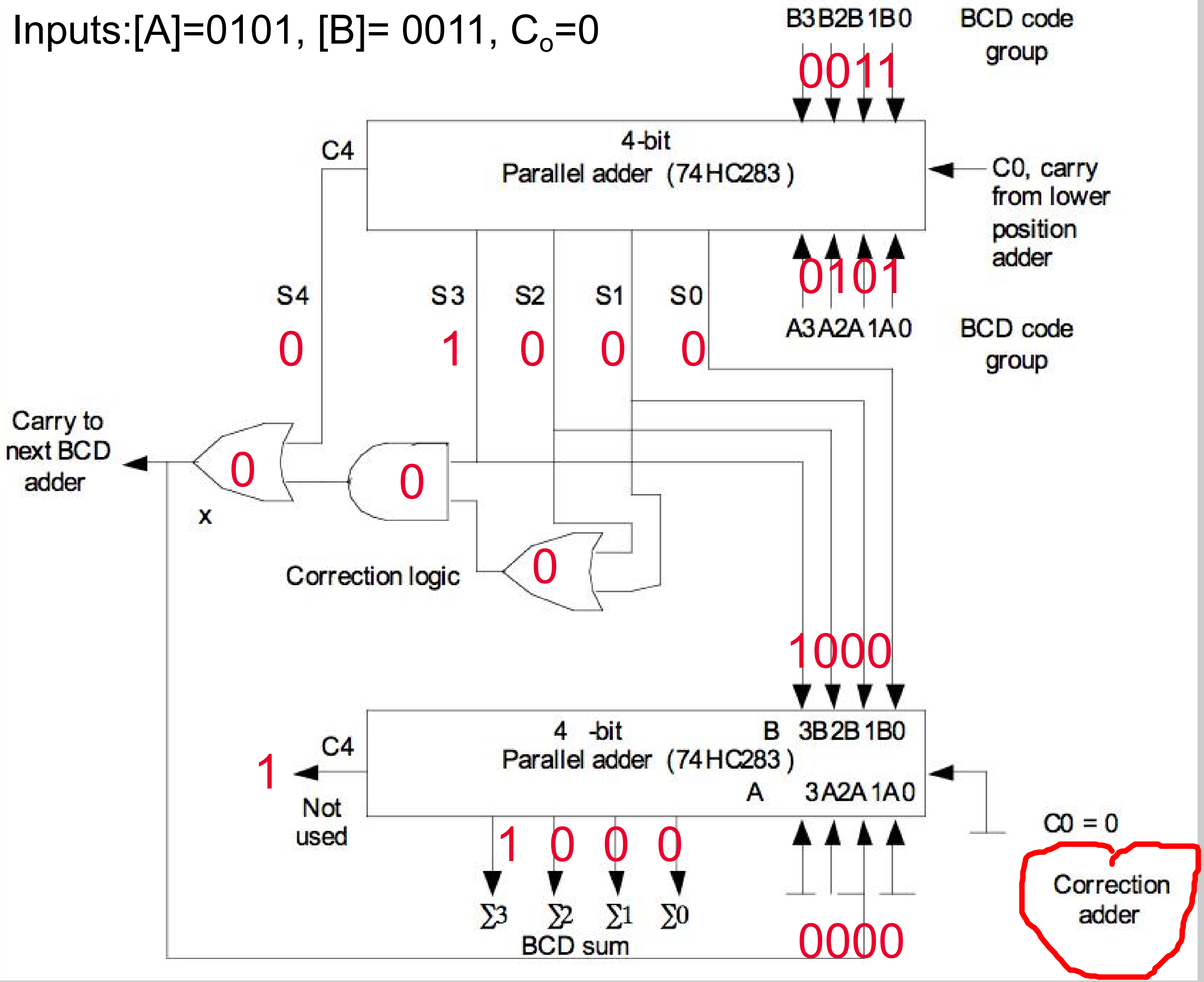
BCD Adder

- Whenever S_4=1 (sums greater than 15)
- Whenever S_3=1 and either S_2 or S_1 or both are 1 (sums 10 to 15)
- The previous table can be expressed as:

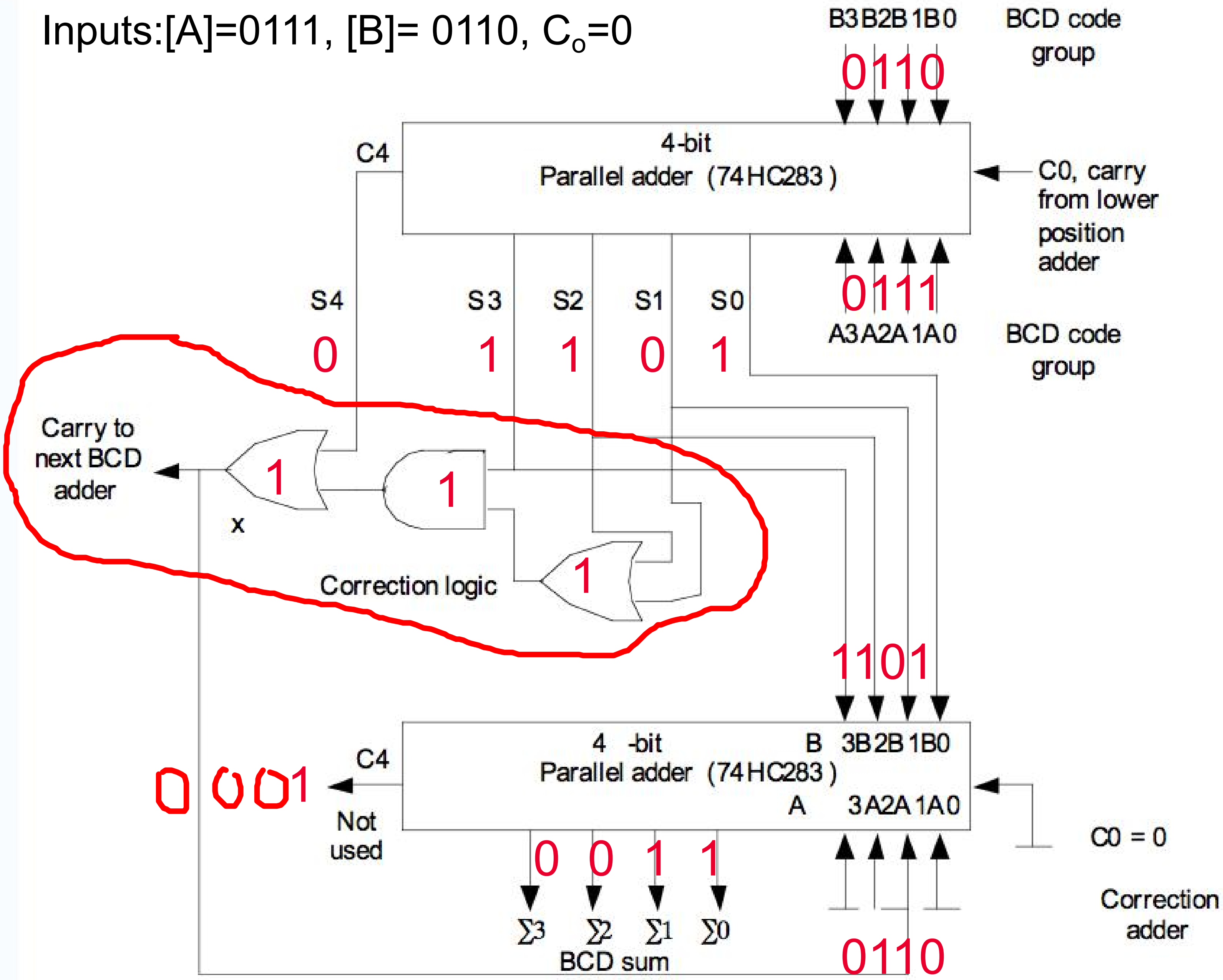
$$X = S_4 + S_3 (S_2 + S_1)$$

So, whenever X = 1 we should add a correction of 0110 to the sum.

Inputs:[A]=0101, [B]= 0011, C_o=0

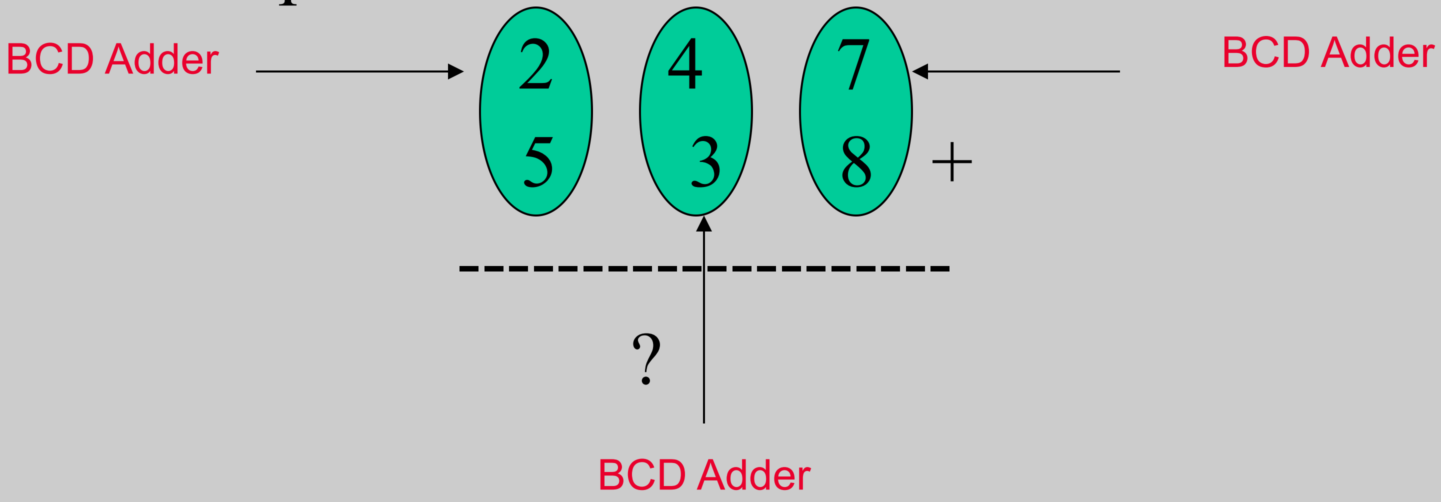


Inputs:[A]=0111, [B]= 0110, C_o=0



Cascading BCD Adders

- The previous circuit is used for adding two decimal digits only. That is, “ 7 + 6 = 13”.
- For adding numbers with several digits, a separate BCD adder for each digit position must be used.
- For example:



BCD Adder

2 4 7

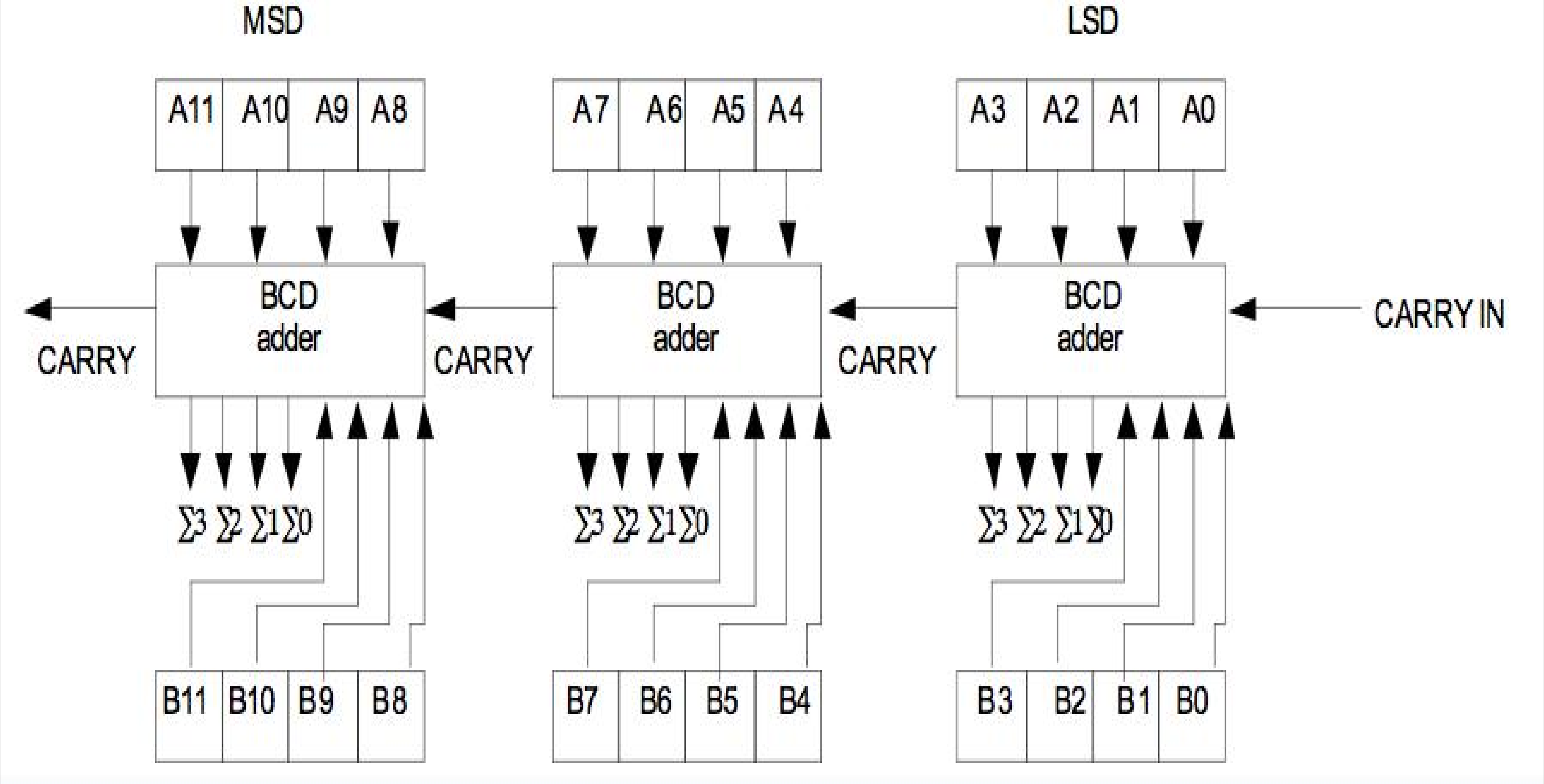
5 3 8 +

?

BCD Adder

BCD Adder

Cascading BCD Adders



Example

- Determine the inputs and the outputs when the above circuit is used to add 538 to 247. Assume a CARRY IN = 0
- Solution:
  - Represent the decimal numbers in BCD

    247 = 0010 0100 0111

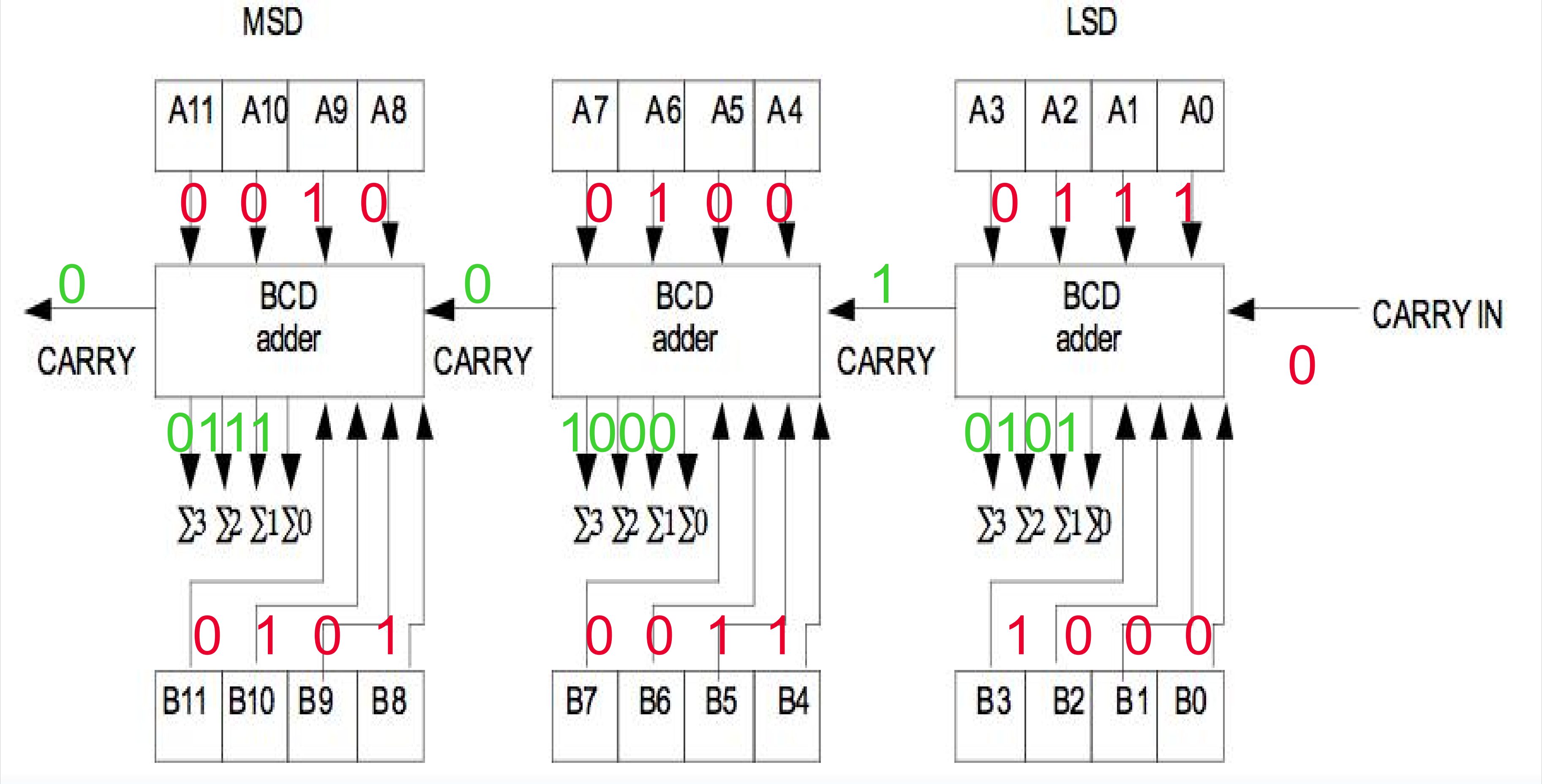
    538 = 0101 0011 1000

    Put these numbers in registers [A] and [B]

    [A] = 0010 0100 0111

    [B] = 0101 0011 1000

Example


MSD
LSD
A11 A10 A9 A8
A7 A6 A5 A4
A3 A2 A1 A0
0 0 1 0
0 1 0 0
0 1 1 1
0
CARRY
BCD adder
0
CARRY
BCD adder
1
CARRY
BCD adder
CARRY IN
0
0111
1000
0101
∑3 ∑2 ∑1 ∑0
∑3 ∑2 ∑1 ∑0
∑3 ∑2 ∑1 ∑0
0 1 0 1
0 0 1 1
1 0 0 0
B11 B10 B9 B8
B7 B6 B5 B4
B3 B2 B1 B0

Two's compliment

- But how do you represent a minus sign electronically in a computer?
- How can you represent it such that arithmetic operations are manageable?
- There are two types of compliments for each number base system.
  - Have the r's complement
  - Have the (r-1)'s complement
- For base 2 have 2's complement and 1's complement

1's Complement

- 1's complement of N is defined as $(2^n - 1) - N$.
  - If n=4 have $(2^n - 1)$ being 1 0000 - 1 = 1111
- So for n=4 would subtract any 4-bit binary number from 1111.
- This is just inverting each bit.
- Example: 1's compliment of 1011001
- is 0100110

1's Complement

For a negative number, we take the positive number and complement every bit.

2's complement

- The 2's complement is defined as 2^n-N
- Can be done by subtraction of N from 2^n or adding 1 to the 1's complement of a number.
- For 6 = 0110
  - The 1's complement is 1001
  - The 2's complement is 1010 (1001+1=1010)

1's Complement

For a negative number, we take the positive number and complement every bit.

2's Complement

For a negative number, we do 1's complement and plus one.

Operation with 2’s complement

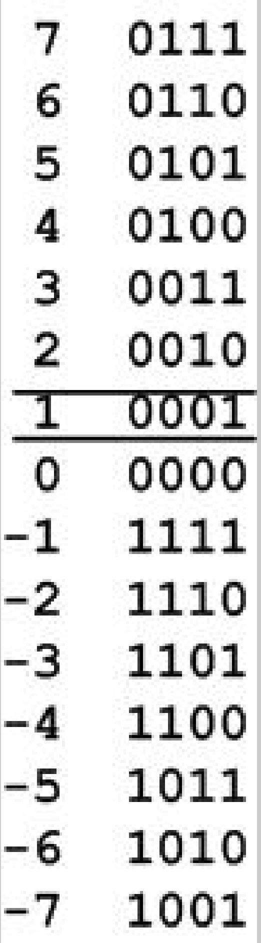
- Add 4 and -6
- Will use the 2’s complement of -6 or 1010

$$\begin{array}{rrrr} & 4 & & 0100 \\ + & \underline{-6} & + & \underline{1010} \\ & -2 & & 1110 \end{array}$$

- And taking the 2’s complement of 1110 get 0001 + 1 = 0010

A 2’s complement table for 4 bits

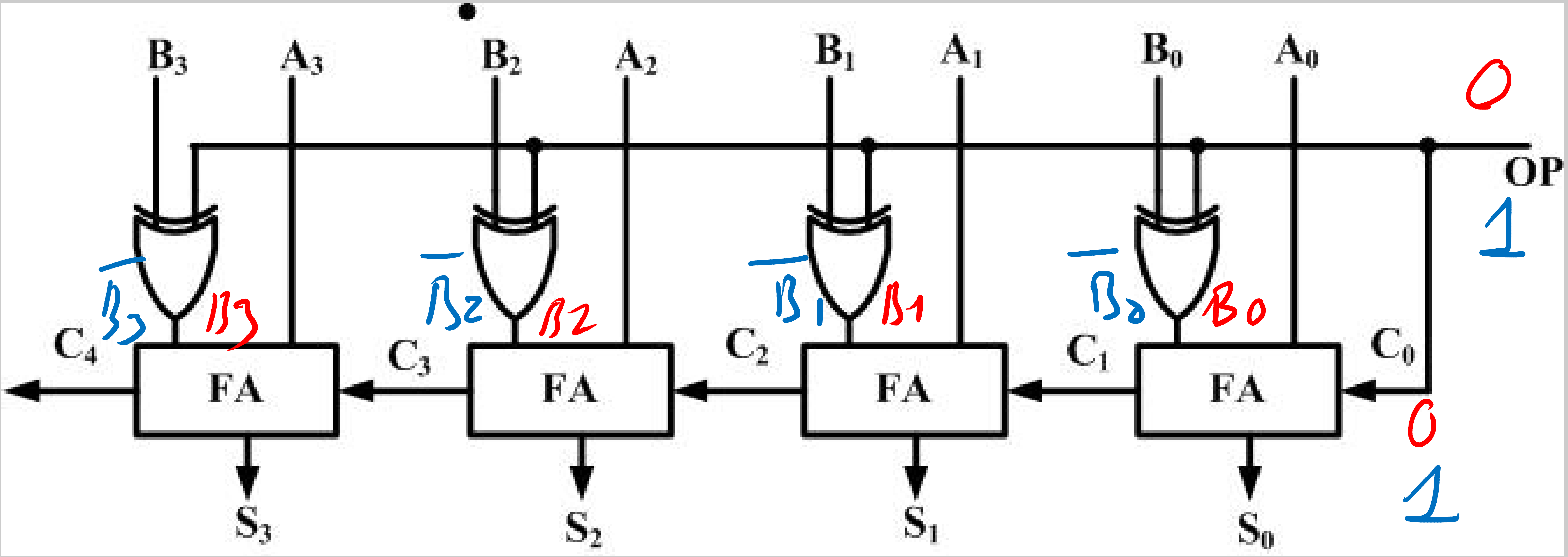
- Listing the values represented.



7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001

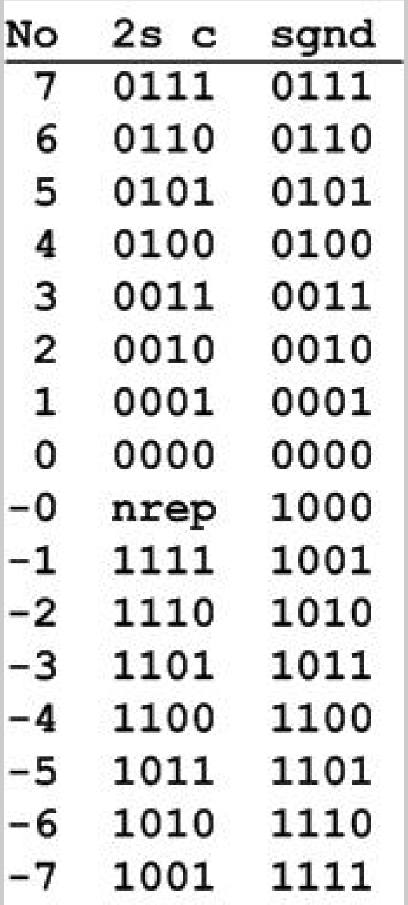
A circuit that does +/-

- A general adder subtractor
- OP=0 for addition/ =1 for subtraction



Another number format

- Signed magnitude – use the MSB to indicate the sign. The remaining bits indicate the magnitude.



No	2s c	sgnd
7	0111	0111
6	0110	0110
5	0101	0101
4	0100	0100
3	0011	0011
2	0010	0010
1	0001	0001
0	0000	0000
-0	nrep	1000
-1	1111	1001
-2	1110	1010
-3	1101	1011
-4	1100	1100
-5	1011	1101
-6	1010	1110
-7	1001	1111

Overflow

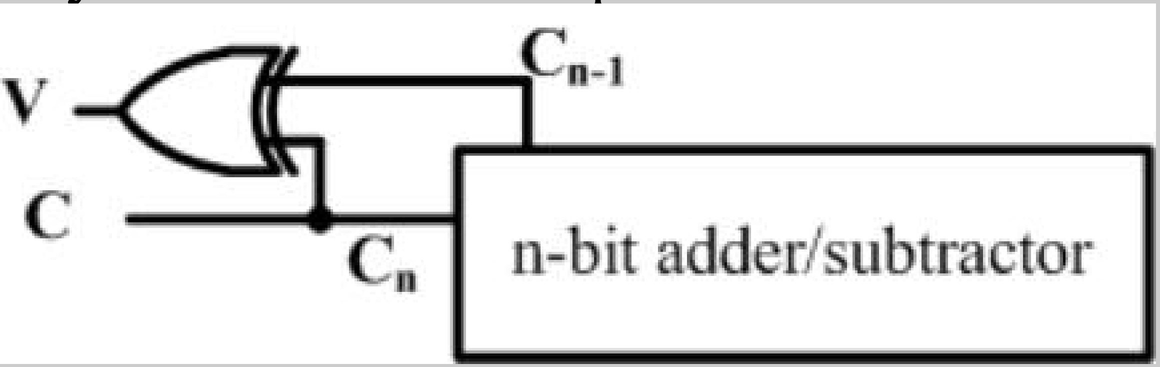
- When adding 2 n-bit numbers it is possilbe to get a n+1 bit result if there is a carry out.
- On paper it is easy just add another bit.
- In 2's complement add a msb 0 for a positive or a msb 1 for a negative.
- In a computer the number of bits that can be used is fixed.

Overflow indication.

- In 8-bit 2’s complement notation the range that can be represented is -127 to +127.
- Then the operation to add +70 to +80 is
  - Carries 0 1
  - +70 0 100 0110
  - +80 0 101 0000
  - +150 1 001 0110
- Also look at the addition of -70 and -80

The other addition

- The addition of -70 and -80
  - Carries 1 0
  - -70 1 011 1010
  - -80 1 011 0000
  - -150 0 110 1010
- The rule – if the carry into the msb position differs from the carry out from the msb position then an overflow h
- The circui
- .


C_{n-1}
V
C
C_n
n-bit adder/subtractor