BINARY ADDER–SUBTRACTOR

The most basic arithmetic operation is the addition of two binary digits.

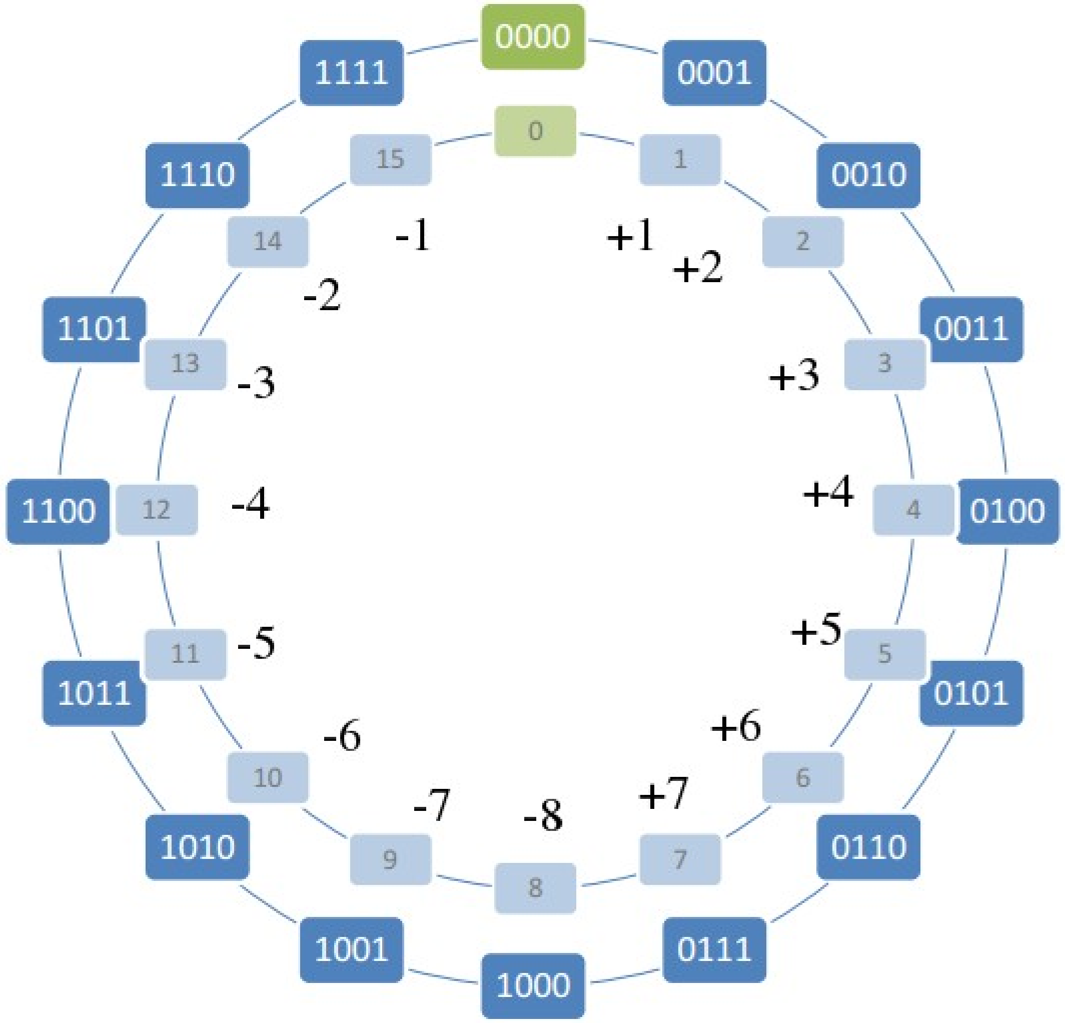
This simple addition consists of four possible elementary operations: 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, and 1 + 1 = 10.

The first three operations produce a sum of one digit, but when both augend and addend bits are equal to 1, the binary sum consists of two digits.

The higher significant bit of this result is called a carry.

- A 4-bit number system allows us to represent 16 values
- Ignoring carries from addition gives us modulo-16 arithmetic

- (**15** + 1) mod 16 = 0
  - and **-1** + 1 = 0
- (**14** + 2) mod 16 = 0
  - and **-2** + 2 = 0
- (**14** + 4) mod 16 = 2
  - and -2 +4 = 2



The two's complement of an N-bit number is defined as the result of subtracting the number from 2^N

So two's complement of 3 is 16 - 3 = 13

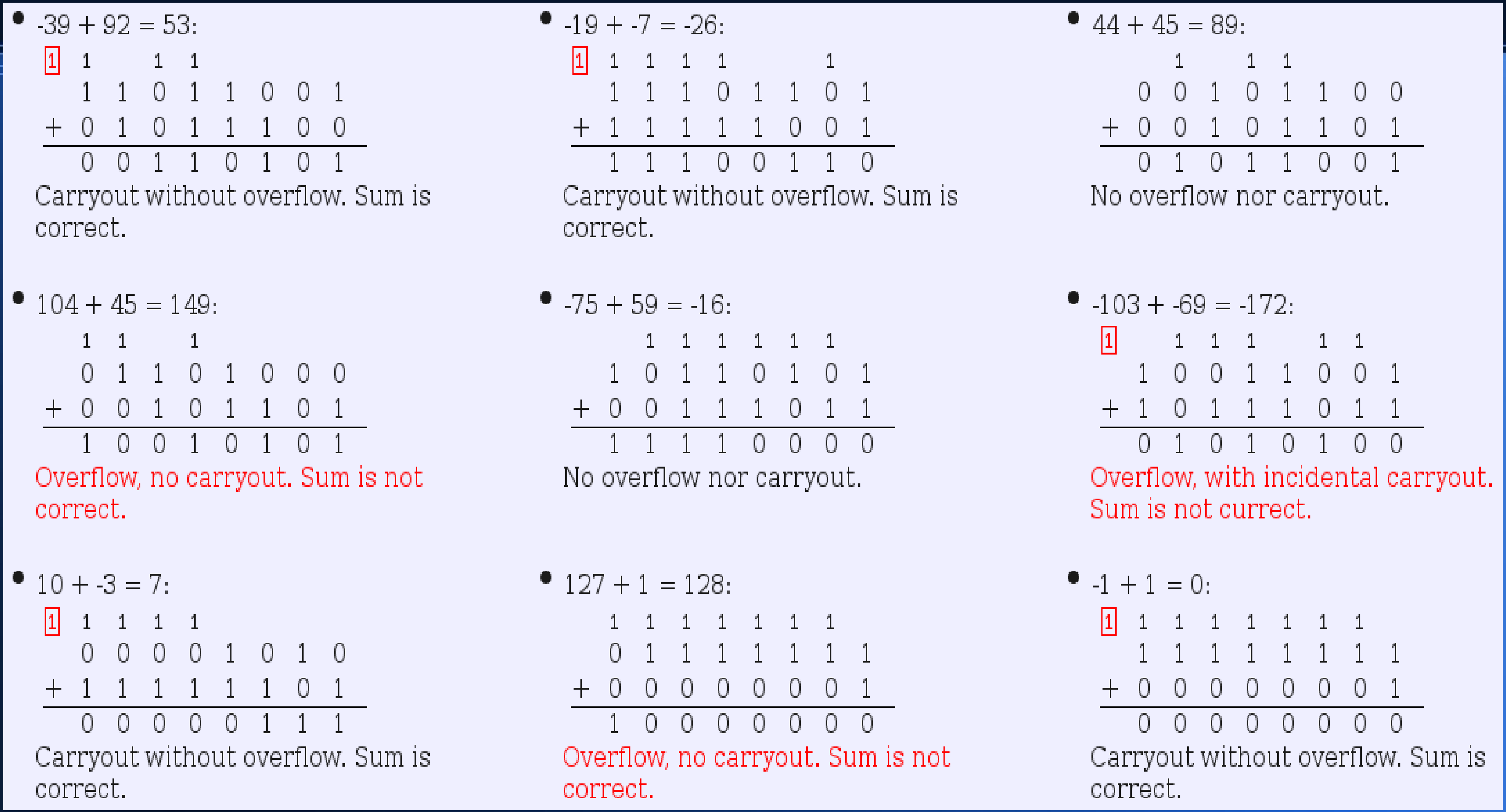
So

5 + -3
(5 + 16 – 3) Mod 16
(5 -3) Mod 16
2

The two's complement of an N-bit number is defined as the complement with respect to 2^N, in other words the result of subtracting the number from 2^N.

This is also equivalent to taking the ones' complement and then adding one, since the sum of a number and its ones' complement is all 1 bits.

BINARY ADDER–SUBTRACTOR



- -39 + 92 = 53:

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

Carryout without overflow. Sum is correct.

- -19 + -7 = -26:

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

Carryout without overflow. Sum is correct.

- 44 + 45 = 89:

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

No overflow nor carryout.

- 104 + 45 = 149:

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

Overflow, no carryout. Sum is not correct.

- -75 + 59 = -16:

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

No overflow nor carryout.

- -103 + -69 = -172:

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

Overflow, with incidental carryout. Sum is not currect.

- 10 + -3 = 7:

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

Carryout without overflow. Sum is correct.

- 127 + 1 = 128:

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

Overflow, no carryout. Sum is not correct.

- -1 + 1 = 0:

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

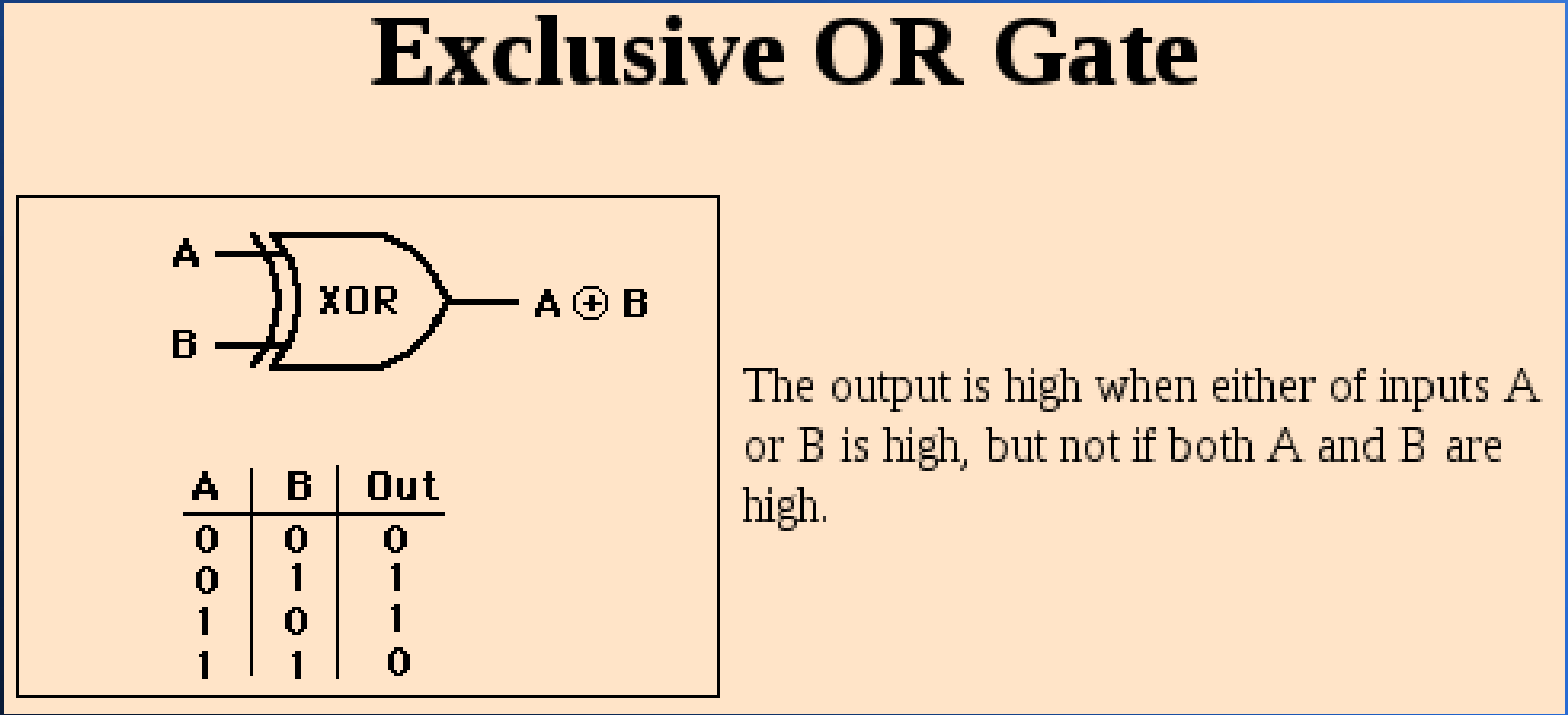
Carryout without overflow. Sum is correct.

Augend. The first of several addends, or "the one to which the others are added," is sometimes called the augend

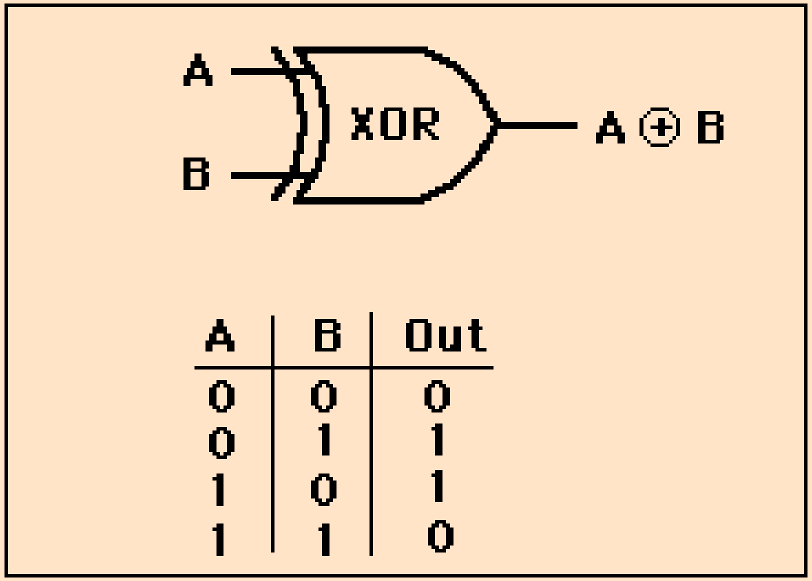
When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher order pair of significant bits.

A combinational circuit that performs the addition of two bits is called a half adder.

One that performs the addition of three bits (two significant bits and a previous carry) is a full adder. The names of the circuits stem from the fact that two half adders can be employed to implement a full adder



Exclusive OR Gate

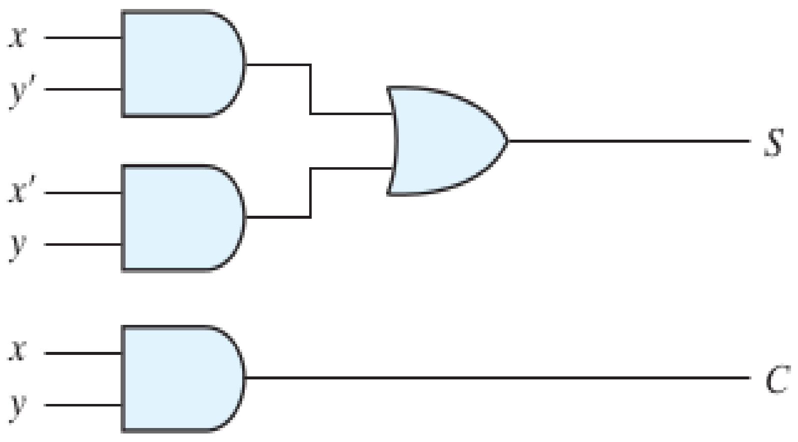


A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

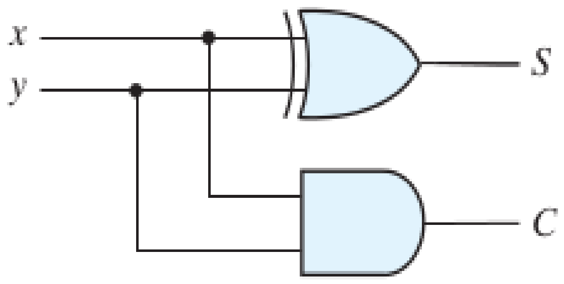
The output is high when either of inputs A or B is high, but not if both A and B are high.

Implementation of half adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

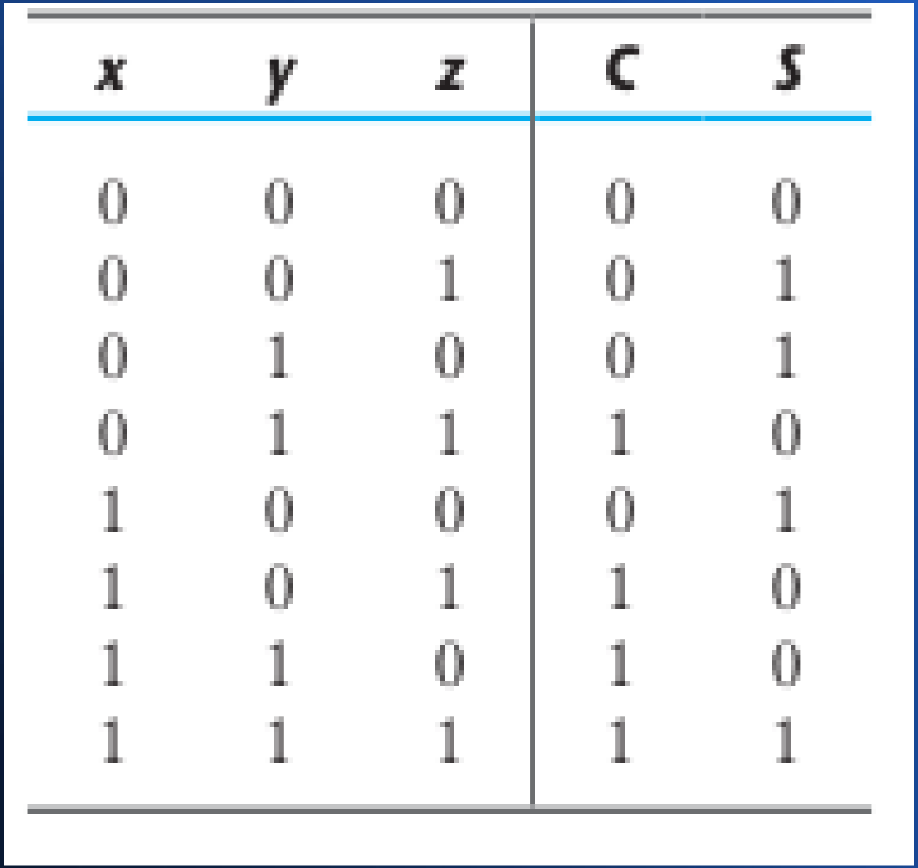


(a) $S = xy' + x'y$
$C = xy$



(b) $S = x \oplus y$
$C = xy$

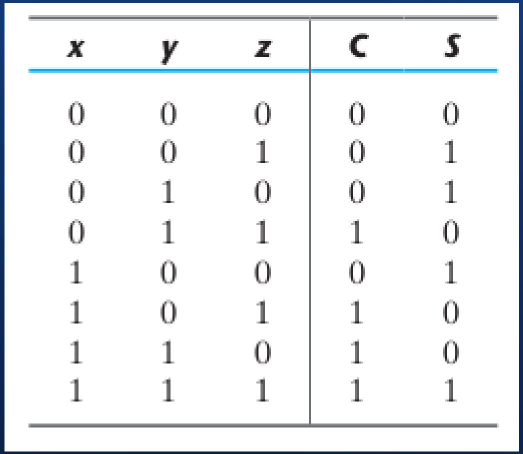
Full Adder



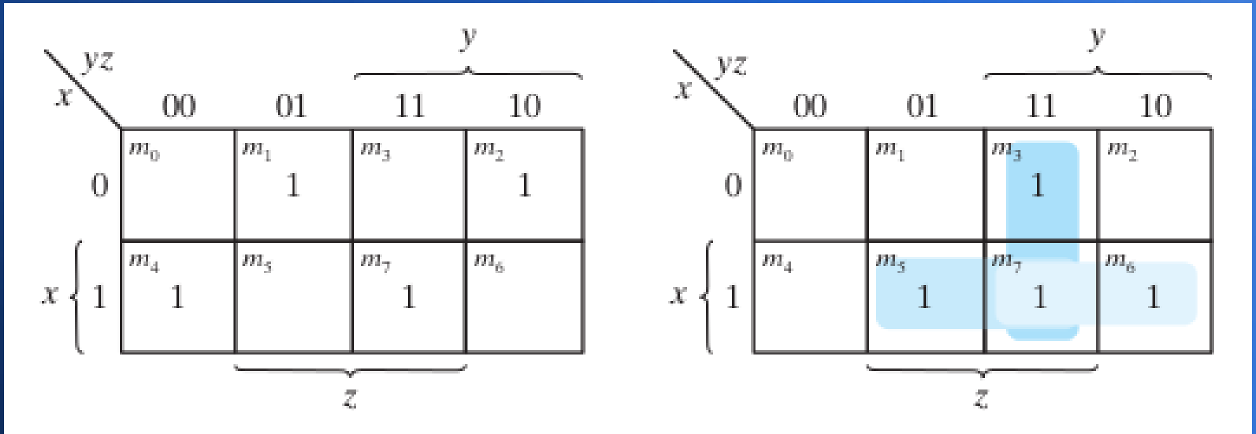
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$

Full Adder



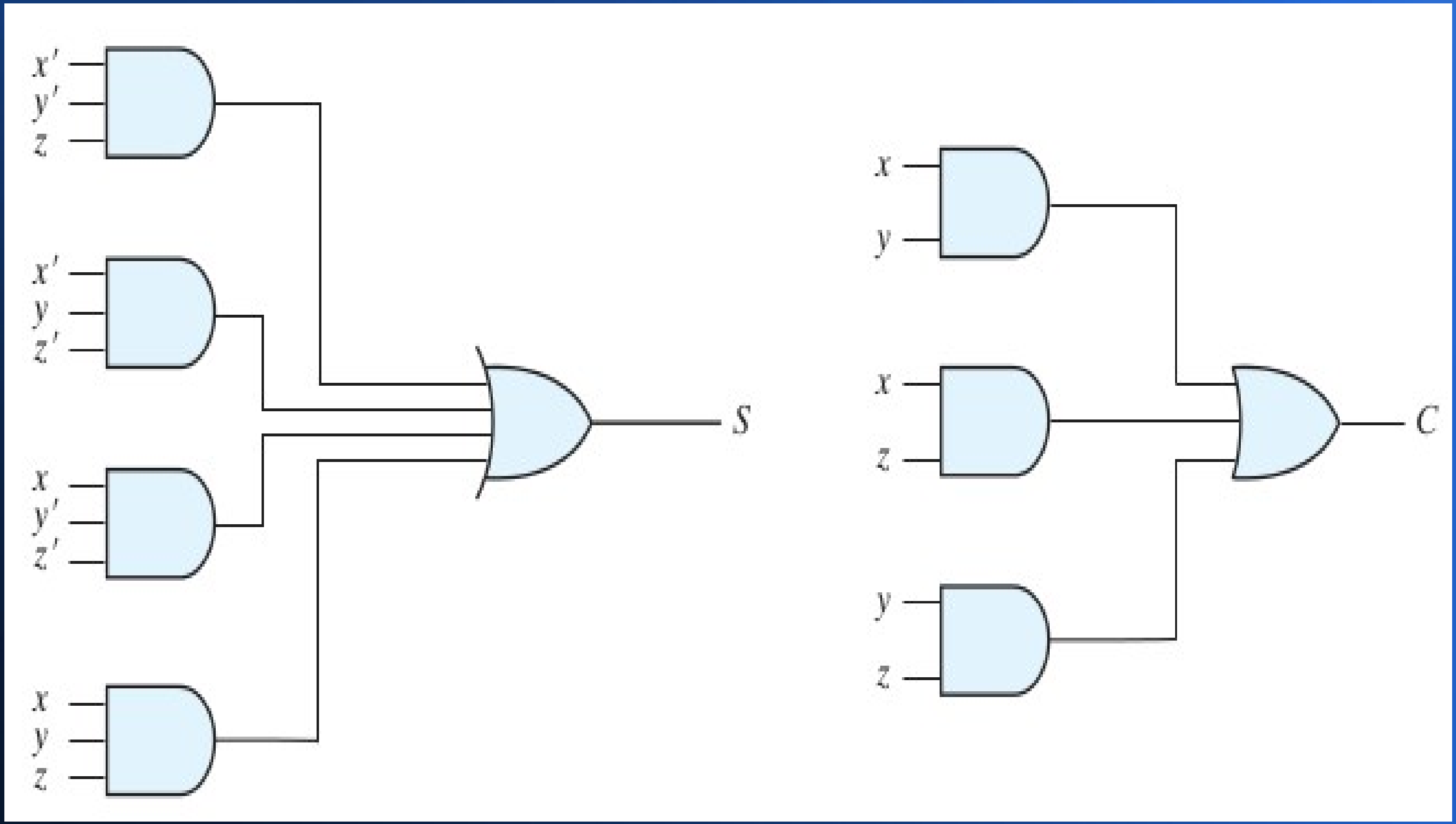
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



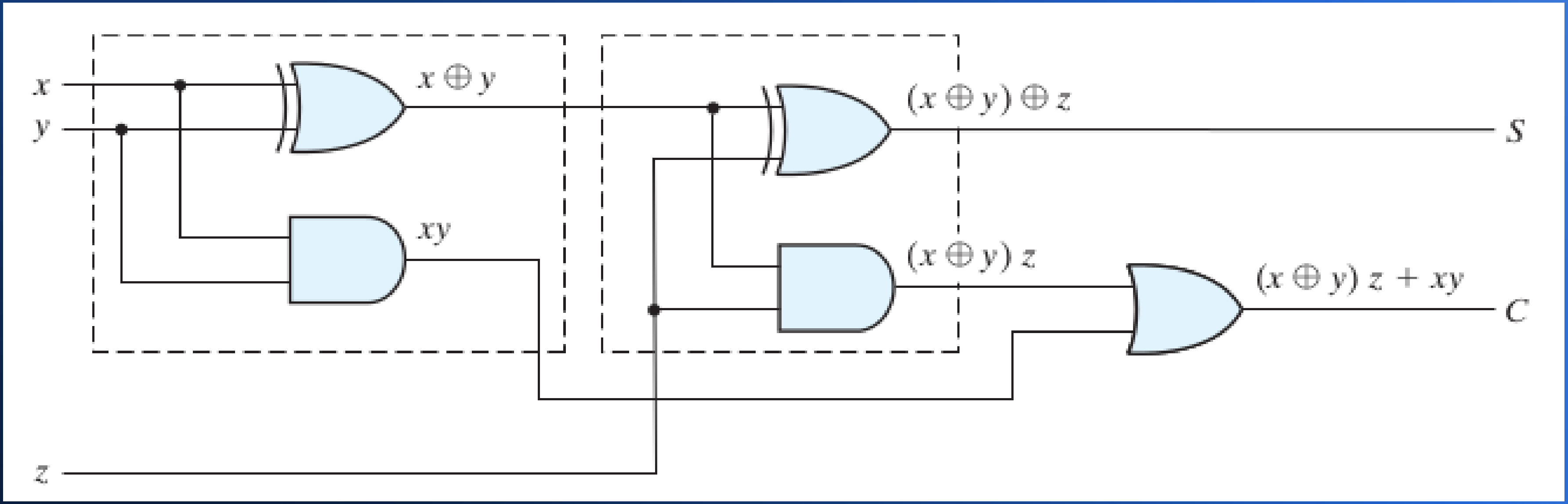
(a) $S = x'y'z + x'yz' + xy'z' + xyz$

(b) $C = xy + xz + yz$

Implementation of full adder in sum-of-products form



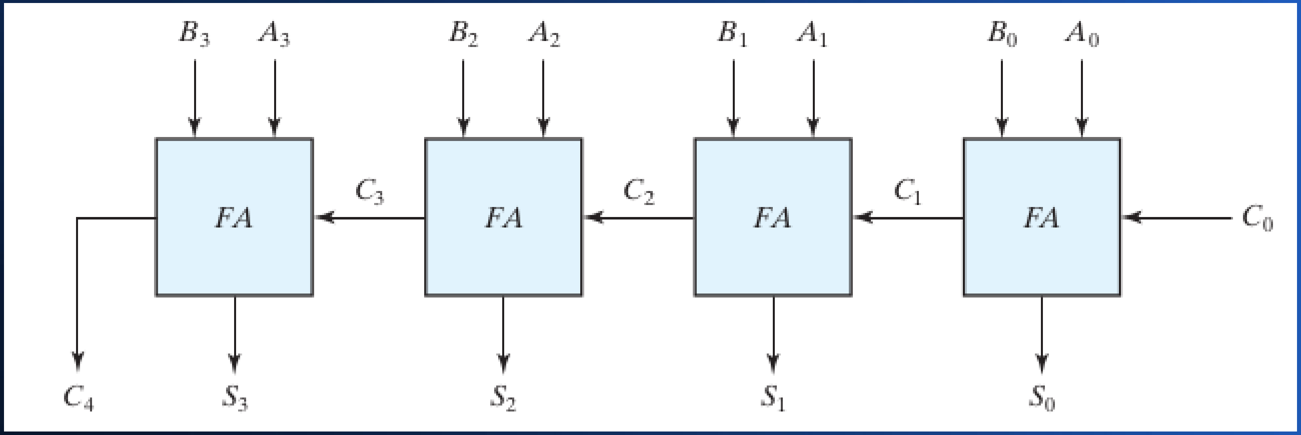
Implementation of full adder with two half adders and an OR gate

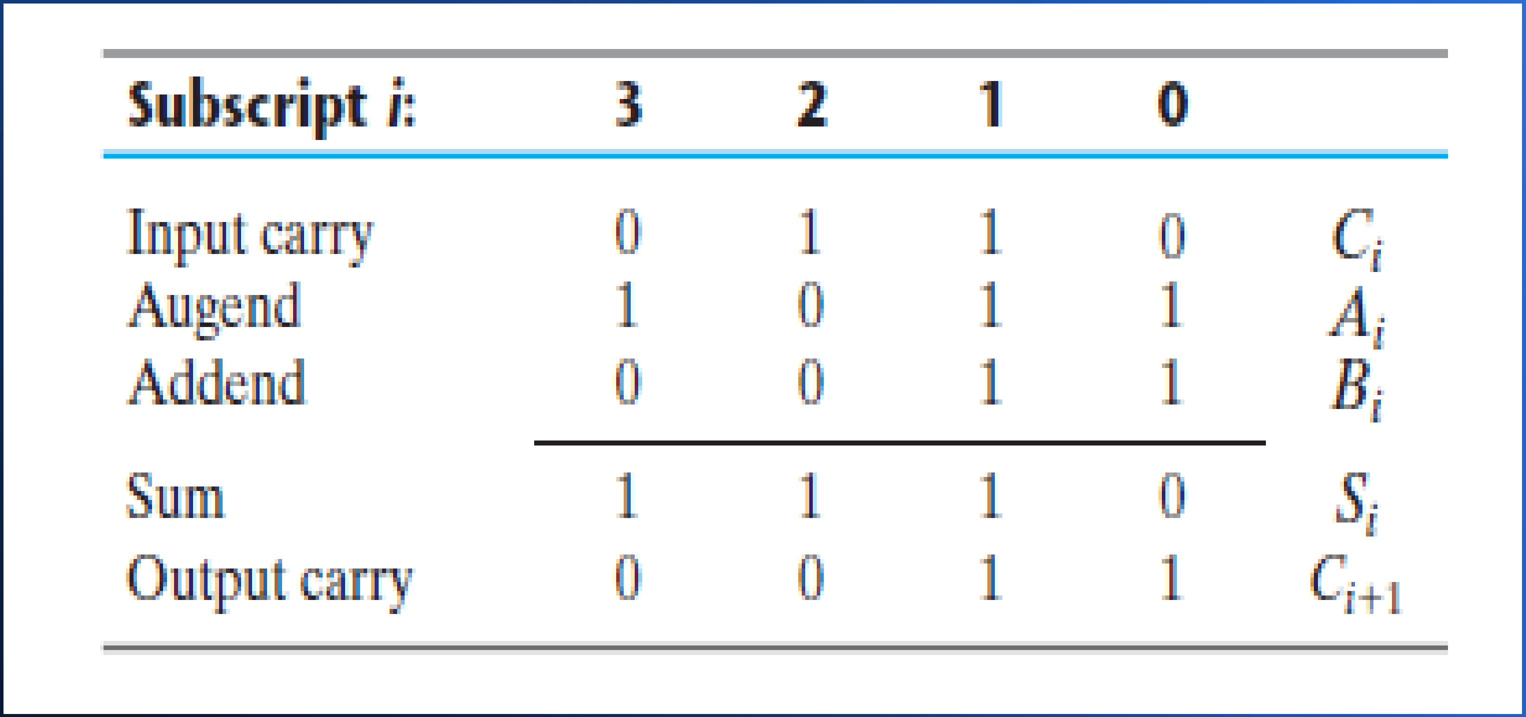


Addition of n-bit numbers requires a chain of n full adders or a chain of one-half adder and n - 1 full adders.

In the former case, the input carry to the least significant position is fixed at 0.

Interconnection of four full-adder (FA) circuits to provide a four-bit binary ripple carry adder.





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}

The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the least significant bit.

The carries are connected in a chain through the full adders. The input carry to the adder is C0, and it ripples through the full adders to the output carry C4.

The S outputs generate the required sum bits. An n-bit adder requires n full adders, with each output carry connected to the input carry of the next higher order full adder.

Carry Propagation

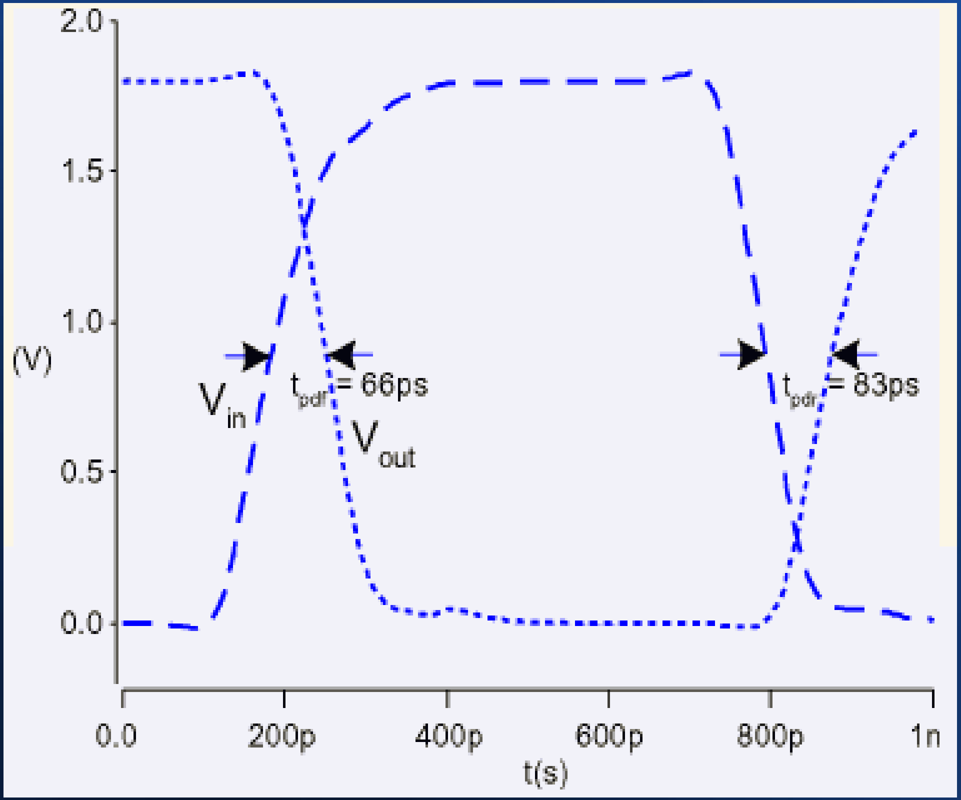
The addition of two binary numbers in parallel implies that all the bits of the augend and addend are available for computation at the same time.

As in any combinational circuit, the signal must propagate through the gates before the correct output sum is available in the output terminals.

The total propagation time is equal to the propagation delay of a typical gate, times the number of gate levels in the circuit.

The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adders.

Inverter Delay



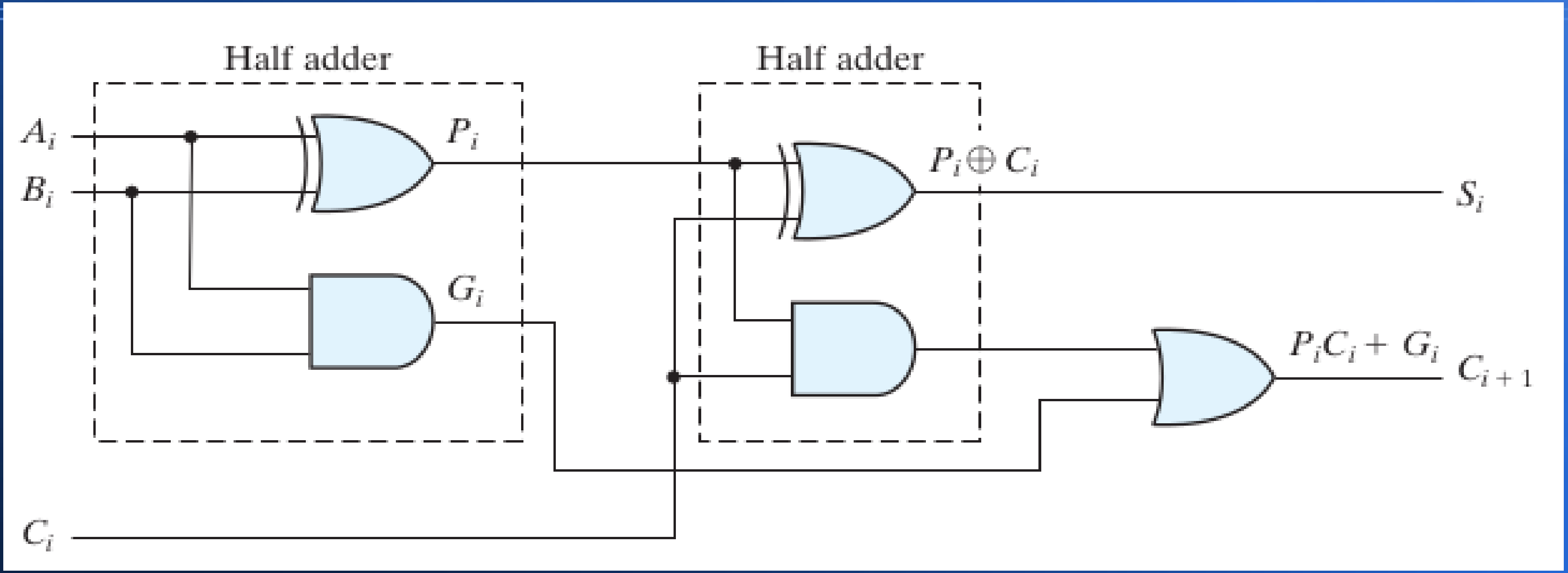
t_{pdr}: rising propagation delay

- From input to rising output crossing $V_{DD}/2$

t_{pdf}: falling propagation delay

- From input to falling output crossing $V_{DD}/2$

Full adder with P and G shown



The signals at Pi and Gi settle to their steady-state values after they propagate through their respective gates. These two signals are common to all half adders and depend on only the input augend and addend bits. The signal from the input carry Ci to the output carry Ci + 1 propagates through an AND gate and an OR gate, which constitute two gate levels. If there are four full adders in the adder, the output carry C4 would have 2 * 4 = 8 gate levels from C0 to C4. For an n-bit adder, there are 2n gate levels for the carry to propagate from input to output.

Gi is called a carry generate, and it produces a carry of 1 when both Ai and Bi are 1, regardless of the input carry Ci.

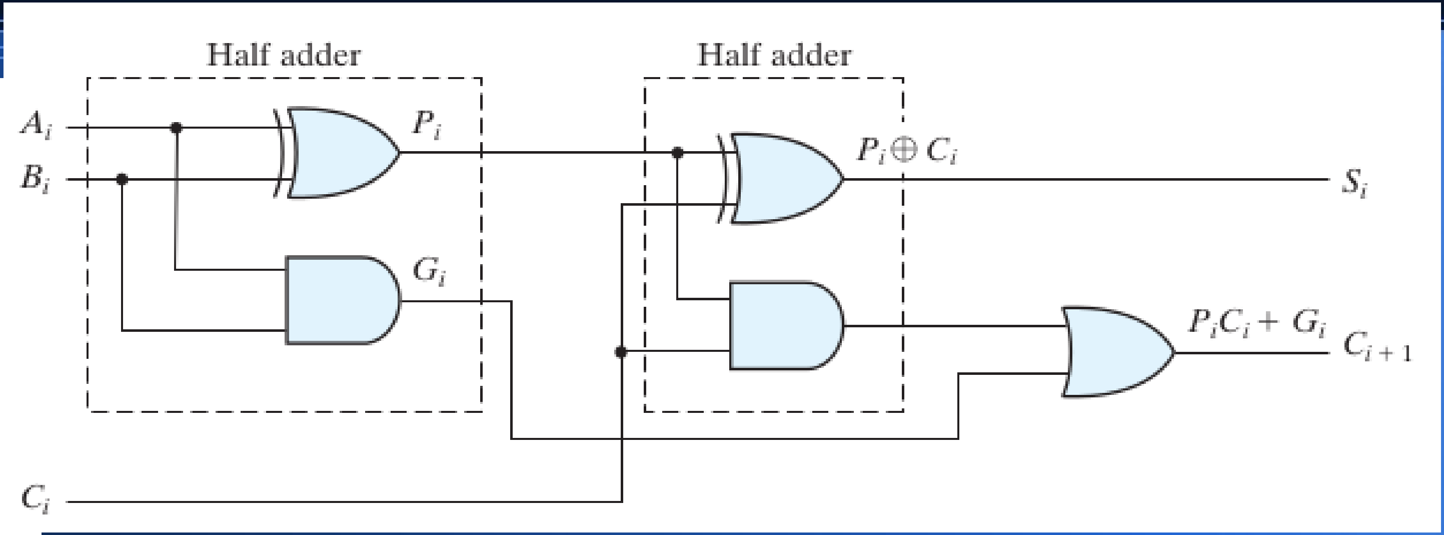
Pi is called a carry propagate, because it determines whether a carry into stage i will propagate into stage i + 1 i.e., whether an assertion of Ci will propagate to an assertion of Ci + 1

Carry lookahead logic

$$C_0 = \text{input carry}$$
$$C_1 = G_0 + P_0C_0$$

$$P_i = A_i \oplus B_i$$
$$G_i = A_iB_i$$

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_iC_i$$



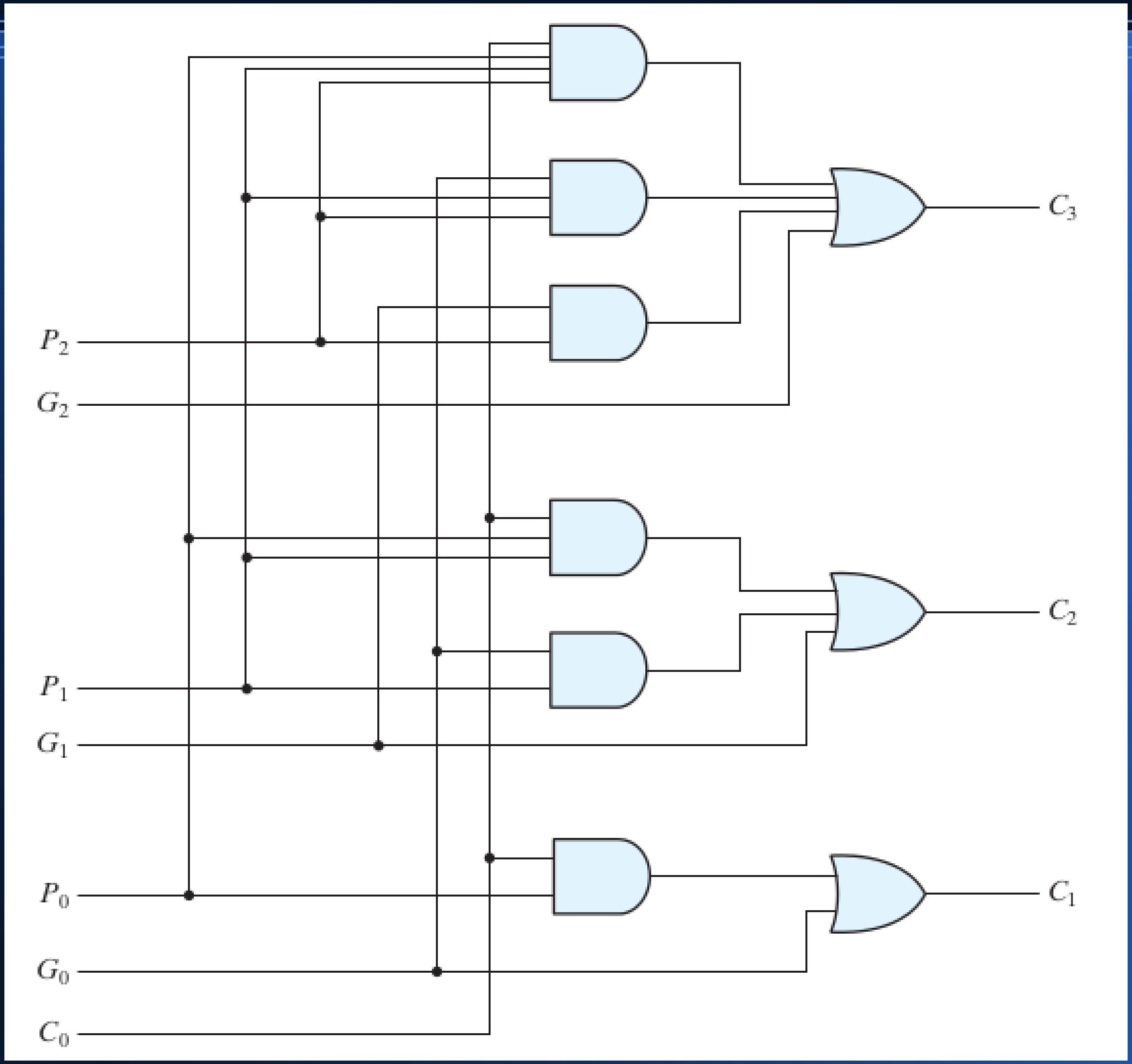
$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

Since the Boolean function for each output carry is expressed in sum-of-products form, each function can be implemented with one level of AND gates followed by an OR gate.

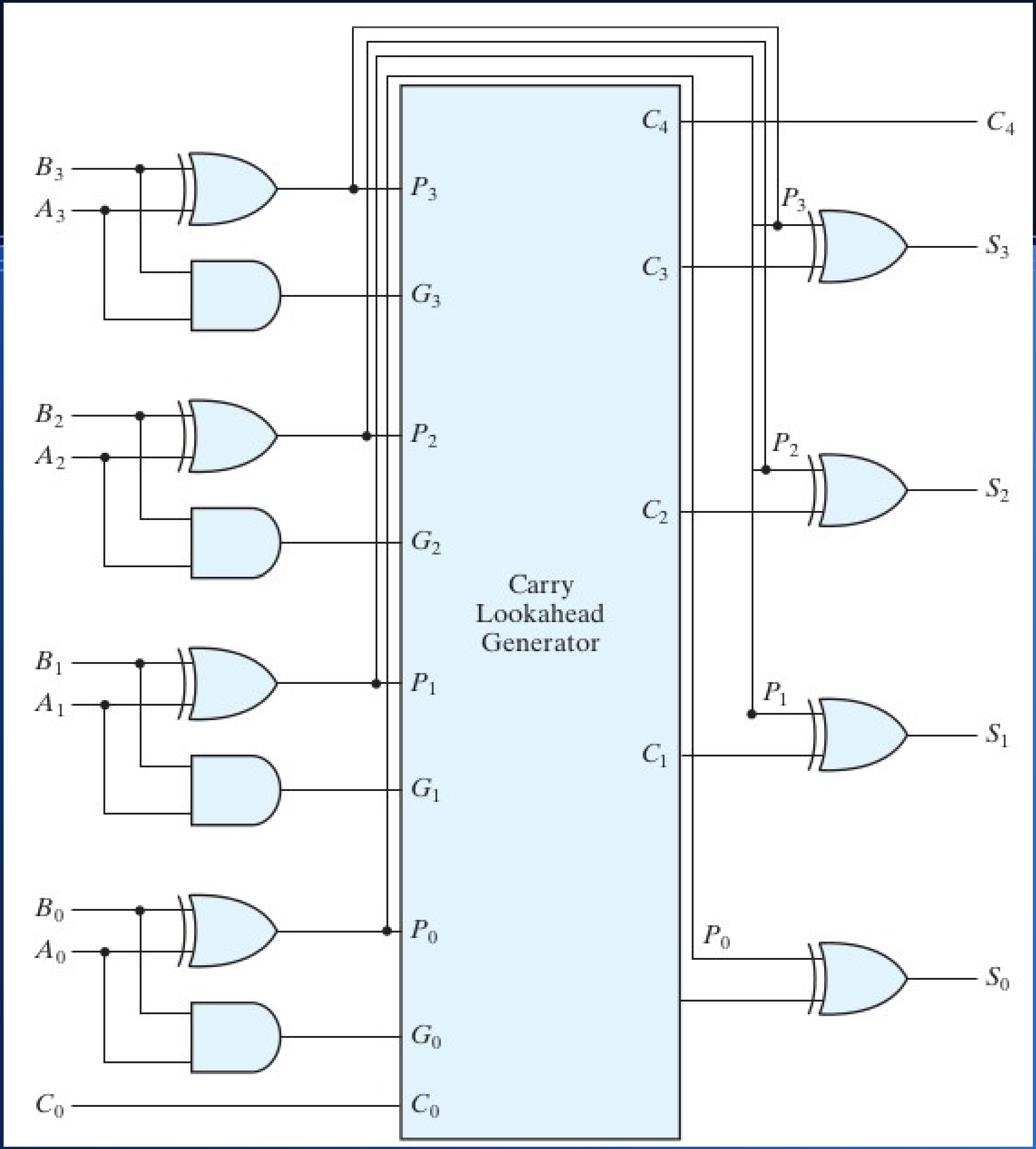
The three Boolean functions for C1, C2, and C3 are implemented in the carry lookahead generator.

Note that this circuit can add in less time because C3 does not have to wait for C2 and C1 to propagate; in fact, C3 is propagated at the same time as C1 and C2. This gain in speed of operation is achieved at the expense of additional complexity (hardware).

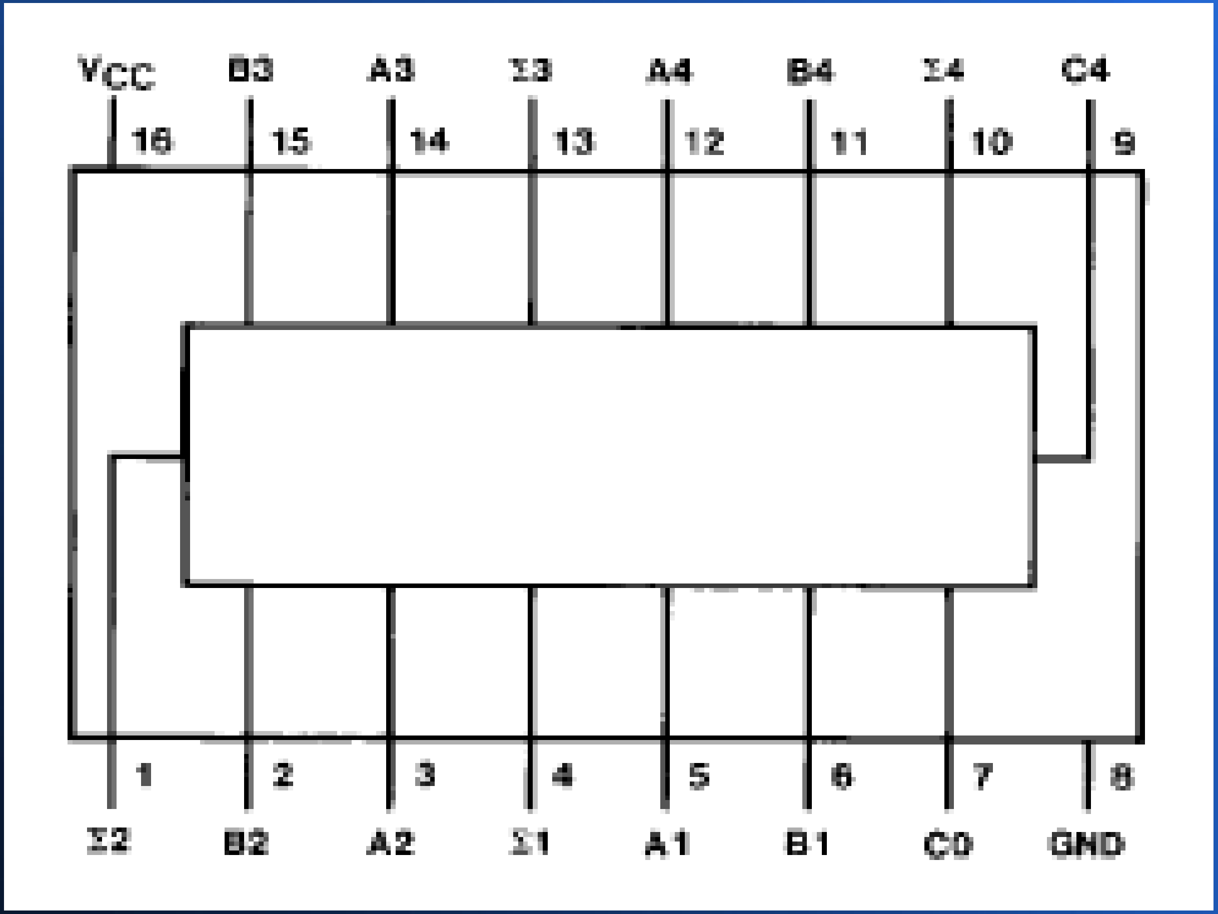
Logic diagram of carry lookahead generator



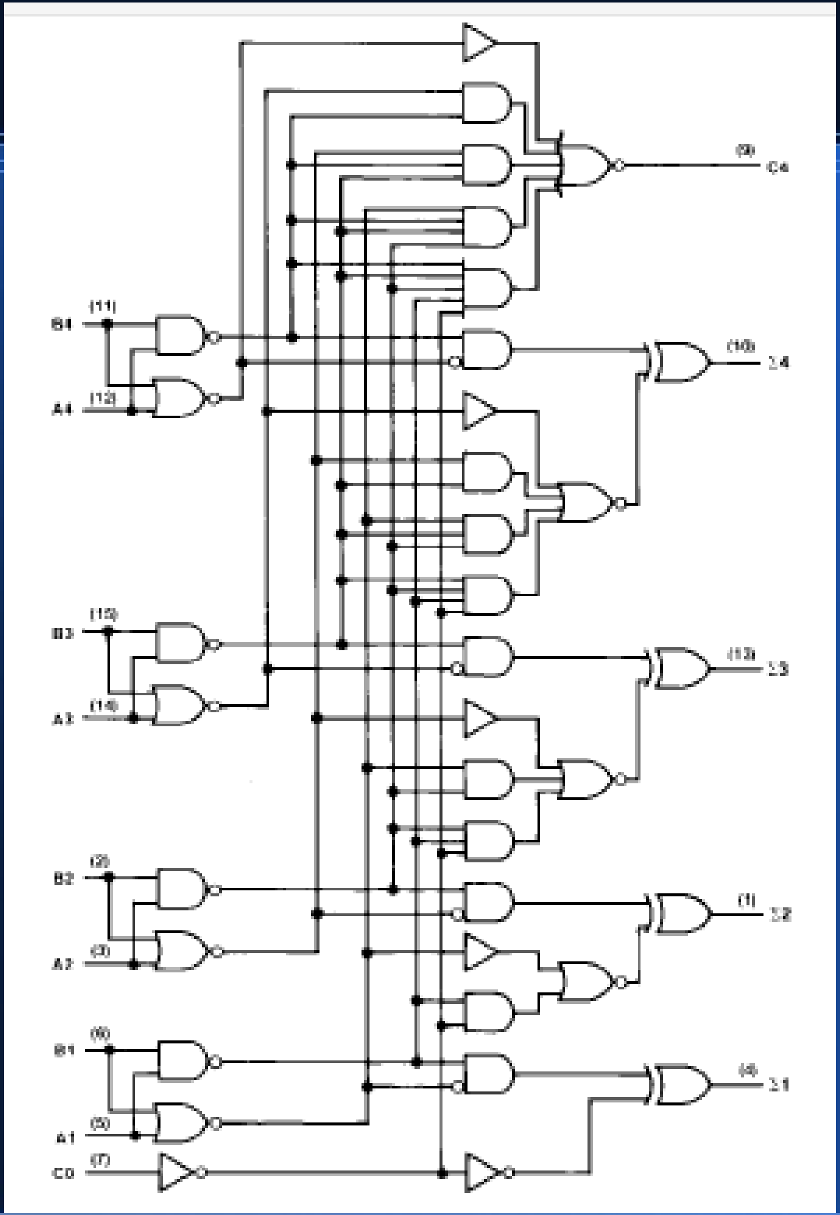
Four-bit adder with carry lookahead



74283 4-bit binary Full Adder Connection Diagram



74283 4-bit binary Full Adder Logic Diagram



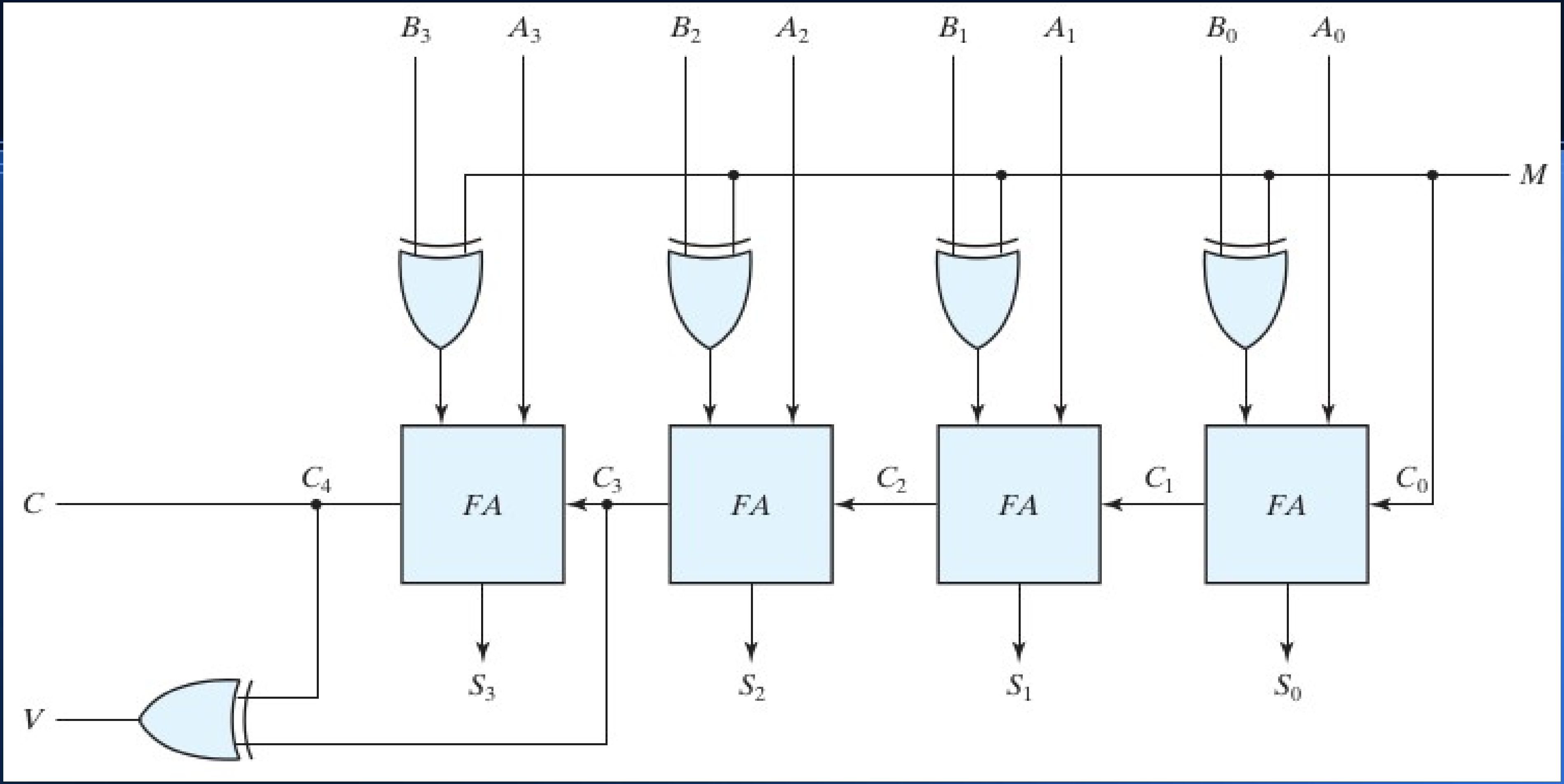
Binary Subtractor

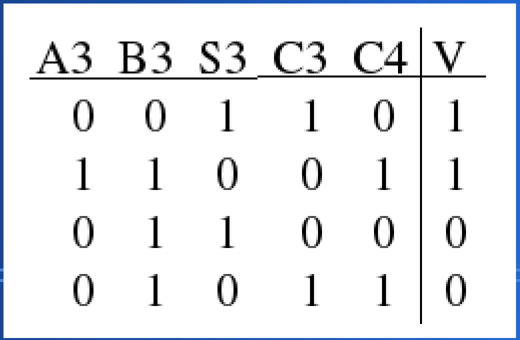
Remember that the subtraction A - B can be done by taking the 2’s complement of B and adding it to A.

The 2’s complement can be obtained by taking the 1’s complement and adding 1 to the least significant pair of bits.

The 1’s complement can be implemented with inverters, and a 1 can be added to the sum through the input carry.

Four-bit adder–subtractor (with overflow detection)





A3	B3	S3	C3	C4	V
0	0	1	1	0	1
1	1	0	0	1	1
0	1	1	0	0	0
0	1	0	1	1	0

The circuit for subtracting A - B consists of an adder with inverters placed between each data input B and the corresponding input of the full adder.

The input carry C0 must be equal to 1 when subtraction is performed.

The operation thus performed becomes A, plus the 1’s complement of B, plus 1.

This is equal to A plus the 2’s complement of B.

For unsigned numbers, that gives A - B if A >= B or the 2's complement of (B – A) if A < B.

For signed numbers, the result is A - B, provided that there is no Overflow.

Binary numbers in the signed-complement system are added and subtracted by the same basic addition and subtraction rules as are unsigned numbers.

Therefore, computers need only one common hardware circuit to handle both types of arithmetic.

The user or programmer must interpret the results of such addition or subtraction differently, depending on whether it is assumed that the numbers are signed or unsigned.

An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position.

If these two carries are not equal, an overflow has occurred

A Simple Formula for Overflow

Let the carry out of the full adder adding the least significant bit be called c0. Then, the carry out of the full adder adding the next least significant bit is c1. Thus, the carry out of the full adder adding the most significant bits is c(k - 1). This assumes that we are adding two k bit numbers. We can write the formula as:

V = c(k-1) XOR c(k-2)

This is effectively XORing the carry-in and the carry-out of the leftmost full adder. Why does this work? The XOR of the carry-in and carry-out differ if there's either a 1 being carried in, and a 0 being carried out, or if there's a 0 being carried in, and a 1 being carried out.

When does that happen? Let's look at each case:

Case 1: 0 carried in, and 1 carried out
If a 0 is carried, then the only way that 1 can be carried out is if x(k-1) = 1 and y(k-1) = 1. That way, the sum is 0, and the carry out is 1. This is the case when you add two negative numbers, but the result is non-negative.

Case 1: 1 carried in, and 0 carried out
The only way 0 can be carried out if there's a 1 carried in is if x(k-1) = 0 and y(k-1) = 0. In that case, 0 is carried out, and the sum is 1. This is the case when you add two non-negative numbers and get a negative result.

(0+1=0 and 0+1=1 carry in = carry out)