Half Adder and Full Adder

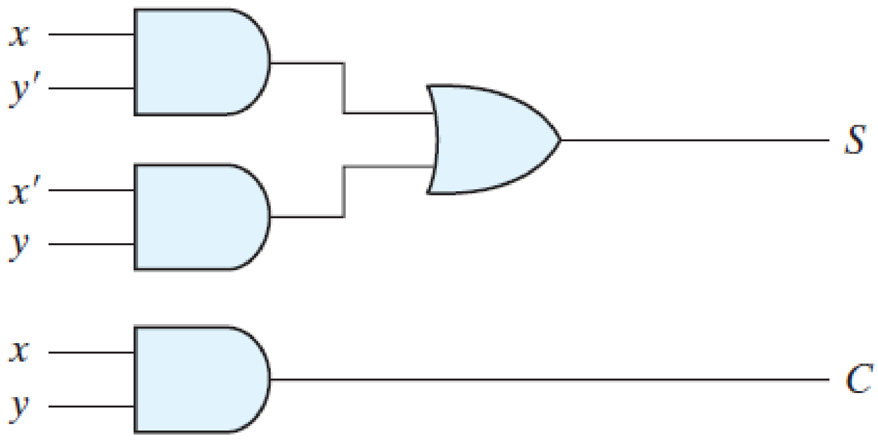
Sparsh Mittal

Half Adder

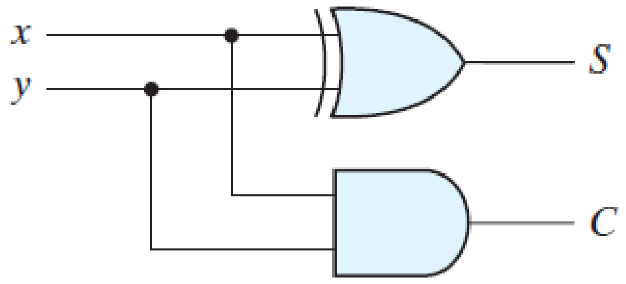
Half Adder

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

$$S = x'y + xy'$$
$$C = xy$$



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



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

Full adder

- Addition of *n*-bit binary numbers requires the use of a full adder, and the process of addition proceeds on a bit-by-bit basis, right to left, beginning with the least significant bit.
- After the least significant bit, addition at each position adds not only the respective bits of the words, but must also consider a possible carry bit from addition at the previous position.
- **A full adder** is a combinational circuit that forms the arithmetic sum of three bits
- Input bits: x and y
- Carry from the previous lower significant position: z

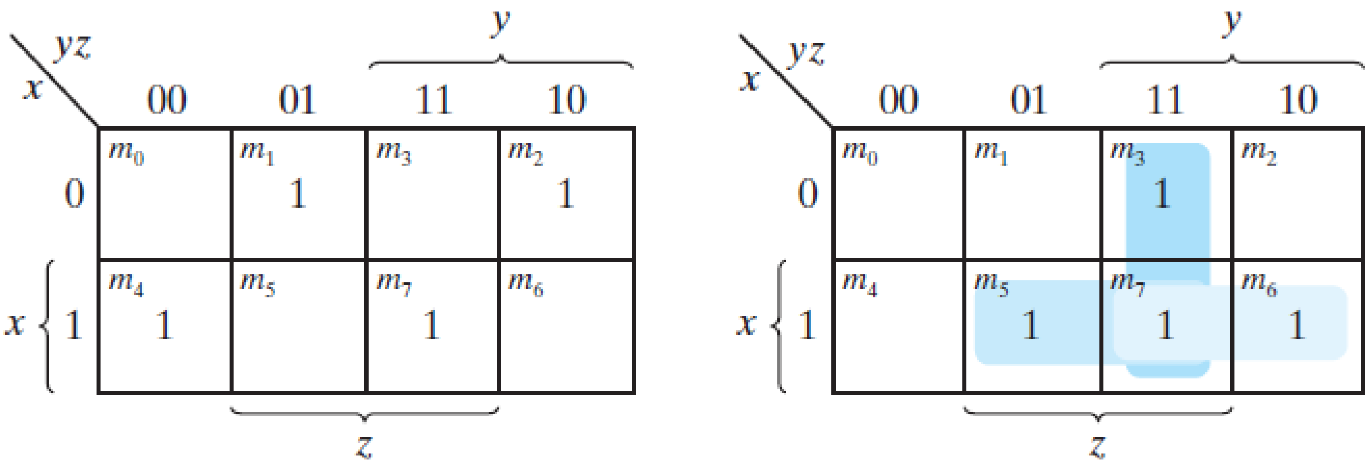
Full adder

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

$$C = xy + xz + yz$$

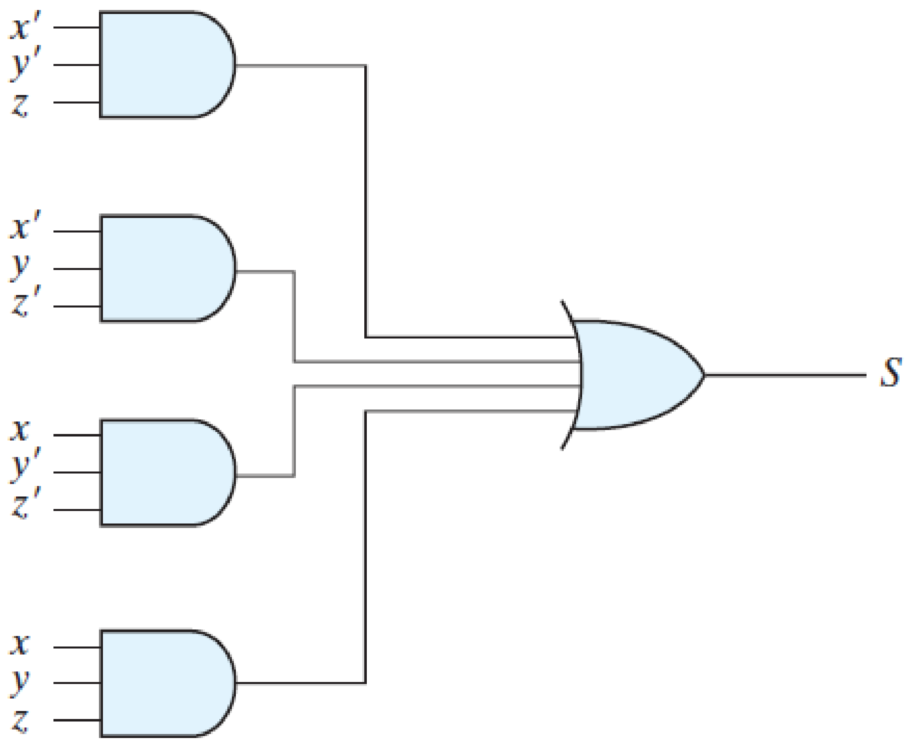
Full Adder

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) $C = xy + xz + yz$



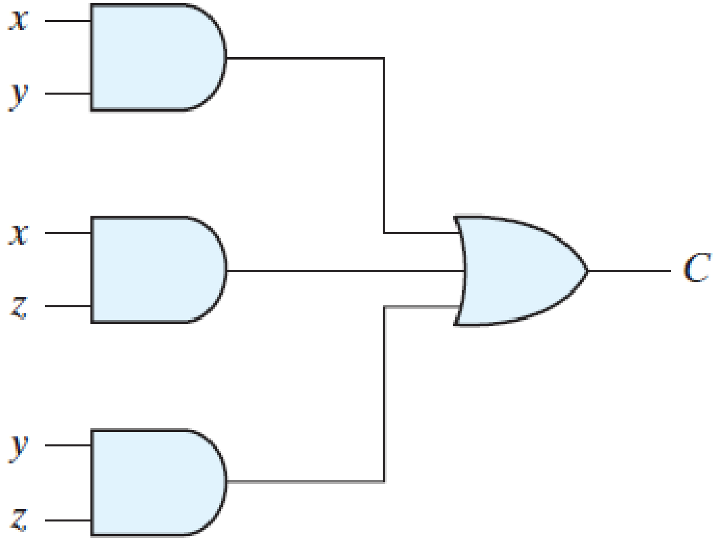


FIGURE 4.7
Implementation of full adder in sum-of-products form

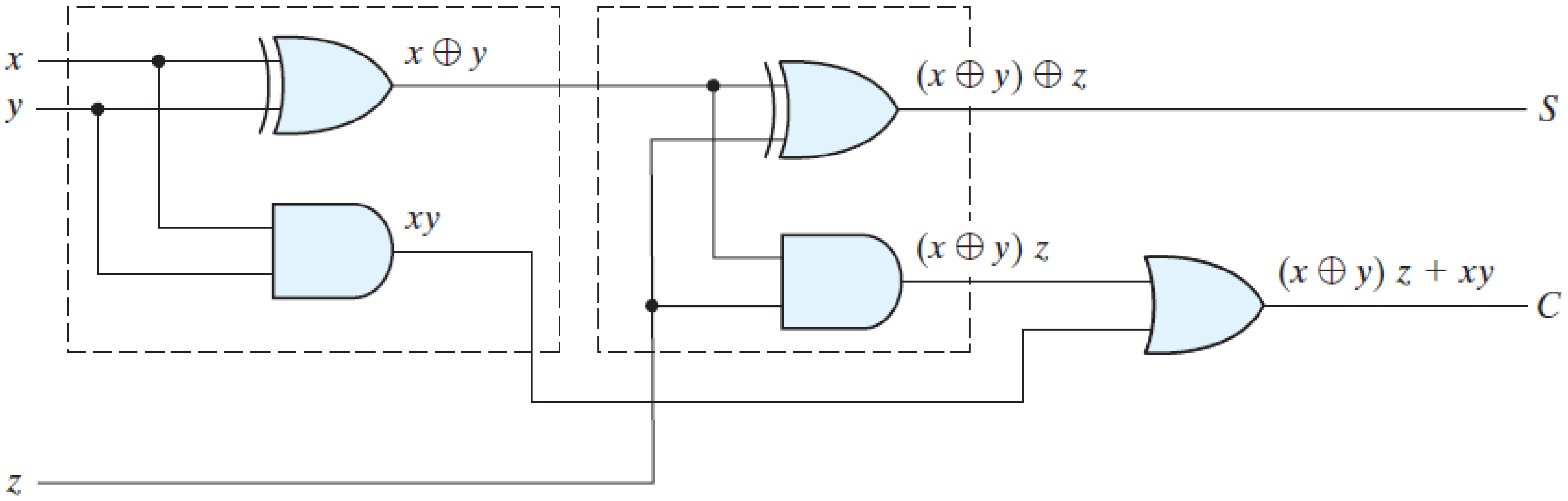


FIGURE 4.8
Implementation of full adder with two half adders and an OR gate

$$
\begin{aligned}
S &= z \oplus (x \oplus y) \\
&= z'(xy' + x'y) + z(xy' + x'y)' \\
&= z'(xy' + x'y) + z(xy + x'y') \\
&= xy'z' + x'yz' + xyz + x'y'z
\end{aligned}
$$

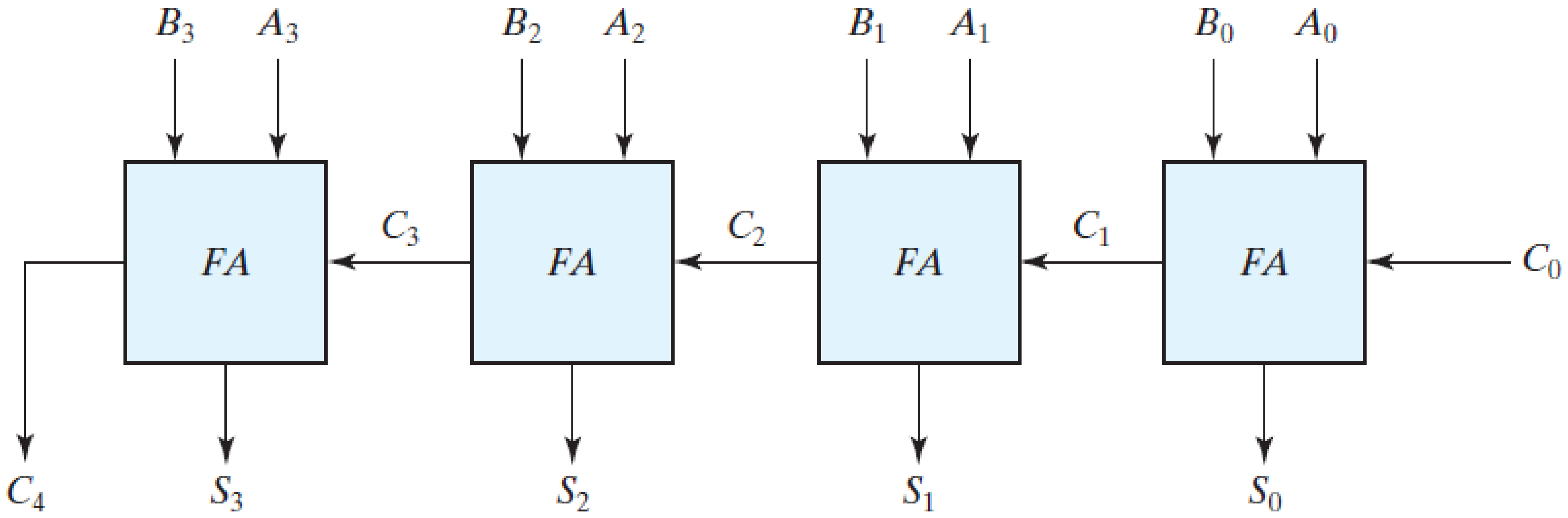
$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

Binary adder

- Addition of n-bit numbers requires a chain of n full adders or a chain of one-half adder and n-1 full adders (assuming Cin at LSB is 0).

Four-bit binary ripple carry adder

The carries are connected in a chain through the full adders.



Example

- $A = 1011$ and $B = 0011$. S= 1110

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 sum bits are thus generated starting from the rightmost position and are available as soon as the corresponding previous carry bit is generated.

Binary subtractor

- 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.
- Operation thus performed becomes A, plus 1's complement of B, plus 1.
- For **unsigned** numbers, that gives A - B if $A \geq B$ or the 2's complement of $(B - A)$ if $A < B$.
- For **signed** numbers, result is A - B, provided that there is no overflow.

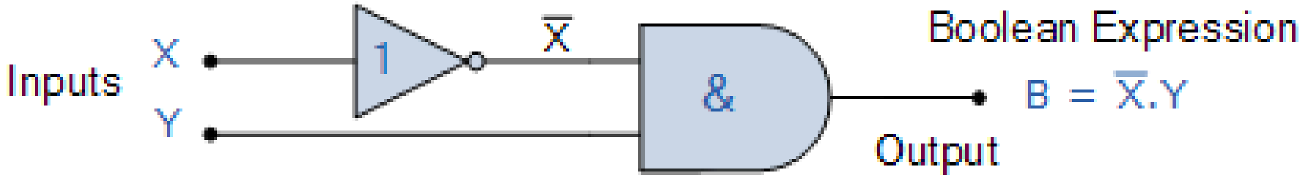
1-bit Binary subtraction

- 0 – 0 = 0
- 0 – 1 = 1 (with a borrow of 1)
- 1 – 0 = 1
- 1 – 1 = 0

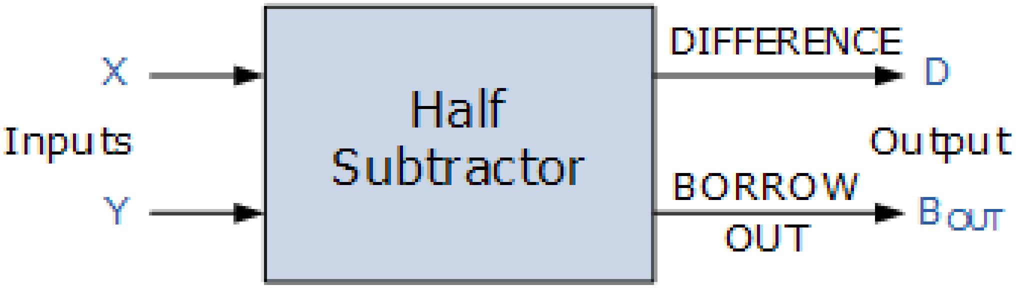
XOR Truth Table		
X	Y	Q
0	0	0
0	1	1
1	0	1
1	1	0

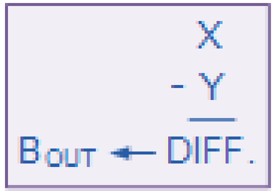
On ignoring the borrow bit, the result of binary subtraction resembles that of an XOR Gate.

Borrow bit is one when input X = 0 and Y = 1.

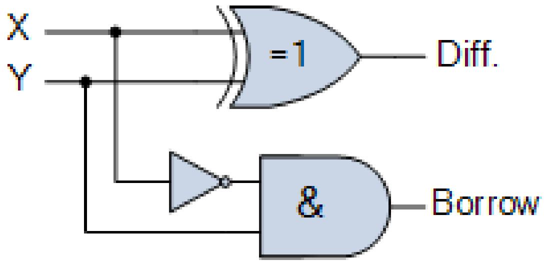


Half subtractor





D = X ⊕ Y
B = X’ Y



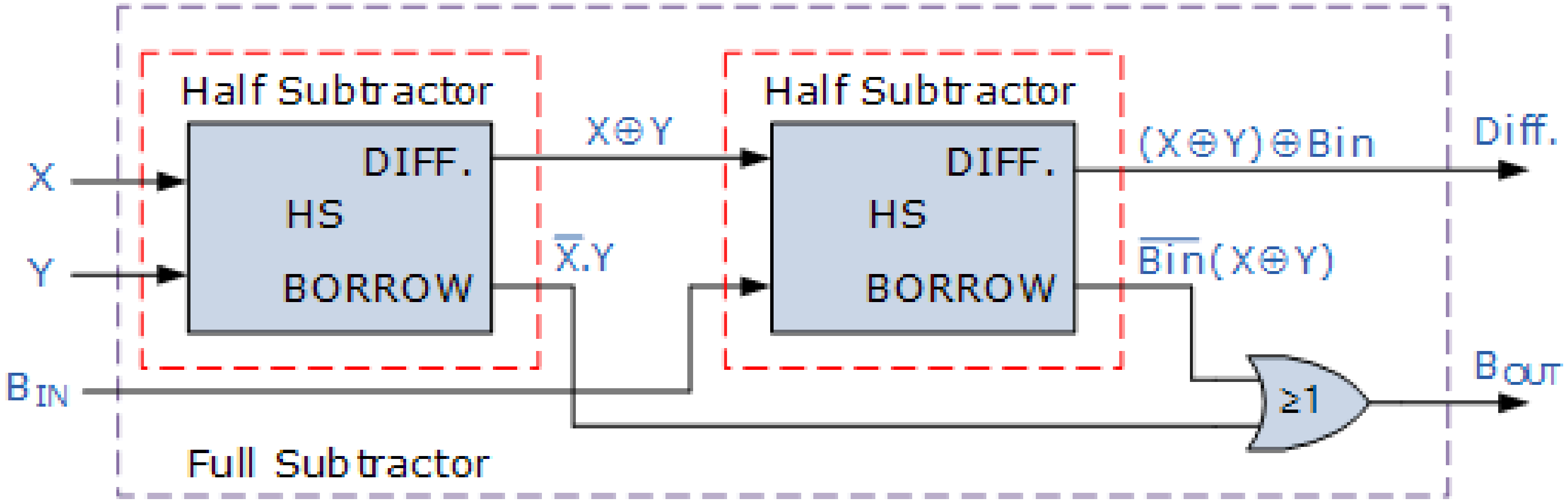
- On comparing half-adder and half-subtractor: sum and diff are exactly same.
- Carry (in adder): XY. Borrow (in subtractor) X’Y

Out of course

Full subtractor

- One major disadvantage of the *Half Subtractor* circuit is that there is no provision for a "Borrow-in" from the previous circuit when subtracting multiple data bits from each other.
- Then we need to produce a "full binary subtractor" circuit to take into account this borrow-in input from a previous circuit.

Full subtractor



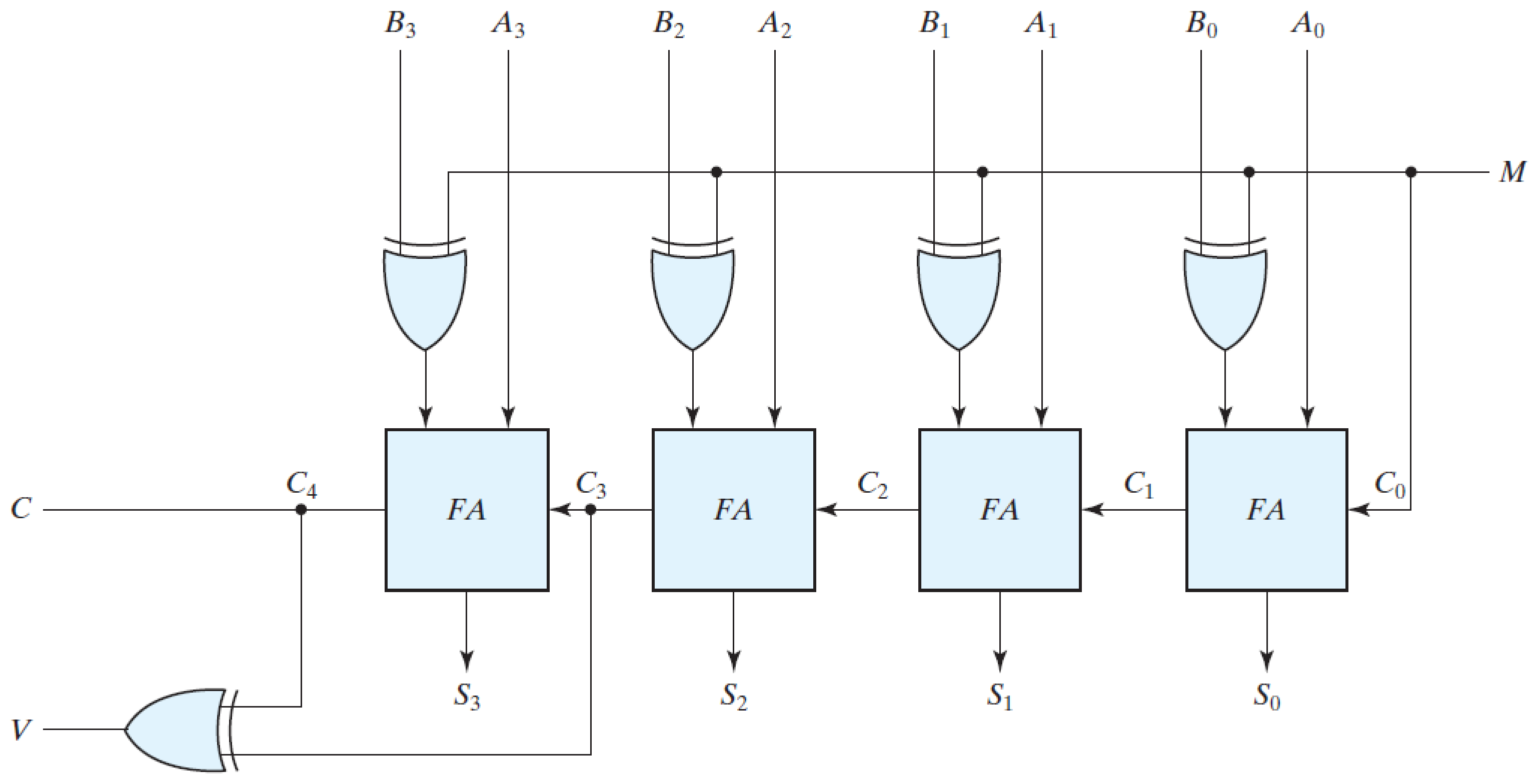
Operation is X-Y

Truth table

B-in	Y	X	Diff.	B-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	1
1	1	1	1	1

Single circuit for add and subtractor

- The mode input M controls the operation.
- $M = 0$, circuit is adder
- $M = 1$ ➔ subtractor.
- $M = 1$ ➔ $B \oplus 1 = B'$ and $C_0 = 1$. ➔ B inputs are all complemented and a 1 is added through the input carry.
- V= for overflow detection



Four-bit adder–subtractor (with overflow detection)

- Addition of two **unsigned numbers**: an overflow is detected from Cout from MSB.
- **Signed number addition**: Two details are important: the leftmost bit always represents the sign, and negative numbers are in 2’s-complement form.
- When two signed numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow.
- An overflow cannot occur after an addition if one number is positive and the other is negative.
- An overflow may occur if the two numbers added are both positive or both negative.

Example (8-bit storage. Range: -128 to 127)

- Both positive: +70 and +80. Sum is +150 ➔ overflow

carries:	0 1	
+70	0	1000110
+80	0	1010000
+150	1	0010110

- Eight-bit result that should have been positive has a negative sign bit (i.e., the eighth bit)
- If, however, the carry out of the sign bit position is taken as the sign bit of the result, then the nine-bit answer so obtained will be correct.
- But since the answer cannot be accommodated within eight bits, we say that an overflow has occurred.

Example (8-bit storage. Range: -128 to 127)

- Both negative: -70 and -80. Sum is -150 ➔ overflow

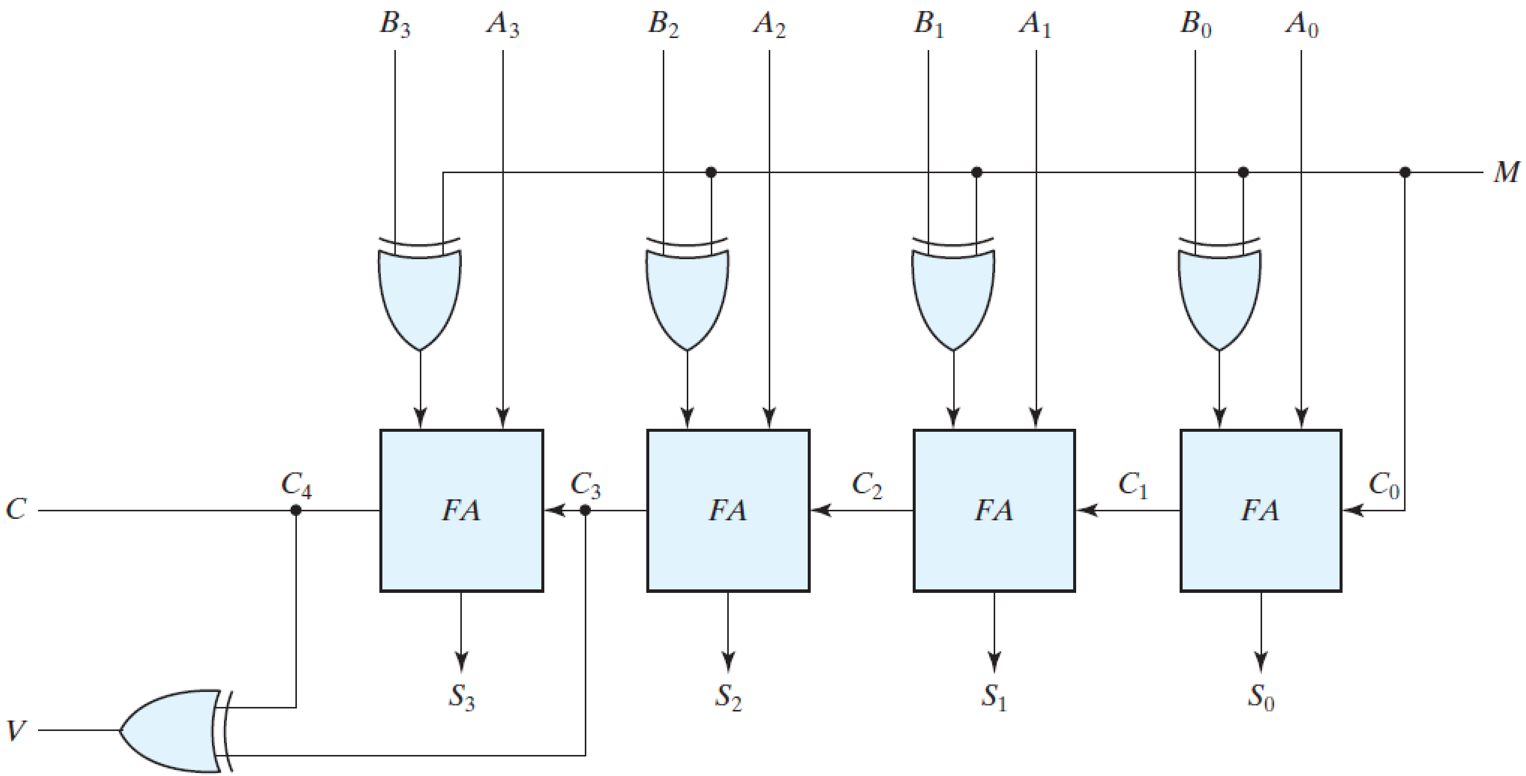
carries:	1 0
−70	1 0111010
−80	1 0110000
−150	0 1101010

Detecting overflow

- An overflow 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.
- Use XOR gate to check inequality

Adder/subtractor

- If the two binary numbers are considered to be unsigned, then the C bit detects a carry after addition or a borrow after subtraction.
- If the numbers are considered to be signed, then the V bit detects an overflow.



Solved Questions

- Assume 5-bit number system, using 2's complement number system.
- Show carry-in and out of MSB to find whether an overflow happens on adding
- (a) 10 and 11
- (b) -12 and -5
- (c) 11 and -12

```
(a)
Carry: 01010
        01010
       +01011
        --------
       10101
        -------
```

The carry-in and out of MSB are different ==> overflow has happened.

```
(b)
Carry:  10000
         10100
        +11011
        ---------
         01111
        ---------
```

The carry-in and out of MSB are different ==> overflow has happened.

- (c)
- Carry: 00000
- 01011
- +10100
- ---------
- 11111
- ---------
- The carry-in and out of MSB is same ==> there is no overflow.