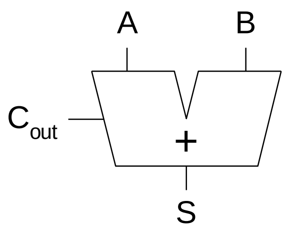
1-Bit Adders

Half Adder

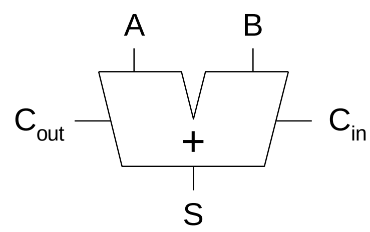


A	B	C_{out}	S
0	0		
0	1		
1	0		
1	1		

S =

C_{out} =

Full Adder



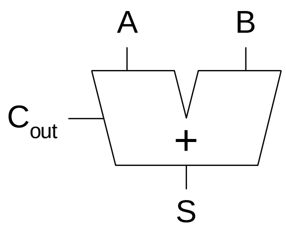
C_{in}	A	B	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

S =

C_{out} =

1-Bit Adders

Half Adder

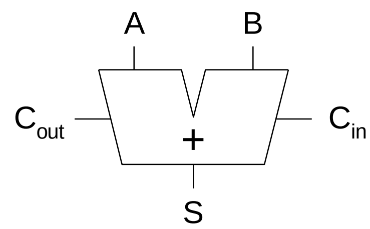


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S =$

$C_{out} =$

Full Adder



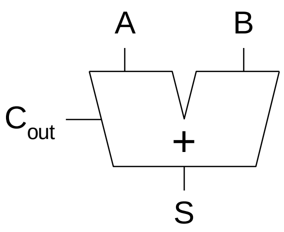
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$S =$

$C_{out} =$

1-Bit Adders

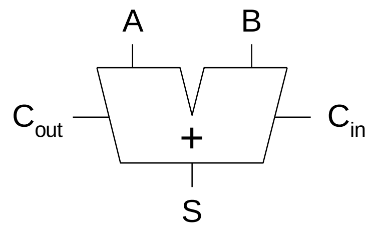
Half Adder



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



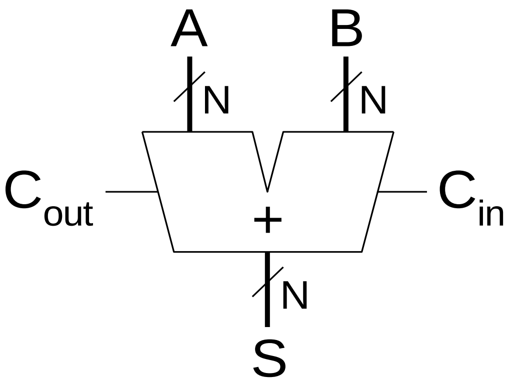
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

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

Multibit Adders (CPAs)

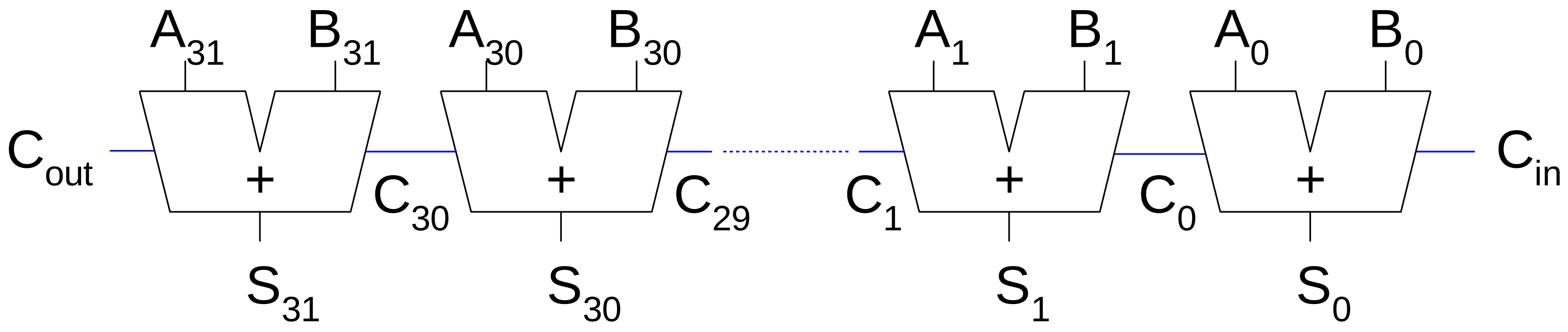
- Types of carry propagate adders (CPAs):
  - Ripple-carry (slow)
  - Carry-lookahead (fast)
  - Prefix (faster)
- Carry-lookahead and prefix adders faster for large adders but require more hardware

Symbol



Ripple-Carry Adder

- Chain 1-bit adders together
- Carry ripples through entire chain
- Disadvantage: **slow**



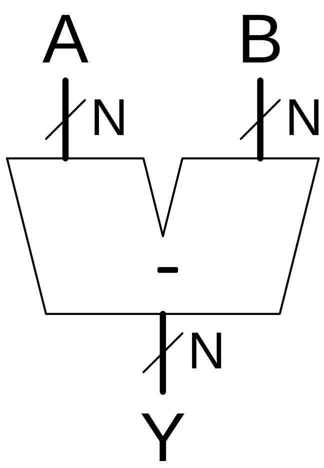
Ripple-Carry Adder Delay

$$t_{\text{ripple}} = N \times t_{FA}$$

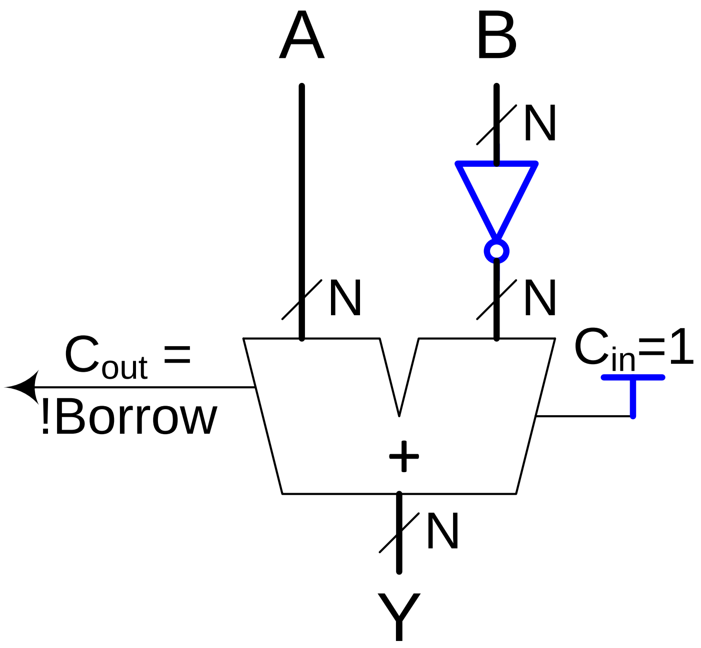
where t_{FA} is the delay of a full adder

Subtractor

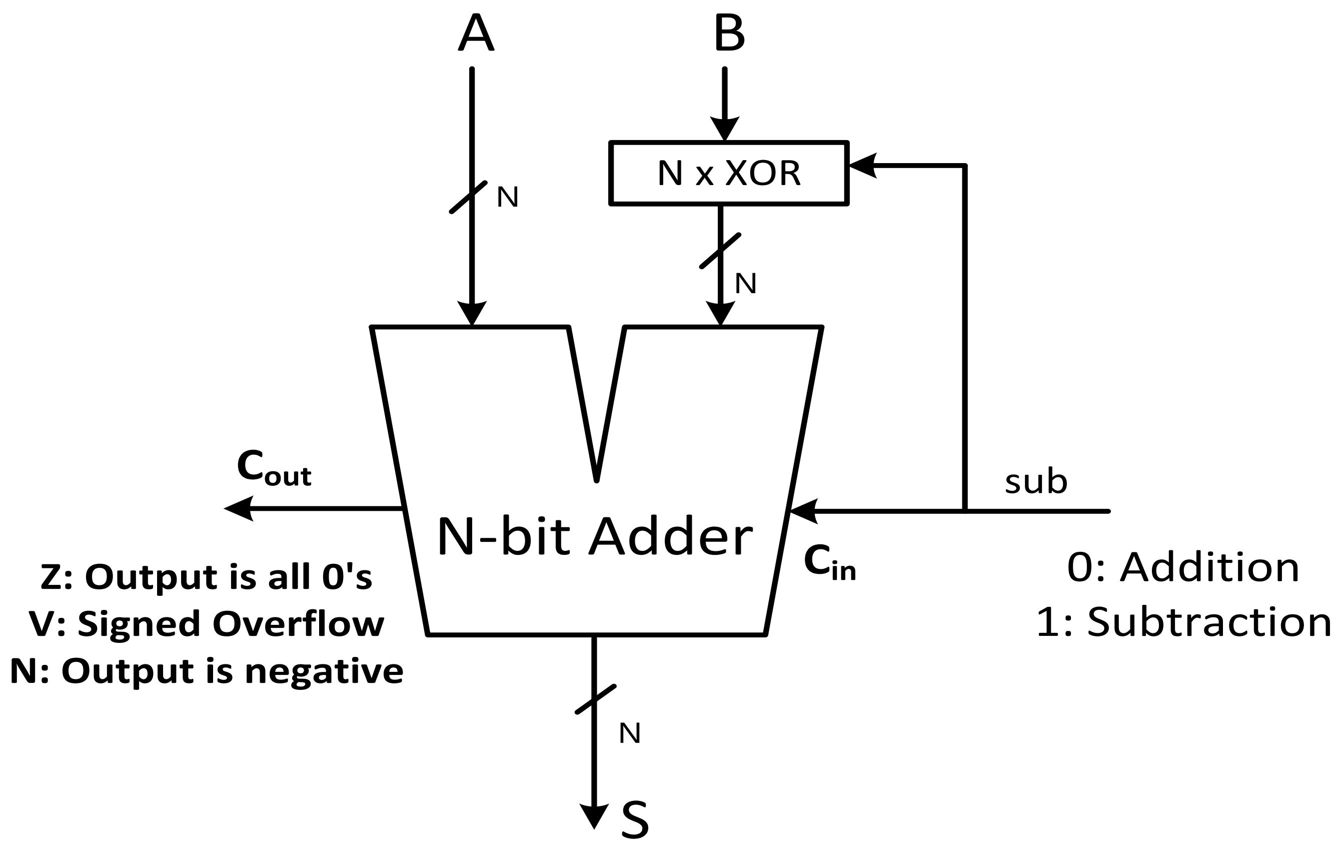
Symbol



Implementation



Adder/Subtractor



Adder/Subtractor Signed Overflow

- In signed addition, overflow occurs when:
  - positive + positive ⇨ negative
  - negative + negative ⇨ positive

positive + negative never generates an overflow

- In signed subtraction, overflow occurs when:
  - positive – negative ⇨ negative
  - negative – positive ⇨ positive

positive – positive or negative – negative

never generates an overflow

Adder/Subtractor, Unsigned Numbers

- If A and B are unsigned numbers:

  sub = 0 means $S = A + B$

  sub = 1 means $S = A + \sim B + 1 = A - B$
- C_{out} is addition overflow indicator:

  If set, it shows 2^N should be added to S
- ! C_{out} (= Borrow) is subtraction underflow indicator:

  If set, S is correctly equal to $S = A - B$

  If not set, 2^N should be deducted from S
- While Z shows output is all zero, ***N and V do not have any meaning*** in unsigned addition/subtraction

Adder/Subtractor, Signed Numbers

- If A and B are signed numbers:

  sub = 0 means S = A +B

  sub = 1 means S = A + ~B + 1 = A – B
- V is addition/subtraction overflow indicator
- Unlike unsigned addition/subtraction, output S can not be reconstructed when overflow occurs.
- Indeed, a new addition/subtraction with extra bits (recall sign-extension) are required to prevent any signed overflow
- While Z shows output is all zero, ***C_{out} do not have any meaning*** in signed addition and/or subtraction

Adder/Subtractor, Elaboration

- A binary adder can ***blindly*** add/subtract signed and unsigned numbers, thanks to the two's complement signed number representation.
  - Warning: above statement is not correct for multiplication and division.
- This is user responsibility to properly threat the output specially when an overflow occurs.
  - In unsigned numbers, C_{out} shows exact error amount.
  - In signed numbers, output should be thrown out, when V (signed overflow) is asserted.

Comparator, Unsigned Numbers

Calculate A + ~B + 1 (i.e. Subtract)	
A == B	

Comparator, Unsigned Numbers

Calculate A + ~B + 1 (i.e. Subtract)	
A == B	Z
A != B	! Z

Comparator, Unsigned Numbers

Calculate A + ~B + 1 (i.e. Subtract)	
A == B	Z
A != B	! Z
A >= B	

Comparator, Unsigned Numbers

Calculate A + ~B + 1 (i.e. Subtract)	
A == B	Z
A != B	! Z
A >= B	C_{out}
A < B	! C_{out}

Comparator, Unsigned Numbers

Calculate A + ~B + 1 (i.e. Subtract)	
A == B	Z
A != B	! Z
A >= B	C_{out}
A < B	! C_{out}
A > B	

Comparator, Unsigned Numbers

Calculate A + ~B + 1 (i.e. Subtract)	
A == B	Z
A != B	$!\,Z$
A >= B	C_{out}
A < B	$!\,C_{out}$
A > B	$C_{out}\ \&\ !\,Z$
A <= B	$!\,C_{out}\ \vert\ Z$

Comparator, Signed Numbers

Calculate A + ~B + 1 (i.e. Subtract)	
A == B	

Comparator, Signed Numbers

Calculate A + ~B + 1 (i.e. Subtract)	
A == B	Z
A != B	! Z

Comparator, Signed Numbers

Calculate A + ~B + 1 (i.e. Subtract)	
A == B	Z
A != B	! Z
A >= B	

Comparator, Signed Numbers

Calculate A + ~B + 1 (i.e. Subtract)	
A == B	Z
A != B	! Z
A >= B	(! N & ! V) | (N & V)
A < B	N xor V

Comparator, Signed Numbers

Calculate A + ~B + 1 (i.e. Subtract)	
A == B	Z
A != B	! Z
A >= B	(! N & ! V) | (N & V)
A < B	N xor V
A > B	

Comparator, Signed Numbers

<table>
<tr><th colspan="2">Calculate A + ~B + 1 (i.e. Subtract)</th></tr>
<tr><td>A == B</td><td>Z</td></tr>
<tr><td>A != B</td><td>! Z</td></tr>
<tr><td>A >= B</td><td>(! N & ! V) | (N & V)</td></tr>
<tr><td>A < B</td><td>N xor V</td></tr>
<tr><td>A > B</td><td>(N xnor V) & ! Z</td></tr>
<tr><td>A <= B</td><td>(N xor V) | Z</td></tr>
</table>

Comparator, Elaboration

- For unsigned numbers, an n-bit binary adder/subtractor in subtraction mode can compare two n-bit unsigned numbers:

  ONLY look at C_{out} (= ! Borrow) and Z

- For signed numbers, above binary subtractor can compare two n-bit signed numbers, even when a signed overflow occurs and output S should be discarded:

  When V = 0, look at N

  When V = 1, N is complemented, thus look at ! N