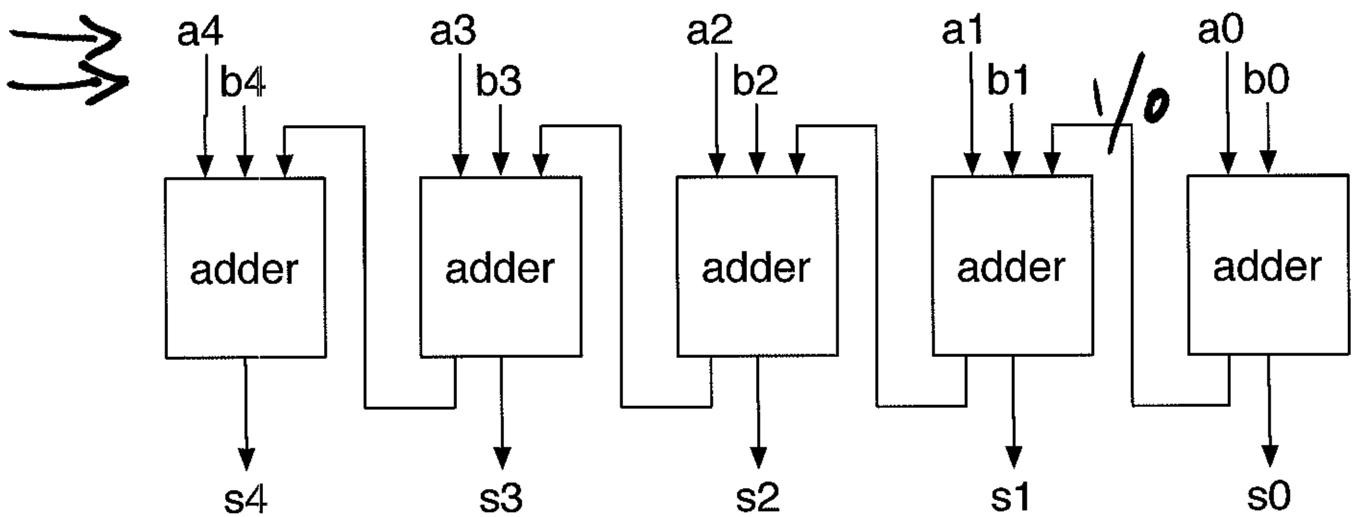
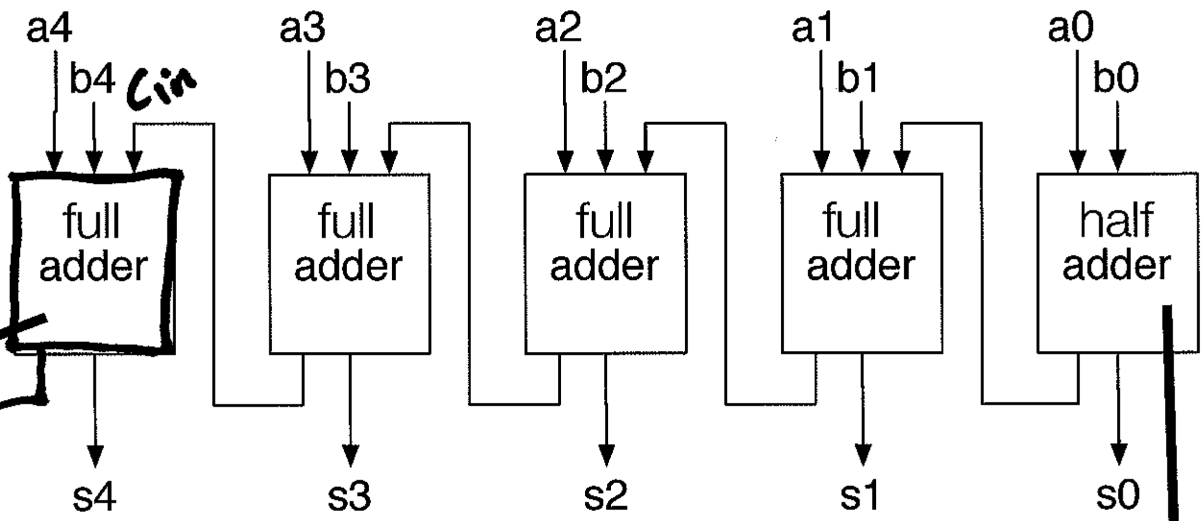
Decimal v. binary addition

$$
\begin{array}{r}
1\,1\,0 \\
4\,3\,5\,8\,2 \\
+\;2\,2\,5\,7\,3 \\
\hline
6\,6\,1\,5\,5
\end{array}
\qquad\qquad
\begin{array}{rl}
1\,1\,1\,1 & \\
0\,1\,0\,1\,1 & (11) \\
+\,0\,0\,1\,0\,1 & (5) \\
\hline
1\,0\,0\,0\,0 &
\end{array}
$$



Ripple carry adder

$(a \oplus b) \oplus c$



a	b	cin	cout	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:

$C_{out} = ab + bc_{in} + ac_{in}$

$S = a \oplus b \oplus c_{in}$

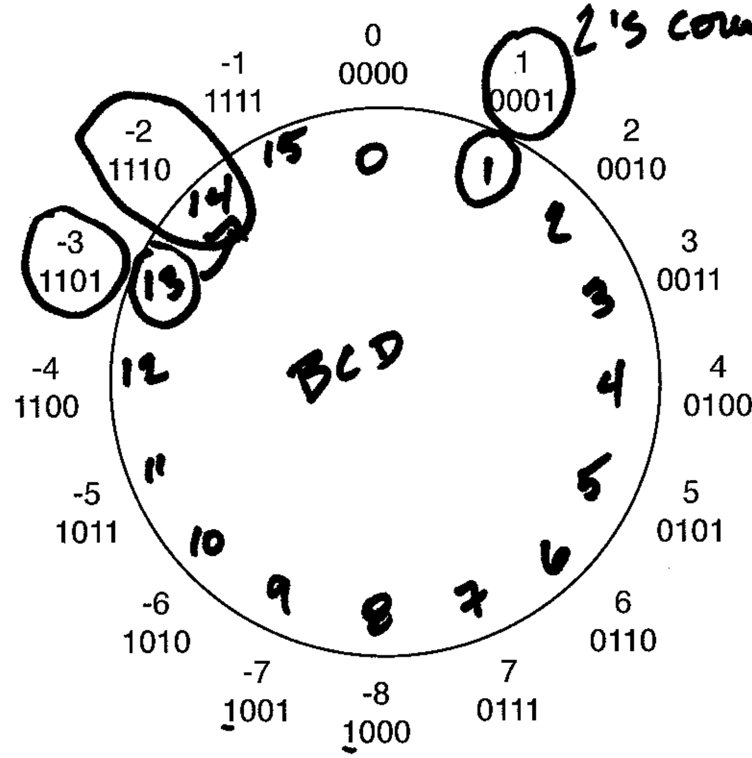
a_i	b_i	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

half adder: $C = ab$

$S = a \oplus b$

Subtraction w. twos complement representation

2's complement



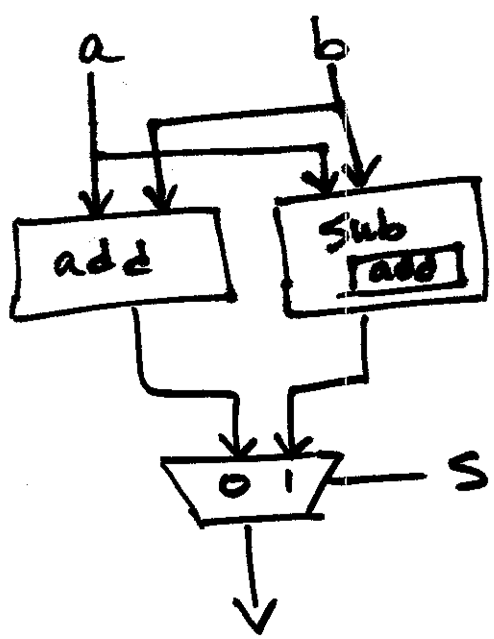
$1 - 3 = -2$

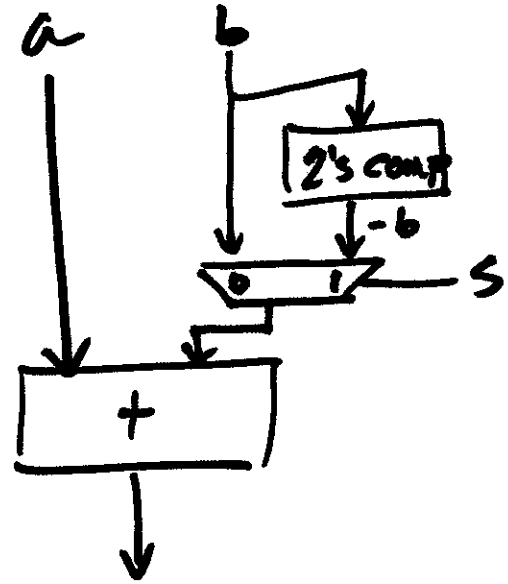
$1 + (-3) = -2$

Can be accomplished with a **twos-complementor** and an **adder**

In class exercise: designing an adder-subtractor

two high-level structures:

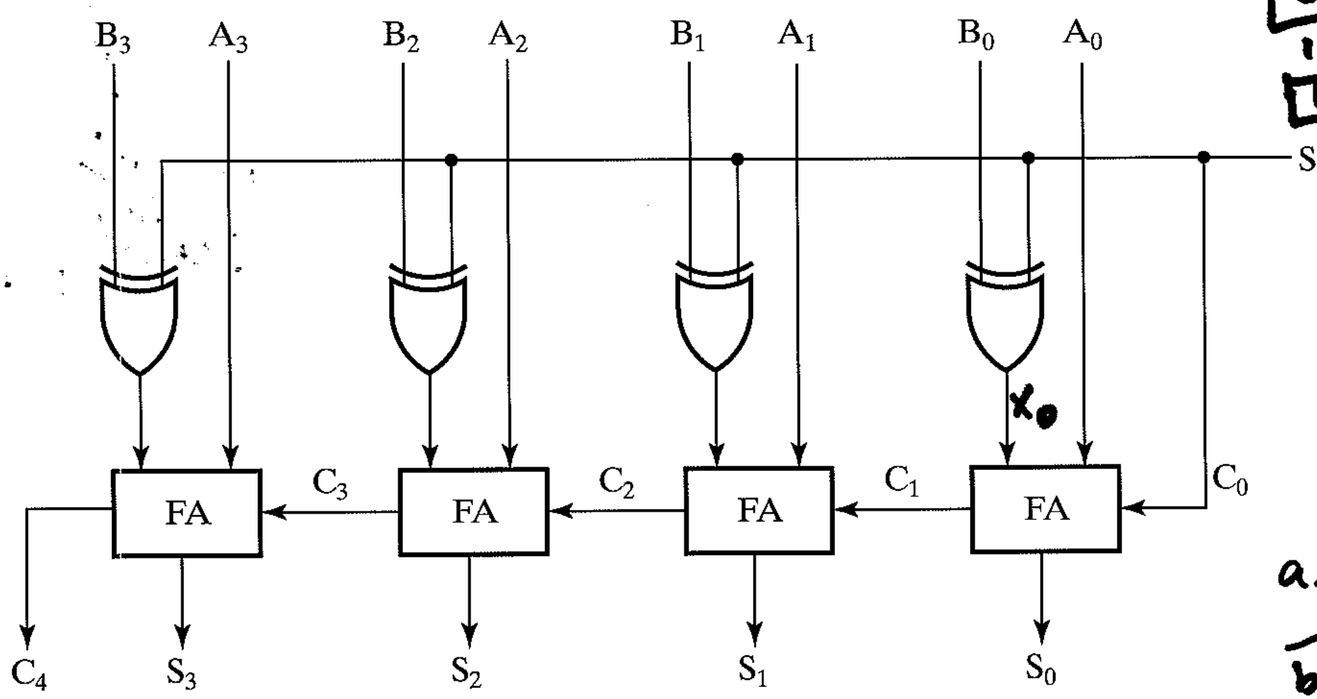




Adder/subtractor for #'s in 2's complement form

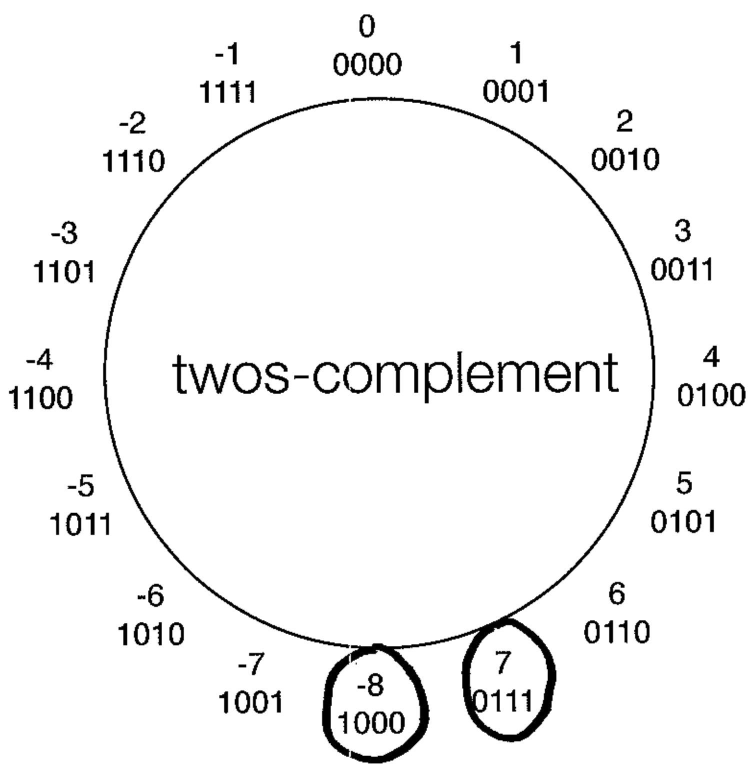
4-7

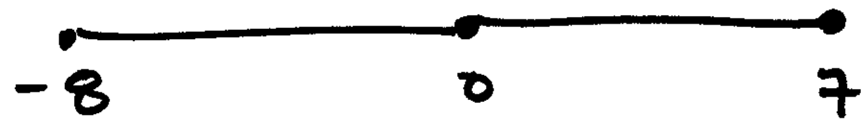
B_0	S	X_0
0	0	0
0	1	1
1	0	1
1	1	0



$$\begin{array}{rcccc} & a_3 & a_2 & a_1 & a_0 \\ & \overline{b_3} & \overline{b_2} & \overline{b_1} & \overline{b_0} \\ + & 0 & 0 & 0 & 1 \\ \hline \end{array} \Bigg] -b$$

Handling overflow





~~0111~~
~~(5) 0101~~
~~(3) 0011~~
~~1000~~ (−8)

1111
(−5) 1011
(−3) 1101
1000 (−8)

~~1000~~
~~(−6) 1010~~
~~(−3) 1101~~
~~0111~~ (7)

0010
(−6) 1010
(3) 0011
1101 (−3)

Handling overflow

c4	c3	c2	c1	c0
	a3	a2	a1	a0
	b3	b2	b1	b0
	s3	s2	s1	s0

full adder

IN | OUT

overflow = $c3 \oplus c4$

a3	b3	c3		c4	s3	overflow
0	0	0	=	0	0	0
0	0	1	≠	0	1	1
0	1	0	=	0	1	0
0	1	1	=	1	0	0
1	0	0	=	0	1	0
1	0	1	=	1	0	0
1	1	0	≠	1	0	1
1	1	1	=	1	1	0

adding two pos #s

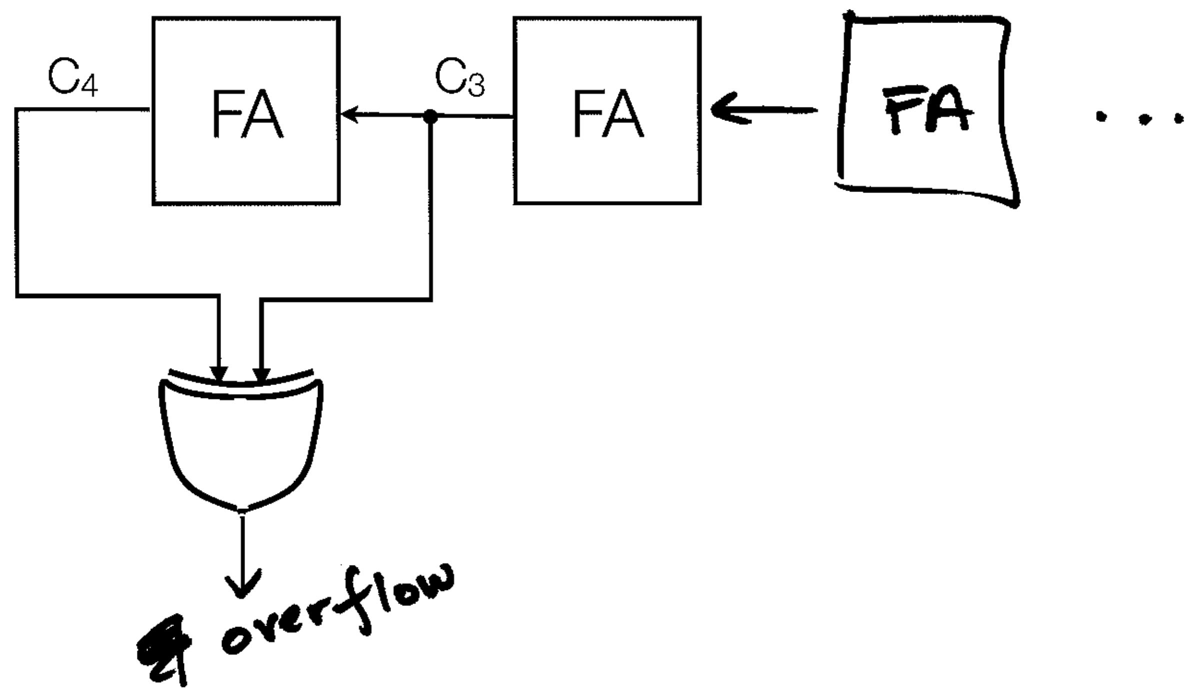
adding two neg. #s

IN | OUT

overflow

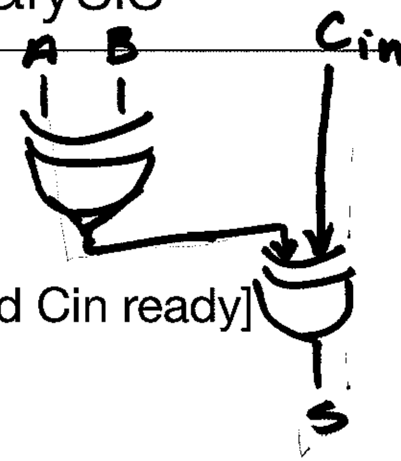
Overflow computation in adder/subtractor

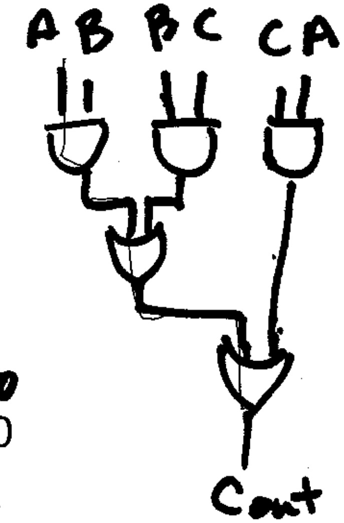
For 2's complement, overflow if 2 most significant carries differ

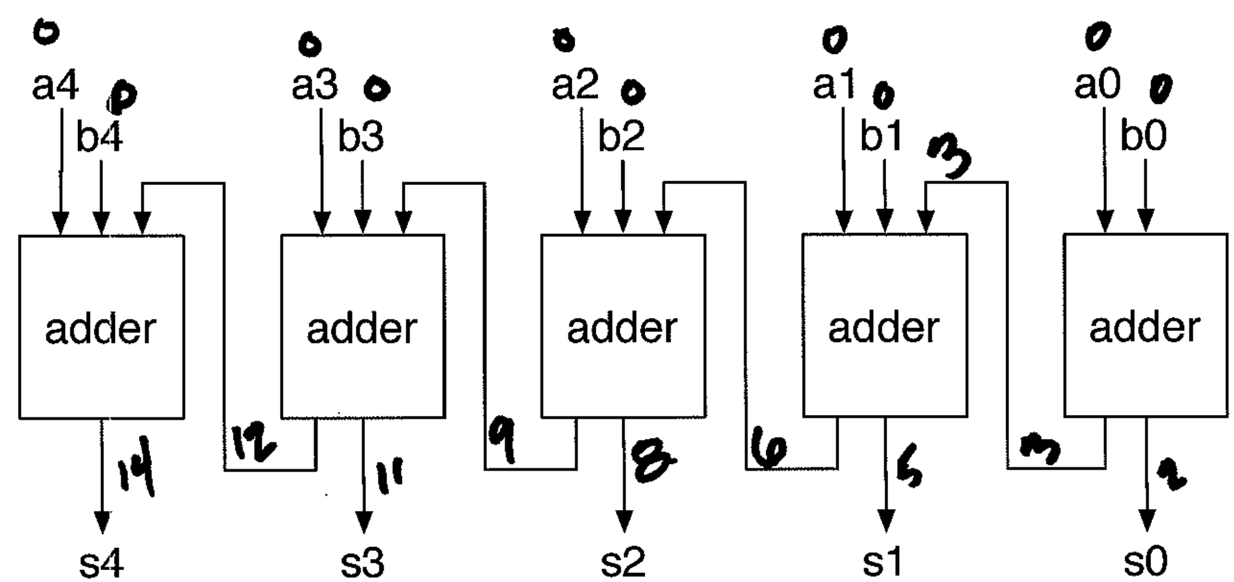


Ripple carry adder delay analysis

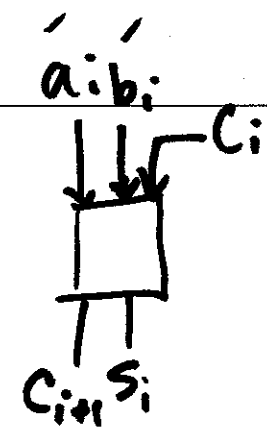
- Assume unit delay, for all gates use 2-input gates
  - S = A ⊕ B ⊕ Cin
    - [S ready __2__ units after A,B and Cin ready]
  - Cout = AB + ACin + BCin
    - [Cout ready __3__ units after A,B and Cin ready]







Carry lookahead adder (CLA)



- Goal: produce an adder of less circuit depth
- Start by rewriting the carry function

$$c_{i+1} = a_i b_i + a_i c_i + b_i c_i$$

$$c_{i+1} = a_i b_i + c_i (a_i + b_i)$$

$$c_{i+1} = g_i + c_i (p_i)$$

carry generate

$g_i = a_i b_i$

carry propagate

$p_i = a_i + b_i$

Carry lookahead adder (CLA) (2)

- Can recursively define carries in terms of propagate and generate signals

$$c_1 = g_0 + \cancel{c_0}^{0} p_0$$

$$c_2 = g_1 + \underline{c_1} p_1$$

$$= g_1 + (\underline{g_0 + c_0 p_0}) p_1$$

$$c_2 = \underline{g_1 + g_0 p_1 + c_0 p_0 p_1}$$

$$c_3 = g_2 + \underline{c_2} p_2$$

$$= g_2 + (\underline{g_1 + g_0 p_1 + c_0 p_0 p_1}) p_2$$

$$= g_2 + g_1 p_2 + g_0 p_1 p_2 + \underline{c_0} p_0 p_1 p_2$$

$g_i = a_i \cdot b_i$
$p_i = a_i + b_i$

- ith carry has i+1 product terms, the largest of which has i+1 literals
- If AND, OR gates can take unbounded inputs: total circuit depth is 2 (SoP form)
- If gates take 2 inputs, total circuit depth is 1 + $\log_2 k$ for k-bit addition

Carry lookahead adder (CLA) (3)

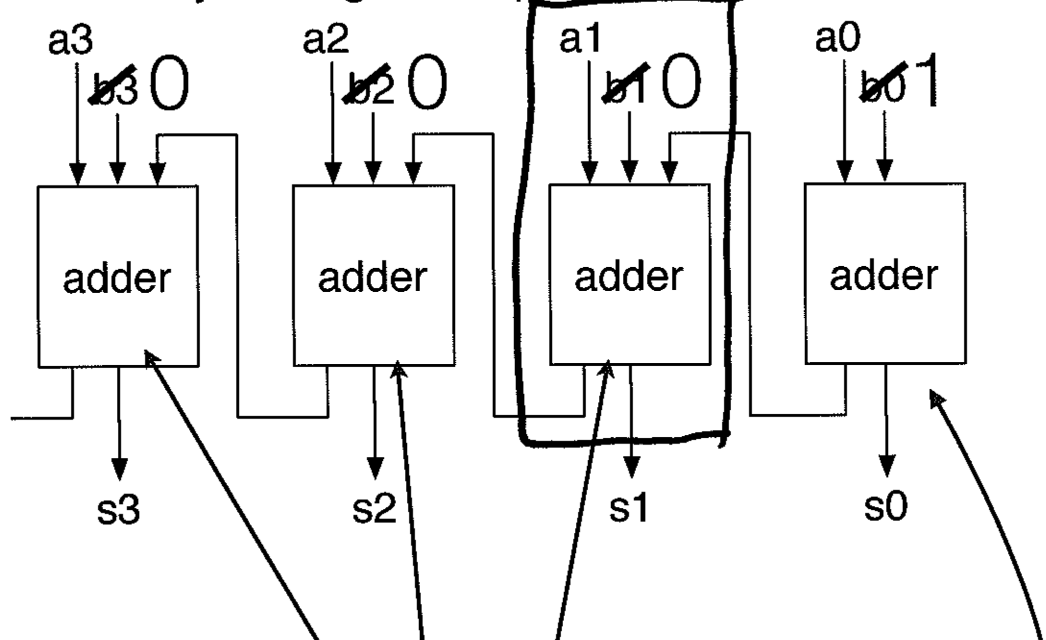
$c_0 = 0$	$s_0 = a_0 \oplus b_0 \oplus c_0$
$c_1 = g_0 + c_0 p_0$	$s_1 = a_1 \oplus b_1 \oplus c_1$
$c_2 = g_1 + g_0 p_1 + c_0 p_0 p_1$	$s_2 = a_2 \oplus b_2 \oplus c_2$
$c_3 = g_2 + g_1 p_2 + g_0 p_1 p_2 + c_0 p_0 p_1 p_2$	$s_3 = a_3 \oplus b_3 \oplus c_3$

Contraction

Contraction is the simplification of a circuit through constant input values.

Contraction example: adder to incrementer

- What is the hardware and delay savings of implementing an incrementer using contraction?



Can be reduced to half-adders

Incrementer circuit

a_0	S	C
0	1	0
1	0	1

$S_0=\overline{a_0}$, $C_0=a_0$