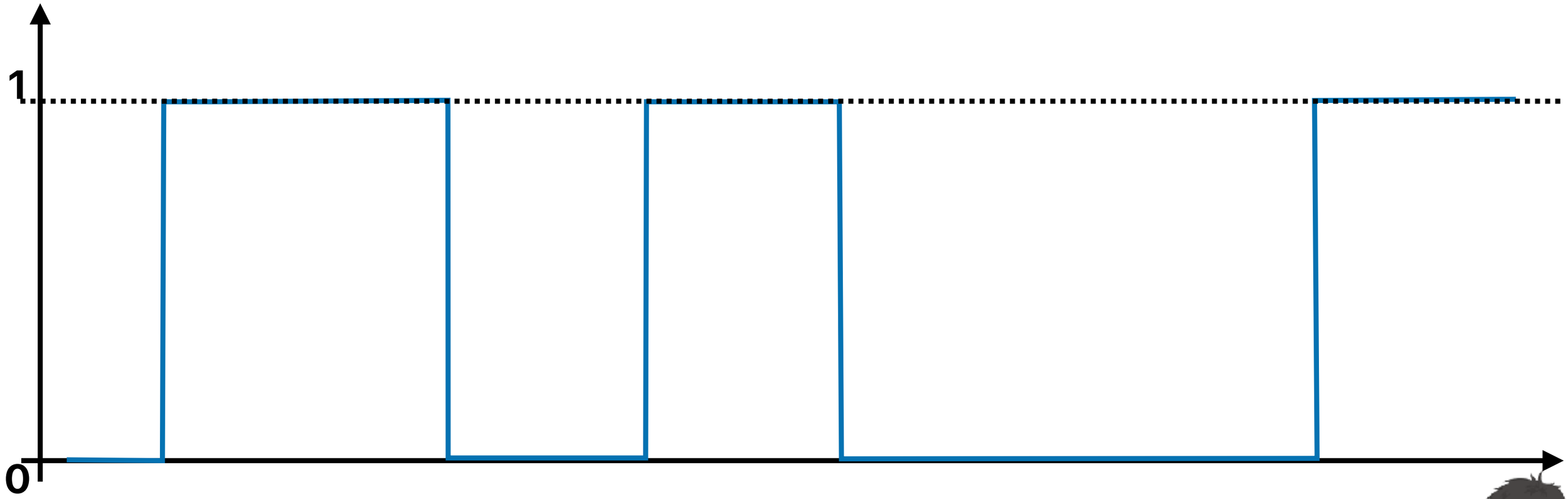
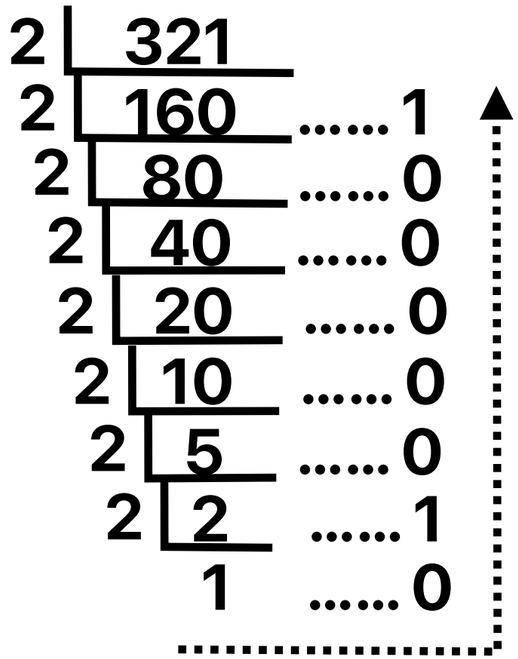
Datapath Components

Prof. Usagi

Recap: Digital circuits only have 0s and 1s…



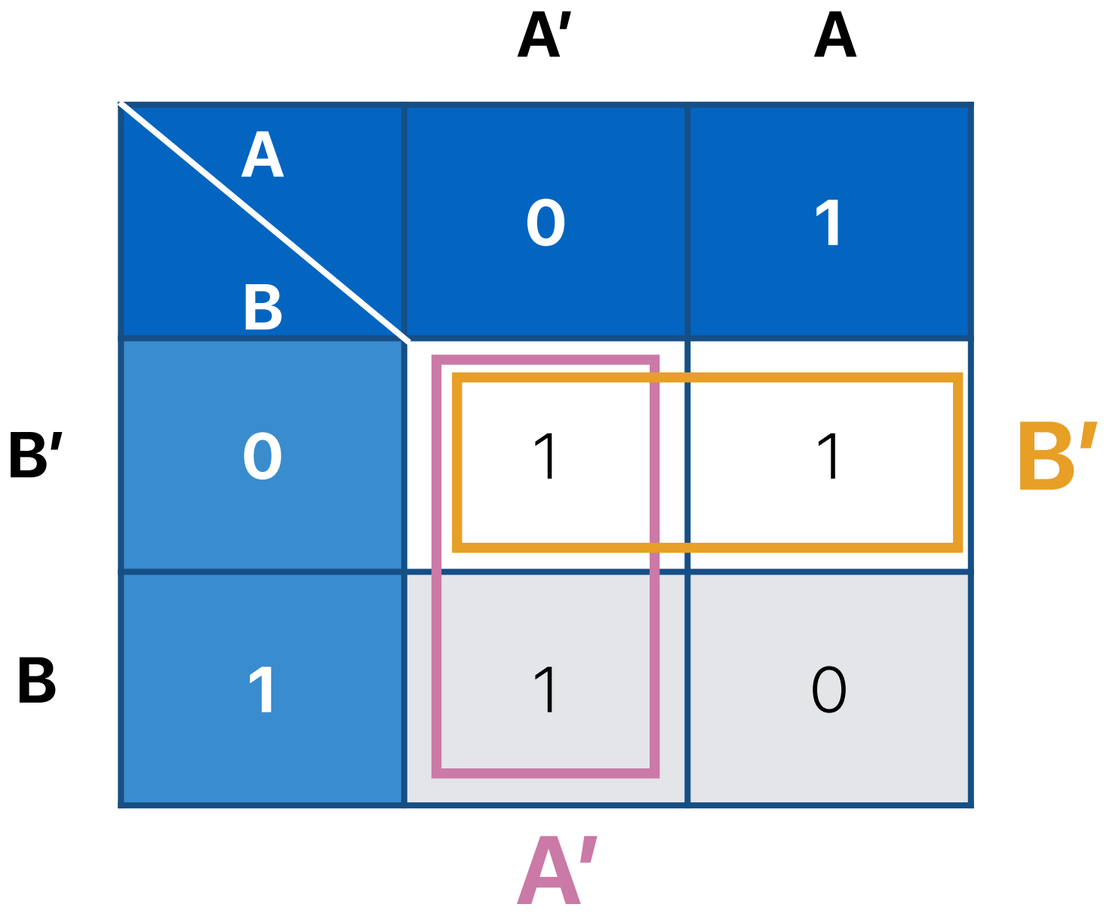
Recap: Converting from decimal to binary



321 = 0b**101000001**

Recap: 2-variable K-map example

Input		Output
A	B	
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

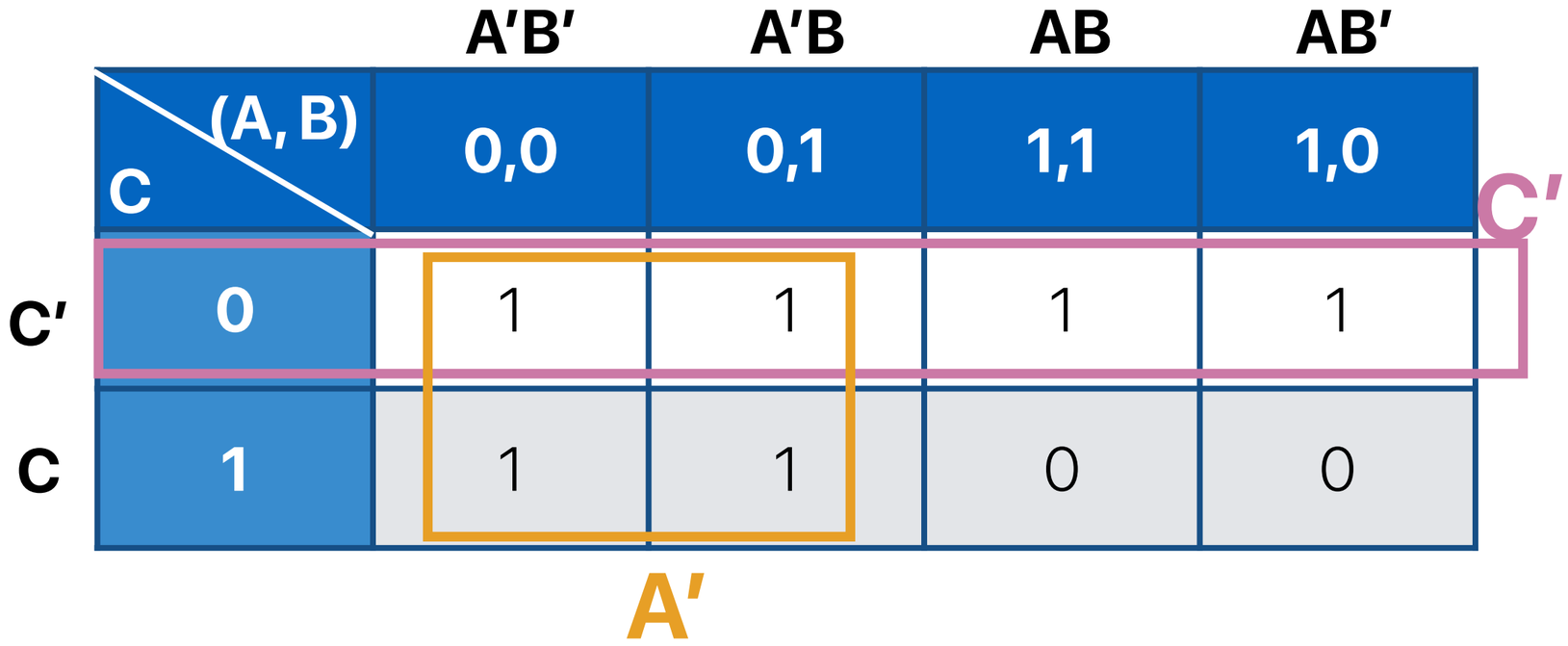


$$F(A, B) = A' + B'$$

Recap: 3-variable K-map

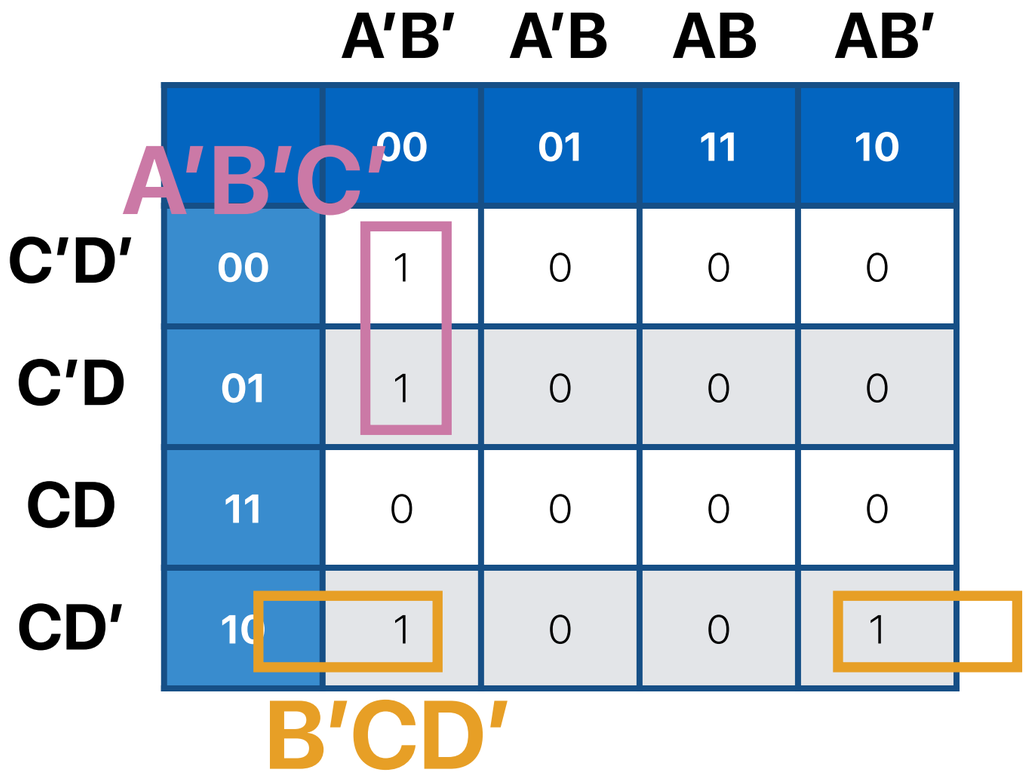
- Reduce to 2-variable K-map — 1 dimension will represent two variables
- Adjacent points should differ by only 1 bit
  - So we only change one variable in the neighboring column
  - 00, 01, 11, 10 — such numbering scheme is so-called **Gray–code**

Input			Output
A	B	C	
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**



Recap: 4-variable K-map

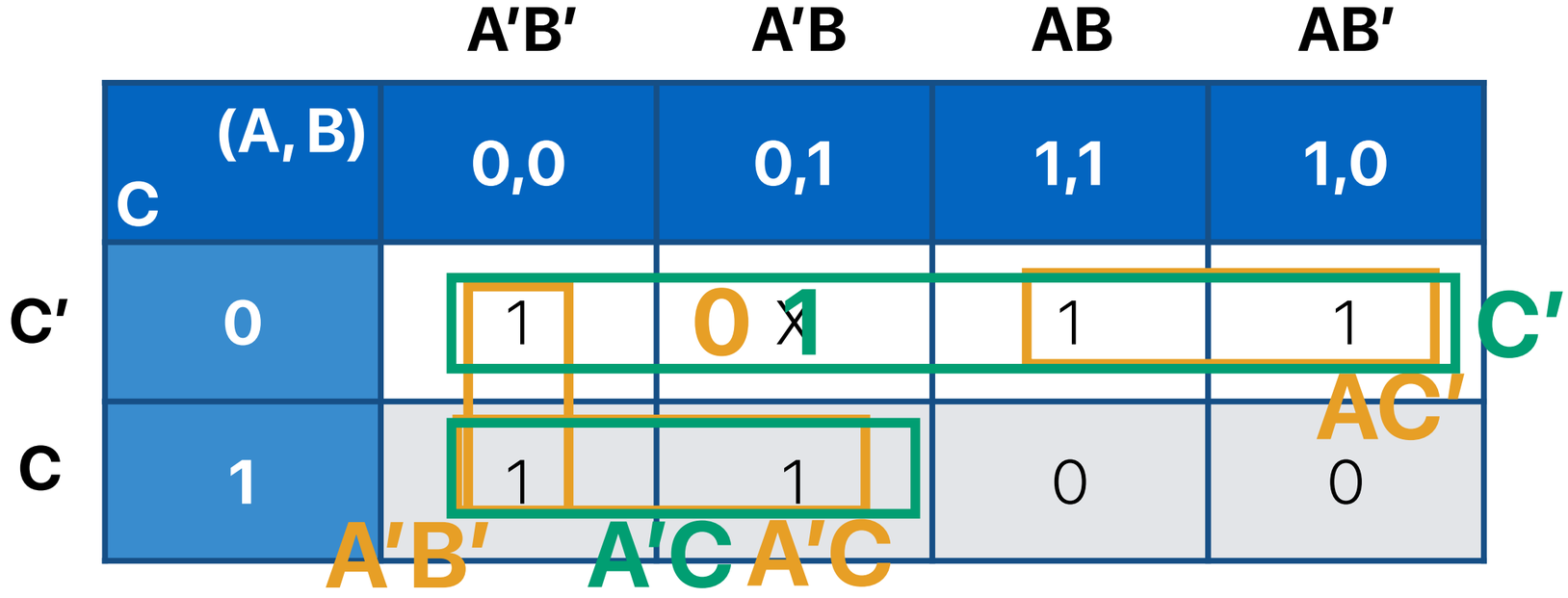
- Reduce to 2-variable K-map — both dimensions will represent two variables
- Adjacent points should differ by only 1 bit
  - So we only change one variable in the neighboring column
  - Use Gray-coding — 00, 01, 11, 10



$$F(A, B, C) = A'B'C' + B'CD'$$

Recap: K-Map with "Don't Care"s

You can treat "X" as either 0 or 1 — depending on which is more advantageous



If we treat the "X" as 0?

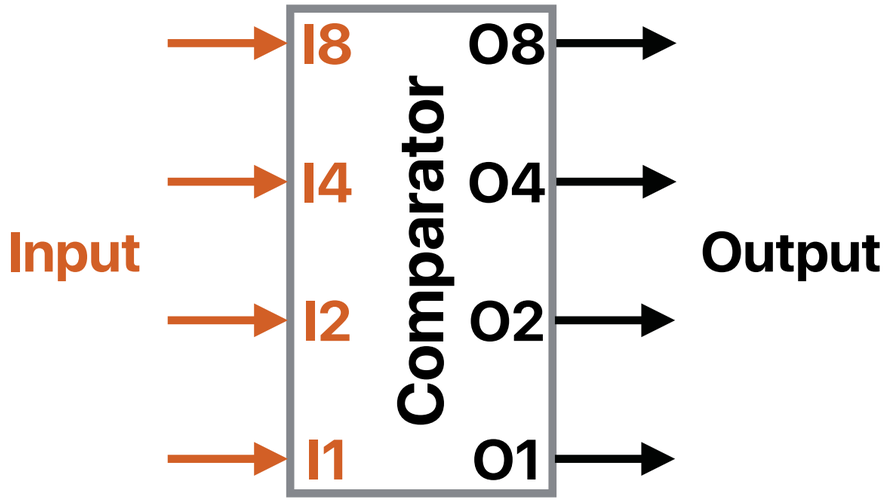
$F(A,B,C)=A'B'+A'C+AC'$

If we treat the "X" as 1?

$F(A,B,C) = C' + A'C$

BCD+1 — Binary coded decimal + 1

- 0x0 — 1
- 0x1 — 2
- 0x2 — 3
- 0x3 — 4
- 0x4 — 5
- 0x5 — 6
- 0x6 — 7
- 0x7 — 8
- 0x8 — 9
- 0x9 — 0
- 0xA — 0xF — Don’t care



Input				Output			
I8	I4	I2	I1	O8	O4	O2	O1
0	0	0	0	**0**	**0**	**0**	**1**
0	0	0	1	**0**	**0**	**1**	**0**
0	0	1	0	**0**	**0**	**1**	**1**
0	0	1	1	**0**	**1**	**0**	**0**
0	1	0	0	**0**	**1**	**0**	**1**
0	1	0	1	**0**	**1**	**1**	**0**
0	1	1	0	**0**	**1**	**1**	**1**
0	1	1	1	**1**	**0**	**0**	**0**
1	0	0	0	**1**	**0**	**0**	**1**
1	0	0	1	**0**	**0**	**0**	**0**
1	0	1	0	**X**	**X**	**X**	**X**
1	0	1	1	**X**	**X**	**X**	**X**
1	1	0	0	**X**	**X**	**X**	**X**
1	1	0	1	**X**	**X**	**X**	**X**
1	1	1	0	**X**	**X**	**X**	**X**
1	1	1	1	**X**	**X**	**X**	**X**

K-maps

Input				Output			
I8	I4	I2	I1	O8	O4	O2	O1
0	0	0	0	**0**	**0**	**0**	**1**
0	0	0	1	**0**	**0**	**1**	**0**
0	0	1	0	**0**	**0**	**1**	**1**
0	0	1	1	**0**	**1**	**0**	**0**
0	1	0	0	**0**	**1**	**0**	**1**
0	1	0	1	**0**	**1**	**1**	**0**
0	1	1	0	**0**	**1**	**1**	**1**
0	1	1	1	**1**	**0**	**0**	**0**
1	0	0	0	**1**	**0**	**0**	**1**
1	0	0	1	**0**	**0**	**0**	**0**
1	0	1	0	**X**	**X**	**X**	**X**
1	0	1	1	**X**	**X**	**X**	**X**
1	1	0	0	**X**	**X**	**X**	**X**
1	1	0	1	**X**	**X**	**X**	**X**
1	1	1	0	**X**	**X**	**X**	**X**
1	1	1	1	**X**	**X**	**X**	**X**

O8

	I8′I4′	I8′I4	I8I4	I8I4′
	00	**01**	**11**	**10**
I2′I1′ **00**	0	0	X	1
I2′I1 **01**	0	0	X	0
I2I1 **11**	0	1	X	X
I2I1′ **10**	0	0	X	X

O4

	I8′I4′	I8′I4	I8I4	I8I4′
	00	**01**	**11**	**10**
I2′I1′ **00**	0	1	X	0
I2′I1 **01**	0	1	X	0
I2I1 **11**	1	0	X	X
I2I1′ **10**	0	1	X	X

O2

	I8′I4′	I8′I4	I8I4	I8I4′
	00	**01**	**11**	**10**
I2′I1′ **00**	0	0	X	0
I2′I1 **01**	1	1	X	0
I2I1 **11**	0	0	X	X
I2I1′ **10**	1	1	X	X

O1

	I8′I4′	I8′I4	I8I4	I8I4′
	00	**01**	**11**	**10**
I2′I1′ **00**	1	1	X	1
I2′I1 **01**	0	0	X	0
I2I1 **11**	0	0	X	X
I2I1′ **10**	1	1	X	X

Outline

- Revisiting the binary number system
- Adders
- Multiplexer

What do we want from a number system?

- Obvious representation of 0, 1, 2, ……
- Represent positive/negative/integer/floating points
- Efficient usage of number space
- Equal coverage of positive and negative numbers
- Easy hardware design
  - Minimize the hardware cost/reuse the same hardware as much as possible
  - Easy to distinguish positive numbers
  - Easy to negation

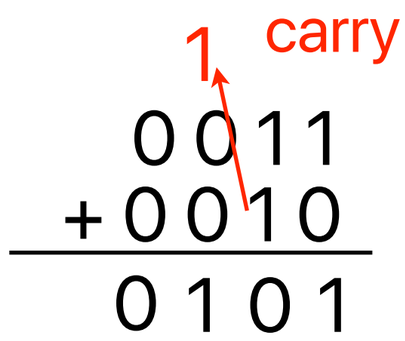
Representing a positive number

- Assume that we have 4 bits

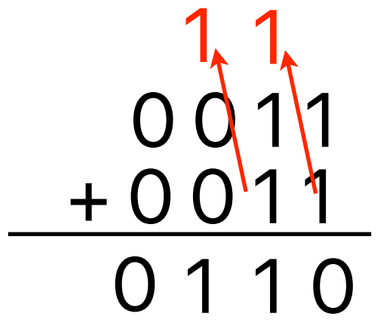
Decimal	Binary	Decimal	Binary
0	0000	4	0100
1	0001	5	0101
2	0010	6	0110
3	0011	7	0111

- Example binary arithmetic

3 + 2 = 5



3 + 3 = 6



Poll close in 1:30

The first proposal

- How many of the following goals can "**simply using the most significant bit as the signed bit**" to represent a negative number fulfill in the number system?
  - ① Obvious representation of 0, 1, 2, ...
  - ② Efficient usage of number space
  - ③ Equal coverage of positive and negative numbers
  - ④ Easy hardware design

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Decimal	Binary	Decimal	Binary
0	0000	-0	1000
1	0001	-1	1001
2	0010	-2	1010
3	0011	-3	1011
4	0100	-4	1100
5	0101	-5	1101
6	0110	-6	1110
7	0111	-7	1111

The first proposal

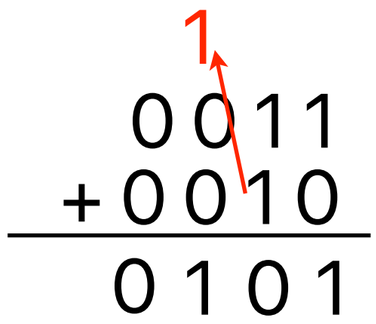
- How many of the following goals can "**simply using the most significant bit as the signed bit**" to represent a negative number fulfill in the number system?
  - ① Obvious representation of 0, 1, 2, ……
  - ② Efficient usage of number space
  - ③ Equal coverage of positive and negative numbers
  - ④ Easy hardware design

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

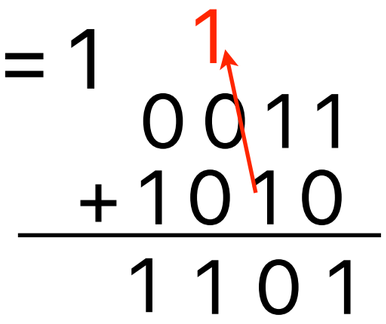
Decimal	Binary	Decimal	Binary
0	0000	-0	1000
1	0001	-1	1001
2	0010	-2	1010
3	0011	-3	1011
4	0100	-4	1100
5	0101	-5	1101
6	0110	-6	1110
7	0111	-7	1111

Can this work?

- 3 + 2 = 5



- 3 + (−2) = 1



= -5 (Not 1)

Doesn't work well and you need a separate procedure to deal with negative numbers!

The first proposal

- How many of the following goals can “simply using the most significant bit as the signed bit” to represent a negative number fulfill in the number system?
  - ① Obvious representation of 0, 1, 2, ……
  - ② Efficient usage of number space
  - ③ Equal coverage of positive and negative numbers
  - ④ Easy hardware design
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Decimal	Binary	Decimal	Binary
0	0000	-0	1000
1	0001	-1	1001
2	0010	-2	1010
3	0011	-3	1011
4	0100	-4	1100
5	0101	-5	1101
6	0110	-6	1110
7	0111	-7	1111

Poll close in 1:30

The second proposal — 1's complement

- How many of the following goals can "1's complement — **flip/not every bit in the corresponding positive number**" to represent a negative number fulfill in the number system?
  - ① Obvious representation of 0, 1, 2, ...
  - ② Efficient usage of number space
  - ③ Equal coverage of positive and negative numbers
  - ④ Easy hardware design

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Decimal	Binary	Decimal	Binary
0	0000	-0	1111
1	0001	-1	1110
2	0010	-2	1101
3	0011	-3	1100
4	0100	-4	1011
5	0101	-5	1010
6	0110	-6	1001
7	0111	-7	1000

The second proposal — 1's complement

- How many of the following goals can "1's complement — **flip/not every bit in the corresponding positive number**" to represent a negative number fulfill in the number system?
  - ① Obvious representation of 0, 1, 2, ...
  - ② Efficient usage of number space
  - ③ Equal coverage of positive and negative numbers
  - ④ Easy hardware design

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Decimal	Binary	Decimal	Binary
0	0000	-0	1111
1	0001	-1	1110
2	0010	-2	1101
3	0011	-3	1100
4	0100	-4	1011
5	0101	-5	1010
6	0110	-6	1001
7	0111	-7	1000

Second proposal: 1's complement

- 3 + 2 = 5

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

- 3 + (−2) = 1

```
overflow  1 1 1 1
          0 0 1 1
        + 1 1 0 1
        ---------
          0 0 0 0   = 0 (Still not 1)
```

Still does not work, but seems closer…

The second proposal — 1's complement

- How many of the following goals can "1's complement — **flip/not every bit in the corresponding positive number**" to represent a negative number fulfill in the number system?
  - ① Obvious representation of 0, 1, 2, ...
  - ② Efficient usage of number space
  - ③ Equal coverage of positive and negative numbers
  - ④ Easy hardware design

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Decimal	Binary	Decimal	Binary
0	0000	-0	1111
1	0001	-1	1110
2	0010	-2	1101
3	0011	-3	1100
4	0100	-4	1011
5	0101	-5	1010
6	0110	-6	1001
7	0111	-7	1000

Poll close in 1:30

The third proposal — 2's complement

- How many of the following goals can "2's complement — **take the 1's complement of corresponding positive number and then +1**" to represent a negative number fulfill in the number system?
  - ① Obvious representation of 0, 1, 2, ...
  - ② Efficient usage of number space
  - ③ Equal coverage of positive and negative numbers
  - ④ Easy hardware design

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Decimal	Binary	Decimal	Binary
0	0000	-1	1111
1	0001	-2	1110
2	0010	-3	1101
3	0011	-4	1100
4	0100	-5	1011
5	0101	-6	1010
6	0110	-7	1001
7	0111	-8	1000

The third proposal — 2's complement

- How many of the following goals can "2's complement — **take the 1's complement of corresponding positive number and then +1**" to represent a negative number fulfill in the number system?
  - ① Obvious representation of 0, 1, 2, ...
  - ② Efficient usage of number space
  - ③ Equal coverage of positive and negative numbers
  - ④ Easy hardware design
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Decimal	Binary	Decimal	Binary
0	0000	-1	1111
1	0001	-2	1110
2	0010	-3	1101
3	0011	-4	1100
4	0100	-5	1011
5	0101	-6	1010
6	0110	-7	1001
7	0111	-8	1000

Does not waste 1111 anymore

Poll close in 1:30

Evaluating 2's complement

- Do we need a separate procedure/hardware for adding positive and negative numbers?

A. No. The same procedure applies
B. No. The same "procedure" applies but it changes overflow detection
C. Yes, and we need a new procedure
D. Yes, and we need a new procedure and a new hardware
E. None of the above

Evaluating 2's complement

- Do we need a separate procedure/hardware for adding positive and negative numbers?

- 3 + 2 = 5

```
   1
  0011
+ 0010
------
  0101
```

- 3 + (-2) = 1

```
  1 1
 1011
+1110
------
 0001   = 1
```

A. No. The same procedure applies
B. No. The same "procedure" applies but it changes overflow detection
C. Yes, and we need a new procedure
D. Yes, and we need a new procedure and a new hardware
E. None of the above

The third proposal — 2's complement

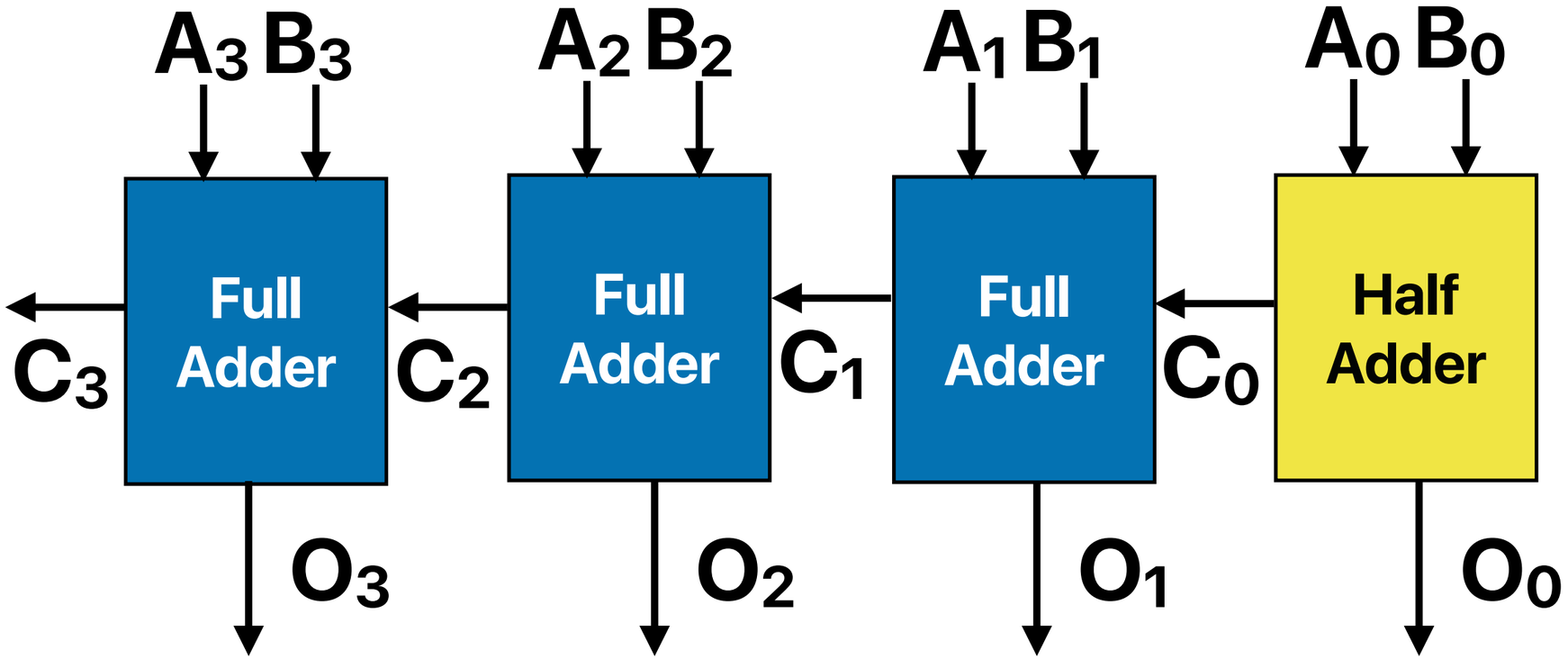
- How many of the following goals can "2's complement — **take the 1's complement of corresponding positive number and then +1**" to represent a negative number fulfill in the number system?
  1. Obvious representation of 0, 1, 2, ...
  2. Efficient usage of number space
  3. Equal coverage of positive and negative numbers
  4. Easy hardware design

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Decimal	Binary	Decimal	Binary
0	0000	-1	1111
1	0001	-2	1110
2	0010	-3	1101
3	0011	-4	1100
4	0100	-5	1011
5	0101	-6	1010
6	0110	-7	1001
7	0111	-8	1000

Adder

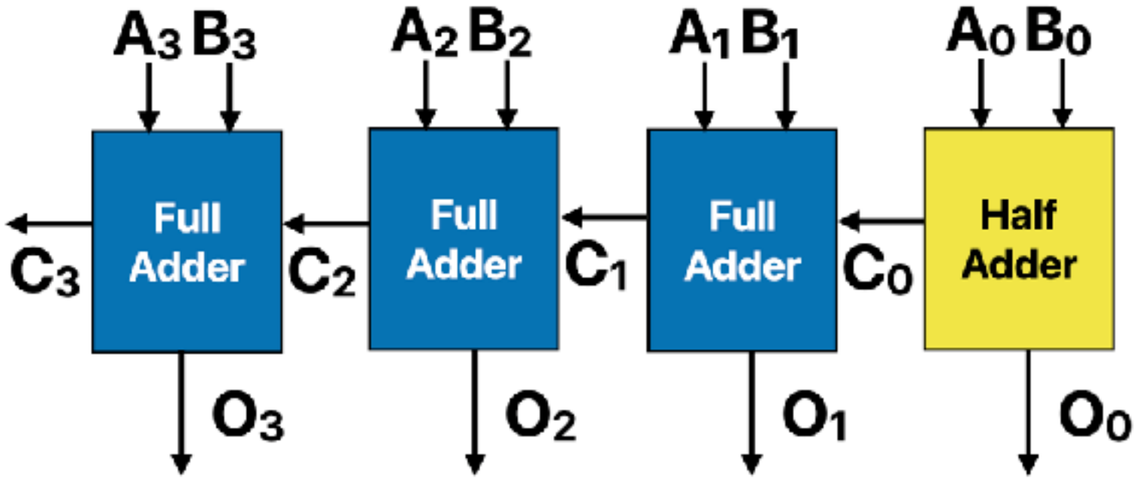
We've built this before!



Poll close in 1:30

If we want to support subtraction?

- If we would like to extend the 4-bit adder that we've built before to support "A-B" with 2's complement, how many of the followings should we add at least?
  - ① Provide an option to use bitwise NOT A
  - ② Provide an option to use bitwise NOT B
  - ③ Provide an option to use bitwise A XOR B
  - ④ Provide an option to add 0 to the input of the half adder
  - ⑤ Provide an option to add 1 to the input of the half adder
  - A. 1
  - B. 2
  - C. 3
  - D. 4
  - E. 5



If we want to support subtraction?

- If we would like to extend the 4-bit adder that we've built before to support "A-B" with 2's complement, how many of the followings should we add at least?
  - ① Provide an option to use bitwise NOT A
  - ② Provide an option to use bitwise NOT B
  - ③ Provide an option to use bitwise A XOR B
  - ④ Provide an option to add 0 to the input of the half adder
  - ⑤ Provide an option to add 1 to the input of the half adder

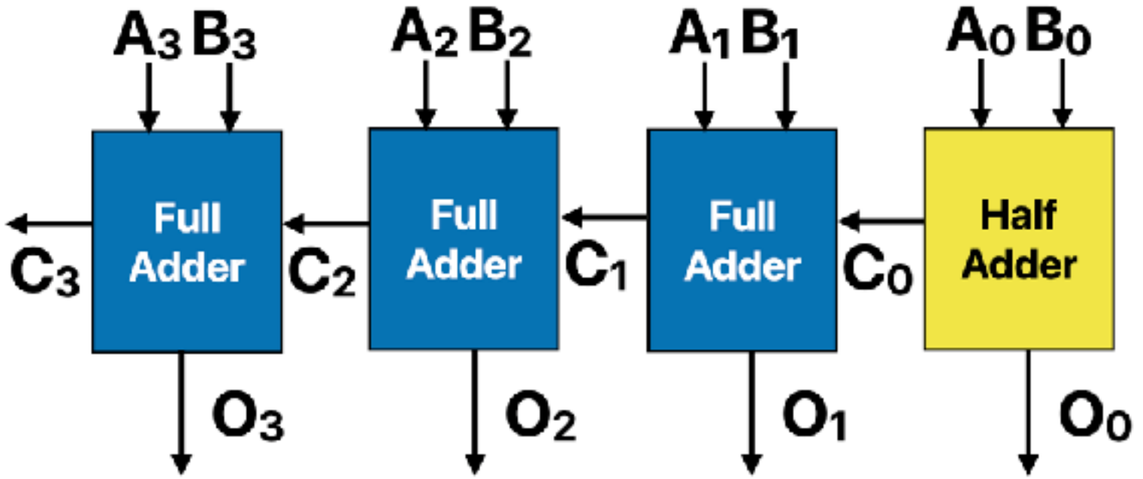
  A. 1

  B. 2

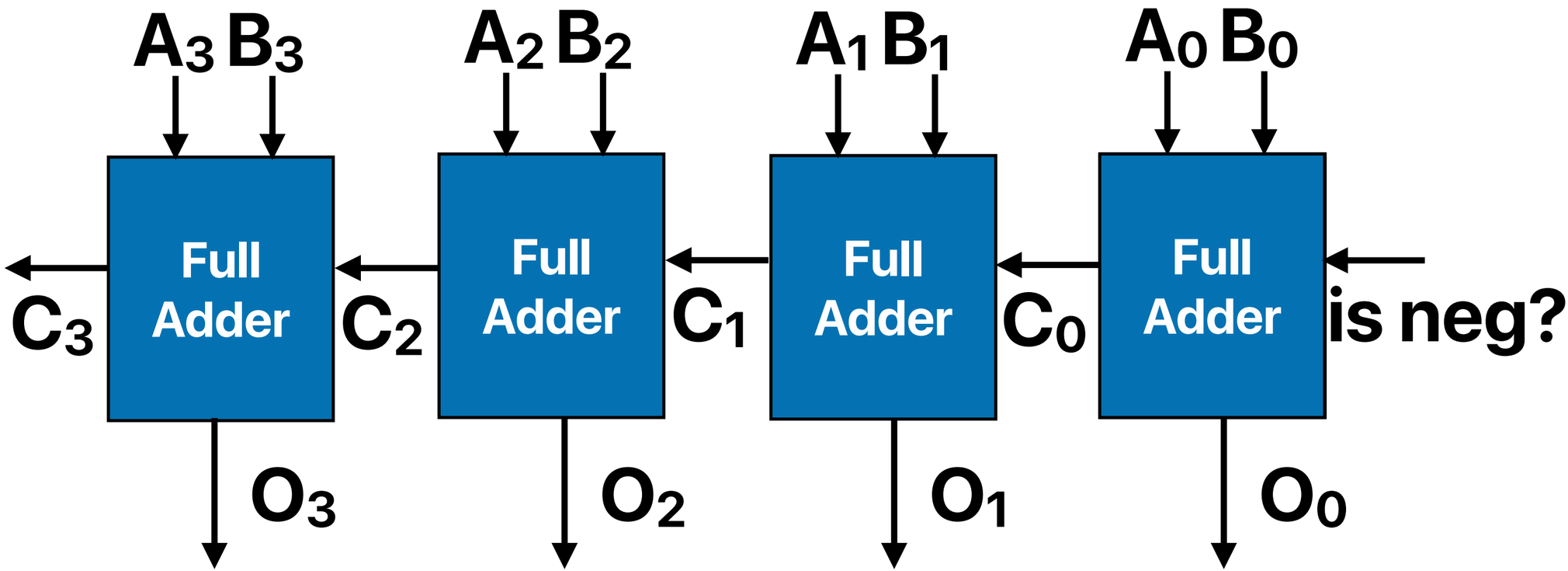
  C. 3

  D. 4

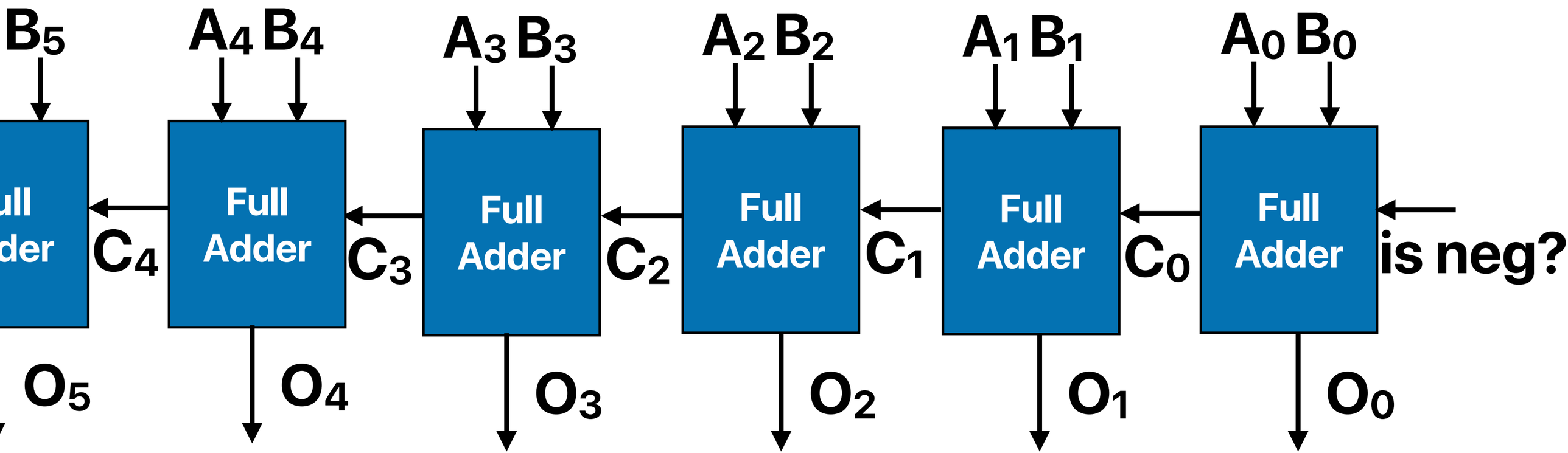
  E. 5



This is what we want!



We can support more bits!



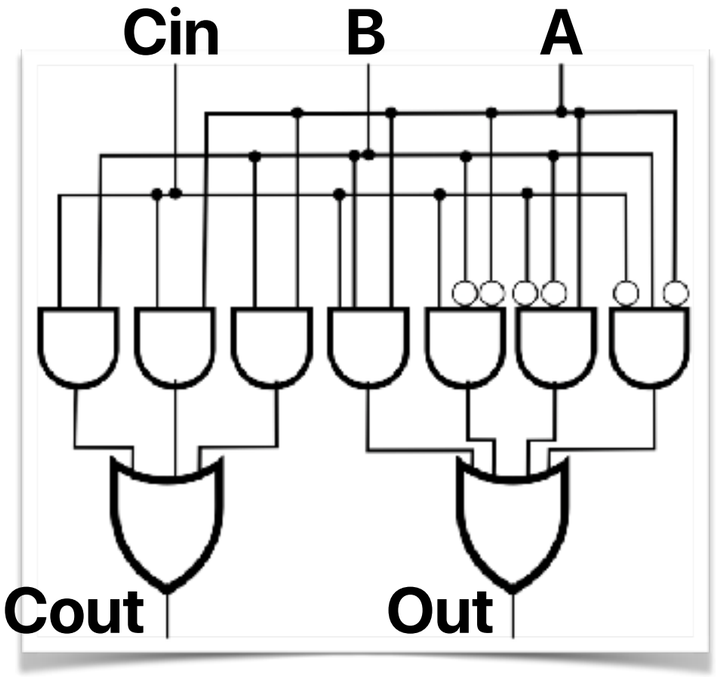
Recap: Full Adder

Input			Output	
A	B	Cin	Out	Cout
0	0	0	**0**	**0**
0	1	0	**1**	**0**
1	0	0	**1**	**0**
1	1	0	**0**	**1**
0	0	1	**1**	**0**
0	1	1	**0**	**1**
1	0	1	**0**	**1**
1	1	1	**1**	**1**

		A′B′	A′B	AB	AB′
	Cout(A, B)	0,0	0,1	1,1	1,0
Cin′	0	0	0	1	0
Cin	1	0	1	1	1

BCin AB ACin

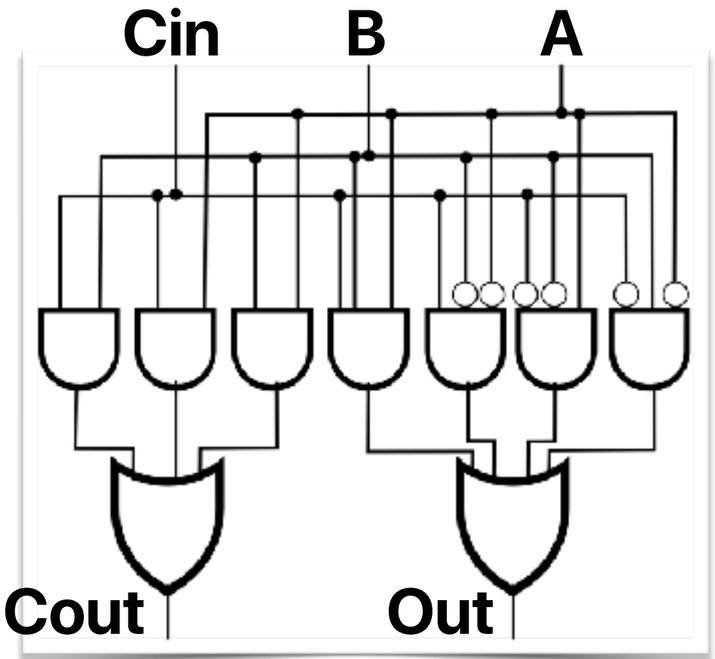
		A′B′	A′B	AB	AB′
	Out(A, B)	0,0	0,1	1,1	1,0
Cin′	0	0	1	0	1
Cin	1	1	0	1	0



Poll close in 1:30

How efficient is the adder?

- One approach estimates transistors, assuming every gate input requires 2 transistors, and ignoring inverters for simplicity. A 2-input gate requires 2 inputs · 2 trans/input = 4 transistors. A 3-input gate requires 3 · 2 = 6 transistors. A 4-input gate: 8 transistors. Wires also contribute to size, but ignoring wires as above is a common approximation.
- Considering the shown 1-bit full adder and use it to build a 32-bit adder, how many transistor do we need?
  A. 1152
  B. 1600
  C. 1664
  D. 1792
  E. 1984

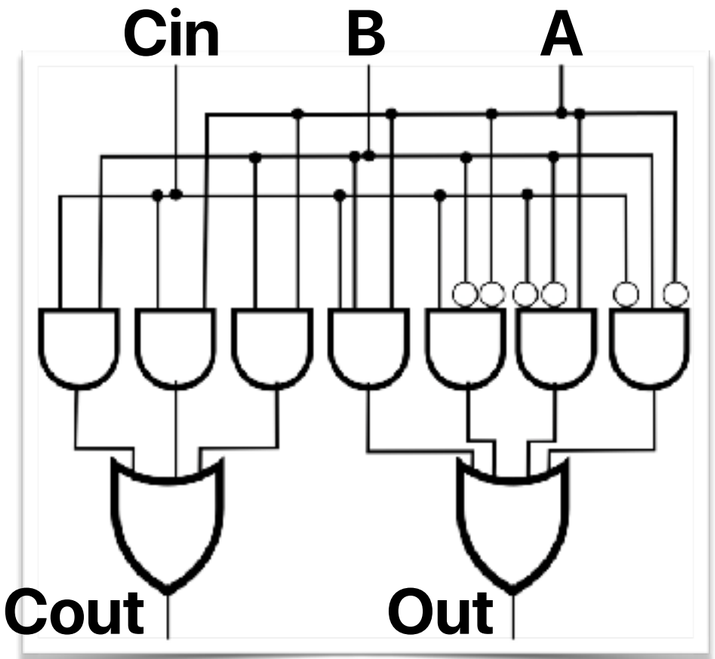


How efficient is the adder?

- One approach estimates transistors, assuming every gate input requires 2 transistors, and ignoring inverters for simplicity. A 2-input gate requires 2 inputs · 2 trans/input = 4 transistors. A 3-input gate requires 3 · 2 = 6 transistors. A 4-input gate: 8 transistors. Wires also contribute to size, but ignoring wires as above is a common approximation.
- Considering the shown 1-bit full adder and use it to build a 32-bit adder, how many transistor do we need?

A. 1152
B. 1600
C. 1664
D. 1792
E. 1984

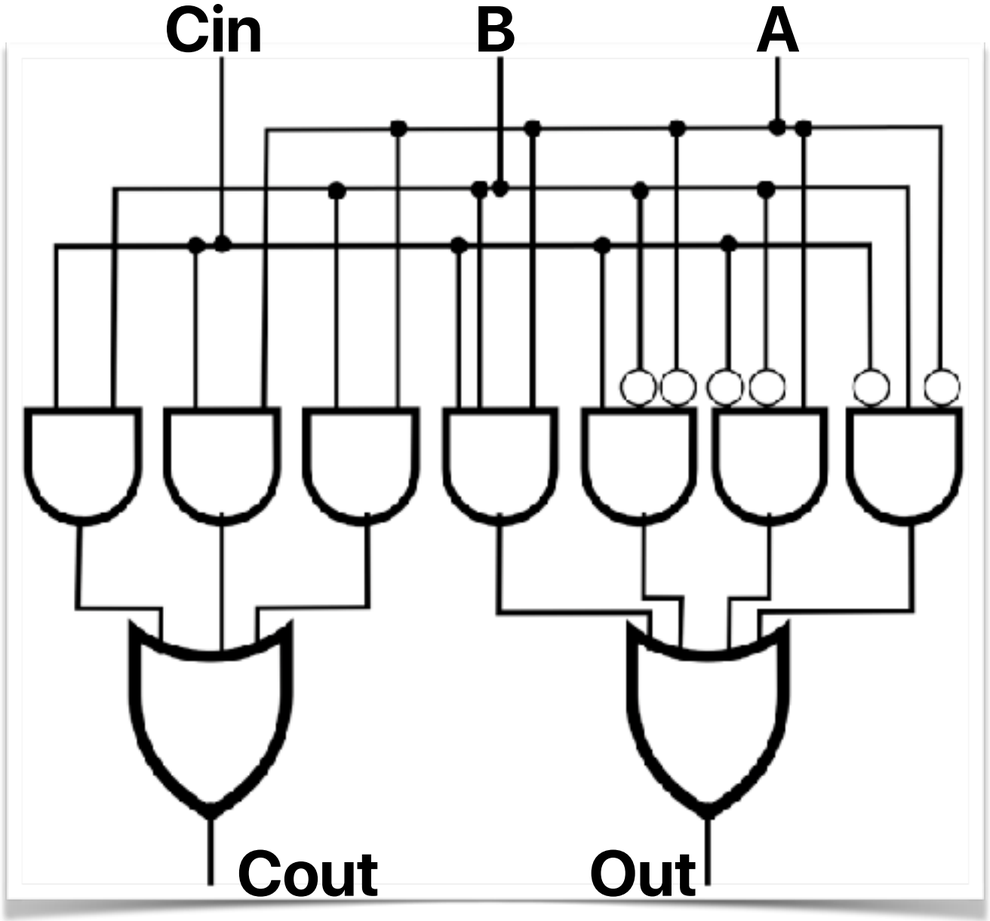
# of 2-inputs: 3
# of 3-inputs: 5
# of 4-inputs: 1
= 3*4 + 5*6 + 1*8 = 50 each



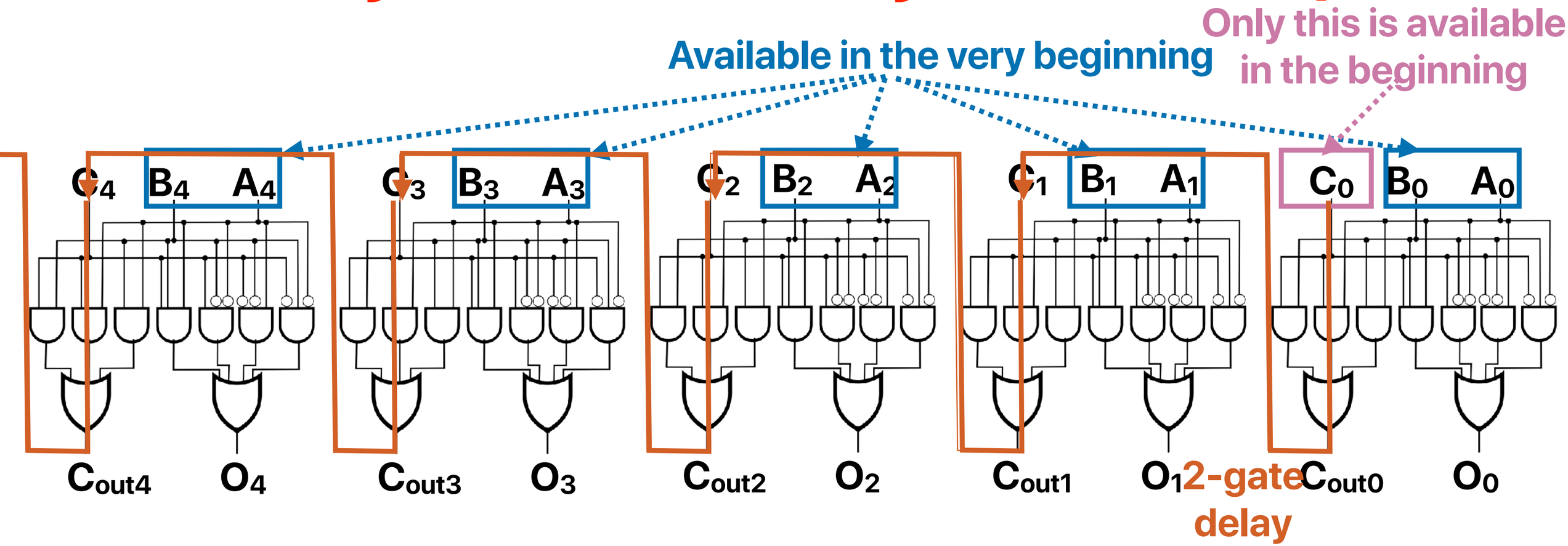
Poll close in 1:30

How efficient is the adder?

- Considering the shown 1-bit full adder and use it to build a 32-bit adder, how many gate-delays are we suffering to getting the final output?
  A. 2
  B. 32
  C. 64
  D. 128
  E. 288



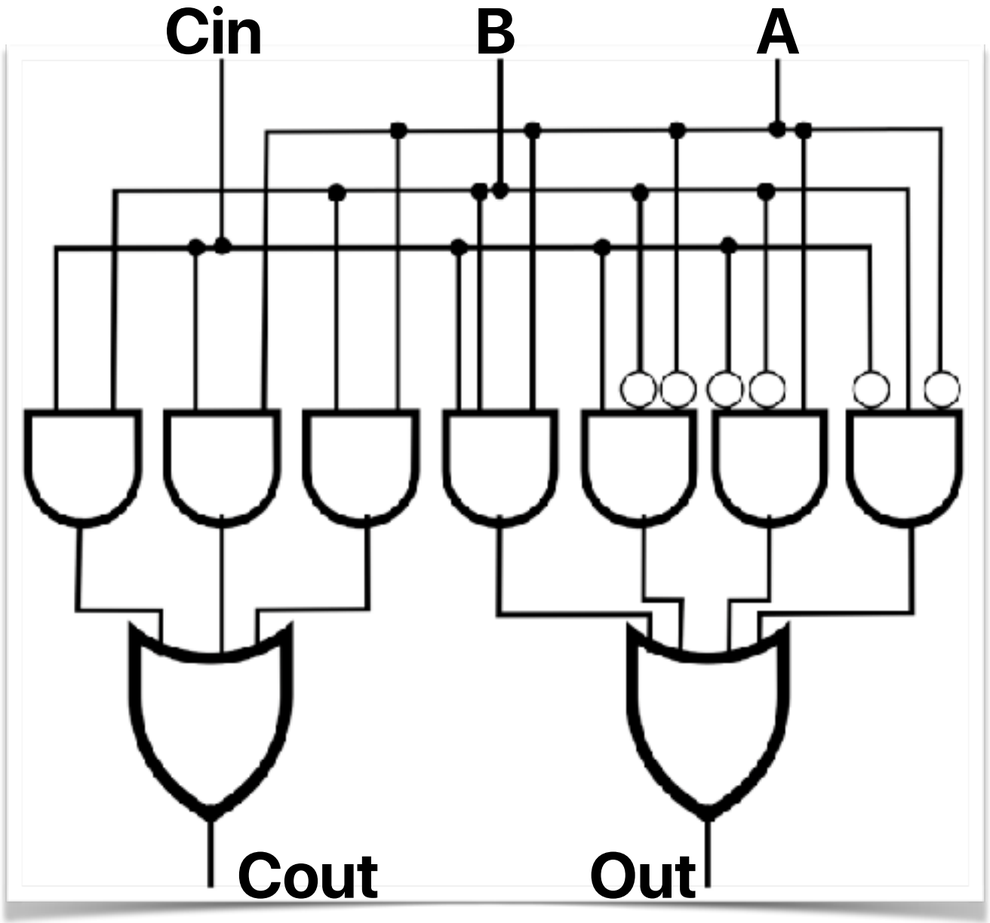
The delay is determined by the "critical path"



Carry-Ripple Adder

How efficient is the adder?

- Considering the shown 1-bit full adder and use it to build a 32-bit adder, how many gate-delays are we suffering to getting the final output?
  - A. 2
  - B. 32
  - C. 64
  - D. 128
  - E. 288



Carry-lookahead adder

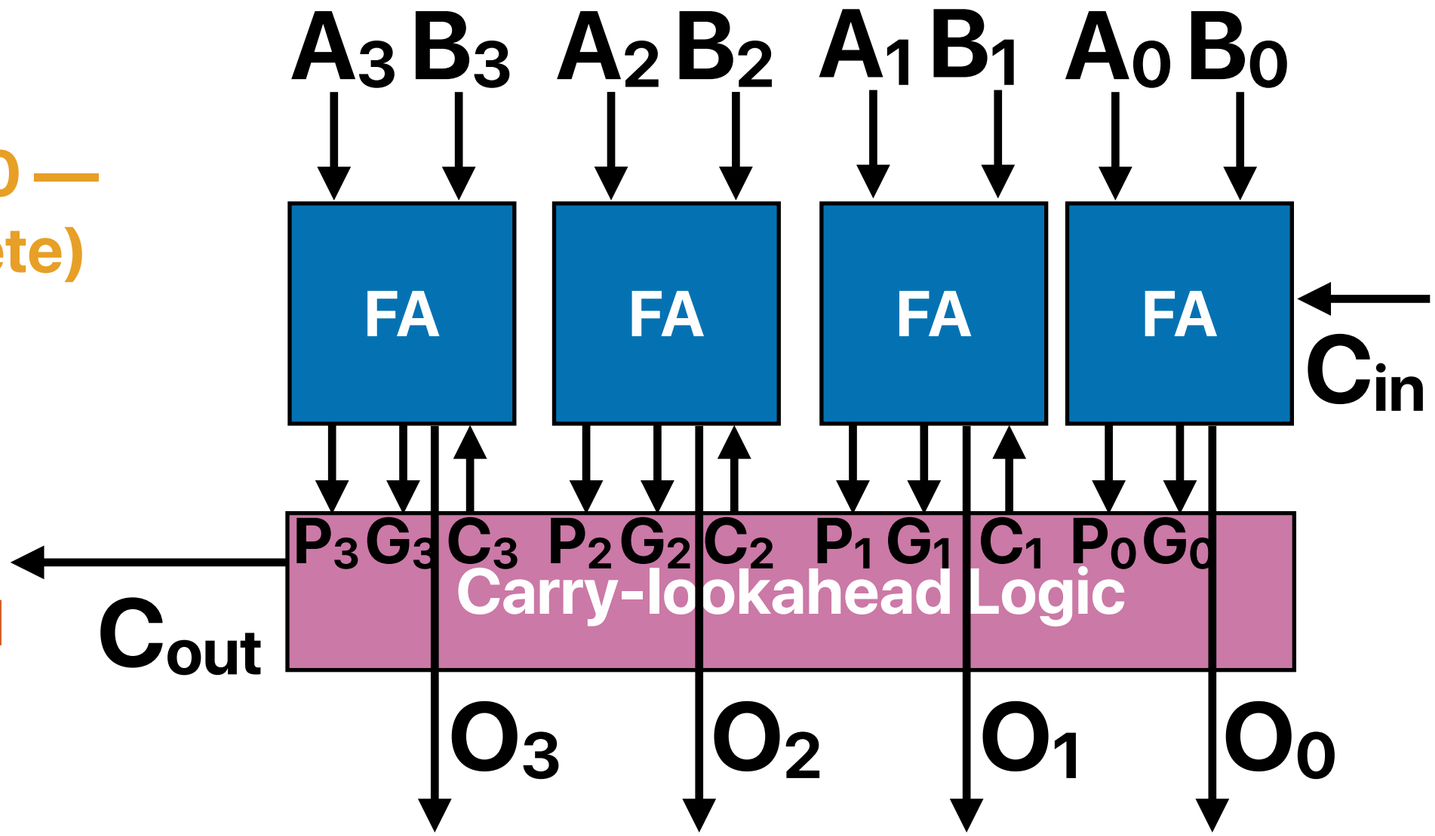
- Uses logic to quickly pre-compute the carry for each digit

Input			Output	
A	B	Cin	Out	Cout
0	0	0	**0**	**0**
0	0	1	**1**	**0**
0	1	0	**1**	**0**
0	1	1	**0**	**1**
1	0	0	**1**	**0**
1	0	1	**0**	**1**
1	1	0	**0**	**1**
1	1	1	**1**	**1**

Both A, B are 0 — no carry (Delete)

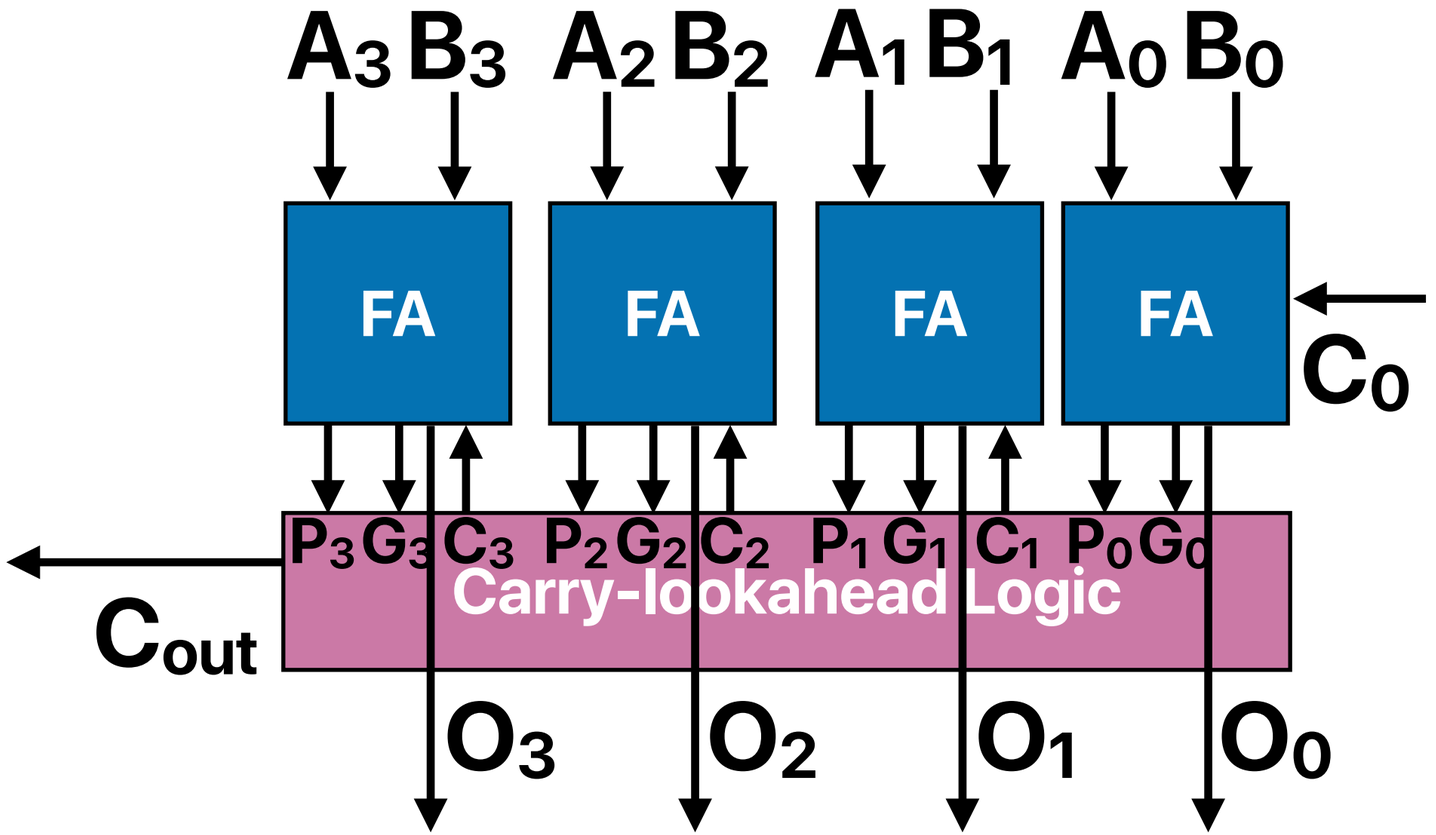
Needs to wait Cin (Propagate)

Both A, B are 1 — must carry (Generate)



CLA (cont.)

- All "G" and "P" are immediately available (only need to look over Ai and Bi), but "c" are not (except the c0).



$$G_i = A_i B_i$$

$$P_i = A_i \text{ XOR } B_i$$

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0)$$
$$= G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2$$
$$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

$$C_4 = G_3 + P_3 C_3$$
$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

Poll close in 1:30

CLA's gate delay

- What's the gate-delay of a 4-bit CLA?
  A. 2
  B. 4
  C. 6
  D. 8
  E. 10

$$G_i = A_i B_i$$

$$P_i = A_i \text{ XOR } B_i$$

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0)$$
$$= G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2$$
$$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

$$C_4 = G_3 + P_3 C_3$$
$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$$
$$+ P_3 P_2 P_1 P_0 C_0$$

CLA's gate delay

- What's the gate-delay of a 4-bit CLA?

A. 2

B. 4

C. 6

D. 8

E. 10

$$G_i = A_i B_i$$

$$P_i = A_i \text{ XOR } B_i$$

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0)$$
$$= G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2$$
$$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

$$C_4 = G_3 + P_3 C_3$$
$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$$
$$+ P_3 P_2 P_1 P_0 C_0$$

Poll close in 1:30

CLA's size

- How many transistors do we need to implement a 4-bit CLA logic?
  A. 38
  B. 64
  C. 92
  D. 116
  E. 128

$$G_i = A_i B_i$$

$$P_i = A_i \text{ XOR } B_i$$

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0)$$
$$= G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2$$
$$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

$$C_4 = G_3 + P_3 C_3$$
$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$$
$$+ P_3 P_2 P_1 P_0 C_0$$

CLA's size

- How many transistors do we need to implement a 4-bit CLA logic?
  - A. 38
  - B. 64
  - C. 92
  - D. 116
  - E. 128

$G_i = A_i B_i$ 4 * 4 = 16

$P_i = A_i \text{ XOR } B_i$ 4 * 4 = 16

$C_1 = G_0 + P_0 C_0$ 4 + 4 = 8

$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0)$

$= G_1 + P_1 G_0 + P_1 P_0 C_0$ 4 + 6 + 6 = 16

$C_3 = G_2 + P_2 C_2$

$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

$C_4 = G_3 + P_3 C_3$ 4 + 6 + 8 + 8 =26

$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$

$+ P_3 P_2 P_1 P_0 C_0$ 4 + 6 + 8 + 10 + 10 = 38

CLA v.s. Carry-ripple

- Size:
  - 32-bit CLA with 4-bit CLAs — requires 8 of 4-bit CLA
    - Each requires 116 for the CLA 4*(4*6+8) for the A+B — 244 gates
    - 1952 transistors
  - 32-bit CRA
    - 1600 transistors **Win!**
- Delay
  - 32-bit CLA with 8 4-bit CLAs
    - 2 gates **Win!**
  - 32-bit CRA
    - 64 gates

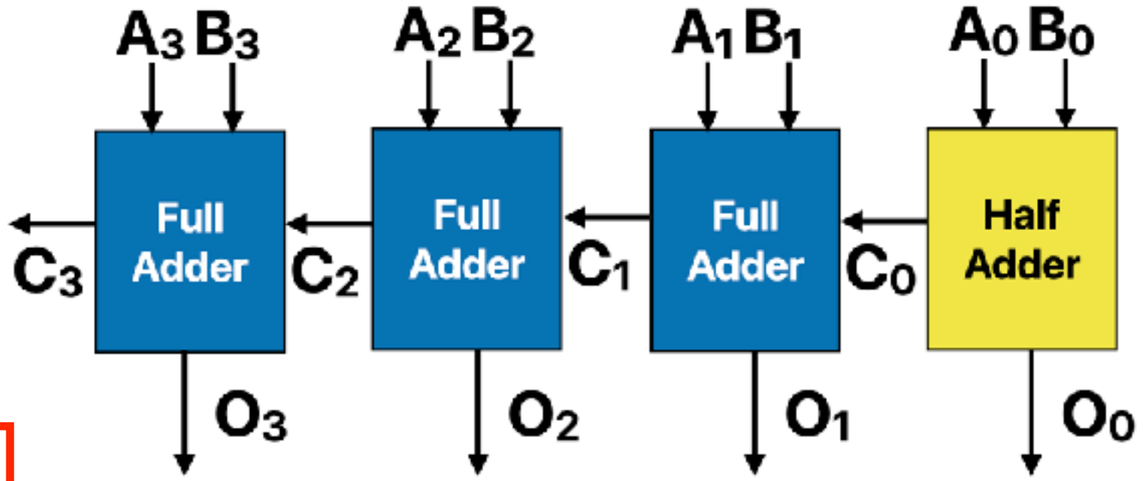
Area-Delay Trade-off!

Recap: If we want to support subtraction?

- If we would like to extend the 4-bit adder that we've built before to support "A-B" with 2's complement, how many of the followings should we add at least?
  - ① Provide an option to use bitwise NOT A
  - ② Provide an option to use bitwise NOT B
  - ③ Provide an option to use bitwise A XOR B
  - ④ Provide an option to add 0 to the input of the half adder
  - ⑤ Provide an option to add 1 to the input of the half adder

  A. 1
  B. 2
  C. 3
  D. 4
  E. 5

How to provide this option



To "NOT" or not to "NOT", that's the question!

Announcement

- Lab 2 due **tonight**
  - Watch the video and read the instruction BEFORE your session
  - There are links on both course webpage and iLearn lab section
  - Submit through iLearn > Labs
- Reading quiz 4 due 4/21 **BEFORE** the lecture
  - Under iLearn > reading quizzes
- Assignment 2 due 4/23
  - Submit on zyBooks.com directly — all challenge questions 2.3-3.5
- Lab 3 due 4/30
  - Watch the video and read the instruction BEFORE your session
  - There are links on both course webpage and iLearn lab section
  - Submit through iLearn > Labs
- Check your grades in iLearn

Electrical
Computer Science
Engineering

120A

つづく