Datapath Components (I)

Prof. Usagi

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

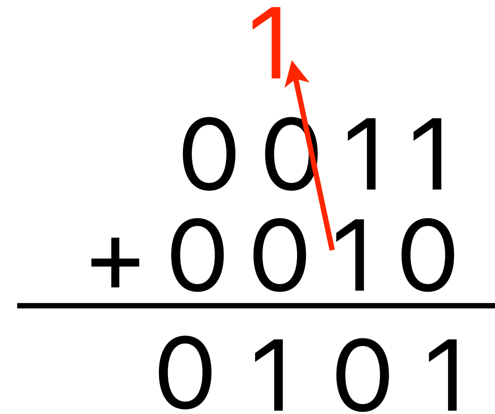
```
  1   carry
  0 0 1 1
+ 0 0 1 0
---------
  0 1 0 1
```

3 + 3 = 6

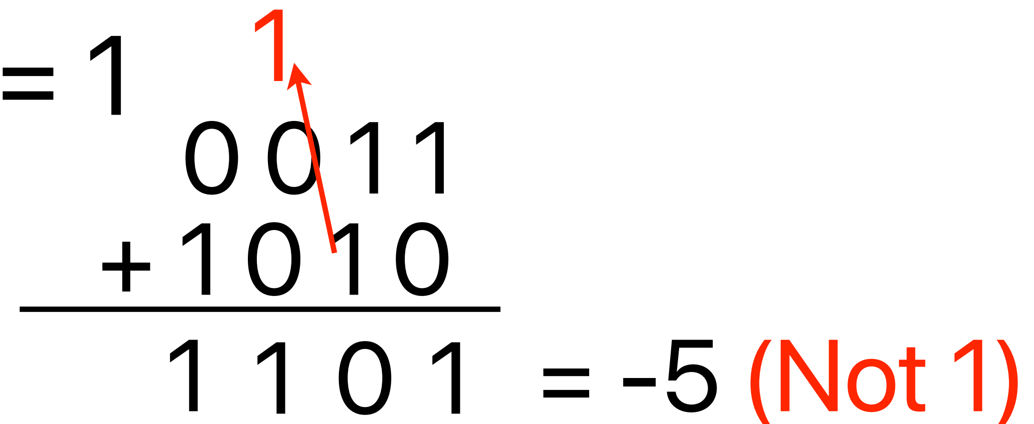
```
  1 1
  0 0 1 1
+ 0 0 1 1
---------
  0 1 1 0
```

Can this work?

- 3 + 2 = 5



- 3 + (-2) = 1



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

Second proposal: 1's complement

- 3 + 2 = 5

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

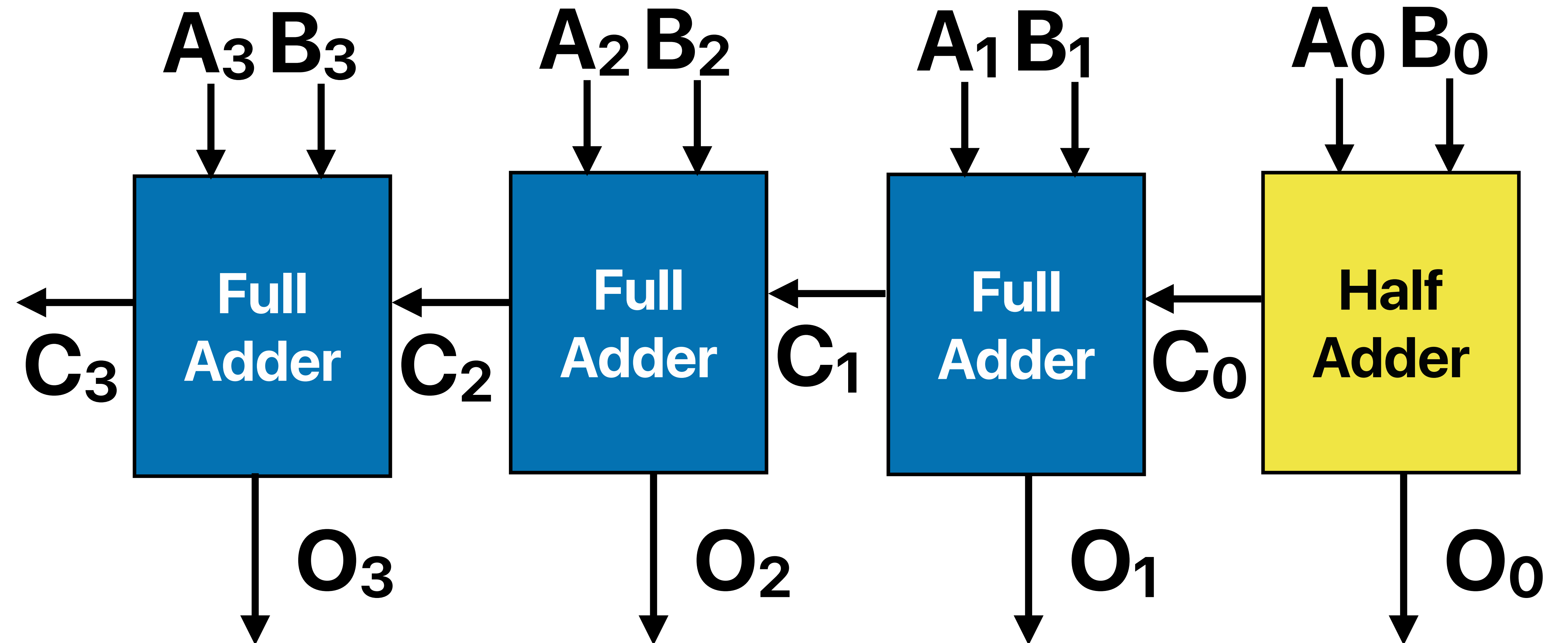
- 3 + (-2) = 1

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

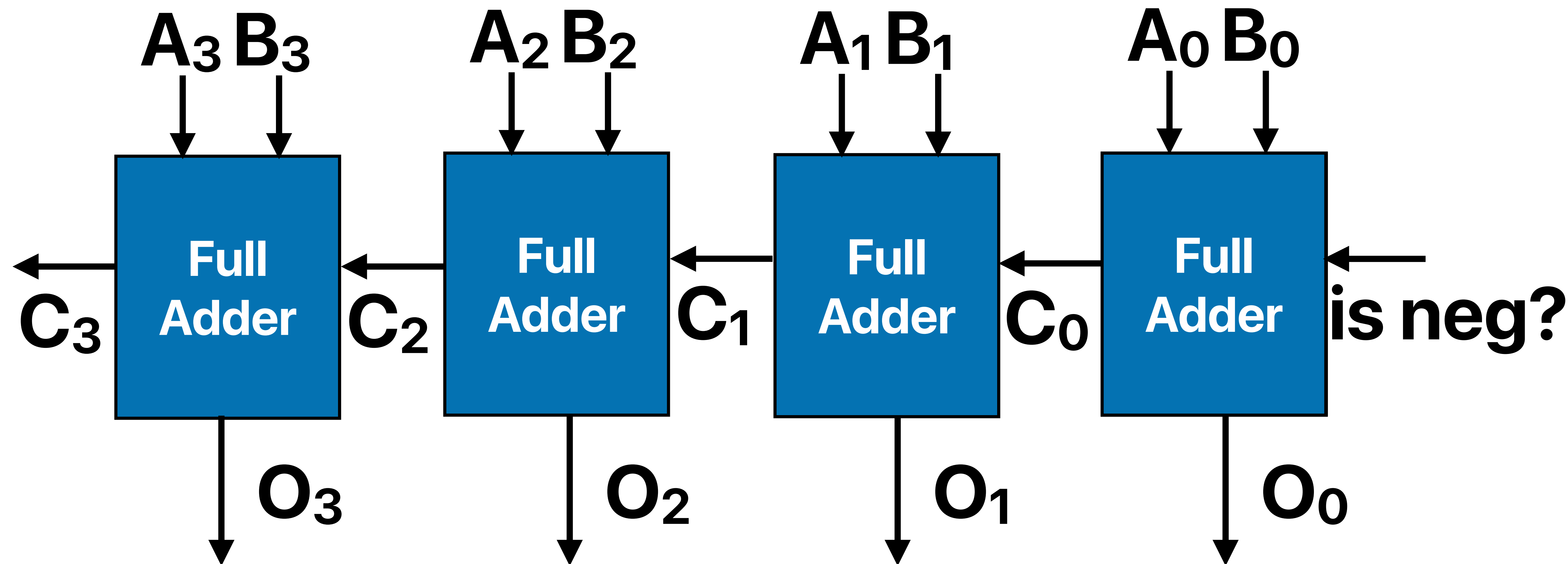
Still does not work, but seems closer...

Adder

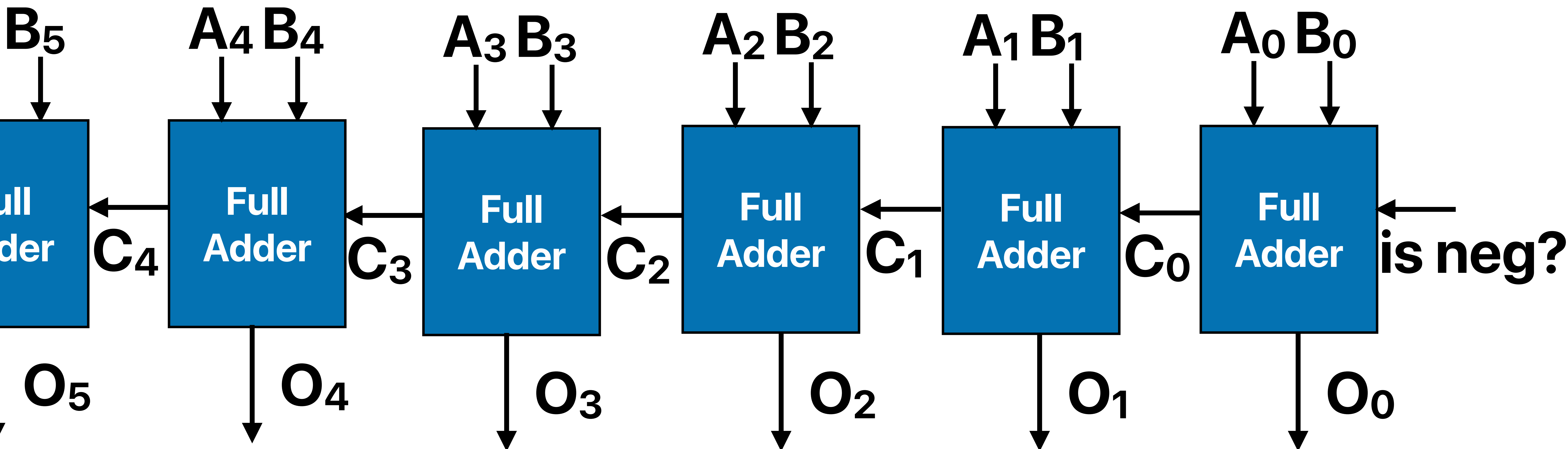
We've built this before!



This is what we want!



We can support more bits!



Carry-lookahead adder

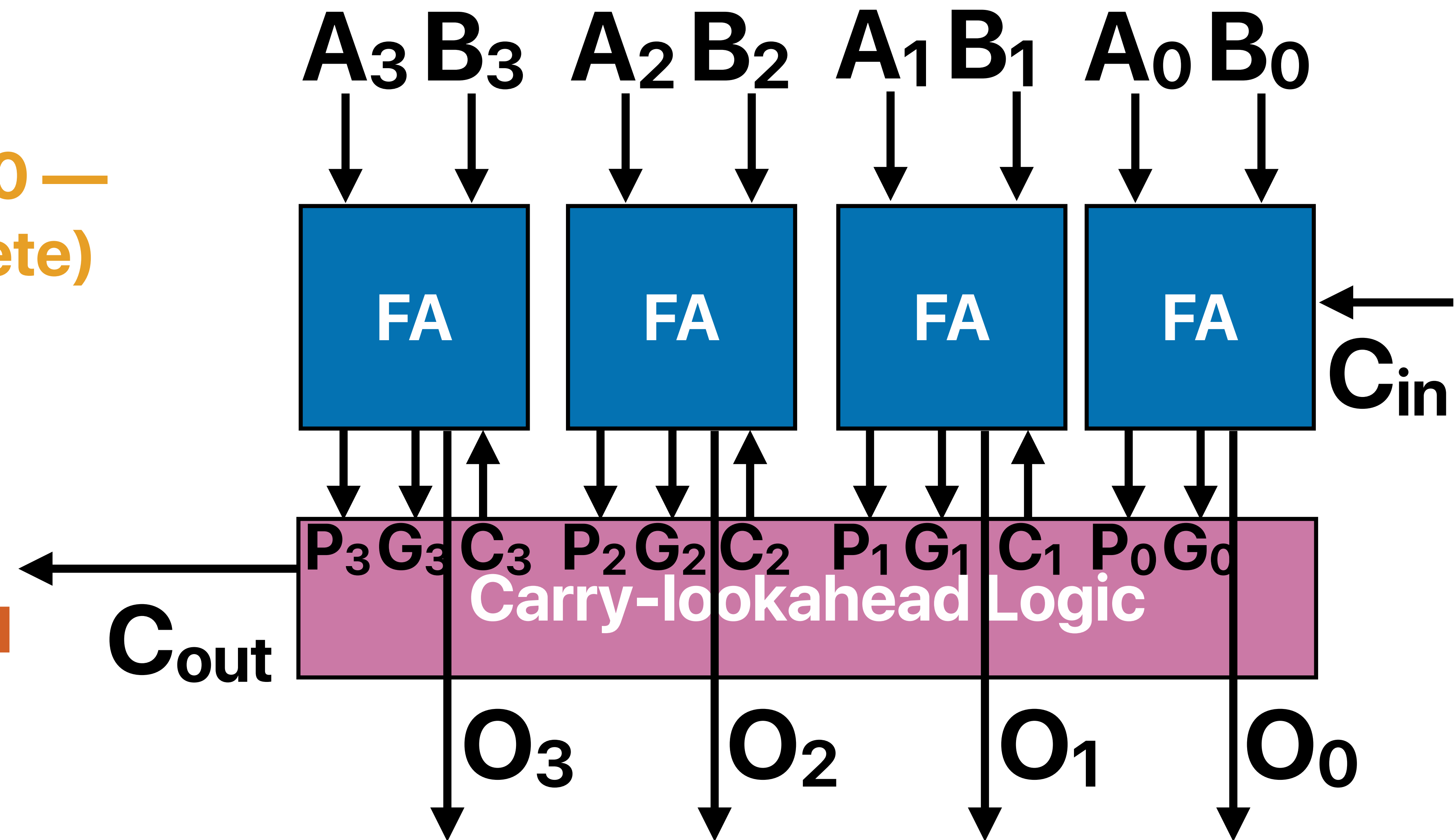
- Uses logic to quickly pre-compute the carry for each digit

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)

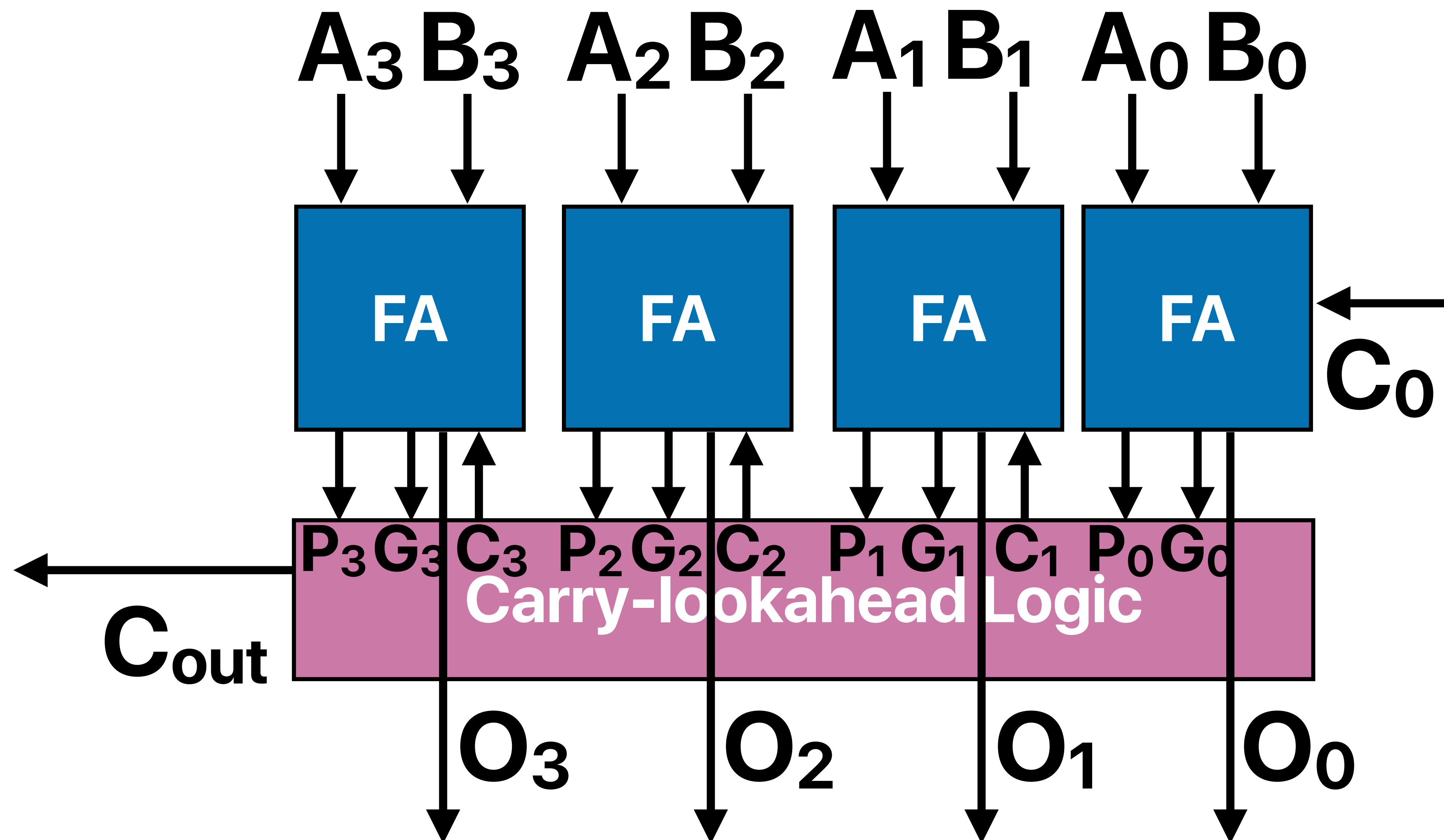
Needs to wait Cin (Propagate)

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).



$G_i = A_iB_i$

$P_i = A_i \text{ XOR } B_i$

$C_1 = G_0 + P_0 C_0$

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

$= G_1 + P_1G_0 + P_1P_0C_0$

$C_3 = G_2 + P_2 C_2$

$= G_2 + P_2 G_1 + P_2 P_1G_0 + P_2 P_1P_0C_0$

$C_4 = G_3 + P_3 C_3$

$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1G_0$

$+ P_3 P_2 P_1P_0C_0$

CLA v.s. Carry-ripple

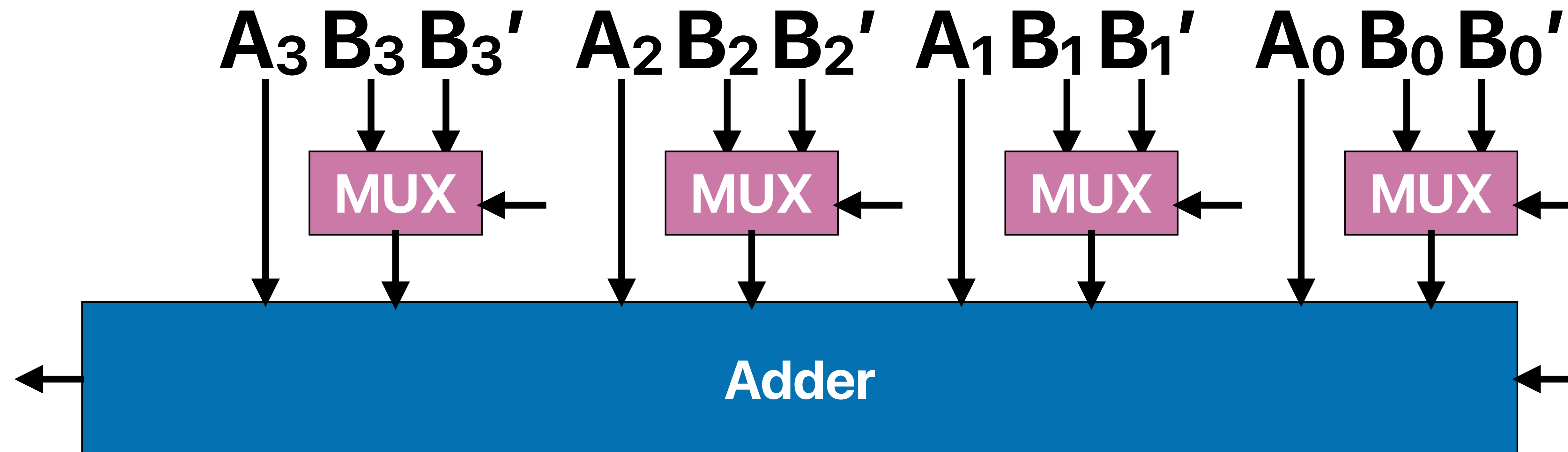
- Size:
  - 32-bit CLA with 4-bit CLAs — requires 8 of 4-bit CLA
    - Each requires 116 for the CLA 4*(4*6+8) for the A+B — 244 gates
    - 1952 transistors
  - 32-bit CRA
    - 1600 transistors **Win!**
- Delay
  - 32-bit CLA with 8 4-bit CLAs
    - 2 gates **Win!**
  - 32-bit CRA
    - 64 gates

Area-Delay Trade-off!

Multiplexer

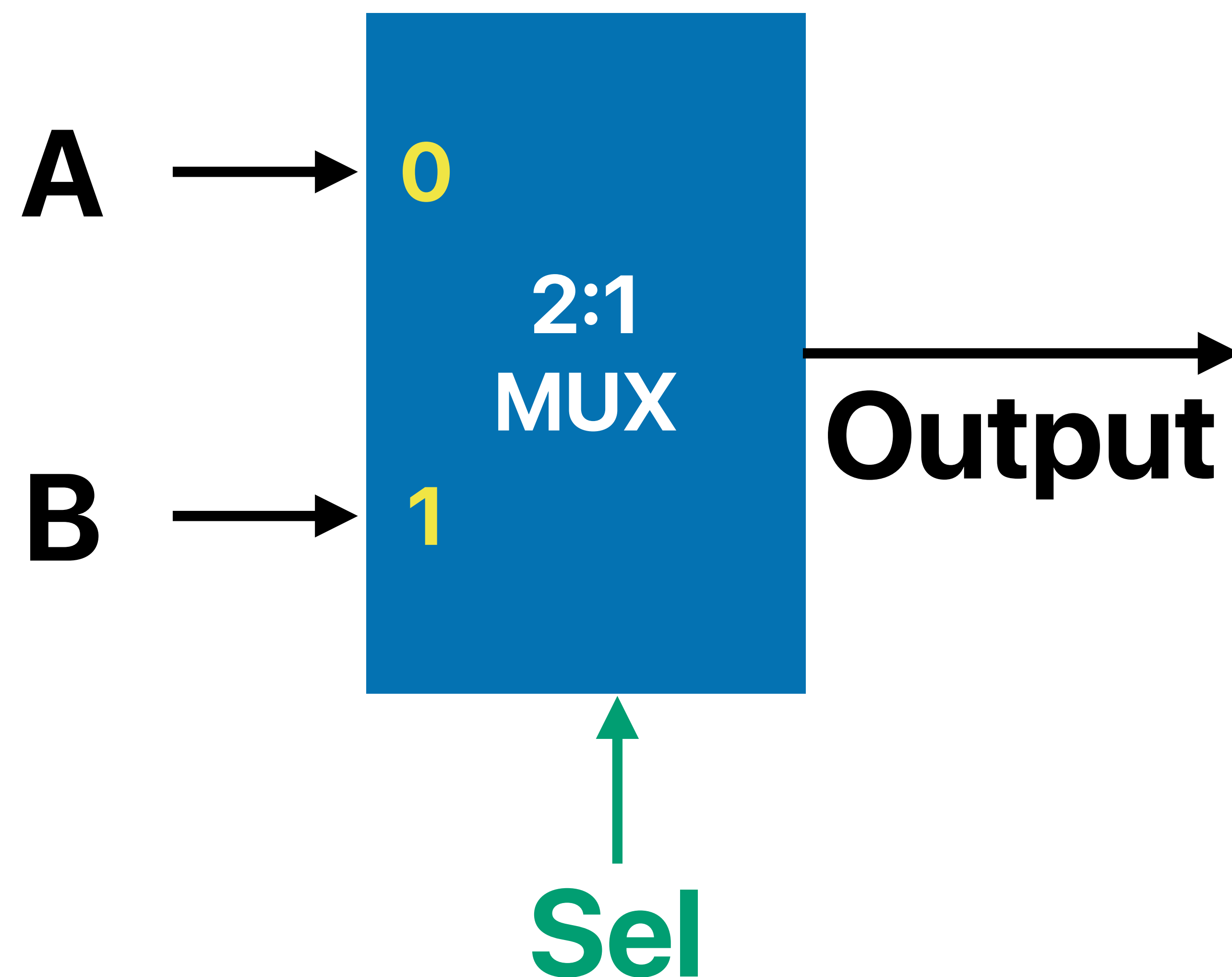
Multiplexer

- Problem — you have multiple possible inputs and you only want to use one of them
  - **N**-to-**M** MUX mean a MUX with **N** inputs, **M** outputs.
- Solution — you need a multiplexer (MUX) to control the output



Let's start with a 2-to-1 MUX

- The MUX has two input ports — numbered as 0 and 1
- To select from two inputs, you need a 1-bit control/select signal to indicate the desired input port



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

Use K-Map

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

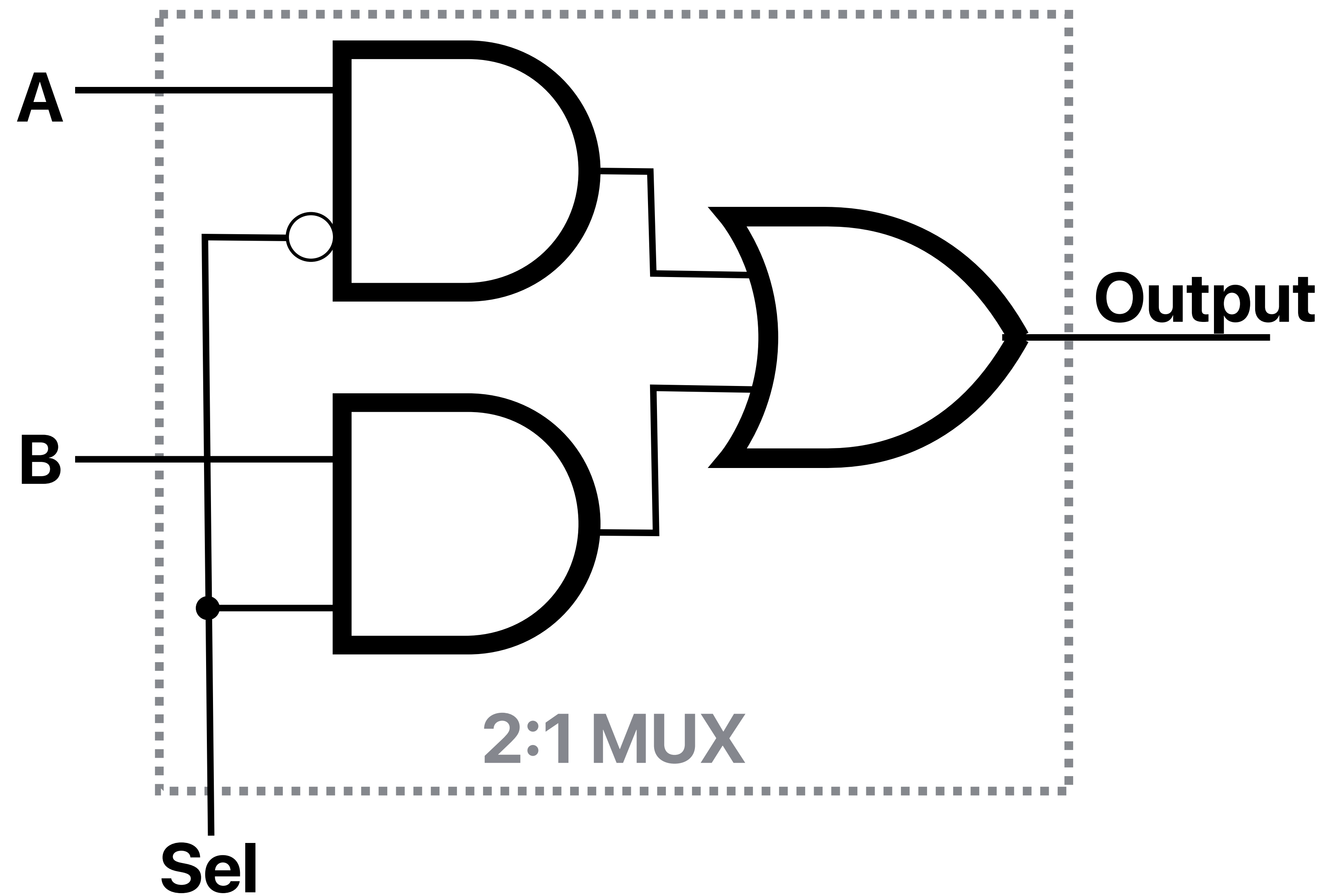
Sel′ means output A
Sel means output B

Output = ASel′ + BSel

(A, B) Sel	A′B′ 0,0	A′B 0,1	AB 1,1	AB′ 1,0
Sel′ 0	0	0	1	1
Sel 1	0	1	1	0

ASel′

BSel



4-to-1 MUX

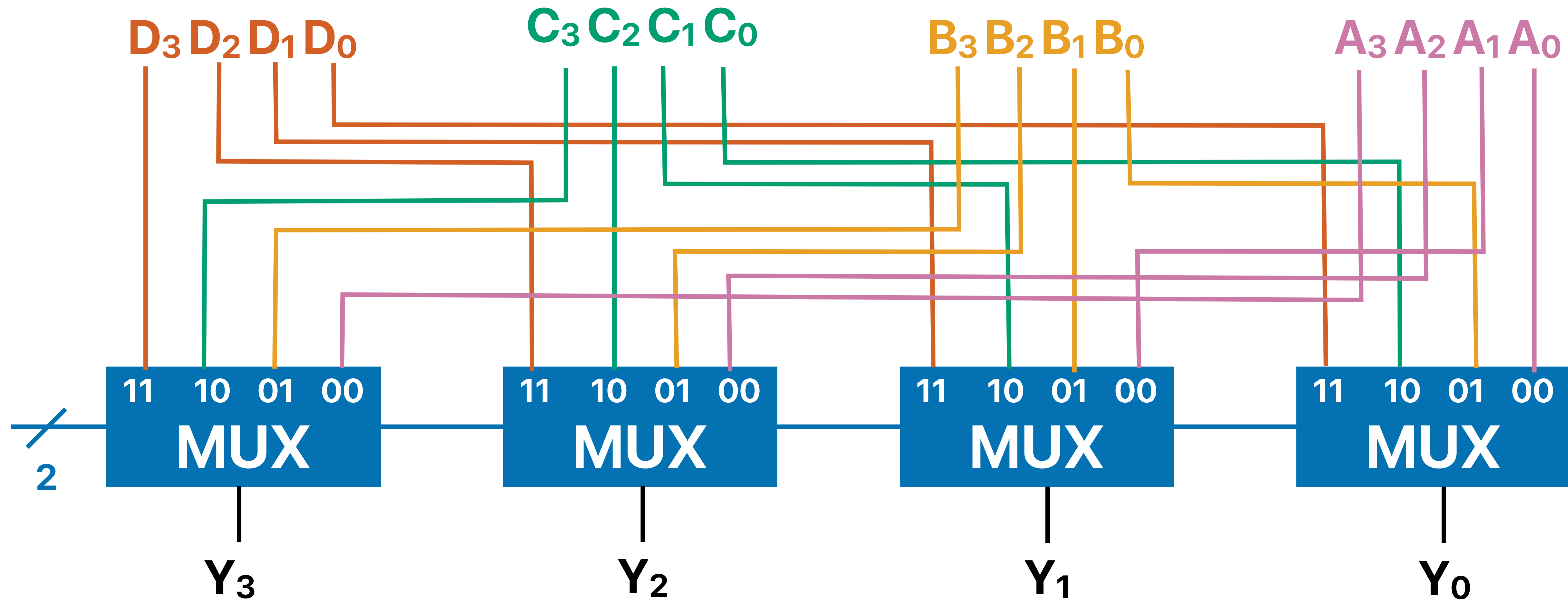
A
B
C
D
Output
4:1 MUX
S0 S1

S0==0 && S1==0 output A
S0==0 && S1==1 output B
S0==1 && S1==0 output C
S0==1 && S1==1 output D

$$Output = AS0'S1' + BS0'S1 + CS0S1' + DS0S1$$

N-bit MUX

- What if we need to output an N-bit (say 4-bit) number from the input set?



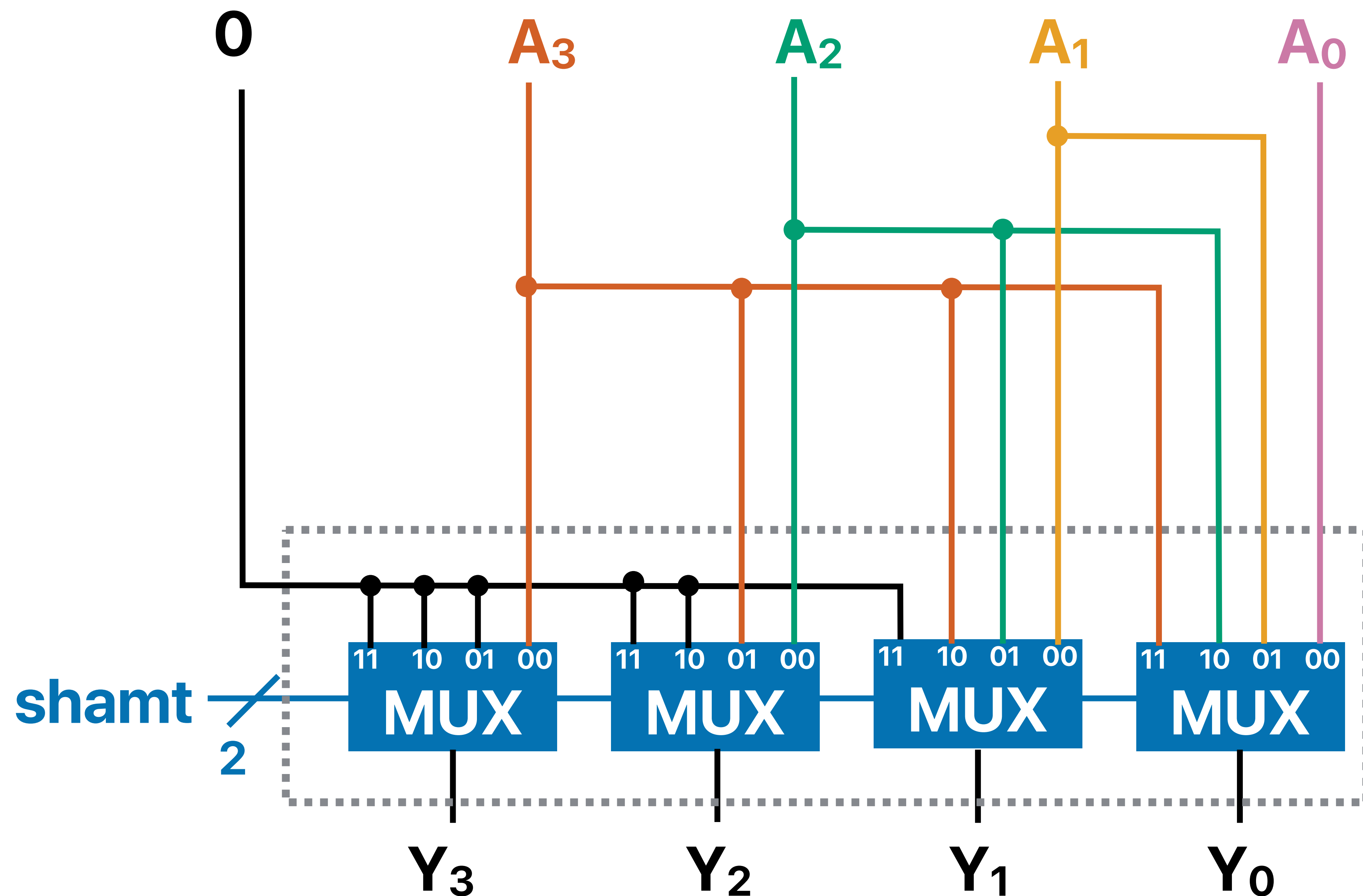
Shifters

Shifters

- Logical shifter: shifts value to left or right and fills empty spaces with 0's
  - 11001 >> 2 = 00110
  - 11001 << 2 = 00100
- Arithmetic shifter: same as logical shifter, but on right shift, fills empty spaces with the old most significant bit
  - Ex: 11001 >>> 2 = 11110
  - Ex: 11001 <<< 2 = 00100
- Rotator: rotates bits in a circle, such that bits shifted off one end are shifted into the other end
  - Ex: 11001 ROR 2 = 01110
  - Ex: 11001 ROL 2 = 00111

https://en.wikipedia.org/wiki/Circular_shift

Shift "Right"



Example:
if S = 11
then
Y3 = 0
Y2 = 0
Y1 = 0
Y0 = A3

Example:
if S = 10
then
Y3 = 0
Y2 = 0
Y1 = A3
Y0 = A2

Example:
if S = 01
then
Y3 = 0
Y2 = A3
Y1 = A2
Y0 = A1

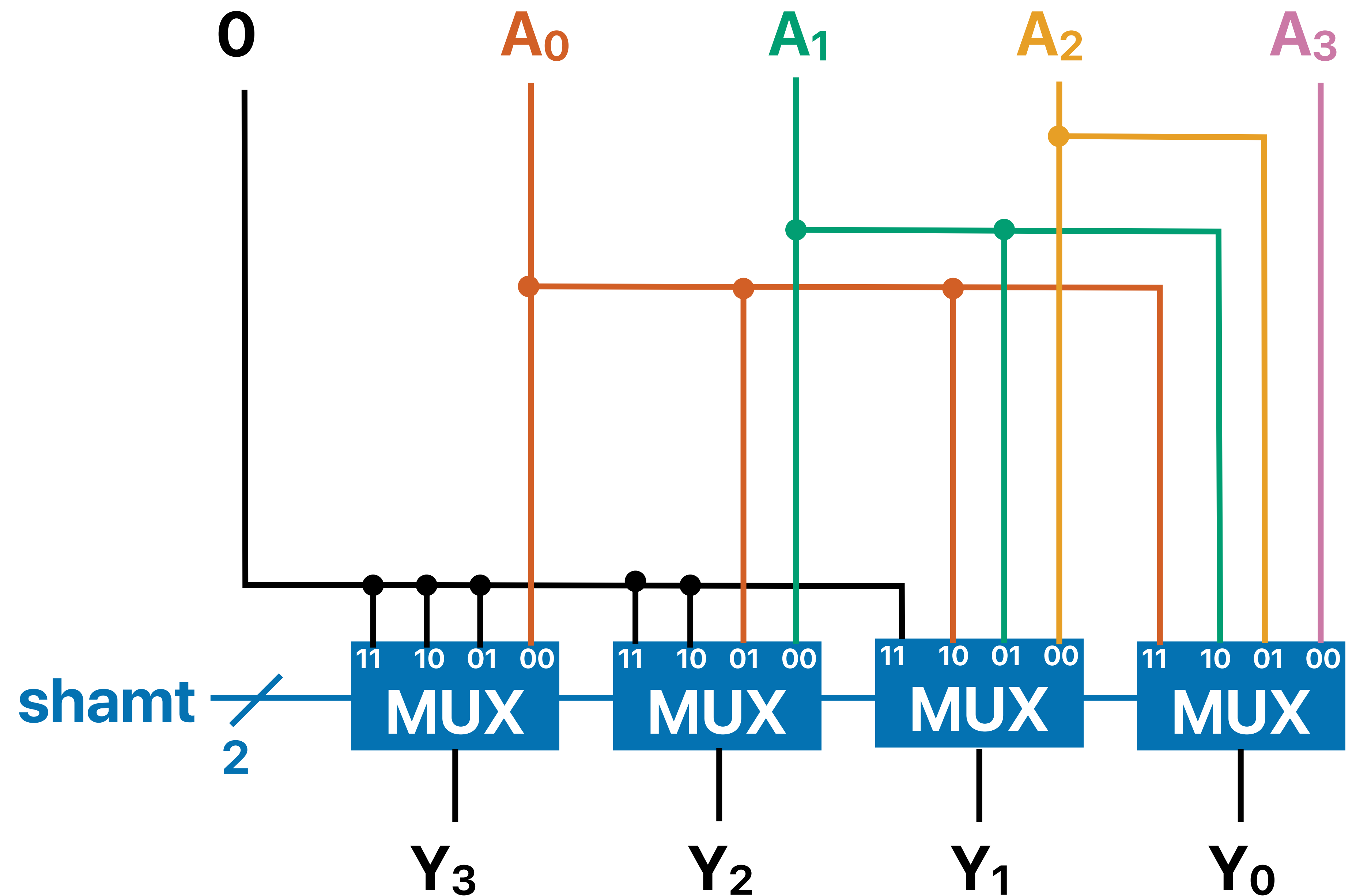
The "chain" of multiplexers determines how many bits to shift

Shift "Left"

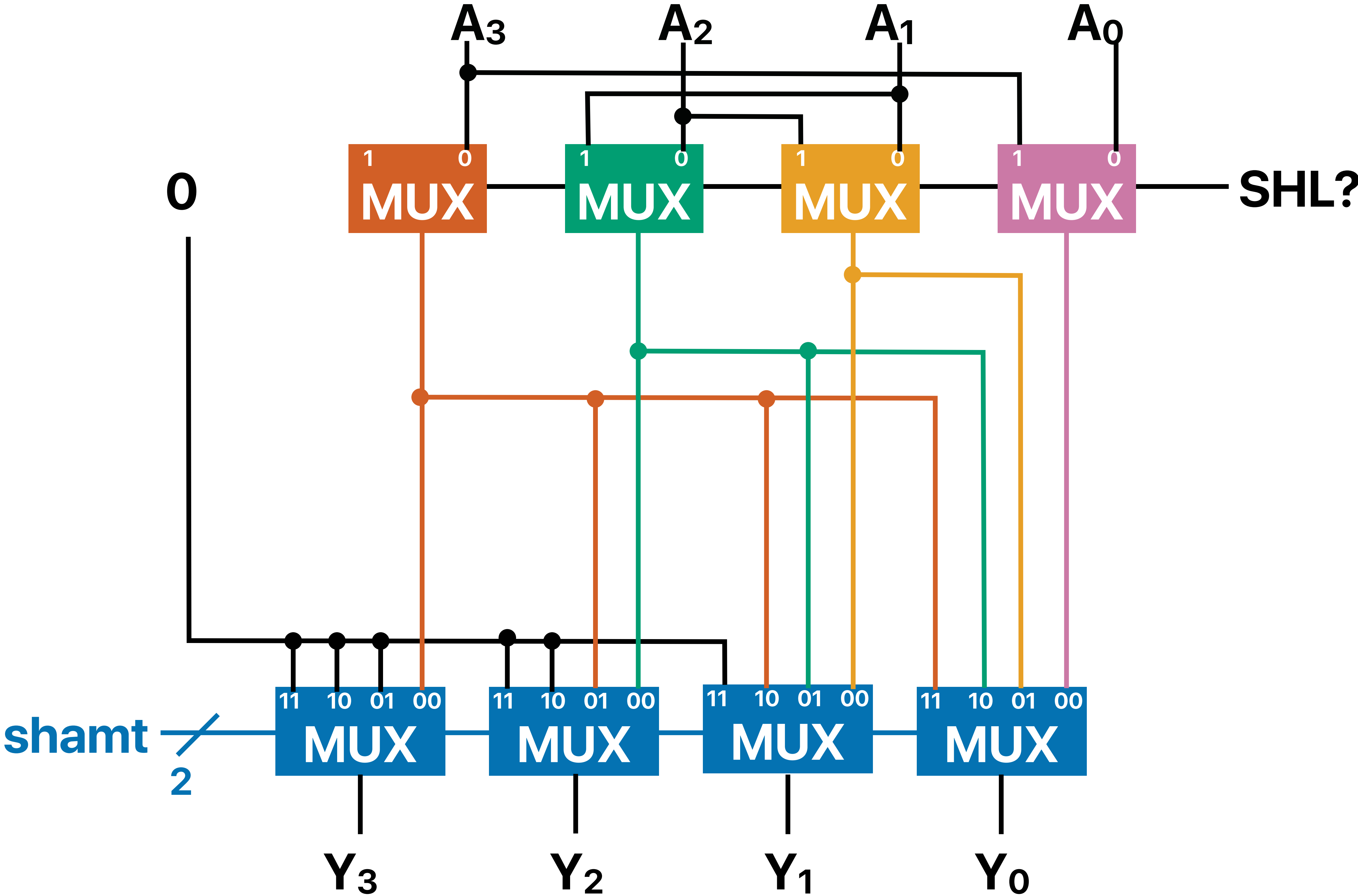
Example:
if S = 01
then
Y3 = A2
Y2 = A1
Y1 = A0
Y0 = 0

Example:
if S = 10
then
Y3 = A1
Y2 = A0
Y1 = 0
Y0 = 0

Example:
if S = 11
then
Y3 = A0
Y2 = 0
Y1 = 0
Y0 = 0



Generic Shifter



Multiplier

Binary multiplication

- Thinking about how you do this by hand in decimal!

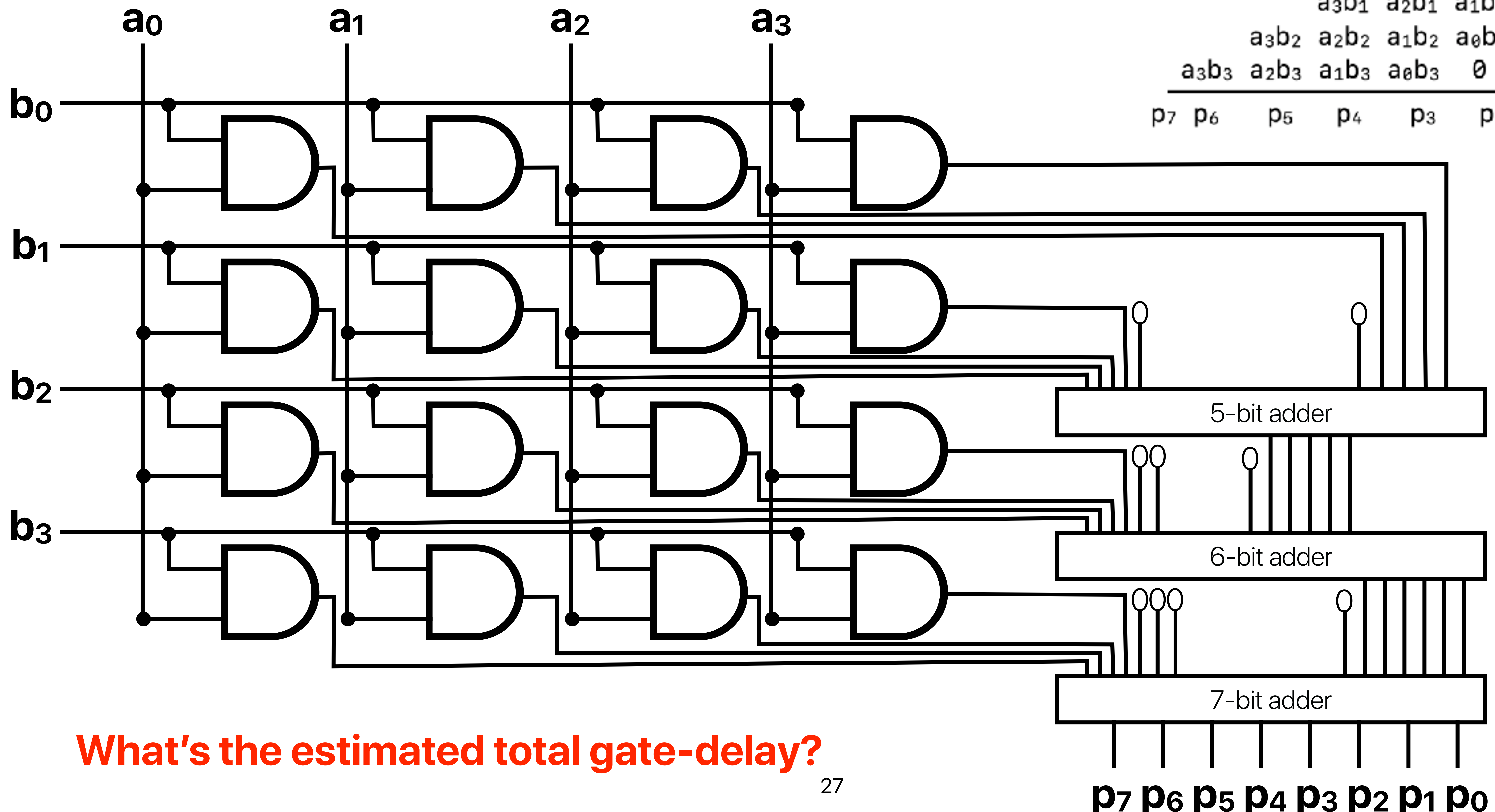
```
      1 2 3 4
    × 5 6 7 8
    ---------
      9 8 7 2
    8 6 3 8
  7 4 0 4
6 1 7 0
-------------
7 0 0 6 6 5 2
```

```
      0 1 1 1
    × 1 1 0 0
    ---------
      0 0 0 0
    0 0 0 0
  0 1 1 1
0 1 1 1
-------------
1 0 1 0 1 0 0
```

				a_3	a_2	a_1	a_0
			×	b_3	b_2	b_1	b_0
				a_3b_0	a_2b_0	a_1b_0	a_0b_0
			a_3b_1	a_2b_1	a_1b_1	a_0b_1	0
		a_3b_2	a_2b_2	a_1b_2	a_0b_2	0	0
	a_3b_3	a_2b_3	a_1b_3	a_0b_3	0	0	0
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

Array style

				a3	a2	a1	a0
			×	b3	b2	b1	b0
				a3b0	a2b0	a1b0	a0b0
			a3b1	a2b1	a1b1	a0b1	0
		a3b2	a2b2	a1b2	a0b2	0	0
	a3b3	a2b3	a1b3	a0b3	0	0	0
p7	p6	p5	p4	p3	p2	p1	p0



What's the estimated total gate-delay?

Divider

Division of positive binary numbers

- Repeated subtraction
  - Set quotient to 0
  - Repeat while (dividend >= divisor)
    - Subtract divisor from dividend
    - Add 1 to quotient
  - When dividend < divisor:
    - Reminder = dividend
    - Quotient is correct

Put everything all together!
ALU — arithmetic logic unit