Numbers in Bits

CS143: lecture 3

Konstantinos Ameranis, June 23

Numbers

- We know a bit represents an answer to a yes-no question
- What yes-no questions do we ask to get a number?
- Think a number from 0-15, what questions should I ask to locate that number?

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

- Is the number >= 8?

- No

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

- Is the number >= 8?
- Is the number >= 4?

- No
- Yes

- Is the number >= 8?
- Is the number >= 4?
- Is the number >= 6?

- No
- Yes
- Yes

- Is the number >= 8?
- Is the number >= 4?
- Is the number >= 6?
- Is the number 7?

- No
- Yes
- Yes
- Yes

- Is the number >= 8?
- Is the number >= 4?
- Is the number >= 6?
- Is the number 7?

- No
- Yes
- Yes
- Yes

7

- [No, Yes, Yes, Yes]

7

- [No, Yes, Yes, Yes]

?

- [Yes, No, Yes, No]

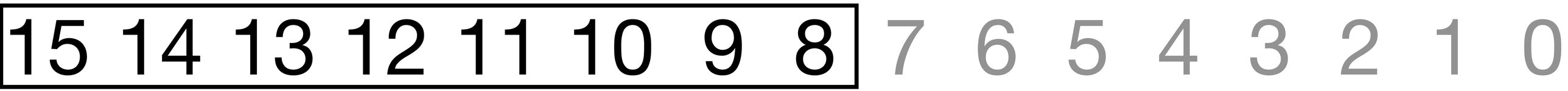
- Is the number >= 8?
- Is the number >= 4?
- Is the number >= 6?
- Is the number 7?

Is the number in the first (higher) half of the remaining numbers?

[Yes, No, Yes, No]

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

[Yes, No, Yes, No]



[Yes, No, Yes, No]

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

[Yes, No, Yes, No]

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

[Yes, No, Yes, No]

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Binary

Number	Encoding
0	
1	
2	
3	
4	
5	
6	
7	

Number	Encoding
8	
9	
10	
11	
12	
13	
14	
15	

Binary

Number	Encoding
0	[no, no, no, no]
1	[no, no, no, yes]
2	[no, no, yes, no]
3	[no, no, yes, yes]
4	[no, yes, no, no]
5	[no, yes, no, yes]
6	[no, yes, yes, no]
7	[no, yes, yes, yes]

Number	Encoding
8	[yes, no, no, no]
9	[yes, no, no, yes]
10	[yes, no, yes, no]
11	[yes, no, yes, yes]
12	[yes, yes, no, no]
13	[yes, yes, no, yes]
14	[yes, yes, yes, no]
15	[yes, yes, yes, yes]

Binary

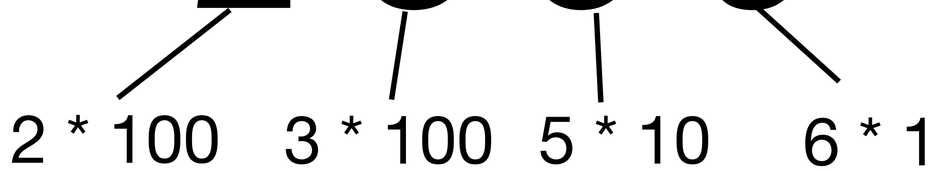
Is there any mathematical significance in this encoding?

Number	Encoding
0	[0, 0, 0, 0]
1	[0, 0, 0, 1]
2	[0, 0, 1, 0]
3	[0, 0, 1, 1]
4	[0, 1, 0, 0]
5	[0, 1, 0, 1]
6	[0, 1, 1, 0]
7	[0, 1, 1, 1]

Number	Encoding
8	[1, 0, 0, 0]
9	[1, 0, 0, 1]
10	[1, 0, 1, 0]
11	[1, 0, 1, 1]
12	[1, 1, 0, 0]
13	[1, 1, 0, 1]
14	[1, 1, 1, 0]
15	[1, 1, 1, 1]

Decimal: 2 3 5 6

Decimal: 2356



- 10 choices per digit
- every time we add one to the tens, we use up all 10 choices

Decimal: 2356

$2 * 10^3 \quad 3 * 10^2 \quad 5 * 10^1 \quad 6 * 10^0$

- 2 choices per digit
- every time we add one to the tens, we use up all 2 choices

Binary: 1 0 1 0

$1 * 2^3$ $0 * 2^2$ $1 * 2^1$ $0 * 2^0$

- 2 choices per digit
- every time we add one to the tens, we use up all 2 choices

Binary: 1 0 1 0

$1 * 2^3$ $0 * 2^2$ $1 * 2^1$ $0 * 2^0$

8 0 2 0 = 10

[Yes, No, Yes, No]

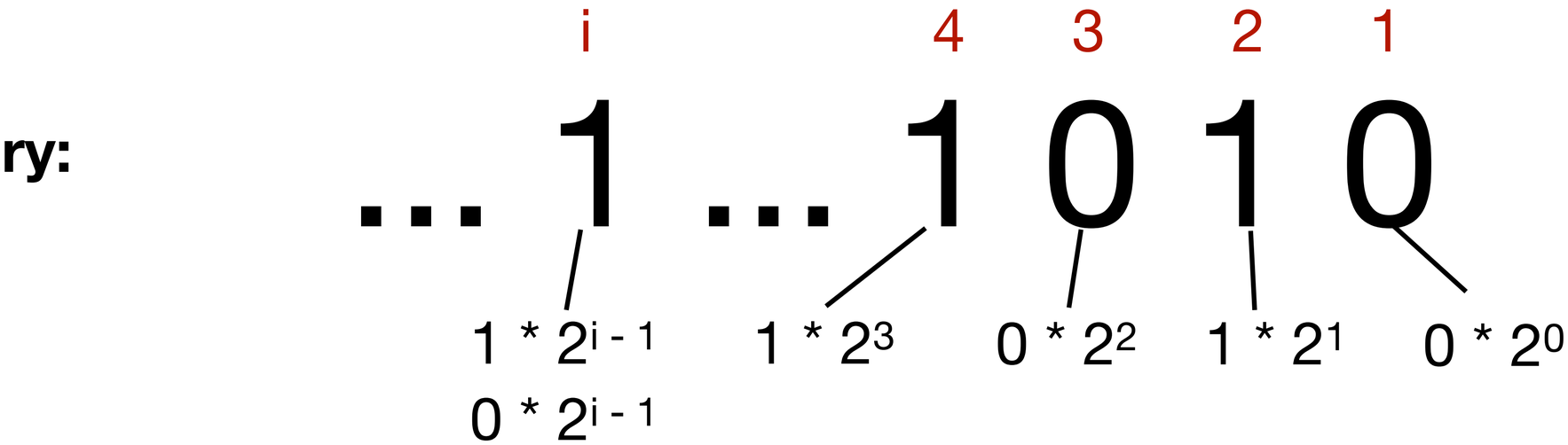
15 14 13 12 11 **10** 9 8 7 6 5 4 3 2 1 0

Binary: 1 0 1 0

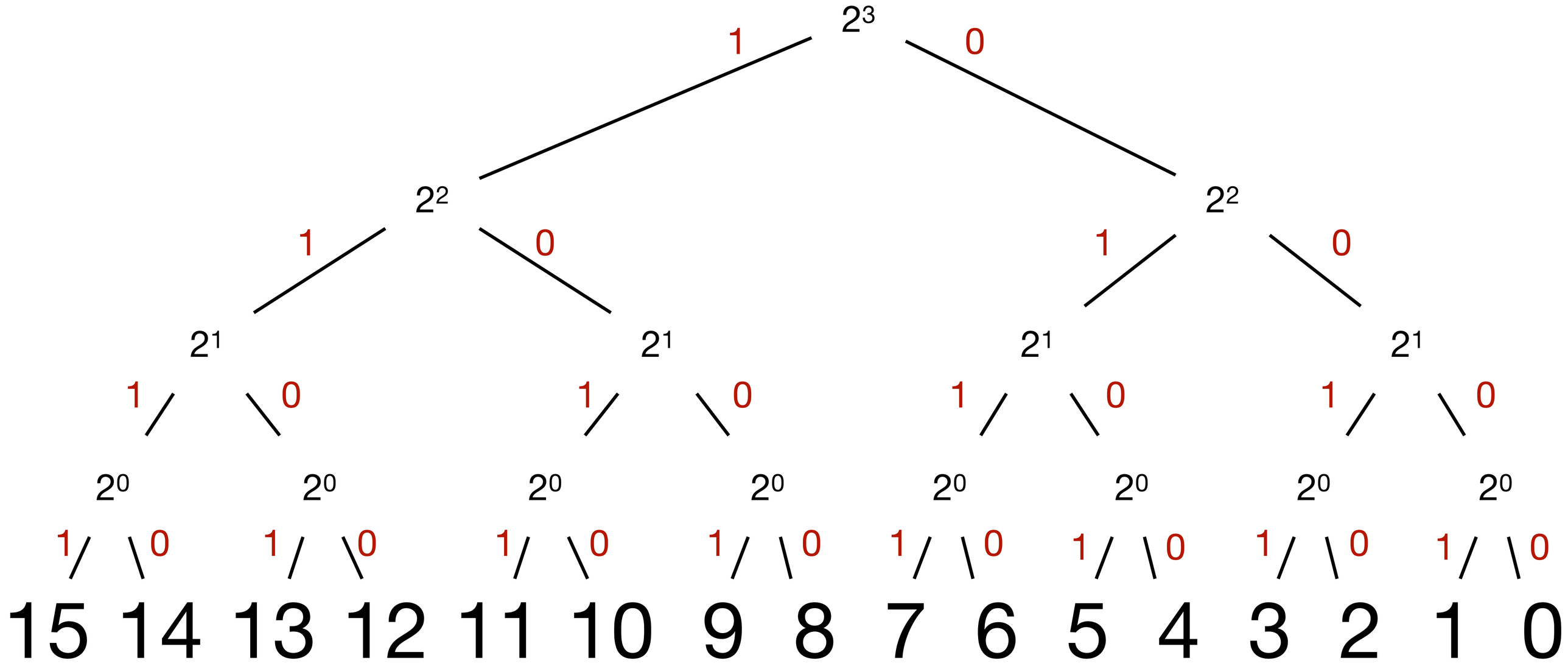
$1 * 2^3$ $0 * 2^2$ $1 * 2^1$ $0 * 2^0$

8 0 2 0 = 10

Binary:

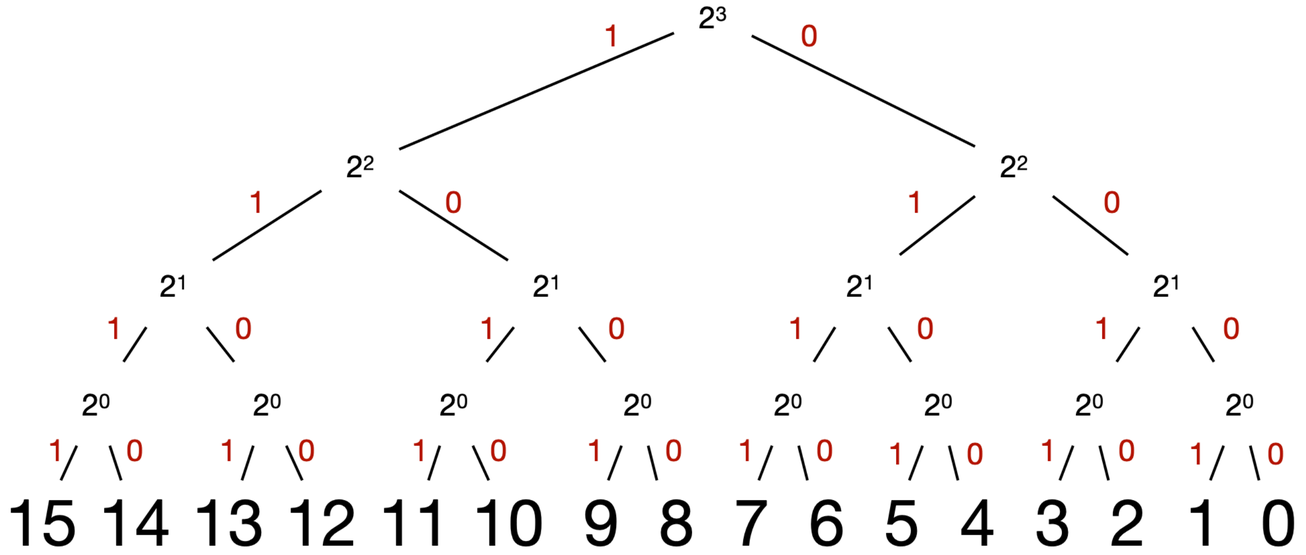


- If the ith bit is 1, the number "has" 2^{i-1}; if the ith bit is 0, the number does not have 2^{i-1}.
- $$Decimal_w(\vec{x}) = \sum_{i=1}^{w} x_{i-1} 2^{i-1}$$


2^3
1
0
2^2
2^2
1
0
1
0
2^1
2^1
2^1
2^1
1
0
1
0
1
0
1
0
2^0
2^0
2^0
2^0
2^0
2^0
2^0
2^0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
0
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

$$Decimal_w(\vec{x}) = \sum_{i=1}^{w} x_{i-1} 2^{i-1}$$

Base-2 Number Interpretation

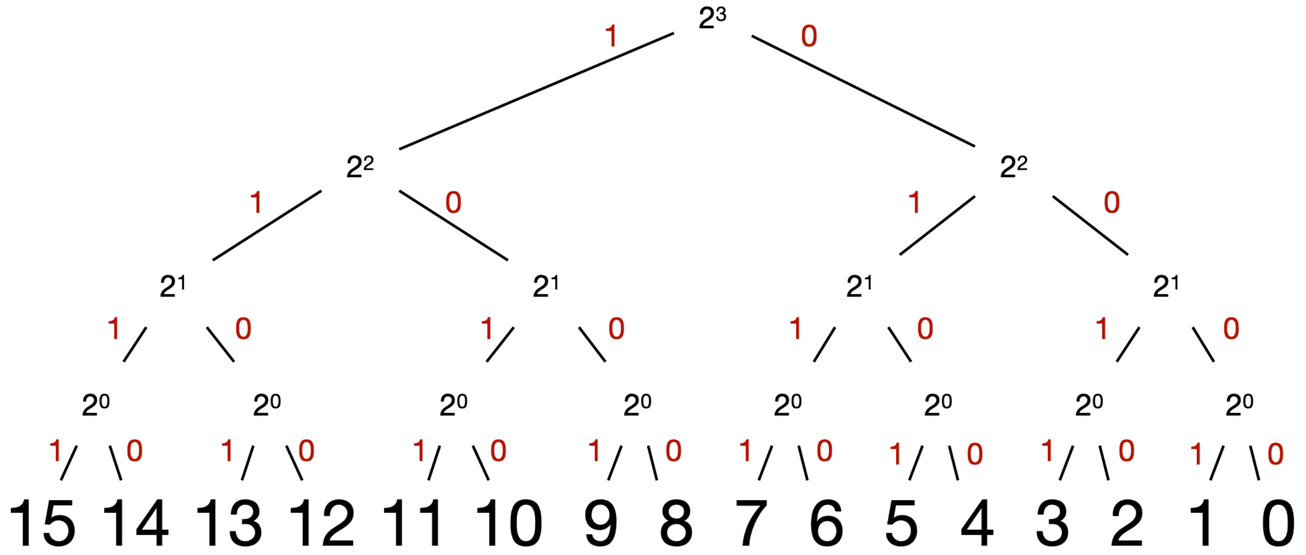


Binary Search Interpretation

- Check: What is the greatest n-bit number? (As bit-pattern?)
- All 1s:
  - Binary Search Interpretation: always going left.
  - Number interpretation: $\forall i$, maximize x_{i-1} --> $x_{i-1} = 1$

$$Decimal_w(\vec{x}) = \sum_{i=1}^{w} x_{i-1} 2^{i-1}$$

Base-2 Number Interpretation

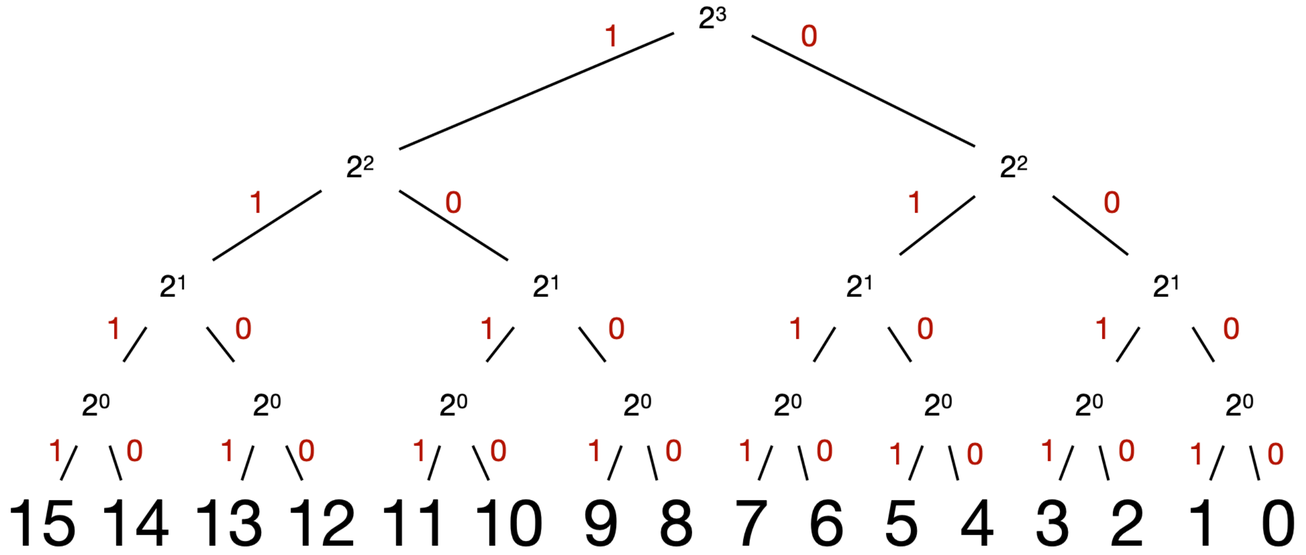


Binary Search Interpretation

- Check: What does a all-1 n-bit number represent in decimal?
- $2^n - 1$
  - Binary Search Interpretation: 1,0,0,0 is 1 more than 0,1,1,1 --> the rightmost number on the left is 1 more than the leftmost number on the right.

$$Decimal_w(\vec{x}) = \sum_{i=1}^{w} x_{i-1} 2^{i-1}$$

Base-2 Number Interpretation



Binary Search Interpretation

- Number Interpretation:
  - $S = 2^{n-1} + 2^{n-2} + \ldots + 2^1 + 2^0$
  - $2S = 2^n + 2^{n-1} + \ldots + 2^2 + 2^1$
  - $S = 2S - S = 2^n + (2^{n-1} - 2^{n-1}) + \ldots + (2^1 - 2^1) - 2^0 = 2^n - 1$

Bits

Know your powers of 2

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

n	2^n
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768

n	2^n
16	65,536
17	131,072
18	262,144
19	524,288
20	1,048,576
21	2,097,152
22	4,194,304
23	8,388,608

n	2^n
24	16,777,216
25	33,554,432
26	67,108,864
27	134,217,728
28	268,435,456
29	536,870,912
30	1,073,741,824
31	2,147,483,648

n	2^n
32	4,294,967,296
64	18,446,744,073,709,600,000

Bits

Know your powers of 2

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

n	2^n
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768

n	2^n
16	65,536
17	131,072
18	262,144
19	524,288
20	1,048,576
21	2,097,152
22	4,194,304
23	8,388,608

n	2^n
24	16,777,216
25	33,554,432
26	67,108,864
27	134,217,728
28	268,435,456
29	536,870,912
30	1,073,741,824
31	2,147,483,648

n	2^n
32	4,294,967,296
64	18,446,744,073,709,600,000

`char`

`short`

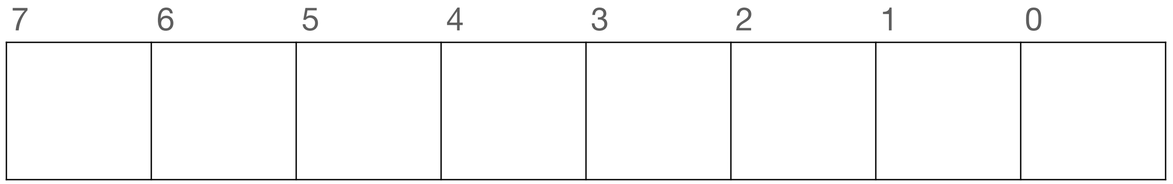
`int`, `float`

`long`, `double`, pointers

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 1 9 9



Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 199

71

128 +

7	6	5	4	3	2	1	0
1							

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 199

7

128 + 64 +

7	6	5	4	3	2	1	0
1	1						

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 199

7

128 + 64 +

7	6	5	4	3	2	1	0
1	**1**	**0**					

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 199

7

128 + 64 +

7	6	5	4	3	2	1	0
1	**1**	**0**	**0**				

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 199

7

128 + 64 +

7	6	5	4	3	2	1	0
1	**1**	**0**	**0**	**0**			

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 199

$$128 + 64 + 4 + 3$$

7	6	5	4	3	2	1	0
1	**1**	**0**	**0**	**0**	**1**		

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 199

128 + 64 + 4 + 2 + 1

7	6	5	4	3	2	1	0
1	**1**	**0**	**0**	**0**	**1**	**1**	

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 199

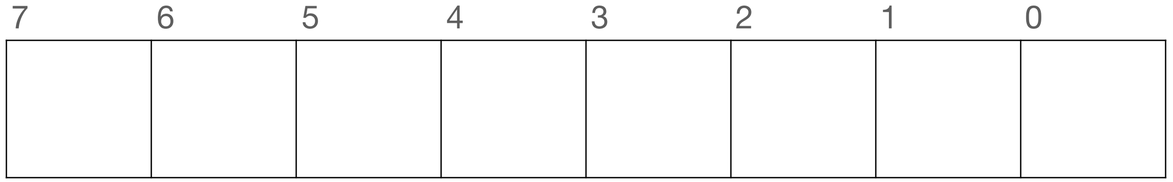
128 + 64 + 4 + 2 + 1

7	6	5	4	3	2	1	0
1	1	0	0	0	1	1	1

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 200



Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 200

72

128 +

7	6	5	4	3	2	1	0
1							

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 200

128 + 64 + 8

7	6	5	4	3	2	1	0
1	**1**						

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 200

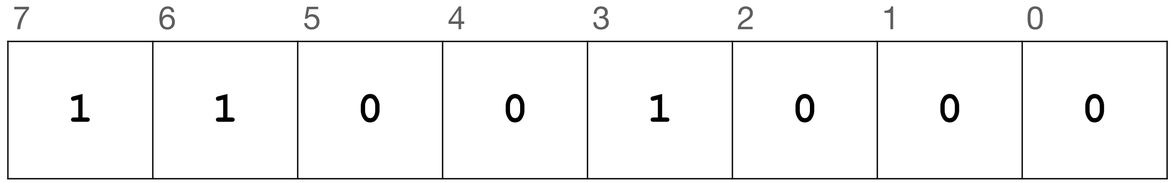
128 + 64 + 8 + 0

7	6	5	4	3	2	1	0
1	**1**	**0**	**0**	**1**			

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

200



Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

199

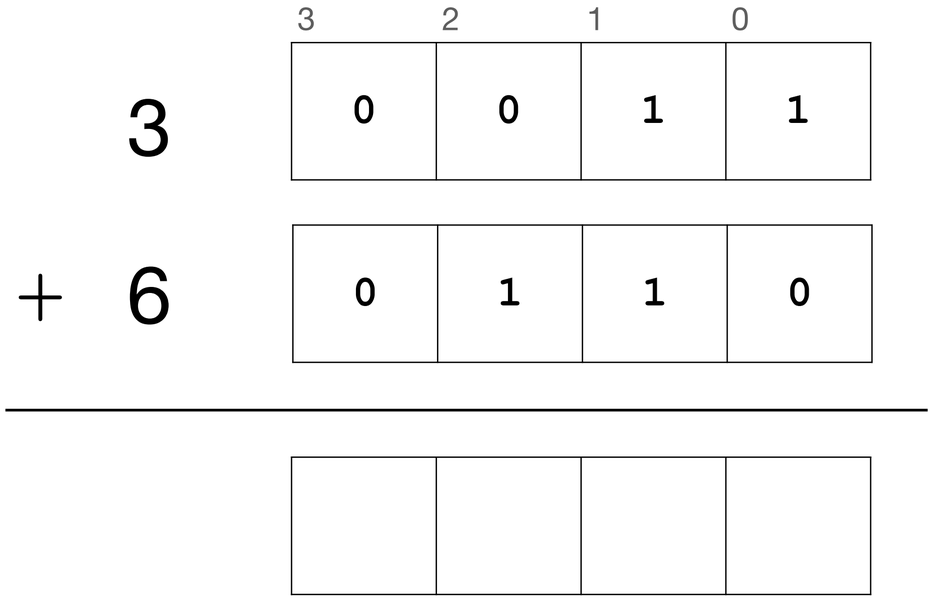
7	6	5	4	3	2	1	0
1	1	0	0	0	1	1	1

\+ 1

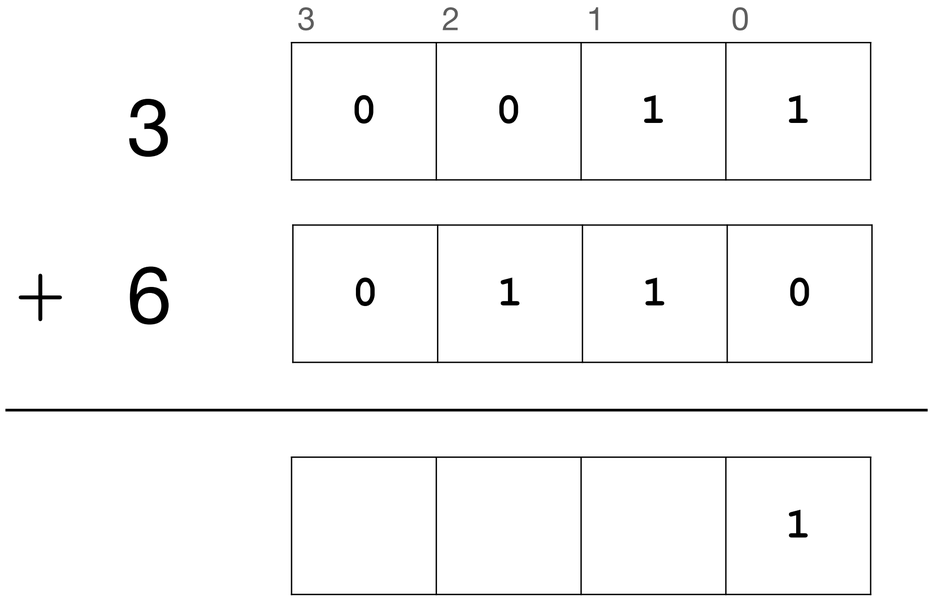
200

7	6	5	4	3	2	1	0
1	1	0	0	1	0	0	0

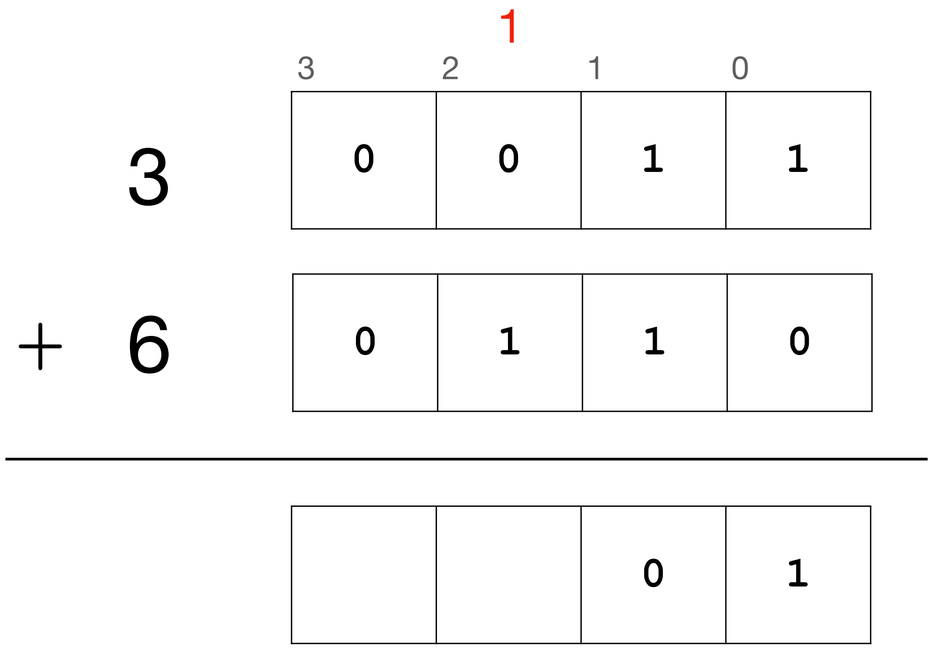
n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128



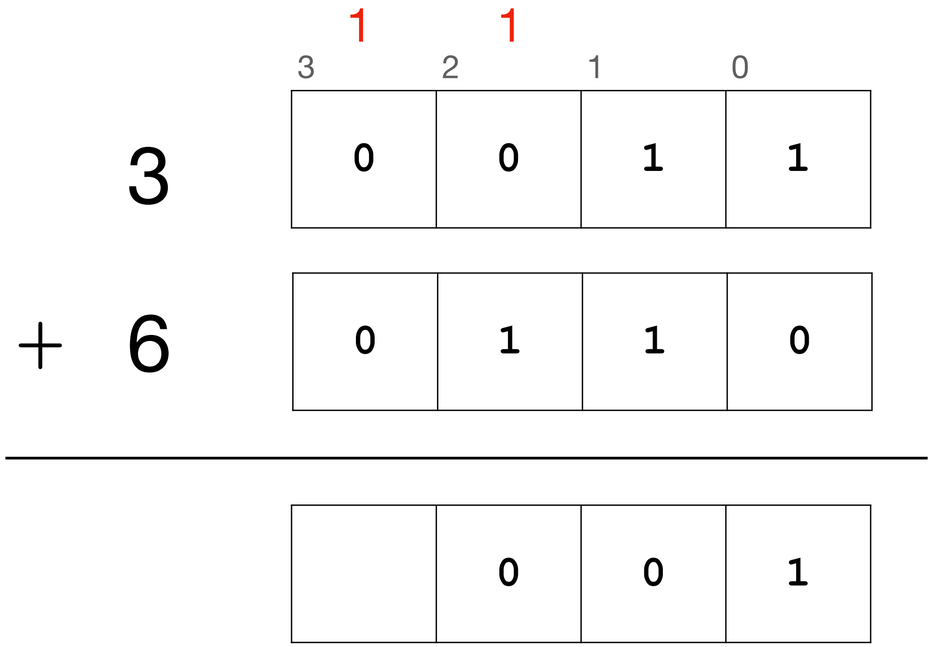
n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128



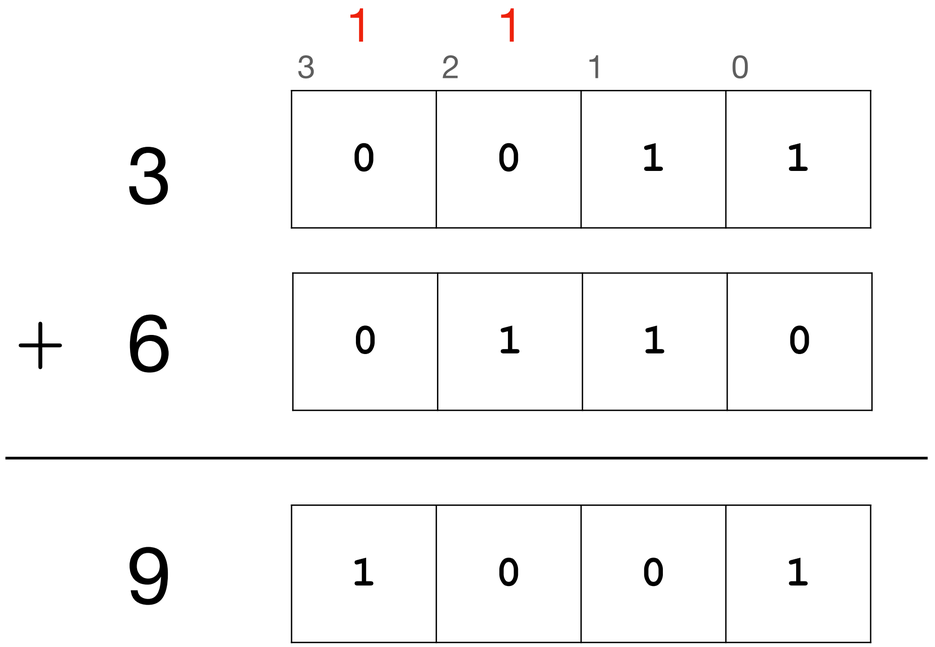
n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128



n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

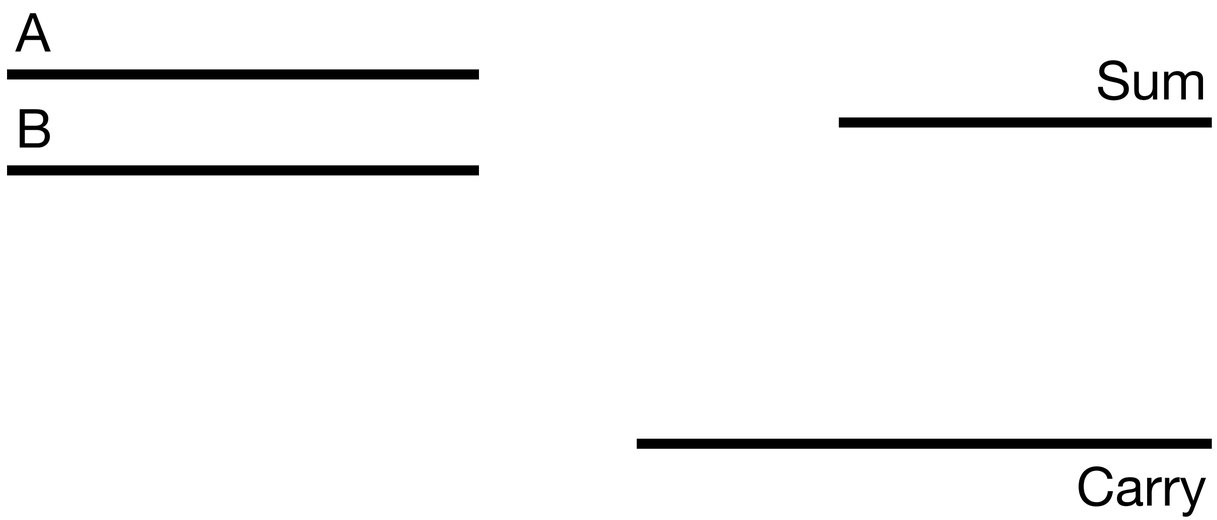


n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

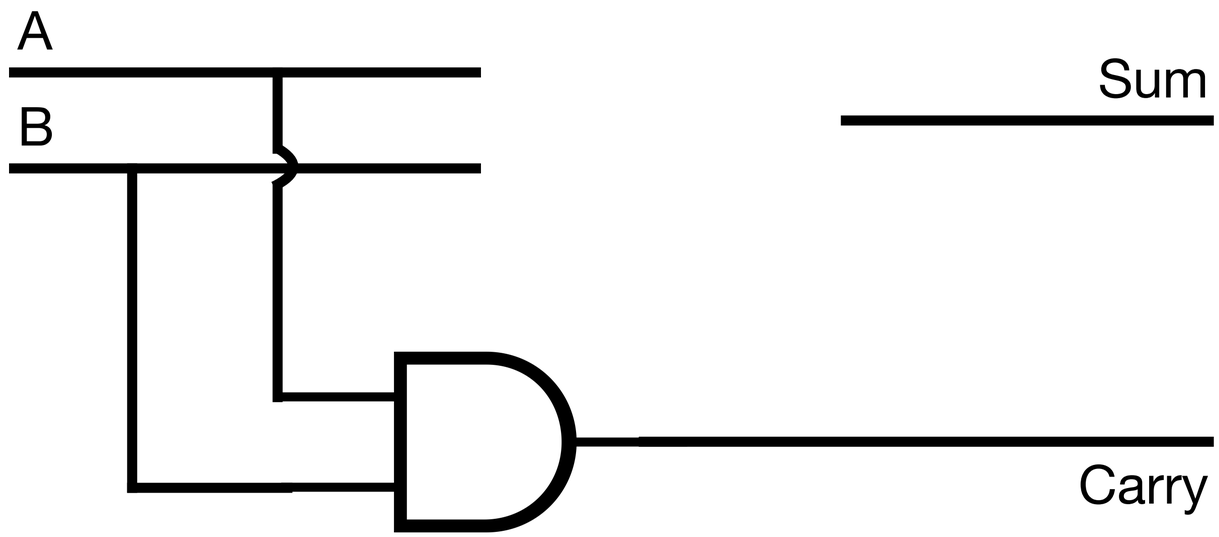


A	B	Carry	Sum
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

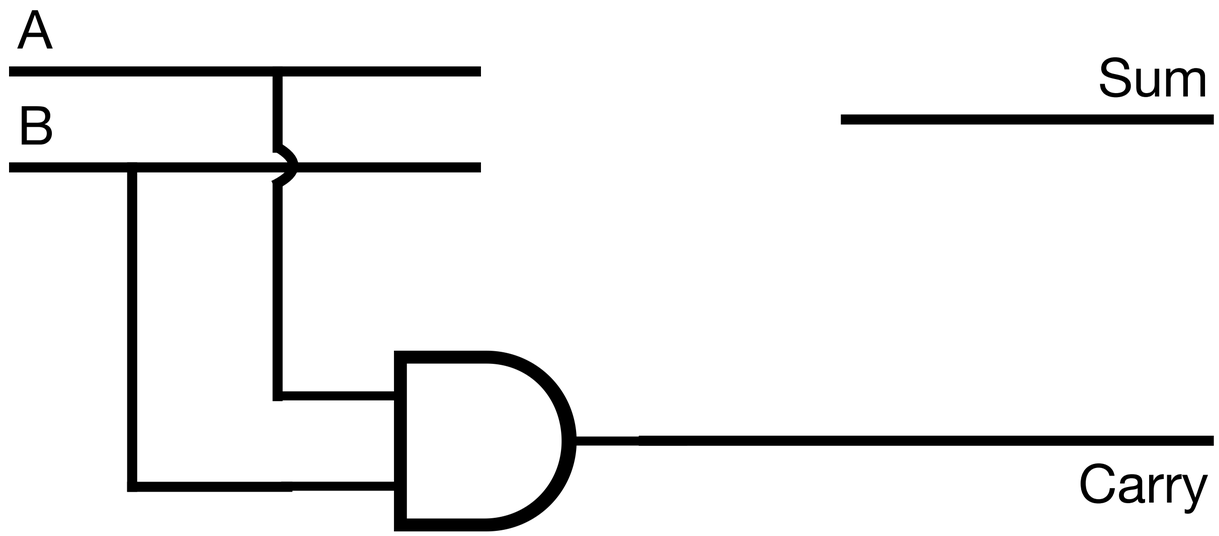
A	B	Carry	Sum
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0



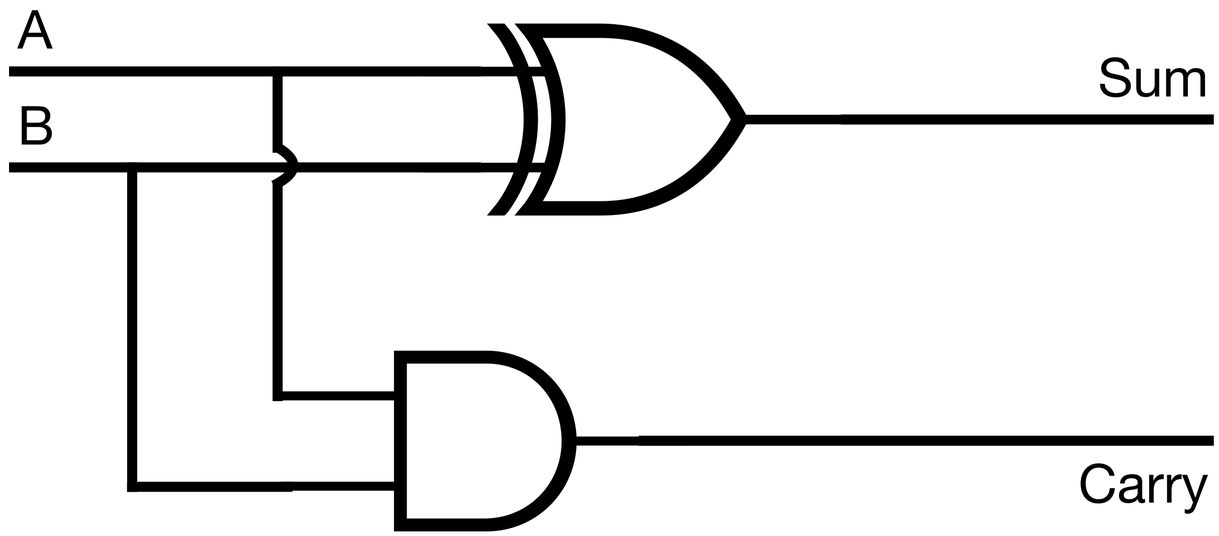
A	B	Carry	Sum
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0



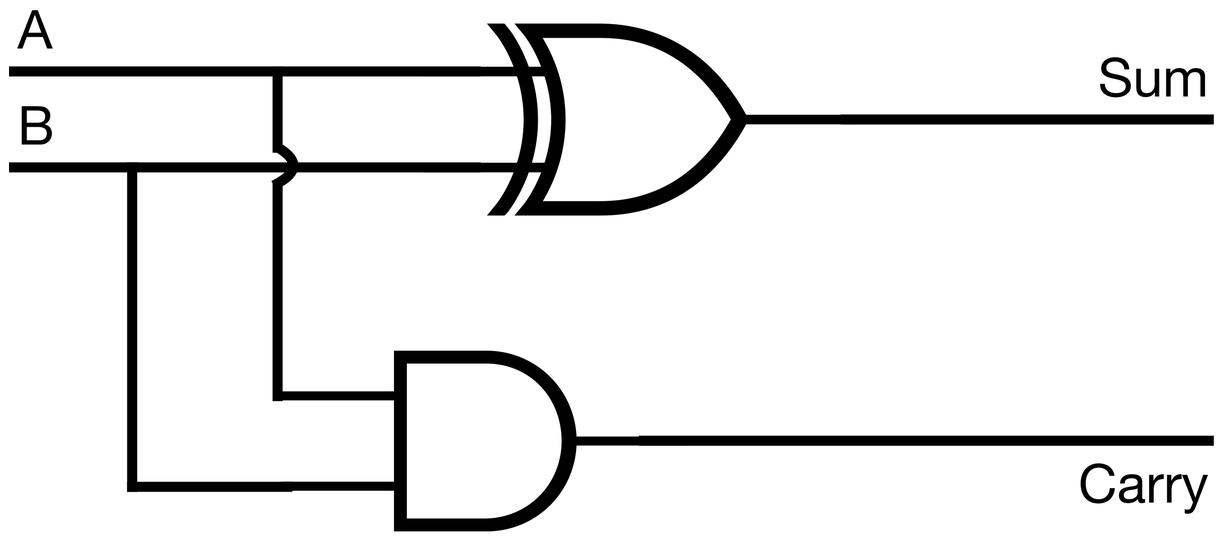
A	B	Carry	Sum
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0



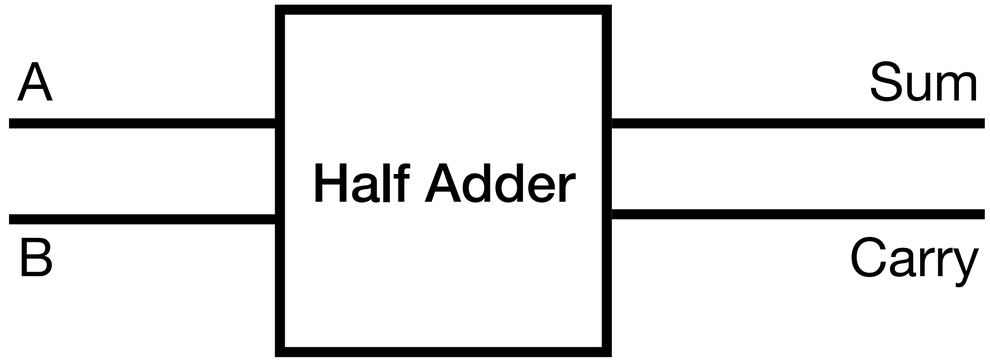
A	B	Carry	Sum
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

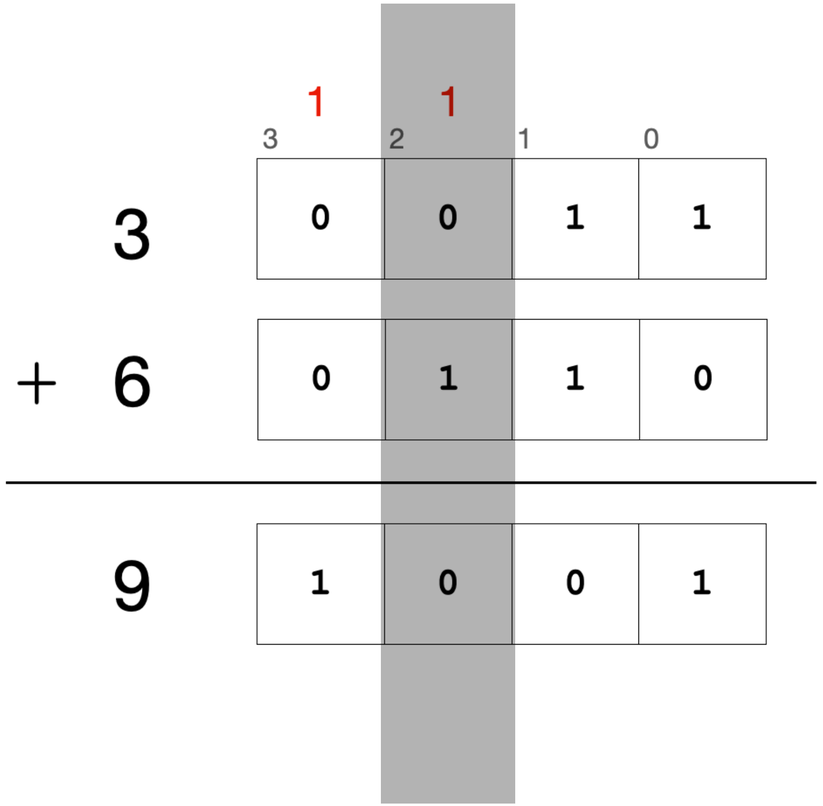


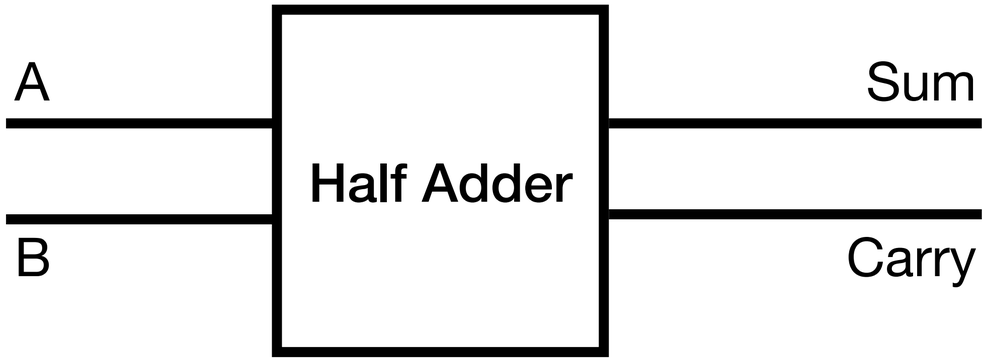
A	B	Carry	Sum
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

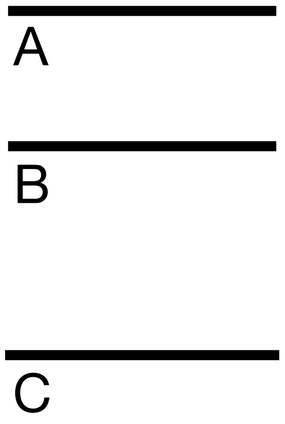


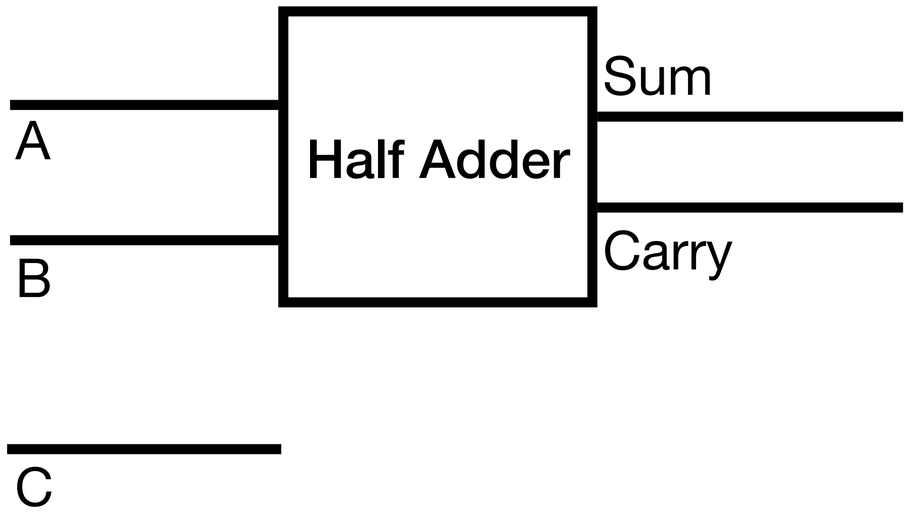
A	B	Carry	Sum
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

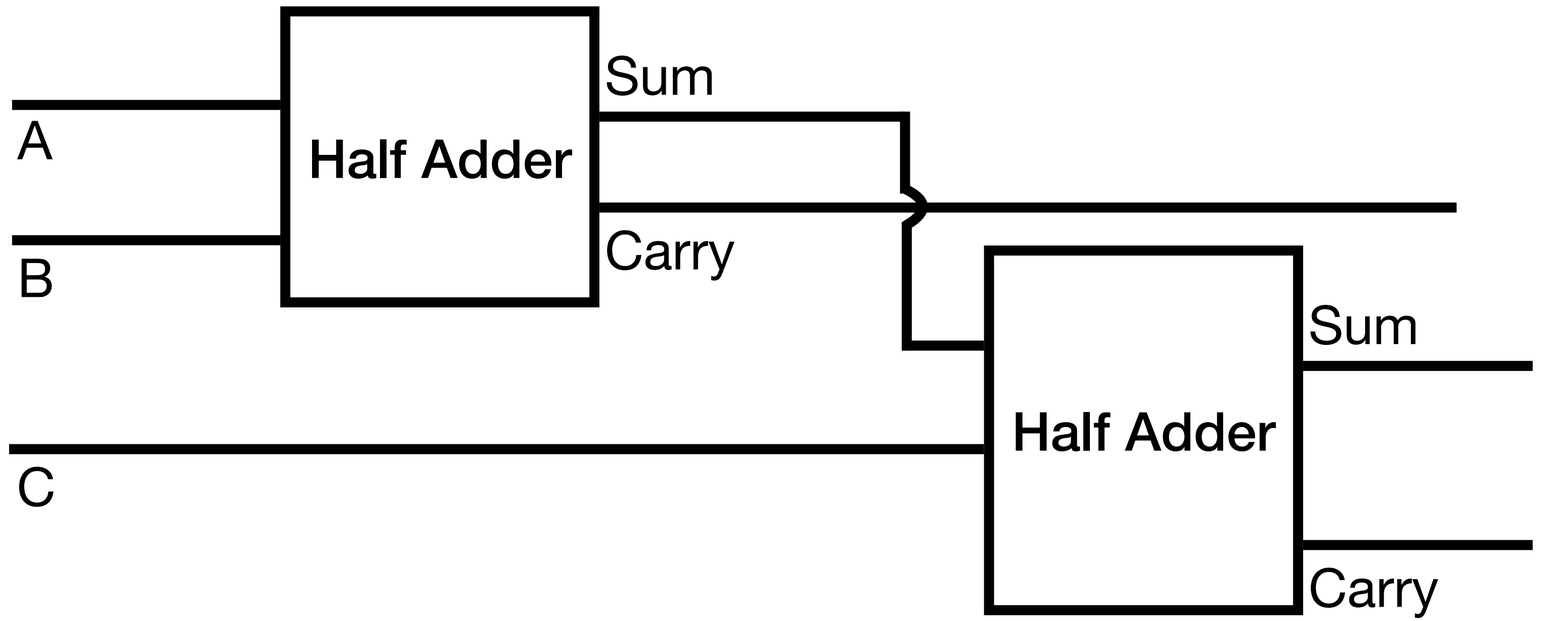


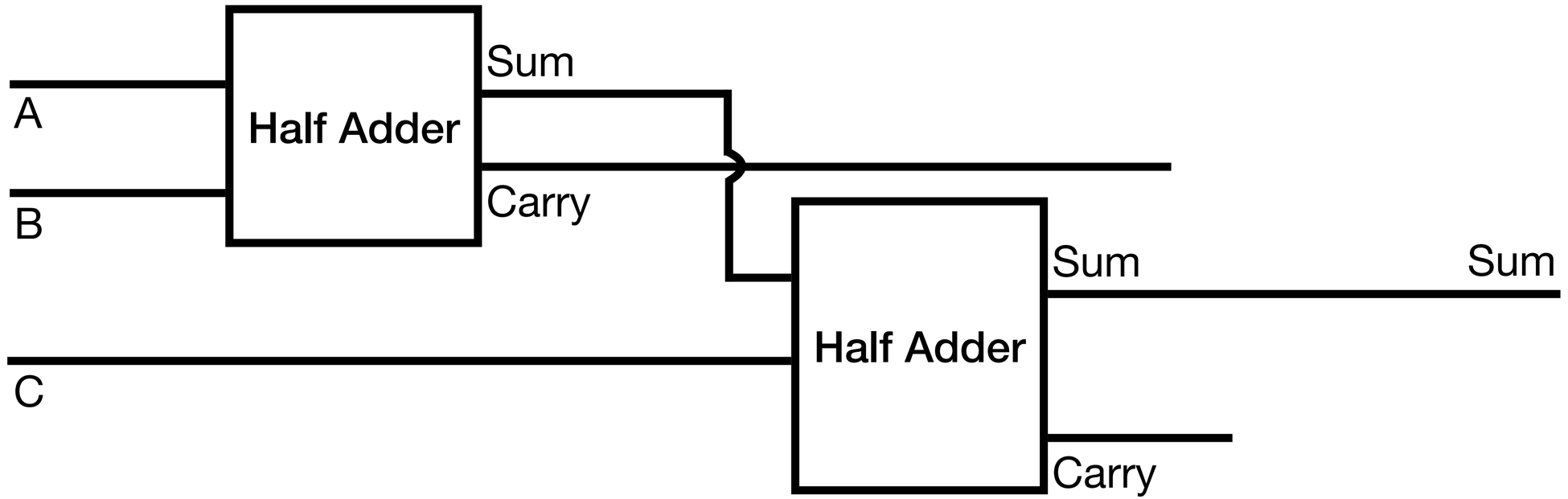

1 1
3 2 1 0
3
0 0 1 1
+ 6
0 1 1 0
9
1 0 0 1

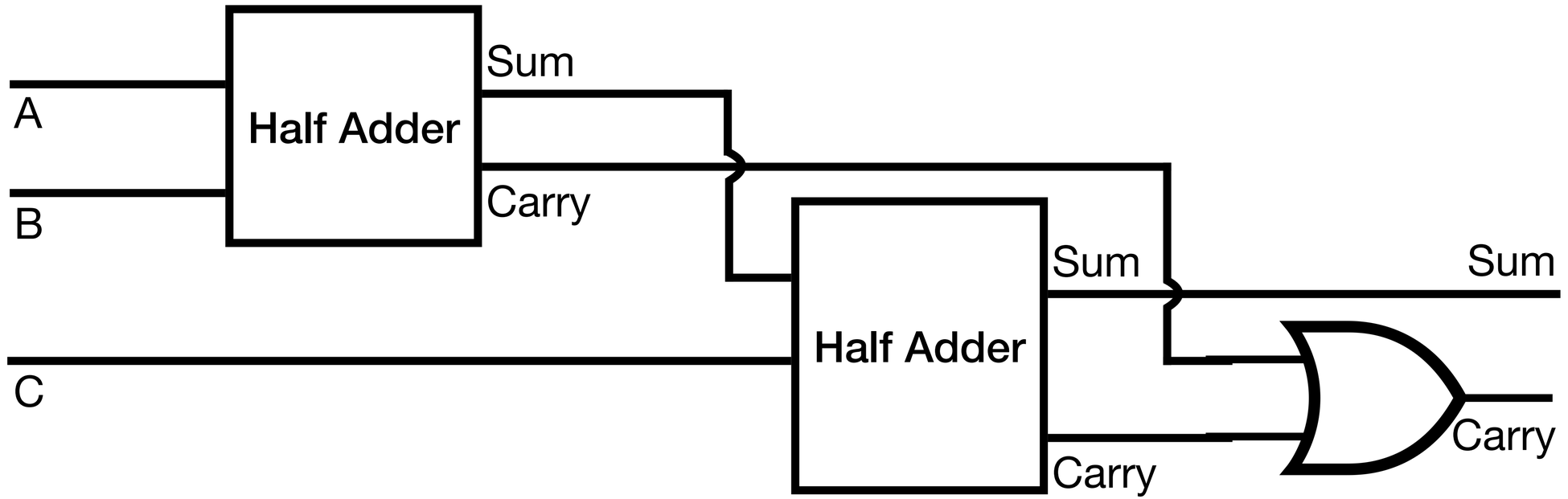

A
B
Half Adder
Sum
Carry

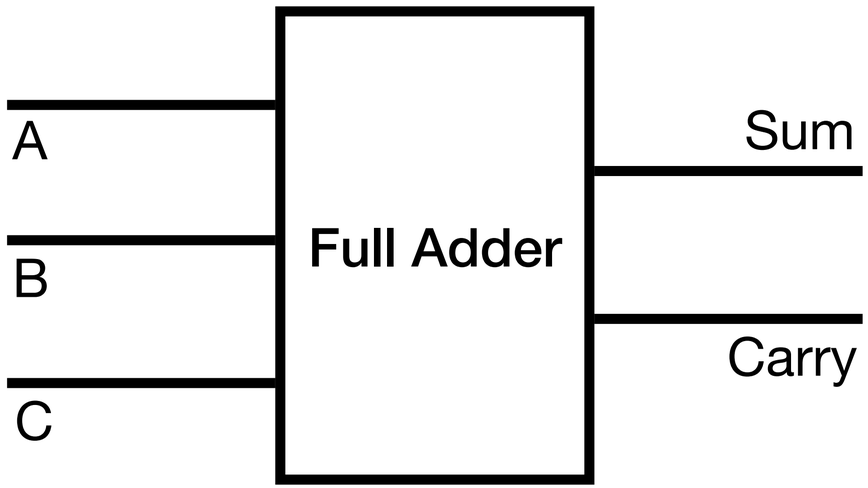

A
B
C

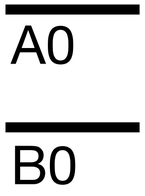

A
B
C
Half Adder
Sum
Carry


A
B
C
Half Adder
Sum
Carry
Half Adder
Sum
Carry

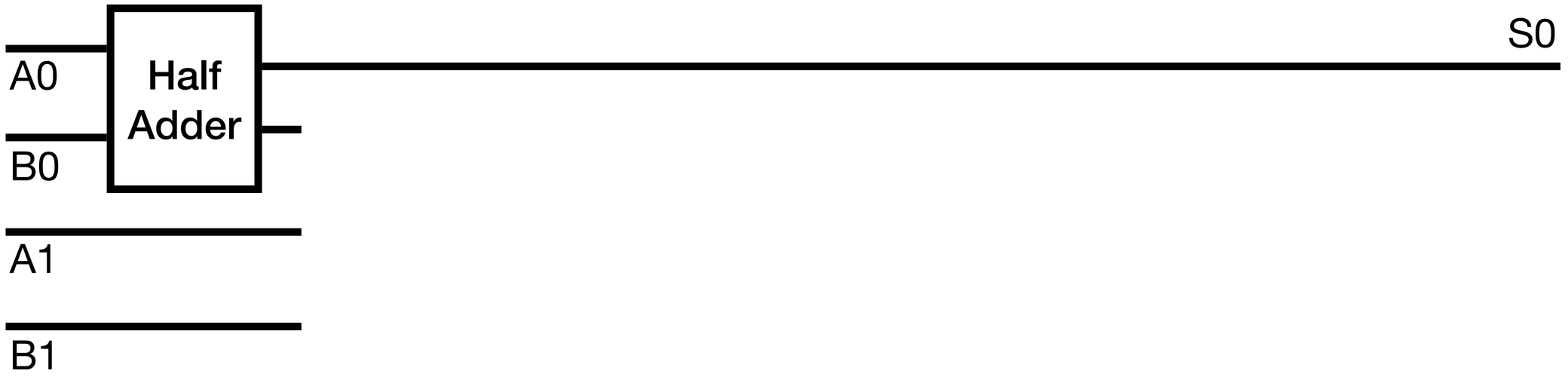

A
B
C
Half Adder
Sum
Carry
Half Adder
Sum
Carry
Sum

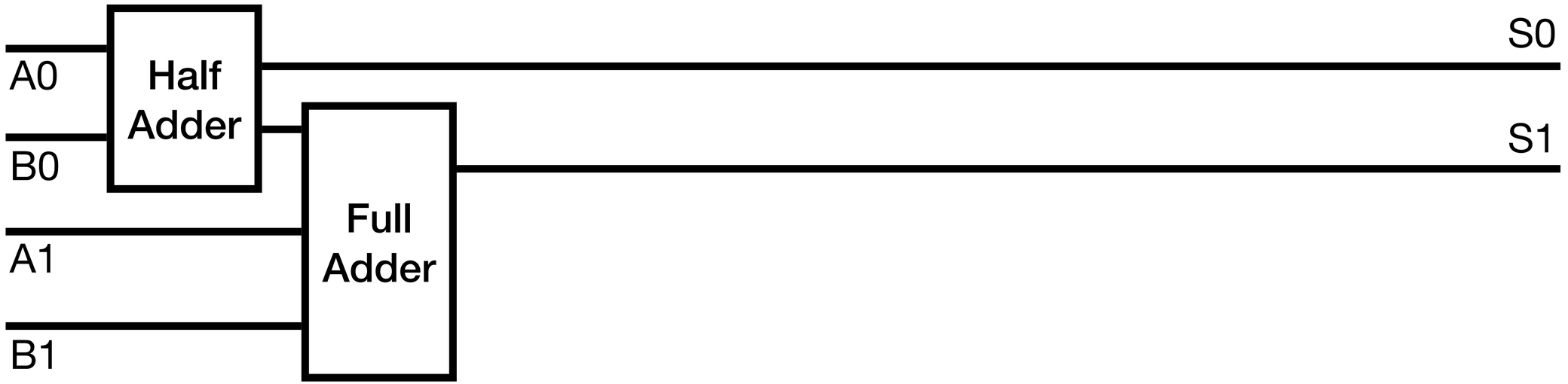

A
B
C
Half Adder
Sum
Carry
Half Adder
Sum
Carry
Sum
Carry

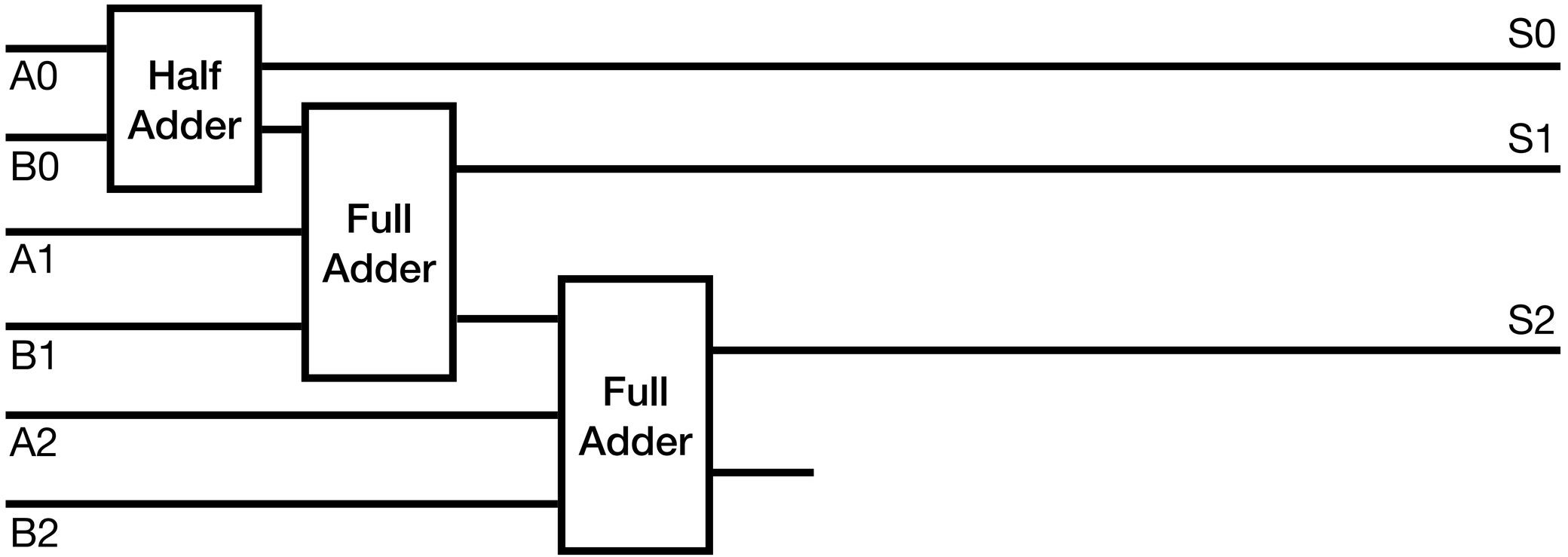

A
B
C
Full Adder
Sum
Carry

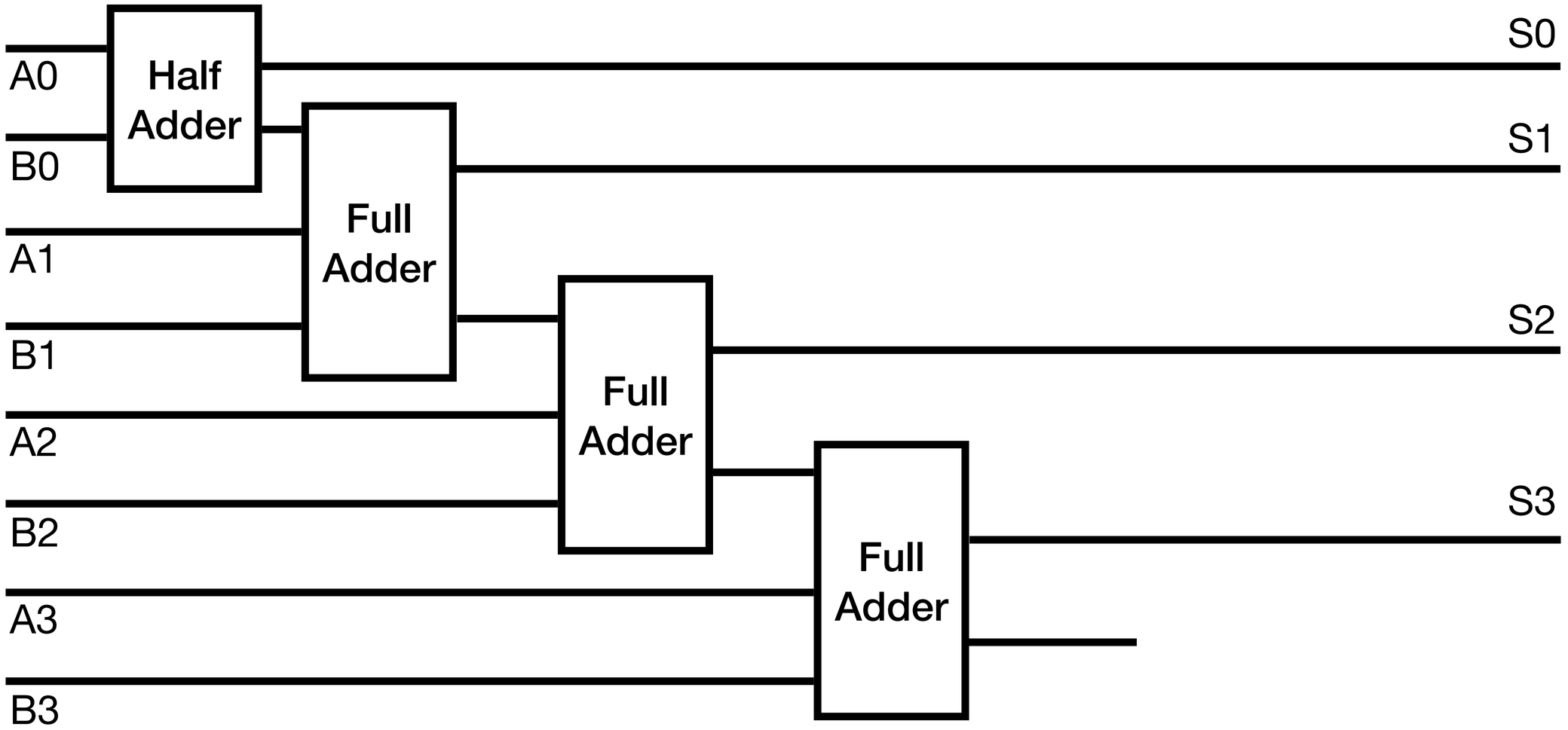

A0
B0

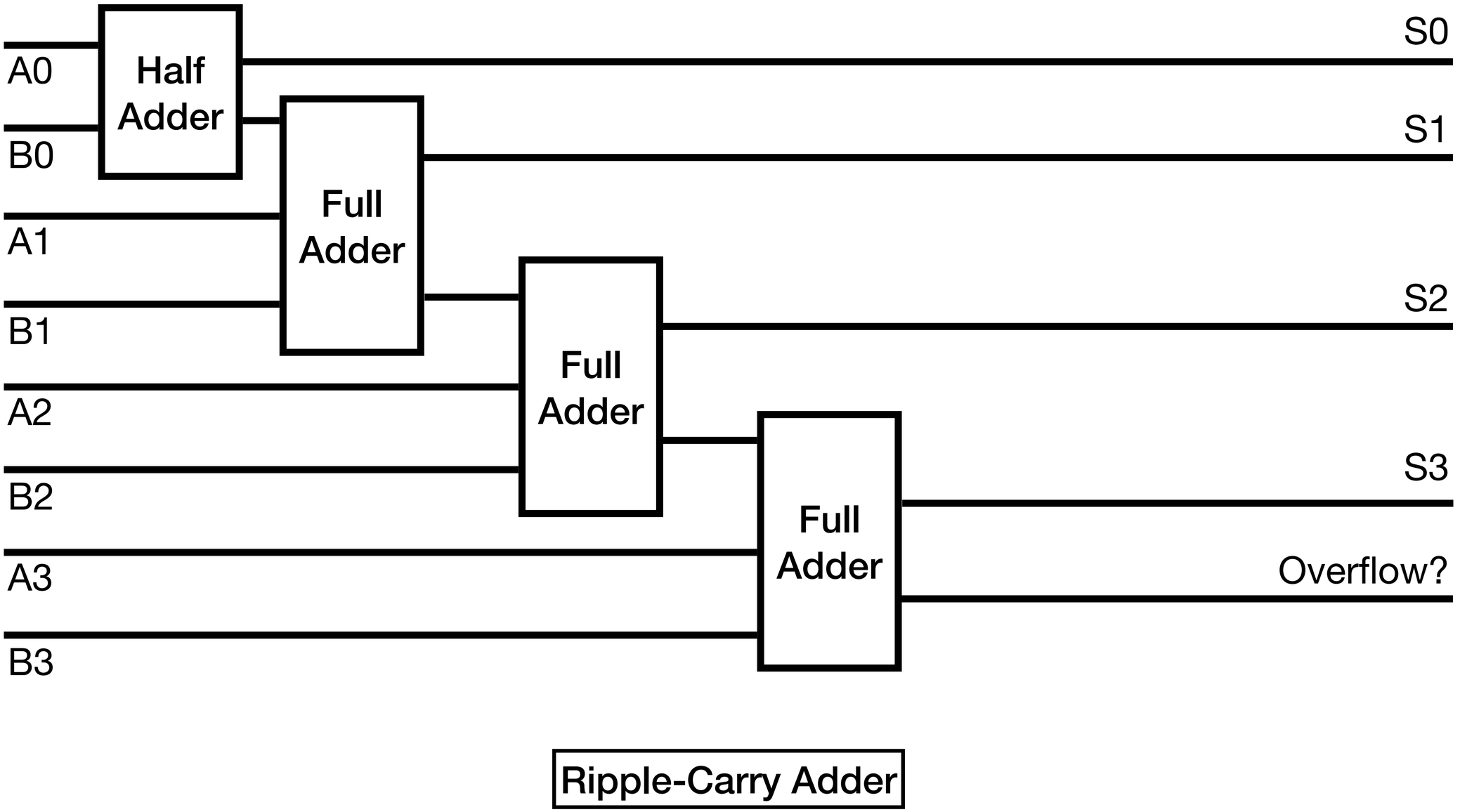

A0
B0
Half
Adder
S0


A0
B0
Half
Adder
S0
A1
B1


A0
B0
A1
B1
Half Adder
Full Adder
S0
S1


A0
B0
A1
B1
A2
B2
Half Adder
Full Adder
Full Adder
S0
S1
S2


A0
B0
A1
B1
A2
B2
A3
B3
Half Adder
Full Adder
Full Adder
Full Adder
S0
S1
S2
S3



Ripple-Carry Adder

Dealing with Complexity

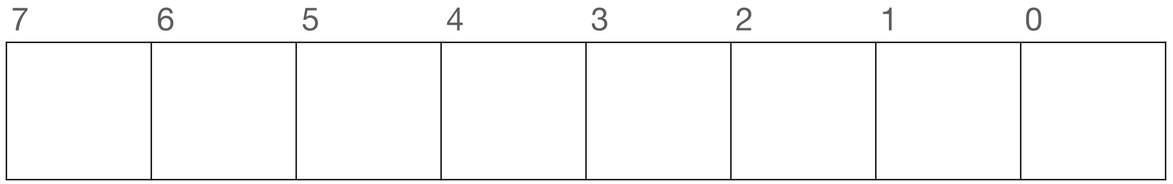
Encapsulation

- Encapsulation hides implementation details when they are not important.
- A system is built upon many layers of encapsulations:
  - Each layer has its own building blocks;
  - Each layer hides details for the layer above.
- Devices: vacuum tubes, relays, transistors
- Digital circuits: Boolean gates
- Logic: Half/Full Adders, 4-bit adders
- Numbers, functions, interfaces, ...

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 2 0



Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 20

```
    4
16 +
```

7	6	5	4	3	2	1	0
0	**0**	**0**	**1**				

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 20

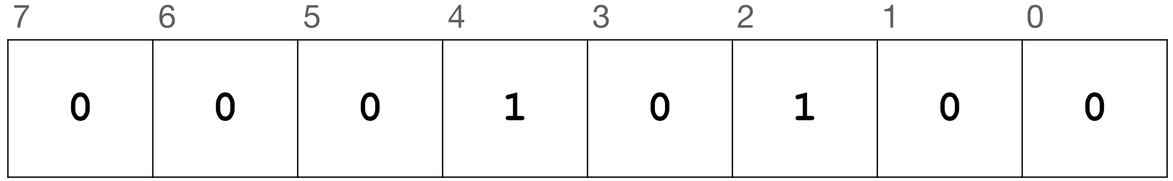
16 + 4

7	6	5	4	3	2	1	0
0	**0**	**0**	**1**	**0**	**1**	**0**	**0**

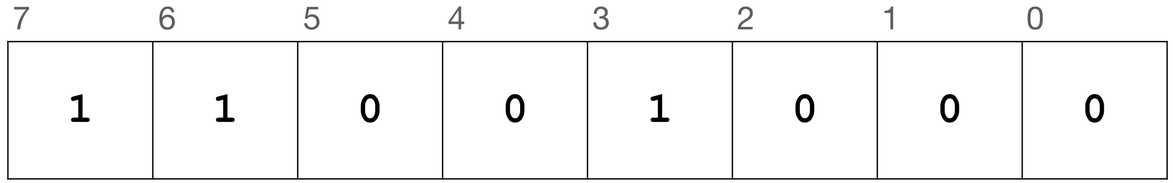
Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

20



200

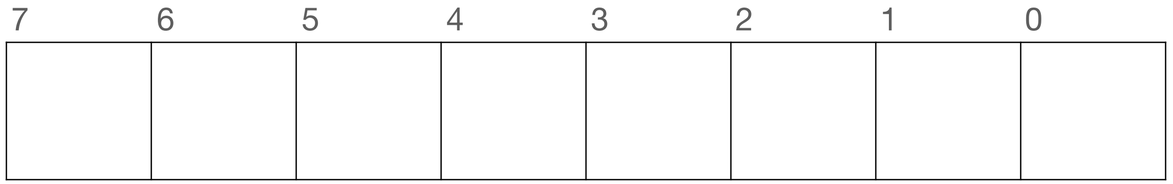


- x 10 can't convert nicely

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 40



Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 40

32 + 8

7	6	5	4	3	2	1	0
0	**0**	**1**					

Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Decimal: 40

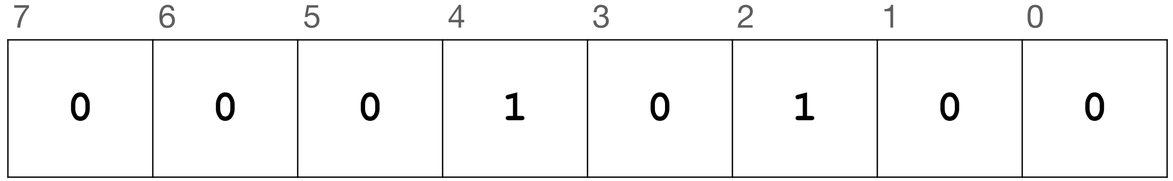
32 + 8

7	6	5	4	3	2	1	0
0	0	1	0	1	0	0	0

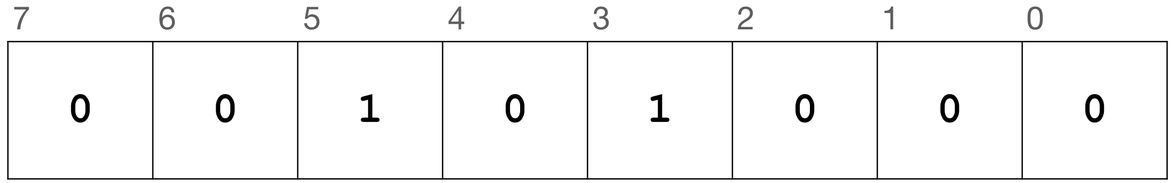
Number Conversions

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

20



40



- x 2 is shifting everything to the left!

Number Conversions

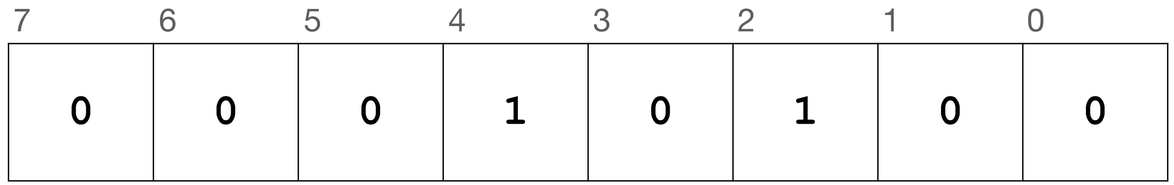
n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

123

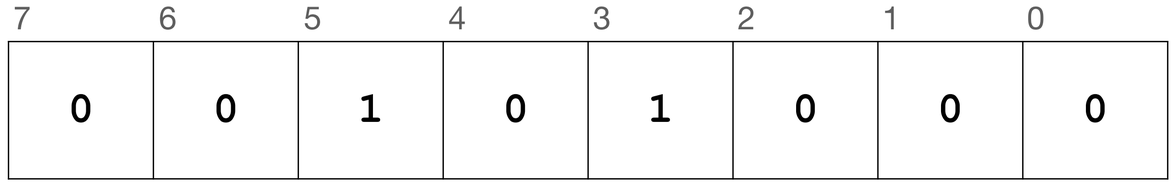
x 10

1230

Decimal



x 2

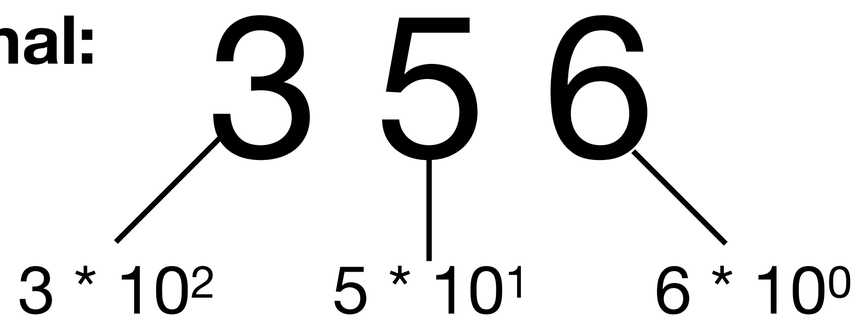


Binary

Hexadecimal Numbers

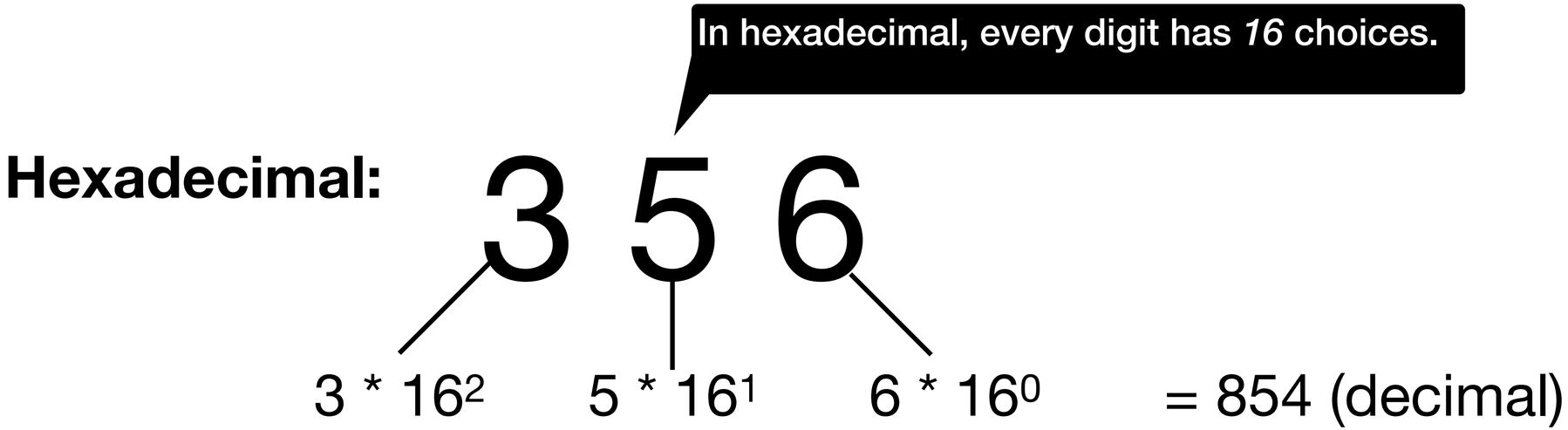
Numbers with even wackier math

Decimal:



Hexadecimal Numbers

Numbers with even wackier math



Hexadecimal Numbers

Numbers with even wackier math

In hexadecimal, every digit has *16* choices.

Hexadecimal: 3 5 6

$3 * 16^2$ $5 * 16^1$ $6 * 16^0$ = 854 (decimal)

										10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Hexadecimal Numbers

Numbers with even wackier math

- Why 16?
- Because 16 is 2^4! So every digit needs exactly 4 bits -- this will turn out to be very convenient
- Usually, we prefix hexadecimal numbers with `0x`

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
										10	11	12	13	14	15

Hexadecimal Numbers

Numbers with even wackier math

n	16^n
0	1
1	16
2	256
3	4096

0x A B

10 11 12 13 14 15

0 1 2 3 4 5 6 7 8 9 A B C D E F

Hexadecimal Numbers

Numbers with even wackier math

n	16^n
0	1
1	16
2	256
3	4096

0x A B

10 * 16 + 11 * 1 = 171

7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	1

								10	11	12	13	14	15		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Hexadecimal Numbers

Numbers with even wackier math

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

0x A B

10 * 16 + 11 * 1 = 171

3	2	1	0	3	2	1	0
1	0	1	0	1	0	1	1

								10	11	12	13	14	15		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

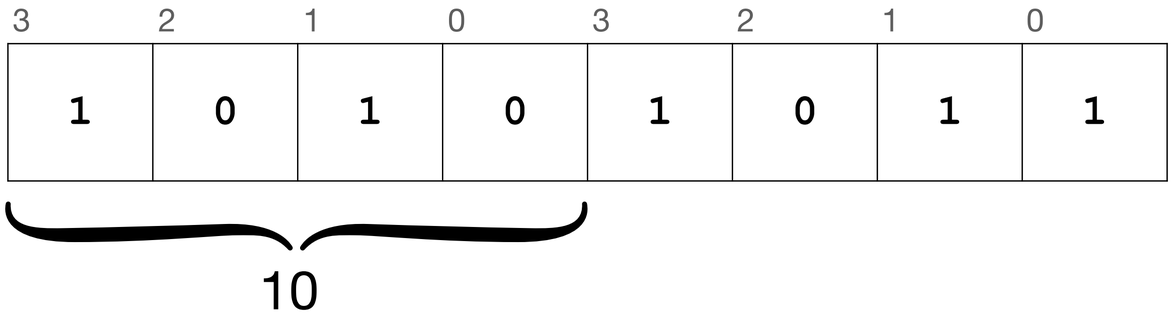
Hexadecimal Numbers

Numbers with even wackier math

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

0x A B

10 * 16 + 11 * 1 = 171



10 11 12 13 14 15

0 1 2 3 4 5 6 7 8 9 A B C D E F

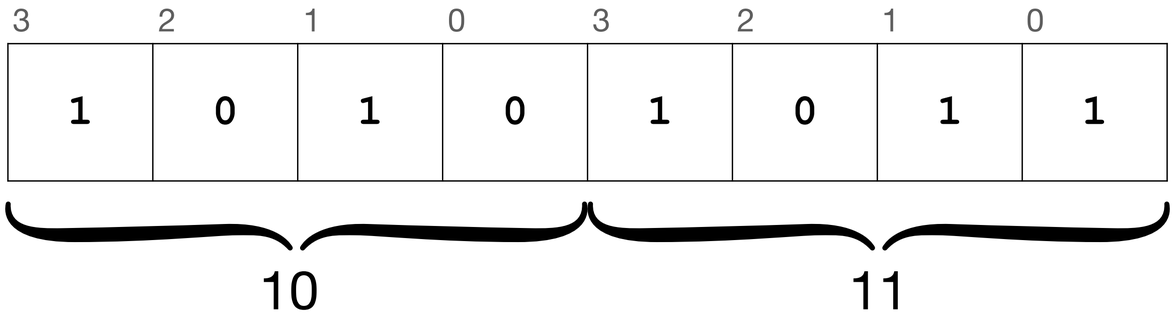
Hexadecimal Numbers

Numbers with even wackier math

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

0x A B

10 * 16 + 11 * 1 = 171



10 11 12 13 14 15

0 1 2 3 4 5 6 7 8 9 A B C D E F

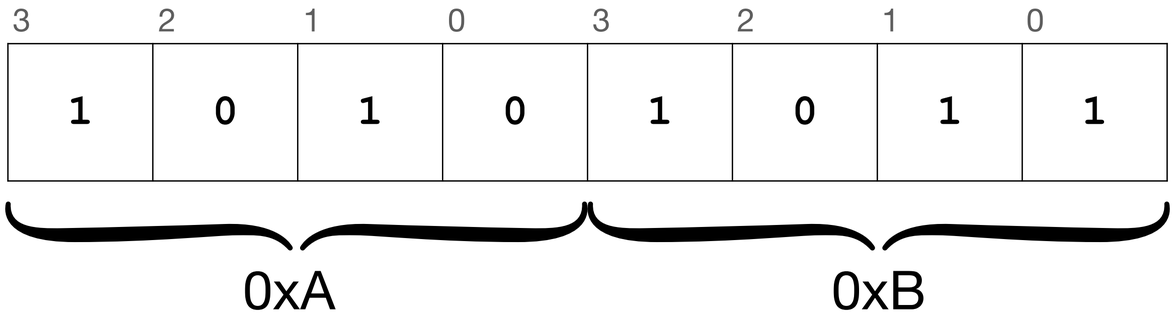
Hexadecimal Numbers

Numbers with even wackier math

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

0x A B

10 * 16 + 11 * 1 = 171



10 11 12 13 14 15

0 1 2 3 4 5 6 7 8 9 A B C D E F

Hexadecimal Numbers

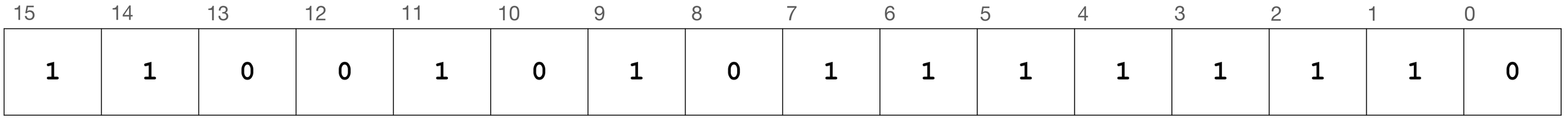
Numbers with even wackier math

- Every hex digit corresponds to 4 binary digits
- We can convert 1 hex digit/4 binary digits at a time!
  - Can't do this with decimal digits

										10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Hexadecimal Numbers

Numbers with even wackier math

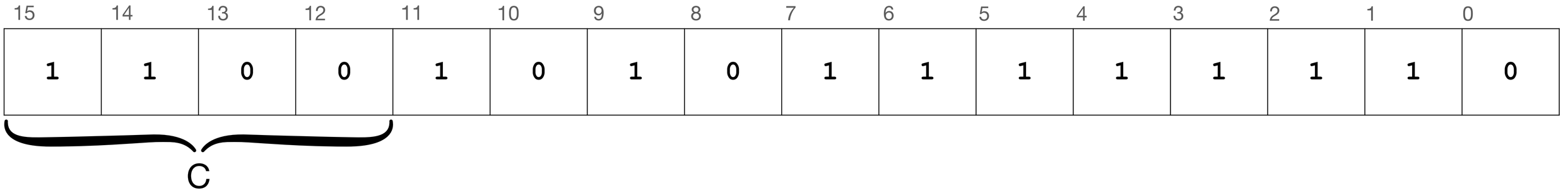


10 11 12 13 14 15

0 1 2 3 4 5 6 7 8 9 A B C D E F

Hexadecimal Numbers

Numbers with even wackier math

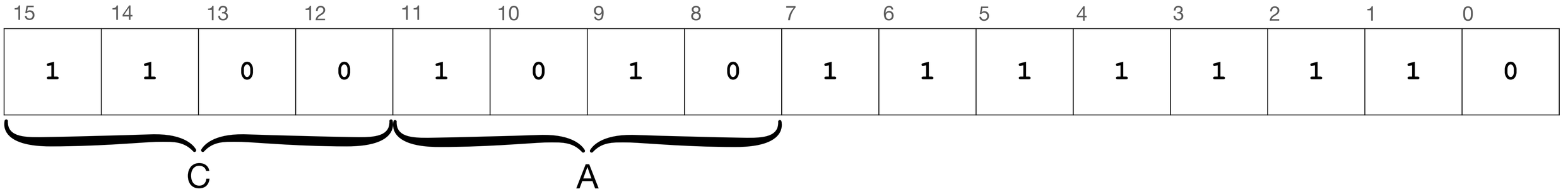


10 11 12 13 14 15

0 1 2 3 4 5 6 7 8 9 A B C D E F

Hexadecimal Numbers

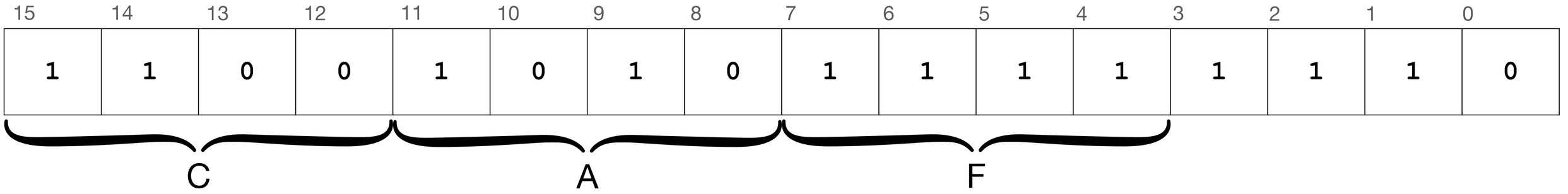
Numbers with even wackier math



0 1 2 3 4 5 6 7 8 9 10 A 11 B 12 C 13 D 14 E 15 F

Hexadecimal Numbers

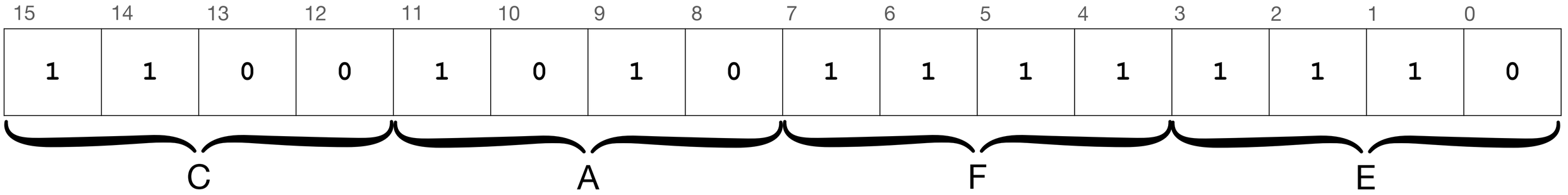
Numbers with even wackier math



0 1 2 3 4 5 6 7 8 9 10 A 11 B 12 C 13 D 14 E 15 F

Hexadecimal Numbers

Numbers with even wackier math

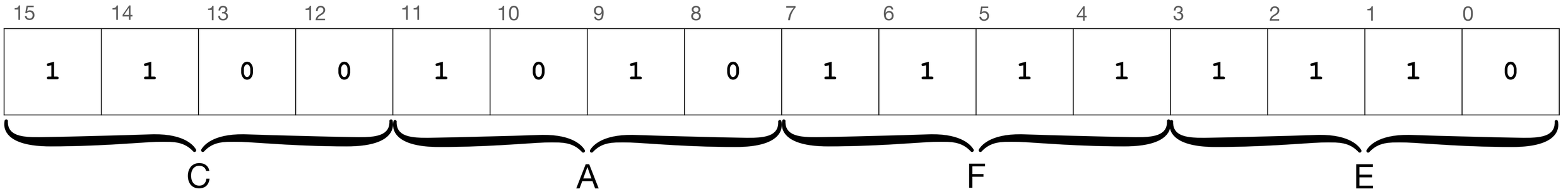


0 1 2 3 4 5 6 7 8 9 A B C D E F

10 11 12 13 14 15

Hexadecimal Numbers

Numbers with even wackier math




```
>>> hex(0b1100101011111110)
'0xcafe'
```

10 11 12 13 14 15

0 1 2 3 4 5 6 7 8 9 A B C D E F

Hexadecimal Numbers

0x123

x 16

0x1230

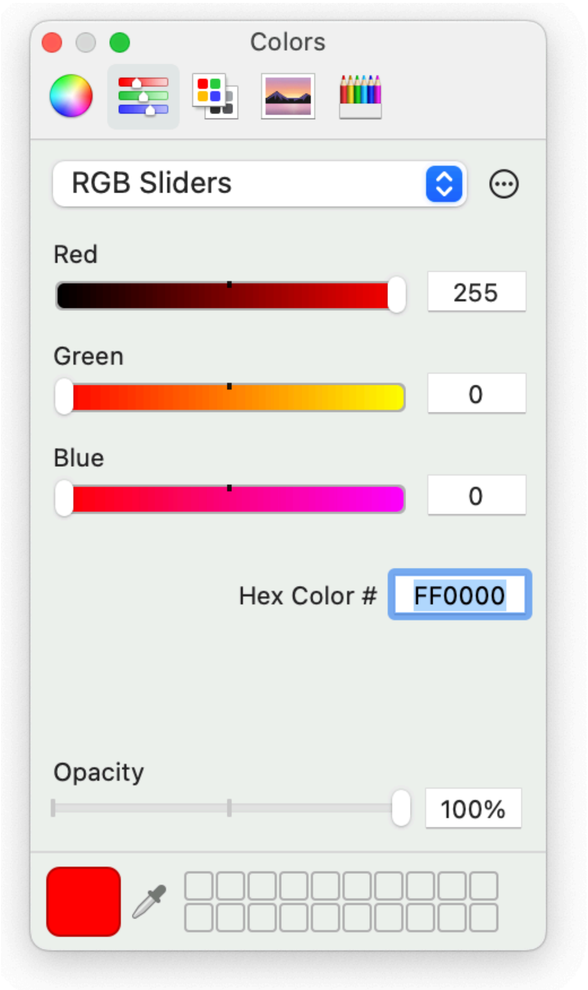
```
>>> bin(0x123)
'0b100100011'
>>> bin(0x1230)
'0b1001000110000'
```

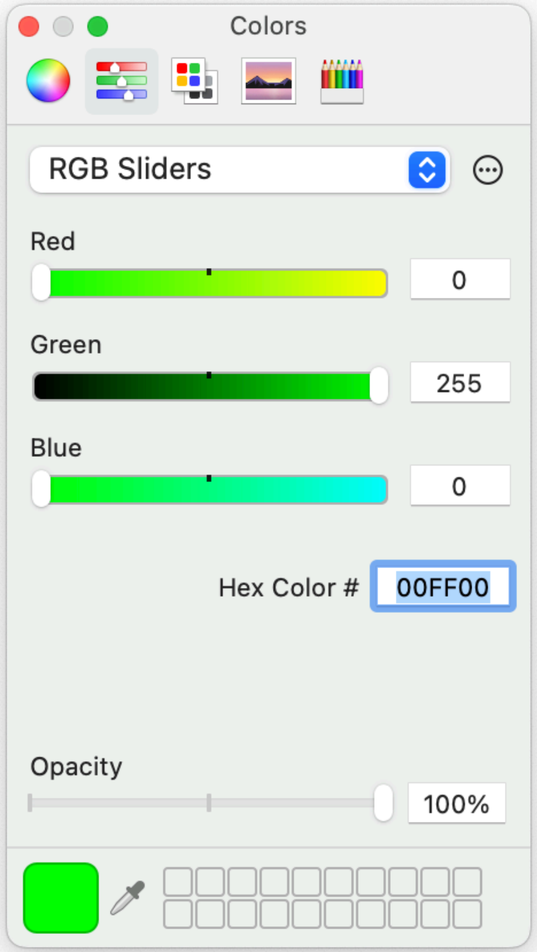
Hexadecimal

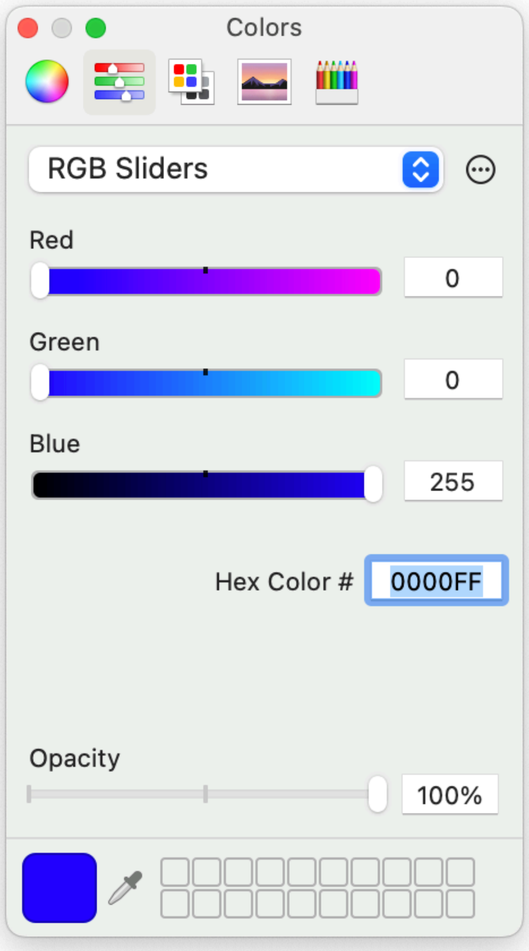
Binary

Hexadecimal Numbers

Hex numbers are everywhere







Numbers in Bits

- Numbers are just bits.
  - We can build logic gates to process these numbers.
  - More operators next time.
- Hexadecimal numbers
  - Tighter correlation between numbers and bits