CS64 Week 5 Lecture 1

Kyle Dewey

Overview

- Recursion in the MIPS calling convention
- Tail call optimization
- Introduction to circuits
- Digital design: single bit adders
- Karnaugh maps

Recursion in MIPS

Quick MIPS Calling Convention Review

- `nested_calls.asm`
- `save_registers.asm`

Recursion

- This same setup handles nested function calls and recursion - we can save `$ra` on the stack
- Example: `recursive_fibonacci.asm`

More Recursion

- What's special about the following recursive function?

```
int recFac(int n, int accum) {
  if (n == 0) {
    return accum;
  } else {
    return recFac(n - 1, n * accum);
  }
}
```

More Recursion

- What's special about the following recursive function?
  - It is *tail recursive* - with the right optimization, uses constant stack space
  - We can do this in assembly - `tail_recursive_factorial.asm`

```
int recFac(int n, int accum) {
  if (n == 0) {
    return accum;
  } else {
    return recFac(n - 1, n * accum);
  }
}
```

Dispelling the Magic: Circuits

Why Binary?

- Very convenient for a circuit
  - Two possible states: on and off
  - `0` and `1` correspond to on and off

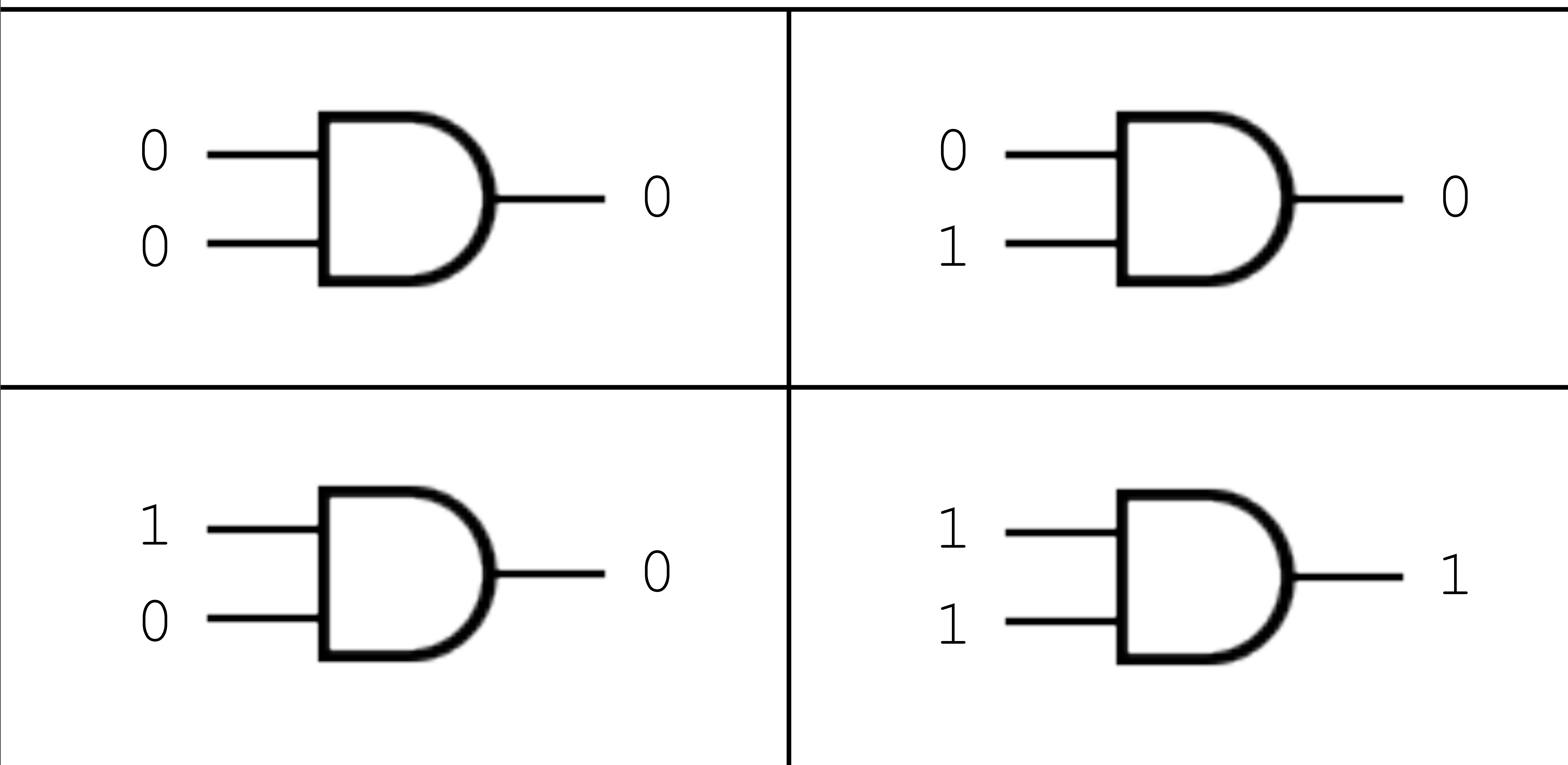
Relationship to Bitwise Operations

- You're already familiar with bitwise OR, AND, XOR, and NOT
- These same operations are fundamental to circuits
  - Basic building blocks for more complex things

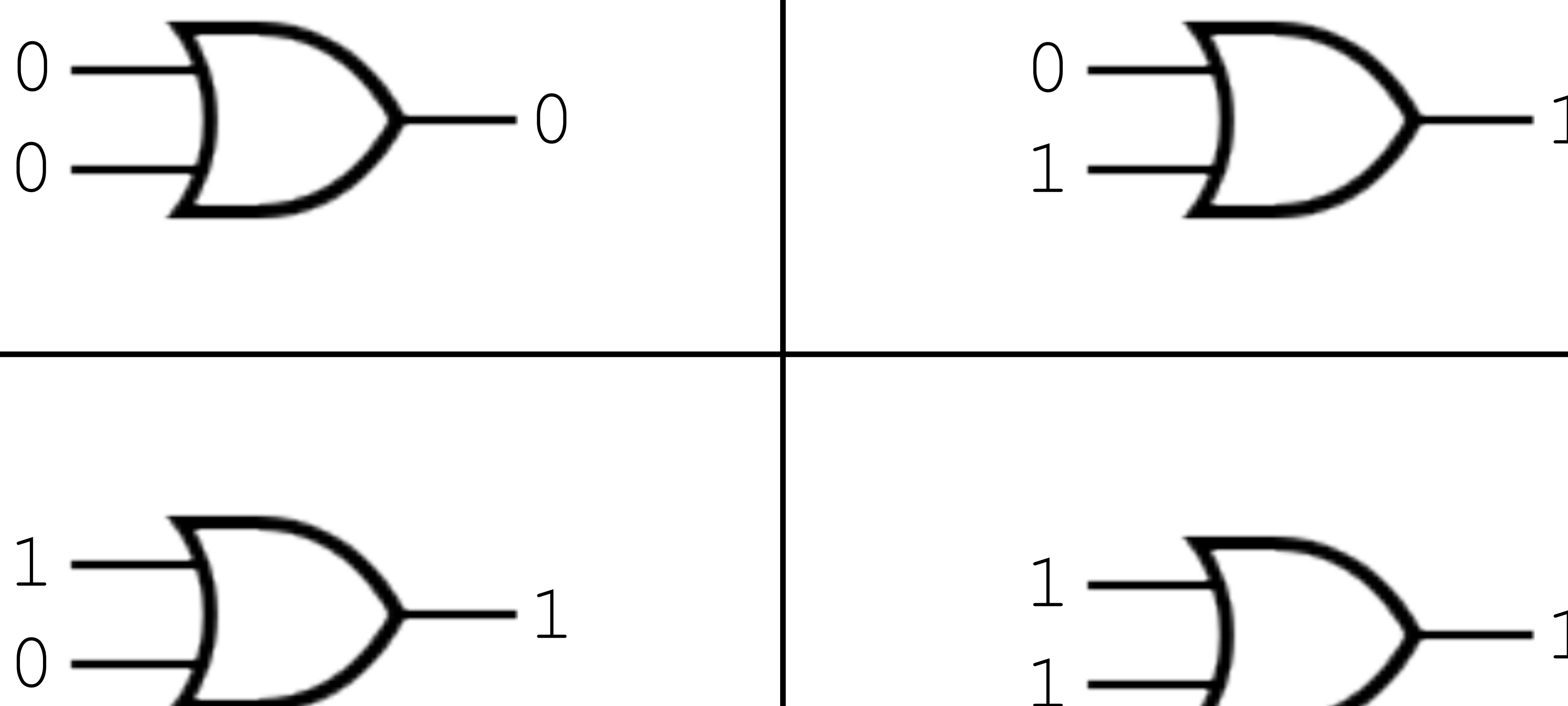
Single Bits

- For the moment, we will deal only with individual bits
- Later, we'll see this isn't actually that restrictive

Operations on Single Bits: AND

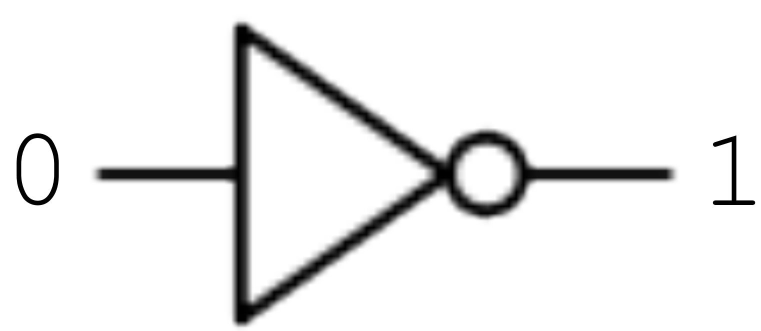


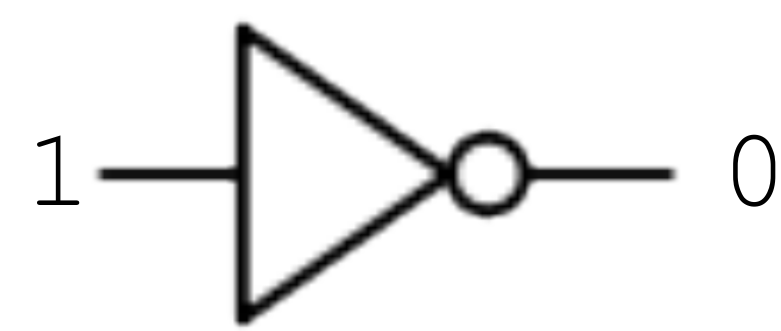
Operations on Single Bits: OR



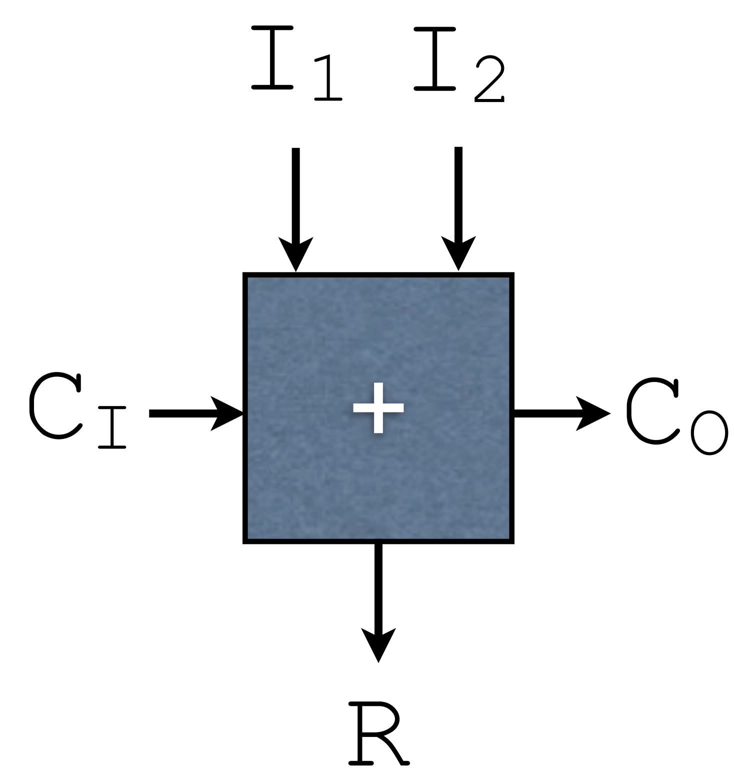
Operations on Single Bits: XOR

Operations on Single Bits: NOT





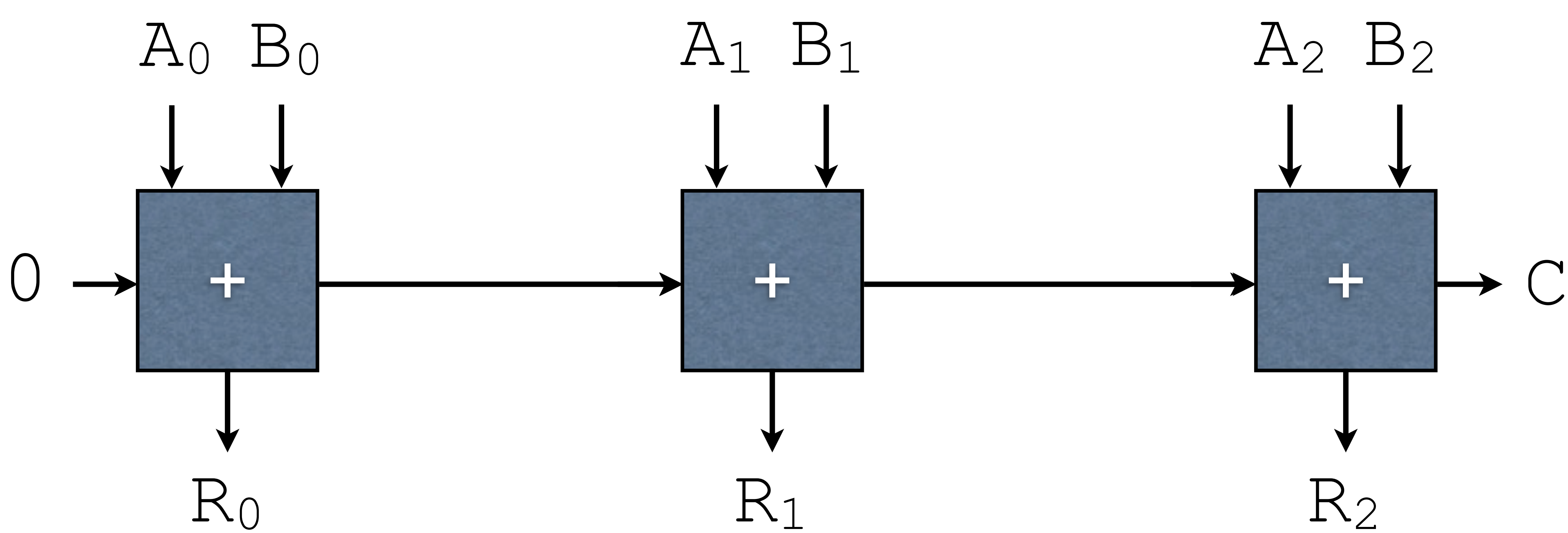
Recall: Single Bit Adders

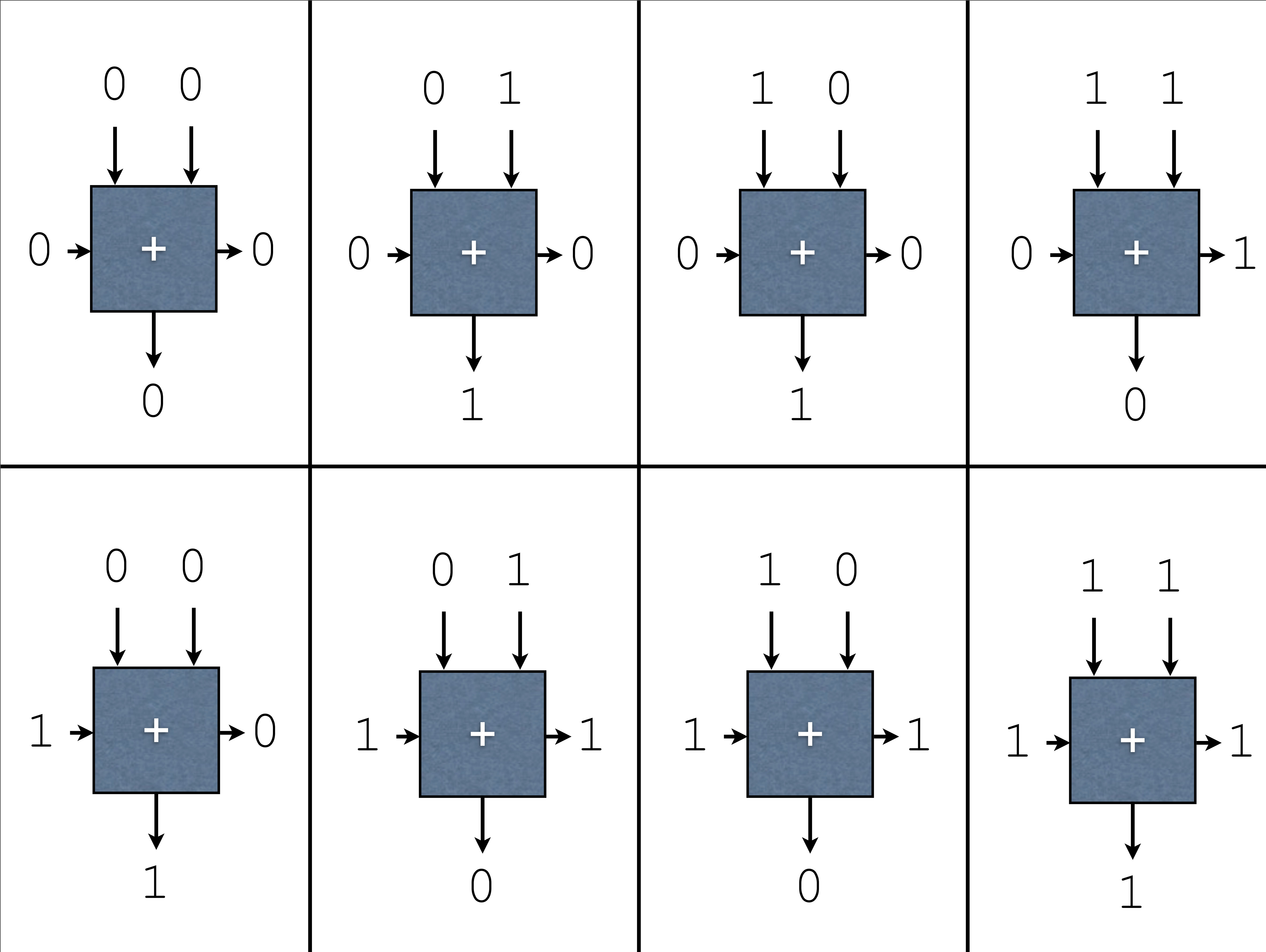


-We had defined a single bit adder that worked like the above...

Stringing them Together

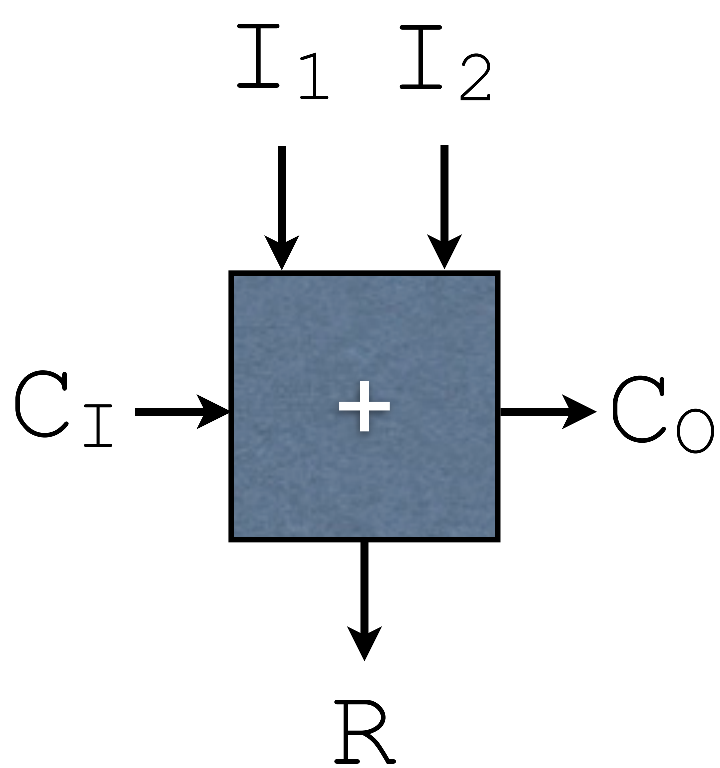
For two three-bit numbers, A and B, resulting in a three-bit result R





0 0
0 + 0
0
0 1
0 + 0
1
1 0
0 + 0
1
1 1
0 + 1
0
0 0
1 + 0
1
0 1
1 + 1
0
1 0
1 + 1
0
1 1
1 + 1
1

As a Truth Table

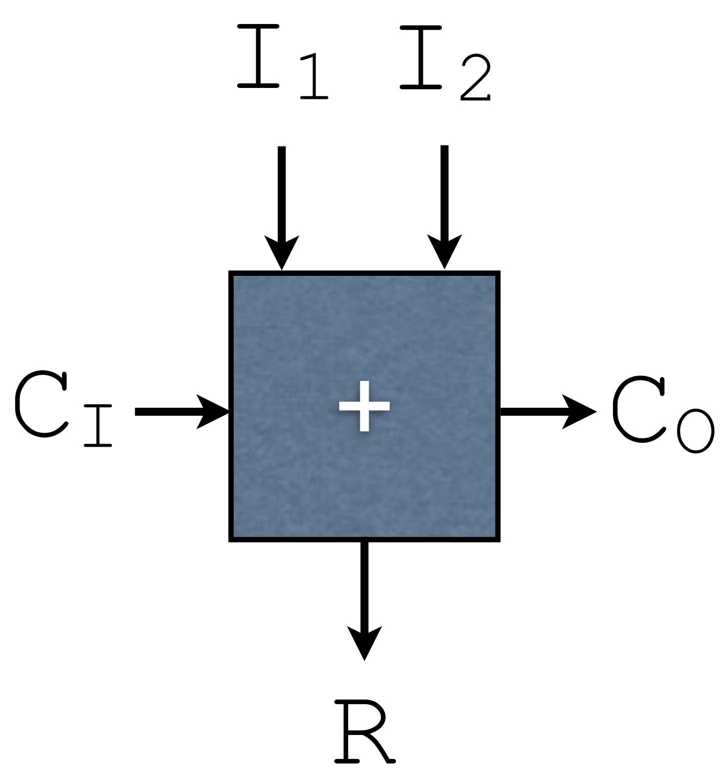


C_I	I_1	I_2	C_O	R
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

–Inputs and outputs are separated by a line

As a Truth Table

Question: how can this be turned into a circuit?



C_I	I_1	I_2	C_O	R
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

-As in, how can we utilize the gates from before to implement this?

Sum of Products

- Variables: A, B, C...
- Negation of a variable: $\overline{A}$, $\overline{B}$, $\overline{C}$...

–Negating a variable is denoted by putting a bar above it

Sum of Products

- Another way to look at OR: sum (+)

$$A + B$$

- Another way to look at AND: multiplication (*)

$A * B$ $\qquad\qquad$ AB

Sum of Products Example

A	B	O
0	0	0
0	1	1
1	0	1
1	1	0

-Say we have this truth table

Sum of Products Example

A	B	O
0	0	0
0	1	1
1	0	1
1	1	0

-We look at the rows with a 1 result

Sum of Products Example

A	B	O
0	0	0
0	1	1
1	0	1
1	1	0

$O = \overline{A}*B$

–For each such row, we generate a product

Sum of Products Example

A	B	O
0	0	0
0	1	1
1	0	1
1	1	0

$$O = \overline{A} * B + A * \overline{B}$$

–For each such row, we generate a product

Sum of Products

C_I	I_1	I_2	C_O	R
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

Question: What would the sum of products look like for this table?

(Note: need one equation for each output.)

Sum of Products

C_I	I_1	I_2	C_O	R
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

Question: What would the sum of products look like for this table?

(Note: need one equation for each output.)

Answer in the presenter notes.

-Using `!A` to mean the negation of `A`
Co = !Ci*I1*I2 + Ci*!I1*I2 + Ci*I1*!I2 + CiI1I2
R = !Ci*!I1*I2 + !Ci*I1*!I2 + Ci*!I1*!I2 + CiI1I2

In-Class Example: Shift Left by 1

Karnaugh Maps

-Material derived from The Essentials of Computer Organization and Architecture 2nd edition, by Linda Null and Julia Lobur

Motivation

- Unnecessarily large programs: bad
- Unnecessarily large circuits: Very Bad™
  - Why?

Motivation

- Unnecessarily large programs: bad
- Unnecessarily large circuits: Very Bad™
  - Why?
    - Bigger circuits = bigger chips = higher cost (non-linear too!)
    - Longer circuits = more time needed to move electrons through = slower

Simplification

- Real-world formulas can often be simplified
  - How might we simplify the following?

```
R = A*B + !A*B
```

–How might we simplify this?

Simplification

- Real-world formulas can often be simplified
  - How might we simplify the following?

```
R = A*B + !A*B
R = B(A + !A)
R = B(true)
R = B
```

Scaling Up

- Performing this sort of algebraic manipulation by hand can be tricky
- We can use *Karnaugh maps* to make it immediately apparent as to what can be simplified

Example

```
R = A*B + !A*B
```

-Start with the sum of products

Example

$$R = A*B + !A*B$$

A	B	O
0	0	0
0	1	1
1	0	0
1	1	1

–Build the truth table

Example

R = A*B + !A*B

A	B	O
0	0	0
0	1	1
1	0	0
1	1	1

A \ B	0	1
0	0	1
1	0	1

-Build the K-map

Example

R = A*B + !A*B

A	B	O
0	0	0
0	1	1
1	0	0
1	1	1

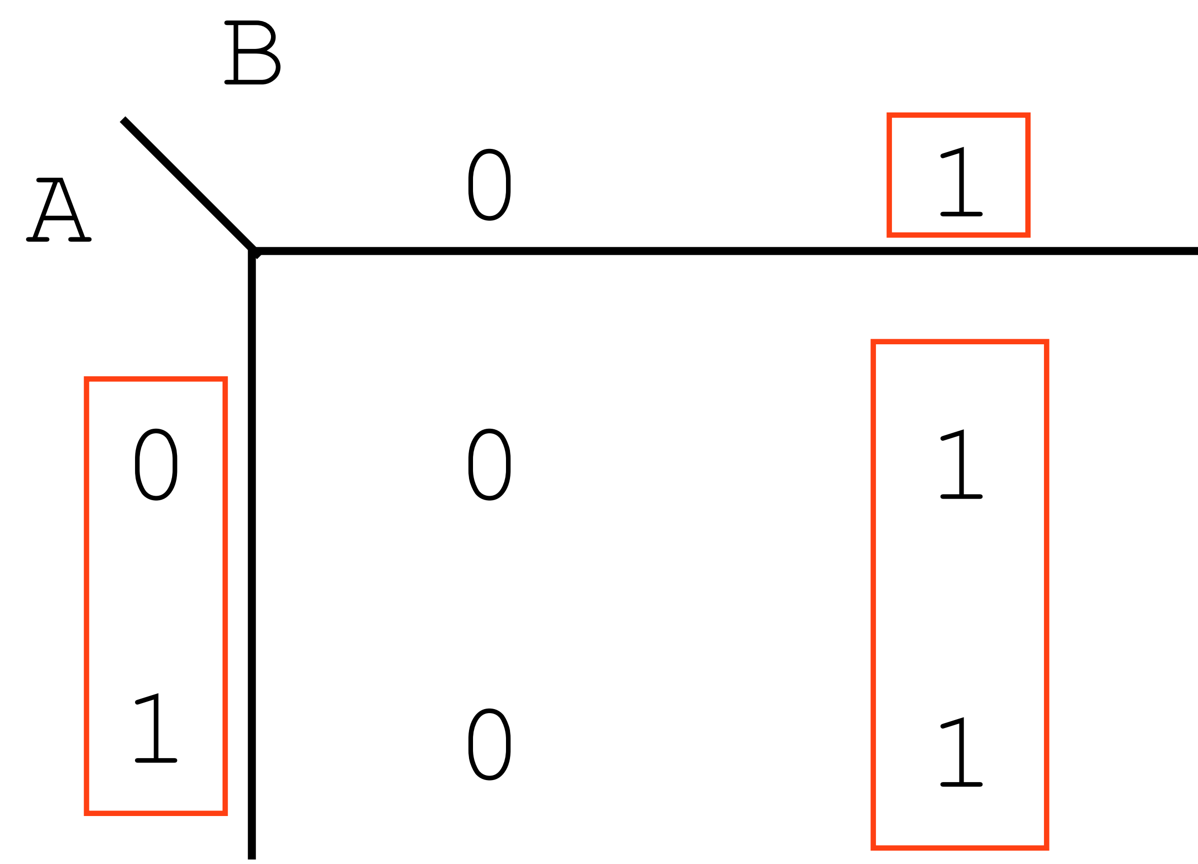
A \ B	0	1
0	0	1
1	0	1

-Group adjacent (row or column-wise, NOT diagonal) 1's in powers of two (groups of 2, 4, 8...)
-

Example

R = A*B + !A*B

A	B	O
0	0	0
0	1	1
1	0	0
1	1	1



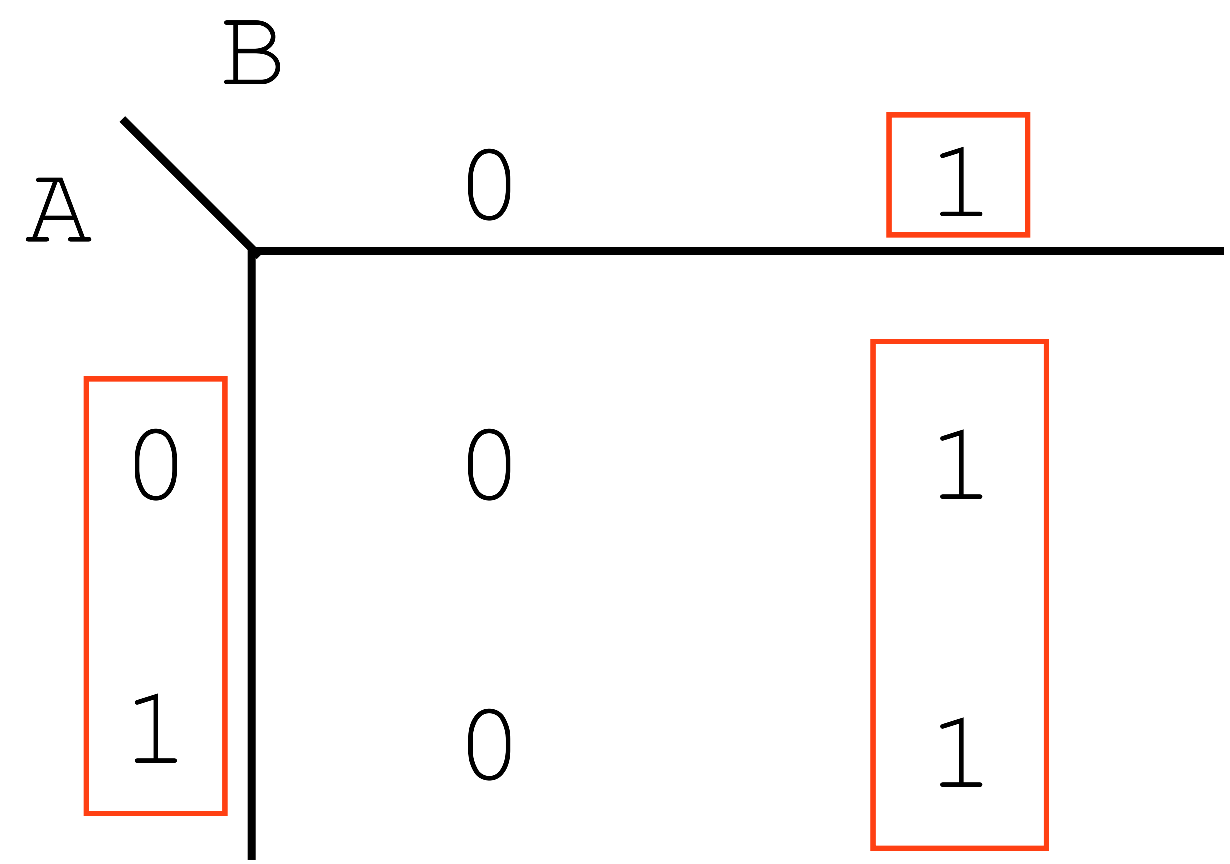
-The values that stay the same are saved, the rest are discarded
-This works because this means that the inputs that differ are irrelevant to the final value, and so they can be removed

Example

R = A*B + !A*B

A	B	O
0	0	0
0	1	1
1	0	0
1	1	1

R = B



-The values that stay the same are saved, the rest are discarded
-This works because this means that the inputs that differ are irrelevant to the final value, and so they can be removed

Three Variables

- We can scale this up to three variables, by combining two variables on one axis
- The combined axis must be arranged such that only one bit changes per position

A \ BC	00	01	11	10
0	?	?	?	?
1	?	?	?	?

Three Variable Example

R = !A!BC + !ABC + A!BC + ABC

-Start with this formula

R = !A!BC + !ABC + A!BC + ABC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

–Build the truth table

R = !A!BC + !ABC + A!BC + ABC

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

A \ BC	00	01	11	10
0	0	1	1	0
1	0	1	1	0

-Build the K-map

$$R = !A!BC + !ABC + A!BC + ABC$$

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

A \ BC	00	01	11	10
0	0	1	1	0
1	0	1	1	0

-Select the biggest group possible, in this case a square

-In order to get the most minimal circuit, we must always select the biggest groups possible

$$R = !A!BC + !ABC + A!BC + ABC$$

A	B	C	R
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

R = C

A \ BC	00	01	11	10
0	0	1	1	0
1	0	1	1	0

-Save the ones that stay the same in a group, discarding the rest

Another Three Variable Example

R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C

-Start with this formula

R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C

A	B	C	R
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

-Build the truth table

$$R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C$$

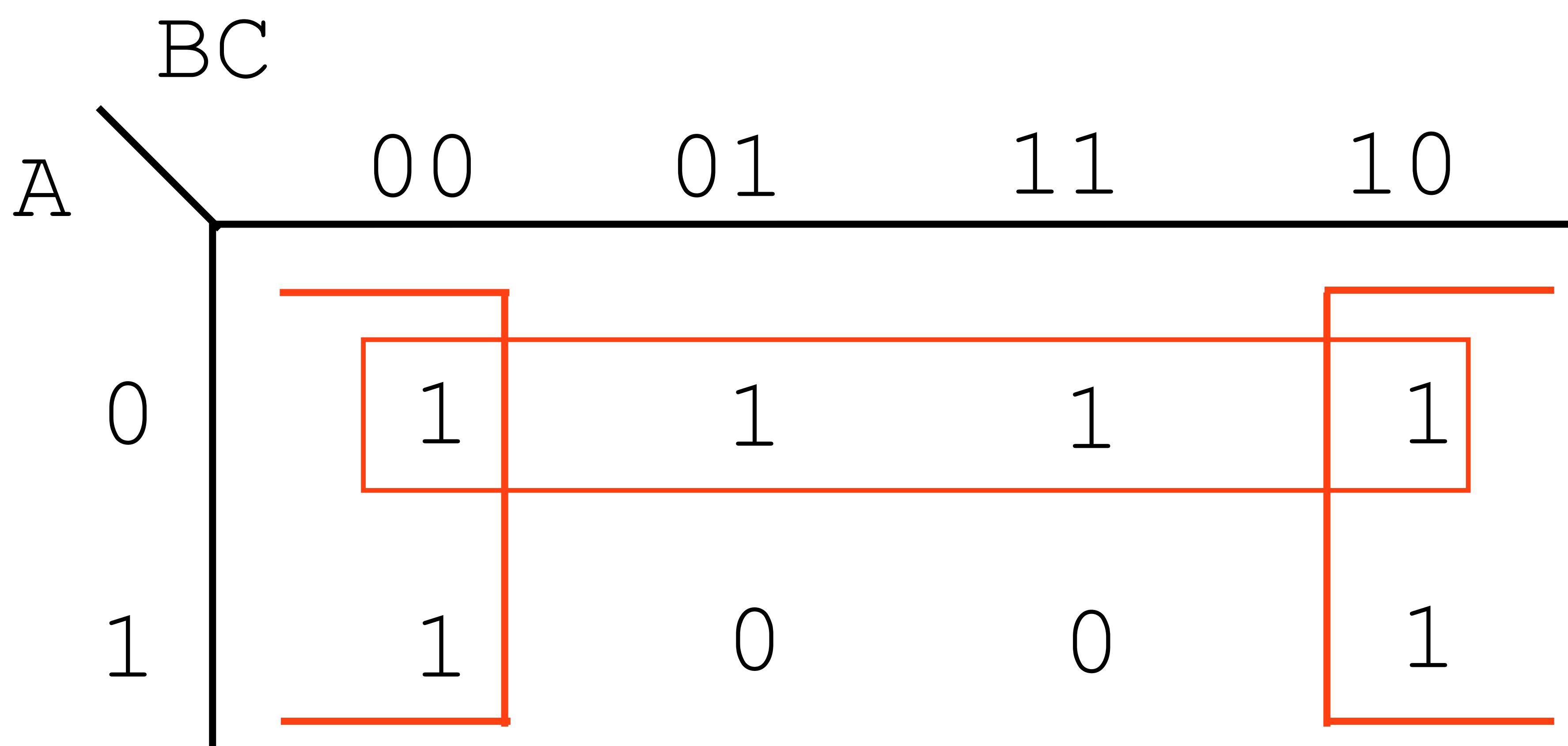
A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

A \ BC	00	01	11	10
0	1	1	1	1
1	1	0	0	1

-Build the K-map

R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C

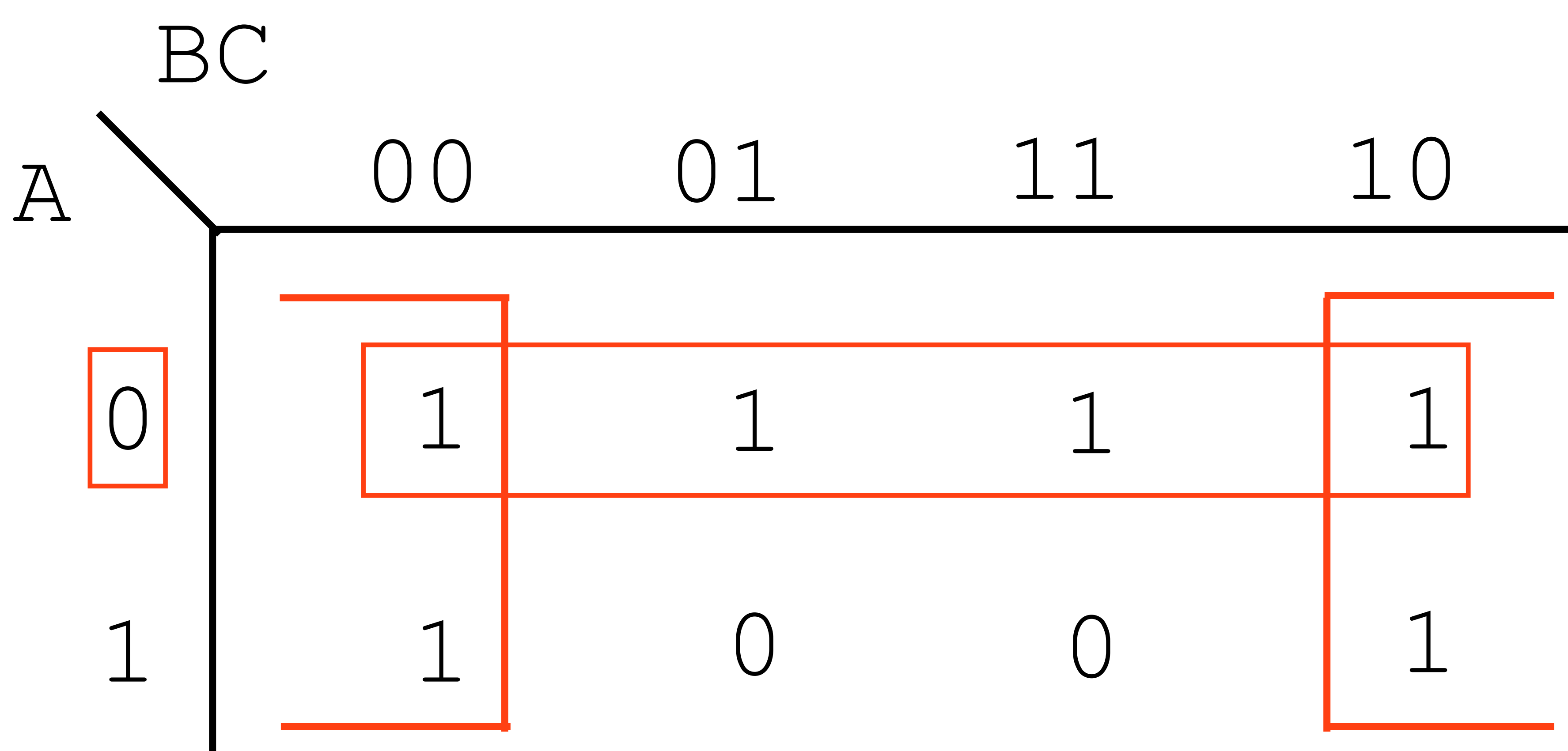
A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



-Select the biggest groups possible
-Note that the values "wrap around" the table

R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C

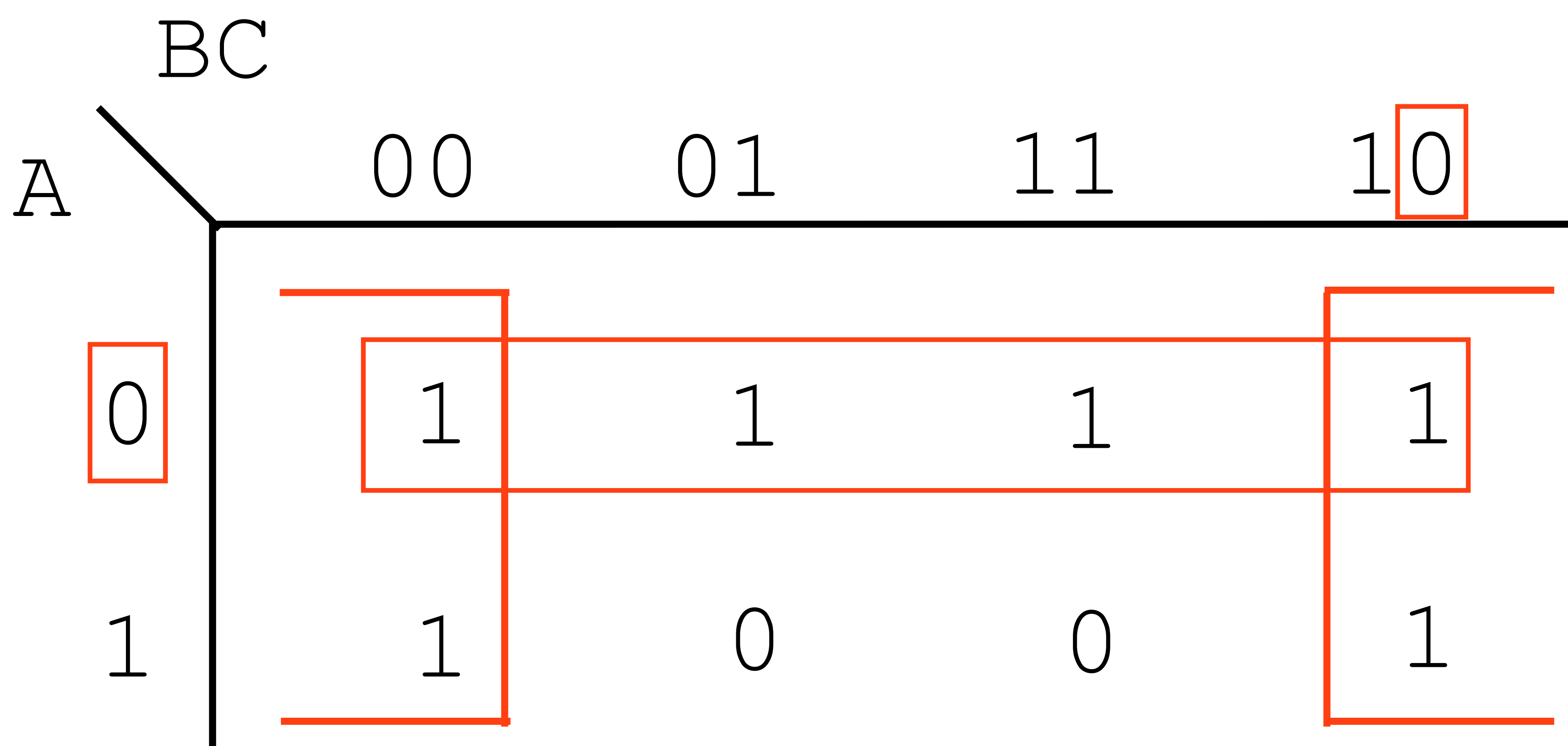
A	B	C	R
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



- Save the ones that stay the same in a group, discarding the rest
- This must be done for each group

R = !A!B!C + !A!BC + !ABC + !AB!C + A!B!C + AB!C

A	B	C	R
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



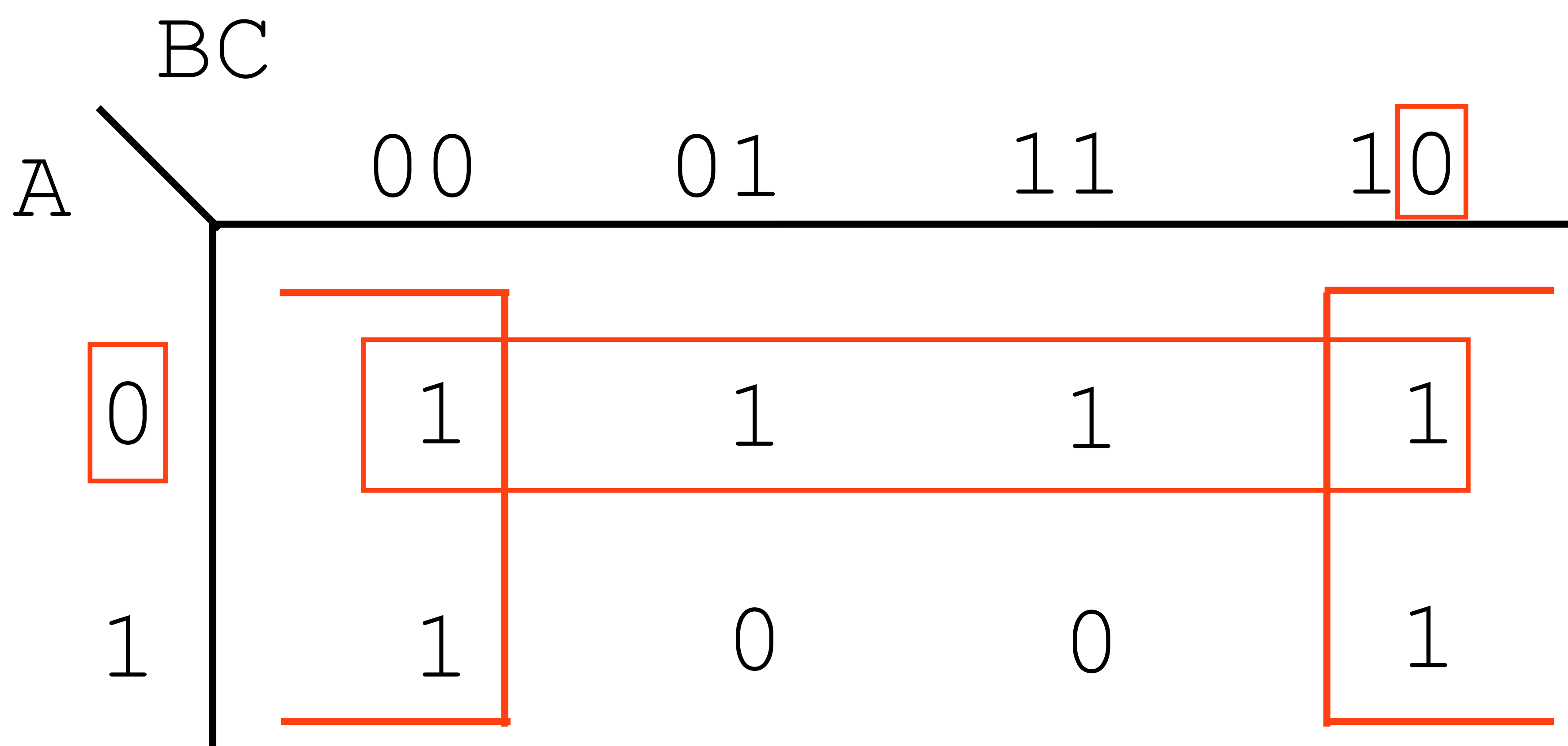
-Save the ones that stay the same in a group, discarding the rest
-This must be done for each group

$$R = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + AB\overline{C}$$

R = !A!B!C + !A!BC + !ABC +
!AB!C + A!B!C + AB!C

A	B	C	R
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

R =!A + !C



-Save the ones that stay the same in a group, discarding the rest
-This must be done for each group

Four Variable Example

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

-Take this formula

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

–For space reasons, we go directly to the K-map

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

- -Group things up
- -The edges logically wrap around!
- -Groups may overlap each other

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

R =!B!C

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

- Look at the bits that don’t change
- First for the cube

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

R =!B!C + !B!D

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

-Look at the bits that don't change
-Second for the cube on the edges

R = !A!B!C!D + !A!B!CD + !A!BC!D + !ABC!D + A!B!C!D + A!B!CD + A!BC!D

R =!B!C + !B!D + !AC!D

AB \ CD	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

-Look at the bits that don’t change
-Third for the line

K-Map Rules in Summary (I)

- Groups can contain only `1`s
- Only `1`s in adjacent groups are allowed (no diagonals)
- The number of `1`s in a group must be a power of two (1, 2, 4, 8...)
- The groups must be as large as legally possible

K-Map Rules in Summary (2)

- All `1`s must belong to a group, even if it's a group of one element
- Overlapping groups are permitted
- Wrapping around the map is permitted
- Use the fewest number of groups possible