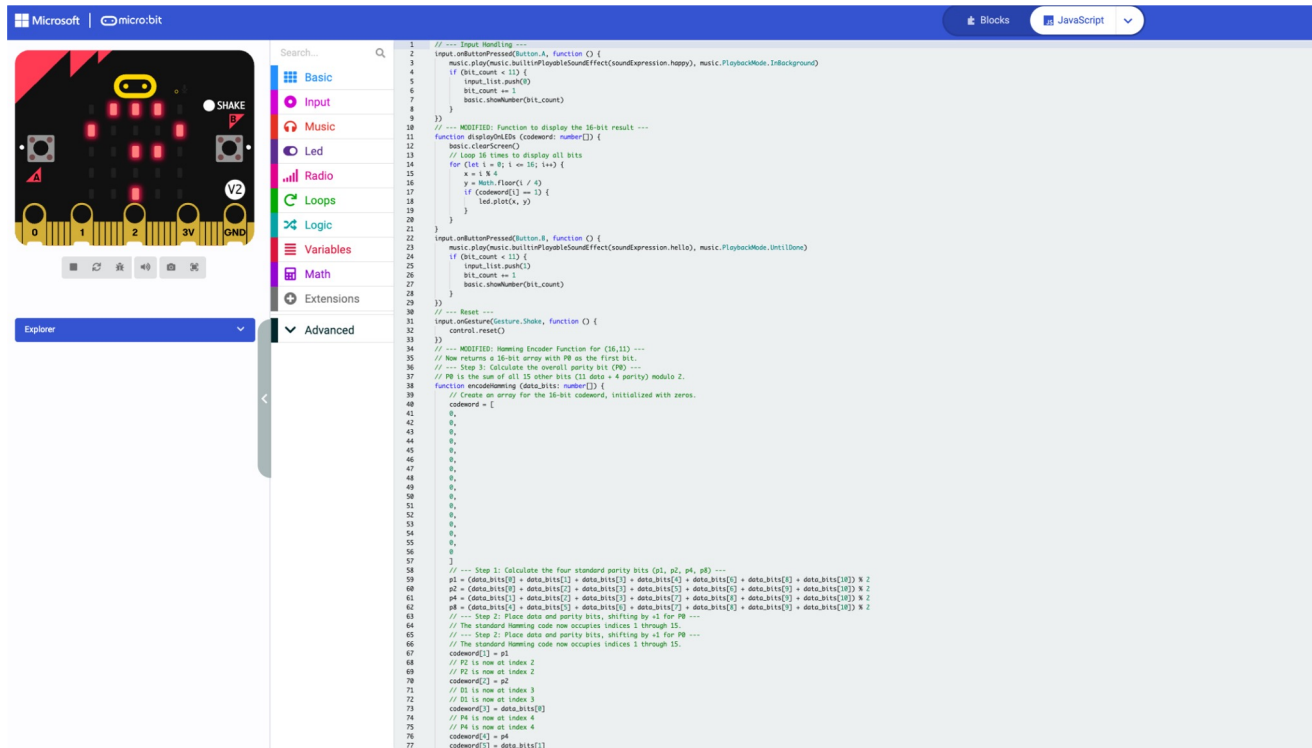


ENR-325/325L Principles of Digital Electronics and Laboratory

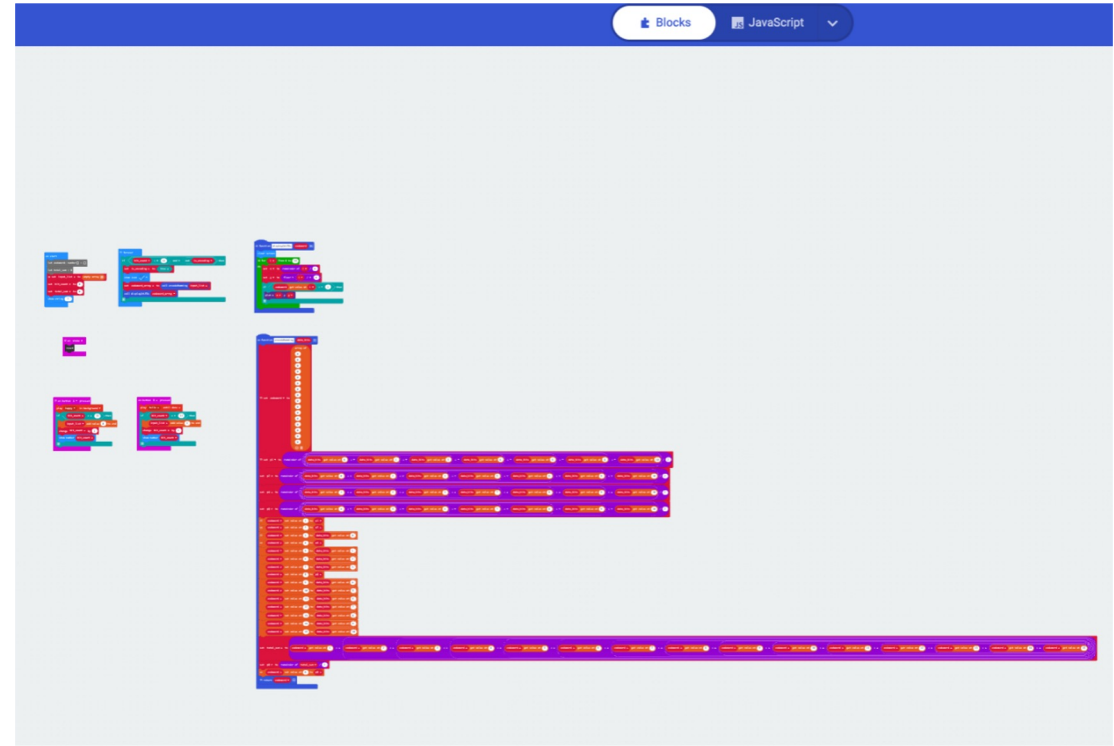
Xiang Li
Fall 2025

Hamming codes can be done in the CS way



The screenshot shows the Microsoft MakeCode editor interface for a micro:bit. On the left is a visual representation of the micro:bit board with pins labeled 0, 1, 2, 3V, and GND. Below it is an 'Explorer' panel. The main area displays JavaScript code for a Hamming encoder. The code includes functions for input handling, displaying LEDs, and encoding data into a 16-bit Hamming code. The encoding process involves calculating four parity bits (p1, p2, p3, p4) and then placing the data and parity bits into a 16-bit array.

```
1 // --- Input Handling ---
2 input.onButtonPressed(Button.A, function () {
3   music.playMusic(BufferPlayEffect(soundExpression.happy), music.PlaybackMode.InBackground)
4   if (bit_count < 12) {
5     input_list.push(0)
6     bit_count += 1
7     basic.showNumber(bit_count)
8   }
9 }
10
11 // --- MODIFIED: Function to display the 16-bit result ---
12 function displayLEDs (codeWord: number[]) {
13   basic.clearScreen()
14   // Loop 16 times to display all bits
15   for (let i = 0; i < 16; i++) {
16     x = i % 4
17     y = Math.floor(i / 4)
18     if (codeWord[i] == 1) {
19       led.plot(x, y)
20     }
21   }
22 }
23
24 input.onButtonPressed(Button.B, function () {
25   music.playMusic(BufferPlayEffect(soundExpression.hello), music.PlaybackMode.UntilDone)
26   if (bit_count < 12) {
27     input_list.push(0)
28     bit_count += 1
29     basic.showNumber(bit_count)
30   }
31 }
32
33 // --- Reset ---
34 input.onGesture(Gesture.Shake, function () {
35   control.reset()
36 })
37
38 // --- MODIFIED: Hamming Encoder Function for (16,12) ---
39 // Now returns a 16-bit array with P0 as the first bit.
40 // --- Step 3: Calculate the overall parity bit (P0) ---
41 // P0 is the sum of all 15 other bits (11 data + 4 parity) modulo 2.
42 function encodeHamming (data: number[]) {
43   // Create an array for the 16-bit codeWord, initialized with zeros.
44   codeWord = [
45     0,
46     0,
47     0,
48     0,
49     0,
50     0,
51     0,
52     0,
53     0,
54     0,
55     0,
56     0,
57     0,
58     0,
59     0
60   ]
61   // --- Step 1: Calculate the four standard parity bits (p1, p2, p3, p4) ---
62   p1 = (data_bits[0] + data_bits[1] + data_bits[2] + data_bits[3] + data_bits[4] + data_bits[5] + data_bits[6] + data_bits[7]) % 2
63   p2 = (data_bits[0] + data_bits[2] + data_bits[4] + data_bits[6] + data_bits[8] + data_bits[10] + data_bits[12] + data_bits[14]) % 2
64   p3 = (data_bits[1] + data_bits[3] + data_bits[5] + data_bits[7] + data_bits[9] + data_bits[11] + data_bits[13] + data_bits[15]) % 2
65   p4 = (data_bits[0] + data_bits[1] + data_bits[4] + data_bits[5] + data_bits[8] + data_bits[9] + data_bits[12] + data_bits[13]) % 2
66   // --- Step 2: Place data and parity bits, shifting by +1 for P0 ---
67   // The standard Hamming code now occupies indices 2 through 15.
68   codeWord[2] = p1
69   // P2 is now at index 2
70   codeWord[3] = p2
71   // P3 is now at index 3
72   codeWord[4] = p3
73   // P4 is now at index 4
74   codeWord[5] = p4
75   codeWord[6] = data_bits[0]
76   codeWord[7] = data_bits[1]
77 }
```



Hamming codes can be done in the EE way

- Before that, we need to acquire some basic skill sets.

Pre-step: Data forms

Step 1: Data manipulation

Step 2: Information storage

Step 3: Interface

Pre-step: Data forms

- Say bye-bye to base 10:

Base 10 (0,1,2,3,4,5,6,7,8,9):

$$(4321)_{10} = 4 \times 10^3 + 3 \times 10^2 + 2 \times 10^1 + 1 \times 10^0$$

Base 2 (0,1):

$$(1011)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

Base 16 (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F):

$$(FF12)_{16} = 15 \times 16^3 + 15 \times 16^2 + 1 \times 16^1 + 2 \times 16^0$$

Looking up how we do base conversions manually and in python.



COE COLLEGE®

The calculation of base 2 are pretty boring compared to base 10

Base 10

$$\begin{array}{r} 324 \\ +123 \\ \hline \end{array}$$

$$\begin{array}{r} 324 \\ -123 \\ \hline \end{array}$$

Base 2

$$\begin{array}{r} 110 \\ +101 \\ \hline \end{array}$$

$$\begin{array}{r} 110 \\ -101 \\ \hline \end{array}$$

Base 10

$$\begin{array}{r} 324 \\ \times 123 \\ \hline \end{array}$$

$$\begin{array}{r} 324 \\ \div 6 \\ \hline \end{array}$$

Base 2

$$\begin{array}{r} 110 \\ \times 101 \\ \hline \end{array}$$

$$\begin{array}{r} 110 \\ \div 10 \\ \hline \end{array}$$

- Your base 10 arithmetic skills can be translated to base 2 ones.
- We will revisit more binary arithmetic operation later, after the logic gates!



COE COLLEGE®

Discuss: the origin of base 16?

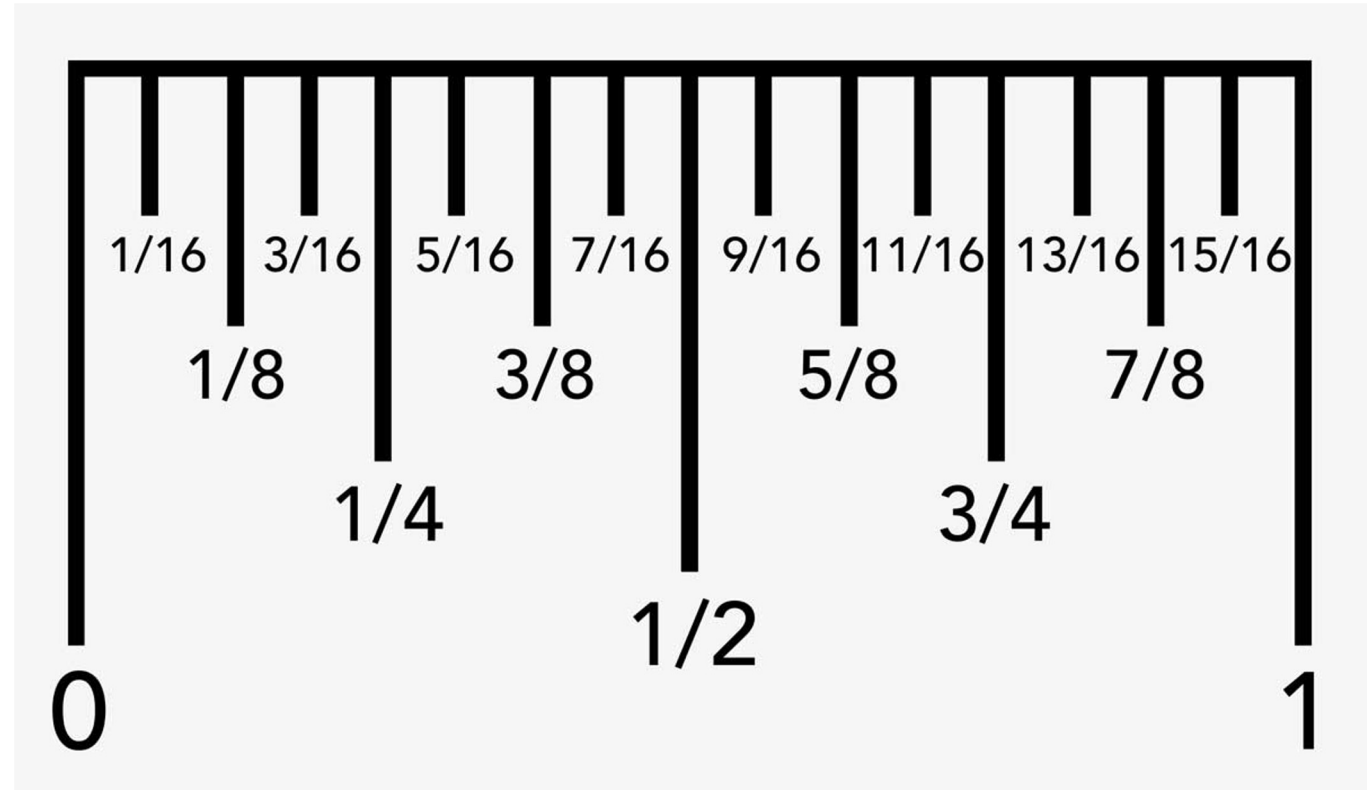
Discuss: the origin of base 16?

My theory:

An easy and fair way to compute with a weightless balance scale.



<https://commons.wikimedia.org/w/index.php?curid=79229218>



<https://www.inchcalculator.com/how-to-read-a-ruler/>

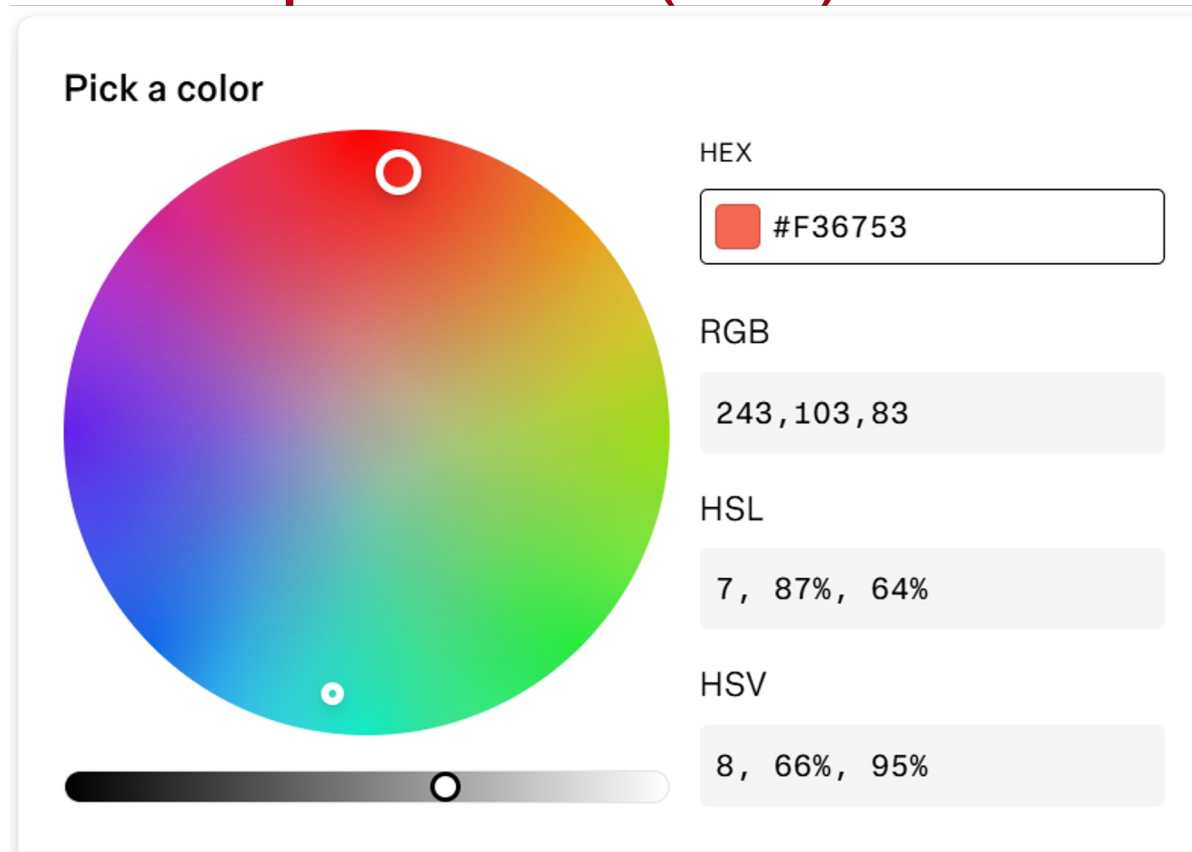
Discuss: why CS loves Hex(decimal) coding

0b : 0011100100101111010

0x : 392FA

Example: why CS loves Hex(decimal) coding

- Example: RGB (8bit) color code or Hex code



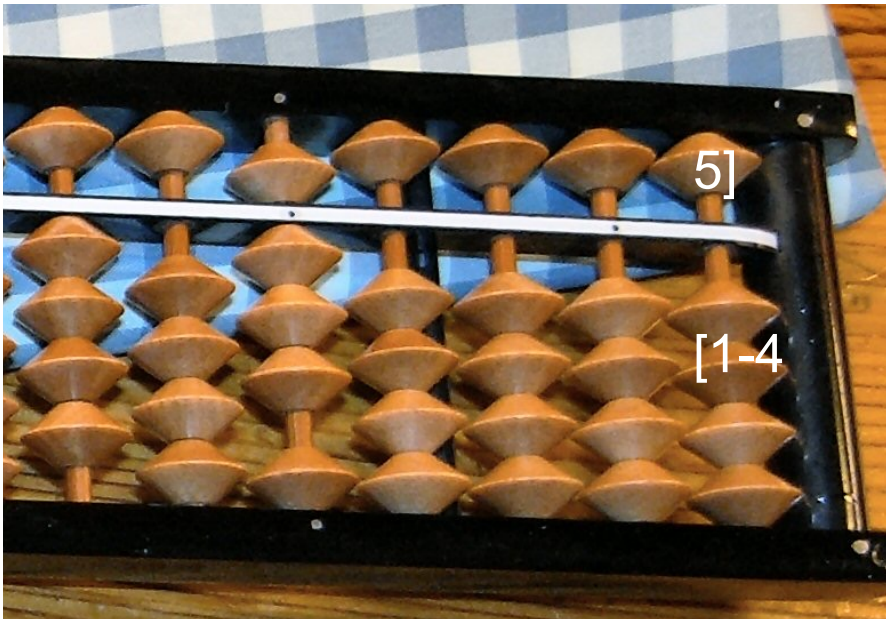
#F36753

R:
Bin(243)=11110011

G:
Bin(103)=01100111

B: Bin(83)=01010011

Before logic gates: why abacus, again?



https://upload.wikimedia.org/wikipedia/commons/9/98/Soroban_%28Abacus%29.JPG

Or presented in this way $[0,1]$

These bits are for counting “not 5”.

These bits are for counting “5”.

- This is forcing more states in a bit.
- Or due to the polarity of $[0,1]$, it is a state vector.
- State vector is a useful tool for cutting-edge computing.

Step 1-2: store data and move data around

- The basic functional unit for digital electronics: gate

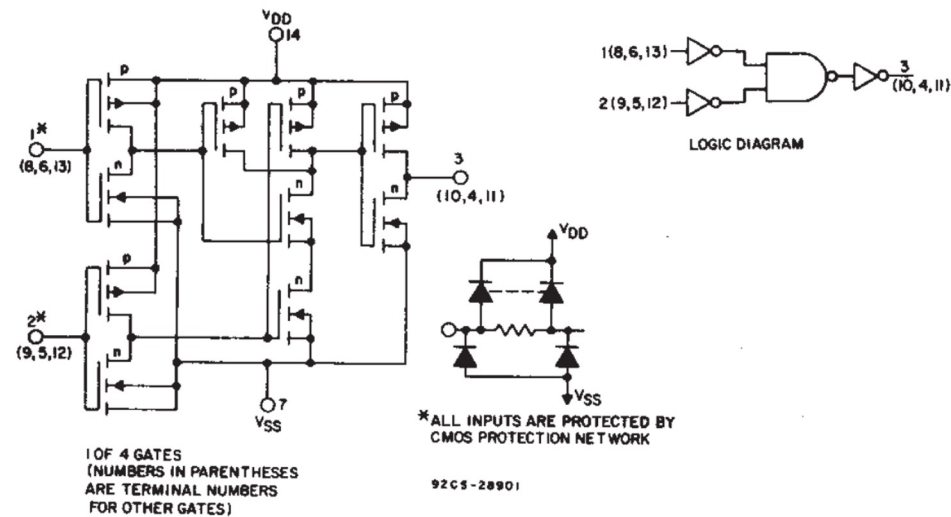
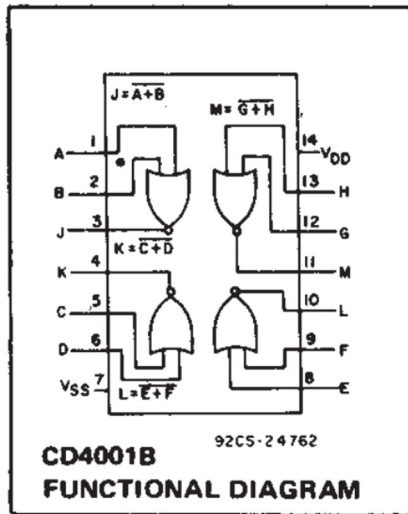
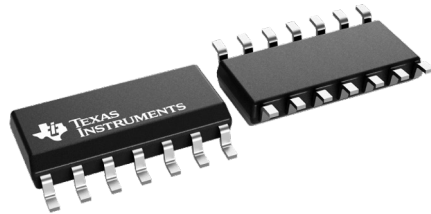
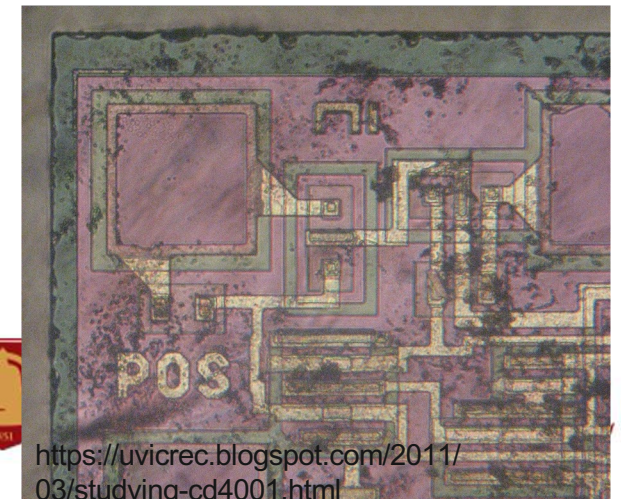
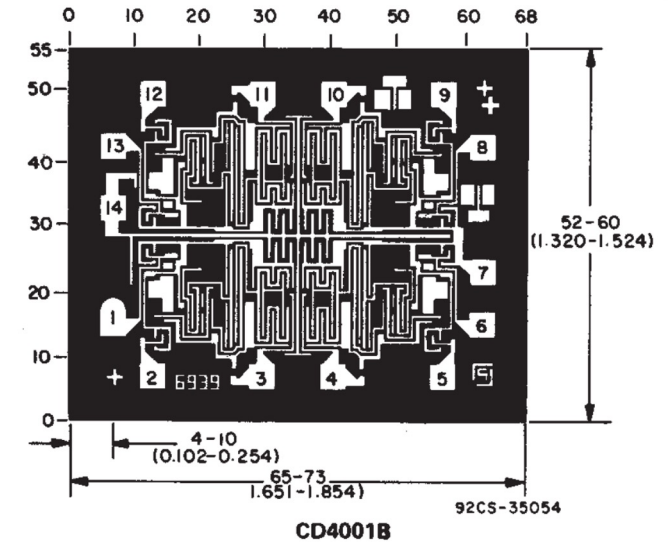


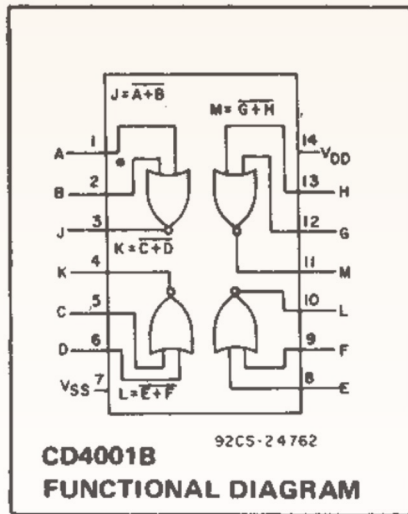
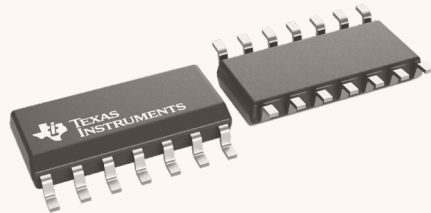
Fig.5 – Schematic and logic diagrams for CD4001B.



<https://uvicrec.blogspot.com/2011/03/studying-cd4001.html>

BTW: the multi-staged abstraction:

Application



EE

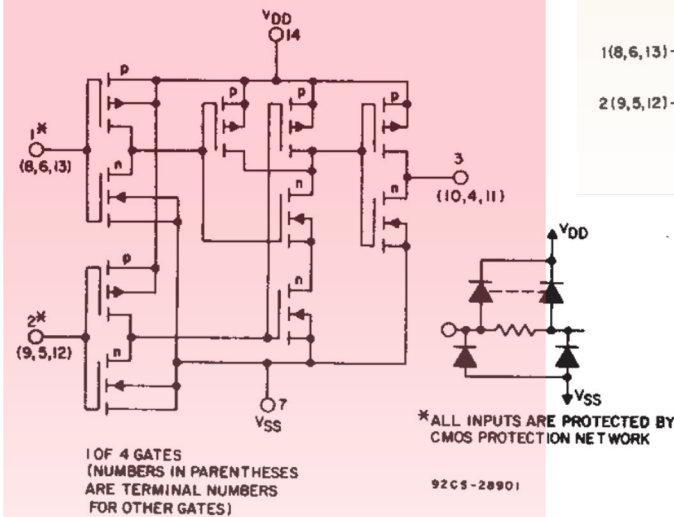
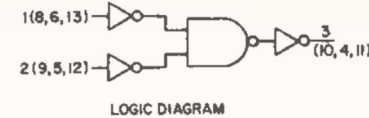
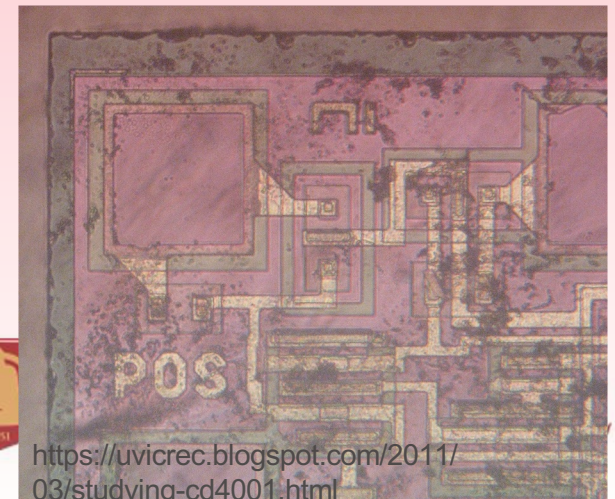
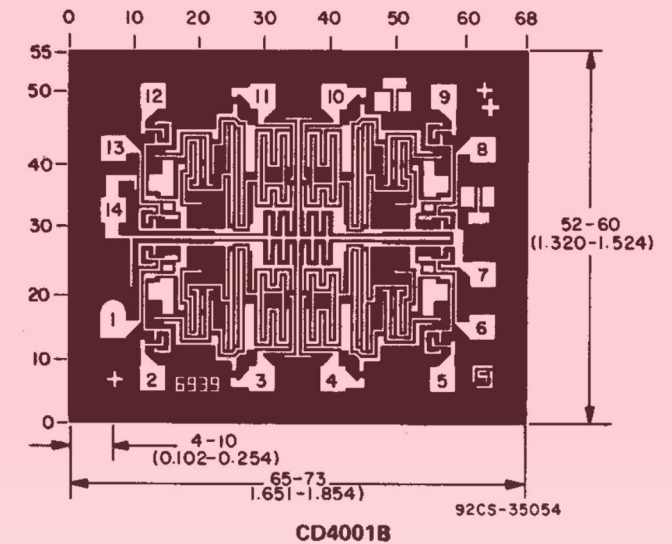


Fig.5 – Schematic and logic diagrams for CD4001B.

CE



MicroE



<https://uvicrec.blogspot.com/2011/03/studying-cd4001.html>

Step 1: data manipulation

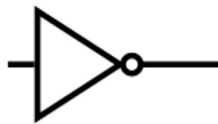
- Boolean operations 1st gen: basic True or False algebra

AND



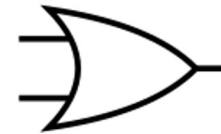
| AND gate truth table | | |
|----------------------|---|---------|
| Input | | Output |
| A | B | A AND B |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

NOT



| Inverter truth table | |
|----------------------|--------|
| Input | Output |
| A | NOT A |
| 0 | 1 |
| 1 | 0 |

OR



| OR gate truth table | | |
|---------------------|---|--------|
| Input | | Output |
| A | B | A OR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Boolean arithmetic symbols are messy

Viable symbols

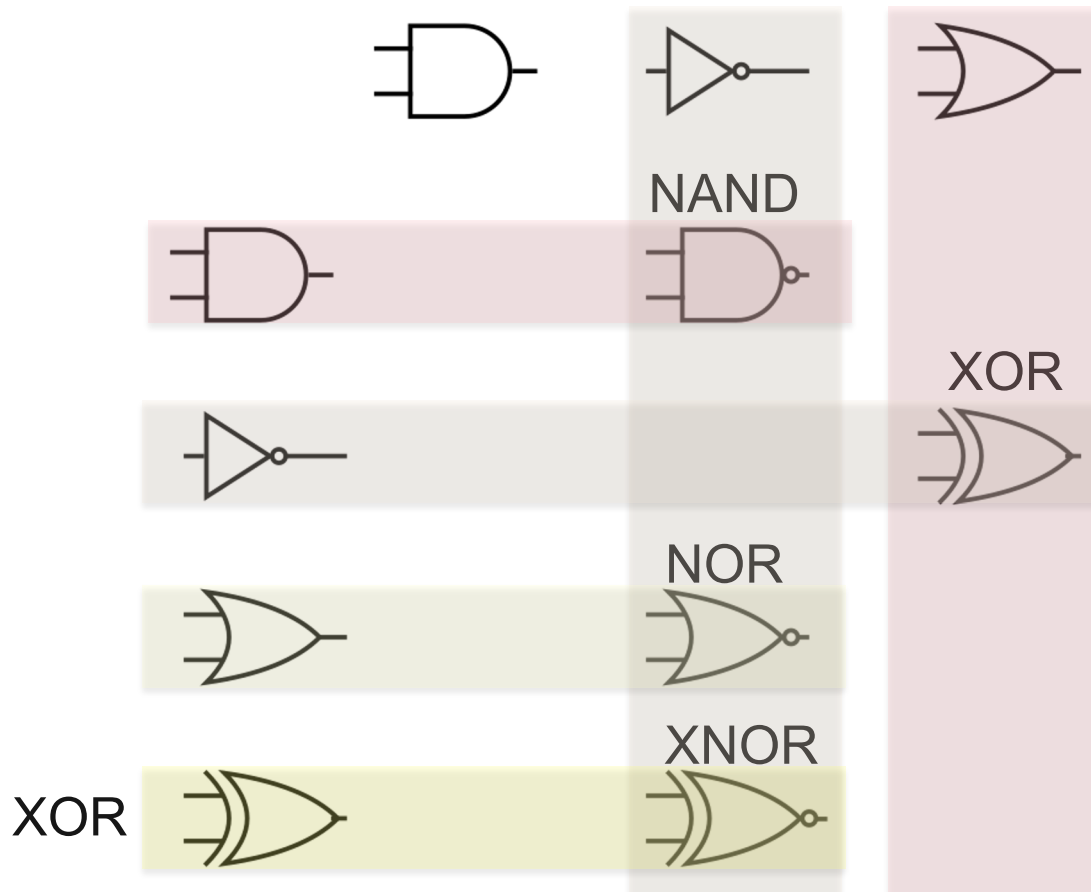
| | |
|-----|-----------|
| AND | • * × & ∧ |
| NOT | ! - _ ' ¬ |
| OR | + ∨ |



COE COLLEGE®

Step 1: data manipulation

- Boolean operations 2nd gen:
“logical logic operation”



“reverse AND”
NAND gate truth table

| Input | | Output |
|-------|---|----------|
| A | B | A NAND B |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

“reverse OR”
NOR gate truth table

| Input | | Output |
|-------|---|---------|
| A | B | A NOR B |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

“exclusive OR”
XOR gate truth table

| Input | | Output |
|-------|---|---------|
| A | B | A XOR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

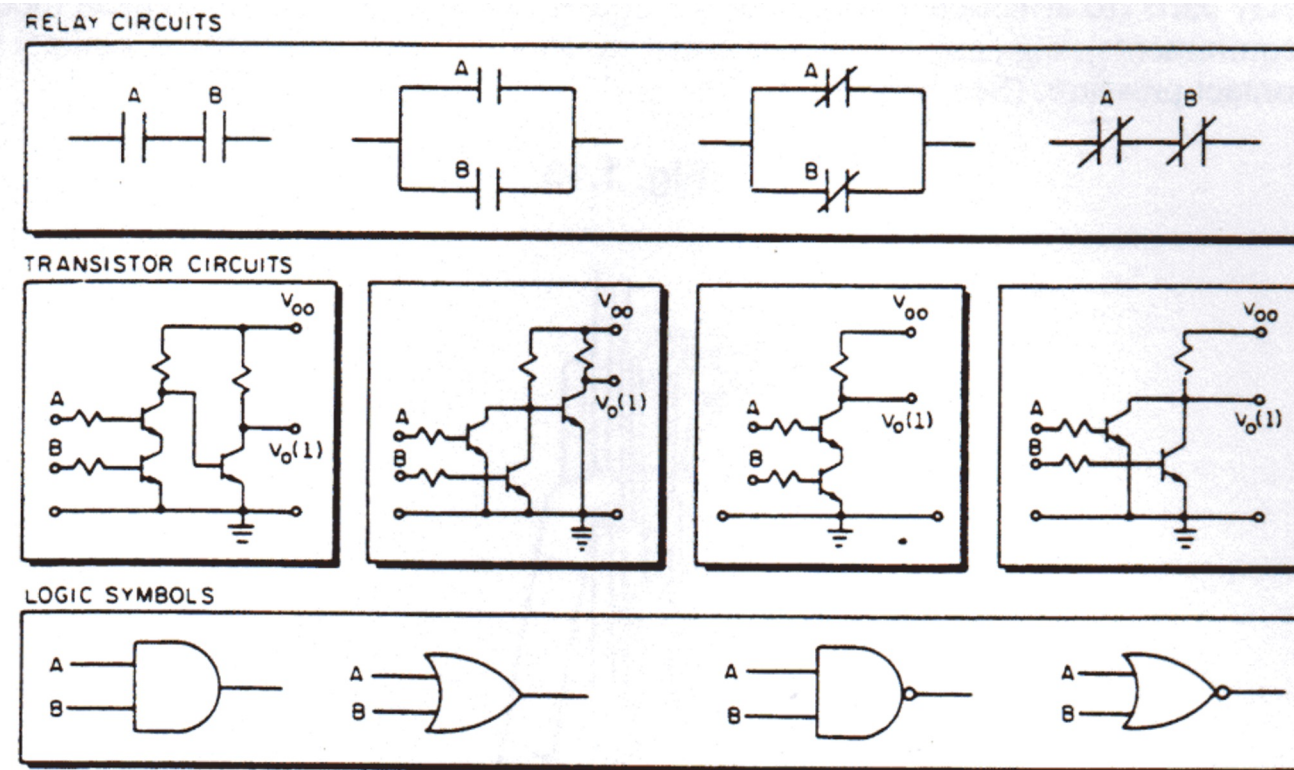
“reverse XOR”
XNOR gate truth table

| Input | | Output |
|-------|---|----------|
| A | B | A XNOR B |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



COE COLLEGE

Logic operation in the early days

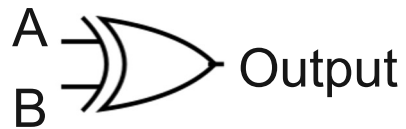


Transistor–transistor logic (TTL)
built with BJT

Fig. 1.12 Symbols used in cam-operated timer control.

Step 1: data manipulation

- A great example of logic gate functions:



XOR gate truth table

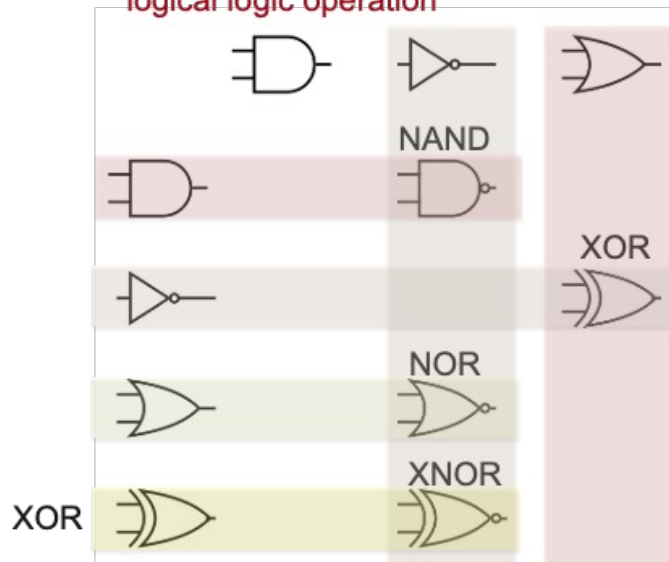
| Input | | Output |
|-------|---|---------|
| A | B | A XOR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Parity check?

“Universal Gate”

- We can use AND, OR, and NOT to build any gates:

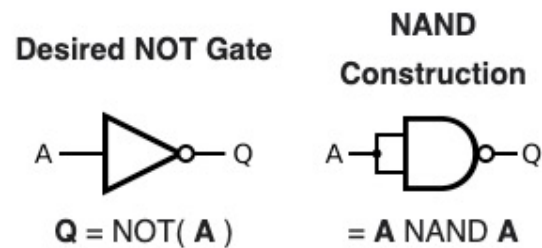
- Boolean operations 2nd gen:
“logical logic operation”



- We can build any gates with NAND gate:

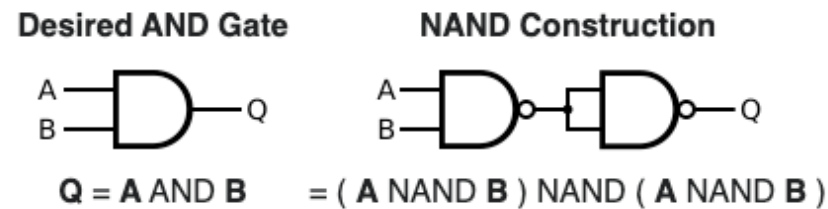
“Universal Gate”

- We can build any gates with NAND gate:



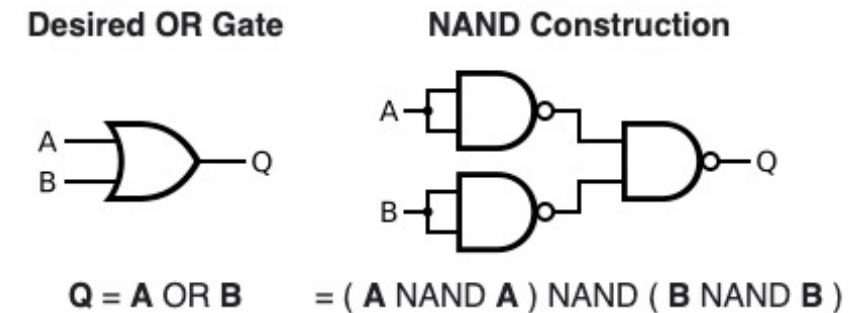
Truth Table

| Input A | Output Q |
|---------|----------|
| 0 | 1 |
| 1 | 0 |



Truth Table

| Input A | Input B | Output Q |
|---------|---------|----------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



Truth Table

| Input A | Input B | Output Q |
|---------|---------|----------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Step 1: data manipulation

- One can dig deeper into the Boolean operations:

Boolean expressions of the 16 functions between two variables.

| Function Name | Function description | Boolean Expression |
|---------------|----------------------|--------------------|
| Null | FALSE (0) | 0 |
| AND | AND | $A \cdot B$ |
| Inhibition | A NOT B | A/B |
| Transfer | A | A |
| Inhibition | B NOT A | B/A |
| Transfer | B | B |
| Exclusive-OR | XOR | $A \oplus B$ |
| OR | OR | $A + B$ |
| NOR | NOR | $A \downarrow B$ |
| XNOR | XNOR | $A \odot B$ |
| Complement | NOT B | B' |
| Implication | A OR NOT B | $A + B'$ |
| Complement | NOT A | A' |
| Implication | NOT A OR B | $A' + B$ |
| NAND | NAND | $A \uparrow B$ |
| Identity | TRUE (1) | 1 |

Fundamental Theorems & Postulates of Boolean Algebra

| | | |
|-------------------------------|--------------------------------------------------------------------|-----------------------------------------------------------------------|
| Identities: | (1) $X + 0 = X$ | (1D) $X \cdot 1 = X$ |
| Null Elements: | (2) $X + 1 = 1$ | (2D) $X \cdot 0 = 0$ |
| Idempotency: | (3) $X + X = X$ | (3D) $X \cdot X = X$ |
| Involution (Double Negation): | (4) $(X')' = X$ | |
| Complements: | (5) $X + X' = 1$ | (5D) $X \cdot X' = 0$ |
| Commutativity: | (6) $X + Y = Y + X$ | (6D) $X \cdot Y = Y \cdot X$ |
| Associativity: | (7) $(X + Y) + Z = X + (Y + Z)$ | (7D) $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$ |
| Distributivity: | (8) $X \cdot Y + X \cdot Z = X \cdot (Y + Z)$ | (8D) $(X + Y) \cdot (X + Z) = X + Y \cdot Z$ |
| Combining: | (9) $X \cdot Y + X \cdot Y' = X$ | (9D) $(X + Y) \cdot (X + Y') = X$ |
| Covering: | (10) $X + X \cdot Y = X$ | (10D) $X \cdot (X + Y) = X$ |
| DeMorgan's Laws: | (12) $(X \cdot Y \cdot Z)' = X' + Y' + Z'$ | (12D) $(X + Y + Z)' = X' \cdot Y' \cdot Z'$ |
| Consensus: | (17) $X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$ | (17D) $(X + Y) \cdot (X' + Z) \cdot (Y + Z) = (X + Y) \cdot (X' + Z)$ |
| Shannon Expansion: | (18) $F(X, Y, Z) = X \cdot F(1, Y, Z) + X' \cdot F(0, Y, Z)$ | (18D) $F(X, Y, Z) = (X + F(0, Y, Z)) \cdot (X' + F(1, Y, Z))$ |

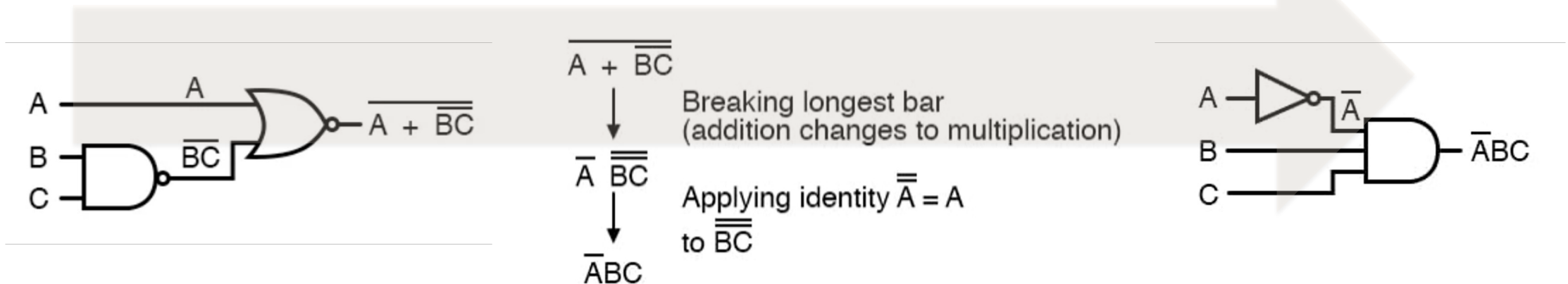
CS211, Rutgers 2013 notes

Step 1: data manipulation

But we will only touch base on two:

- De Morgan's laws (1/2)

$$\begin{array}{l} \text{not (A OR B)} = (\text{not A}) \text{ AND } (\text{not B}) \\ \text{not (A AND B)} = (\text{not A}) \text{ OR } (\text{not B}) \end{array} \quad \left| \begin{array}{l} \overline{(A \cdot B)} \equiv (\bar{A} + \bar{B}) \\ \overline{(A + B)} \equiv (\bar{A} \cdot \bar{B}) \end{array} \right|$$

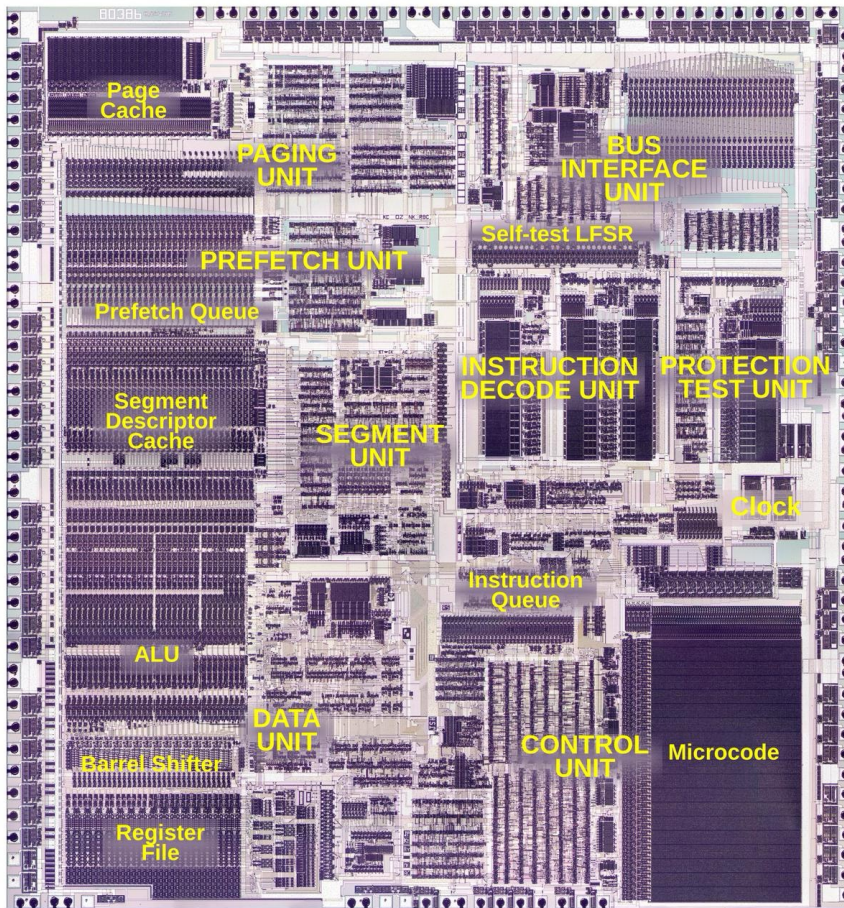


Example from: <https://www.allaboutcircuits.com/textbook/digital/chpt-7/demorgans-theorems/>

De Morgan's laws is a way to **mathematically** simplified a circuit, but not always realistic for IC.

This is the true logic simplification for IC

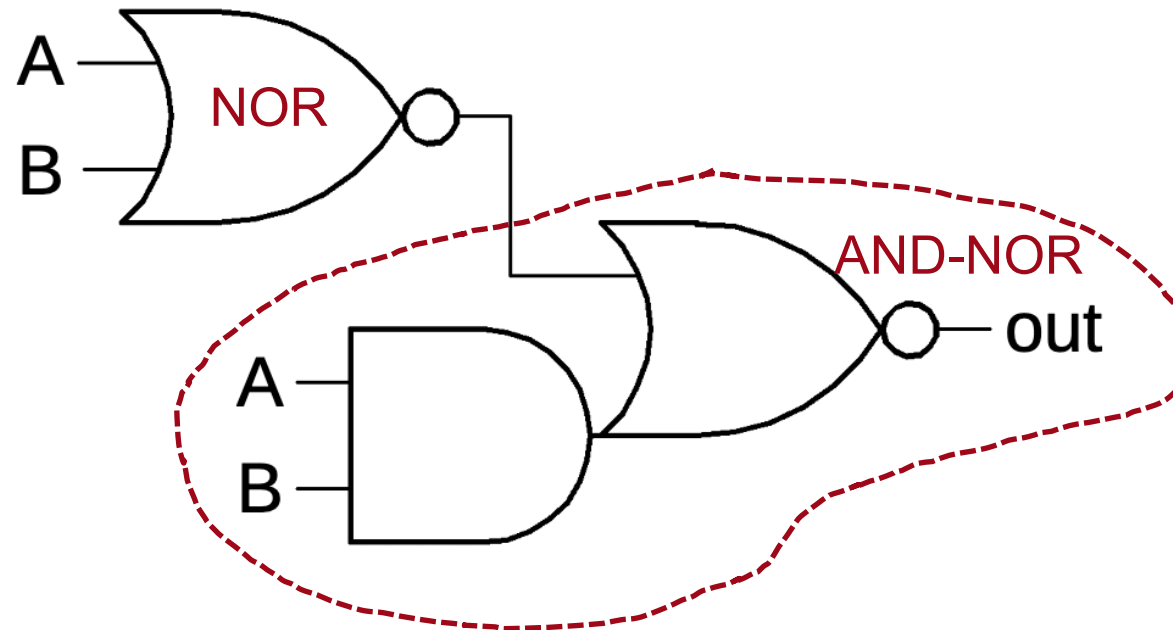
Remember XOR?



This Intel's 386 processor (1985) sure has a lot of it.

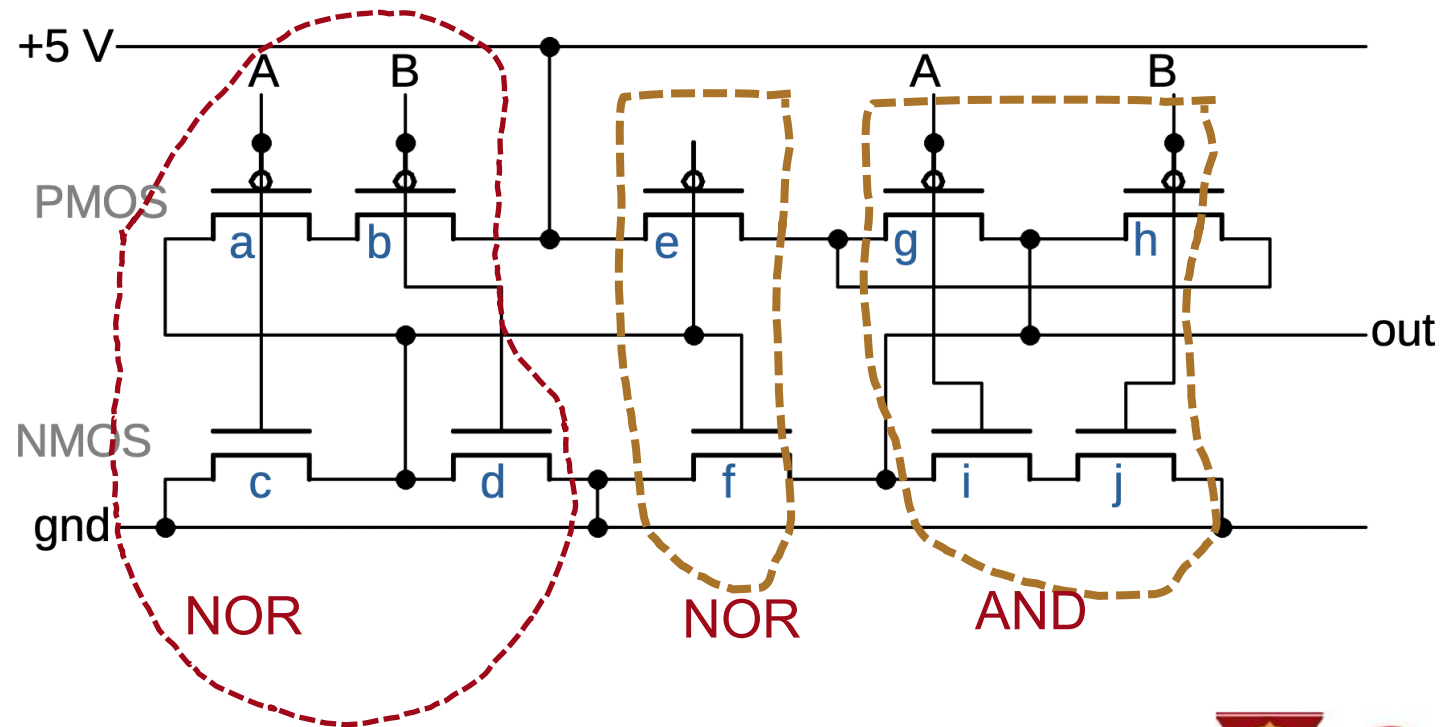
This is the true logic simplification for IC

This is the logic gate it used to generate XOR:

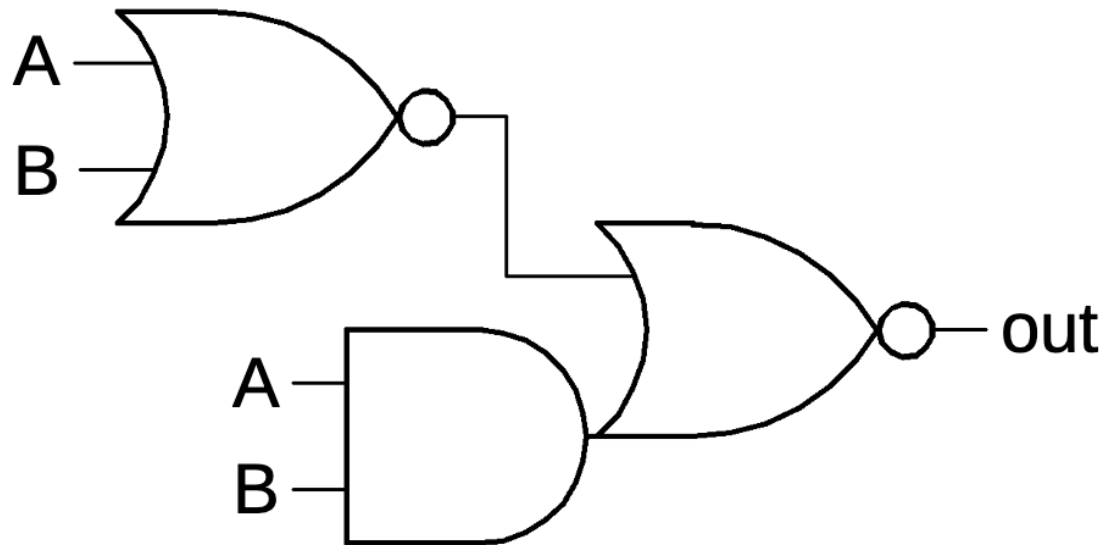


This is the true logic simplification for IC

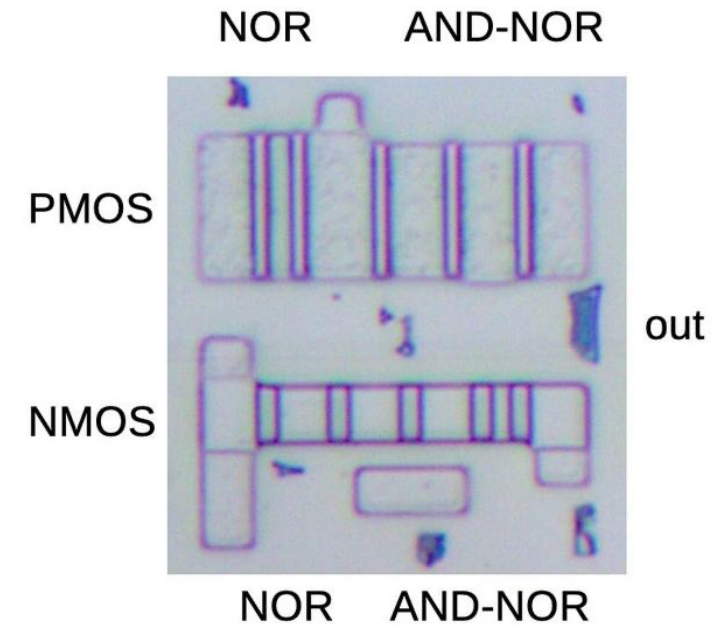
This is the transistor layout it used to generate XOR:



This is the true logic simplification for IC



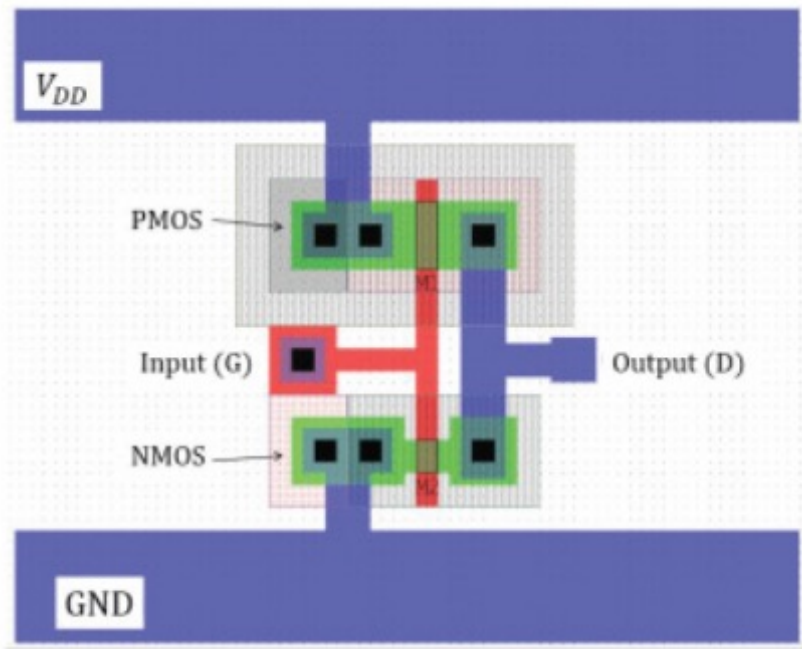
This is the actual XOR gate on IC



This is the true logic simplification for IC

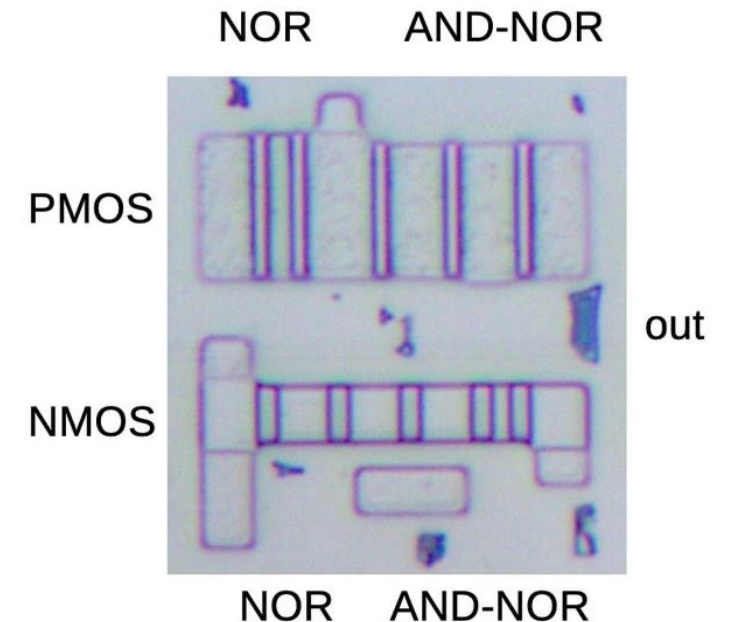
P well and N well design wise, not much differ from this invertor design:

CMOS invertor including electrodes



DOI: 10.1145/2755563

This is the actual XOR gate on IC

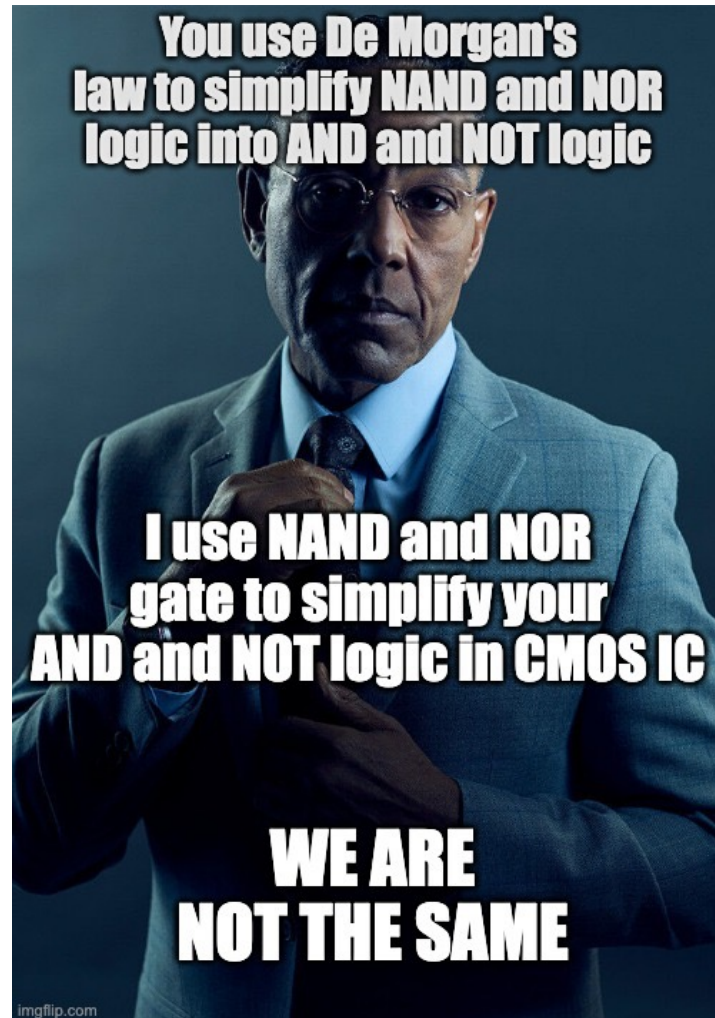


<https://www.righto.com/2023/12/386-xor-circuits.html>

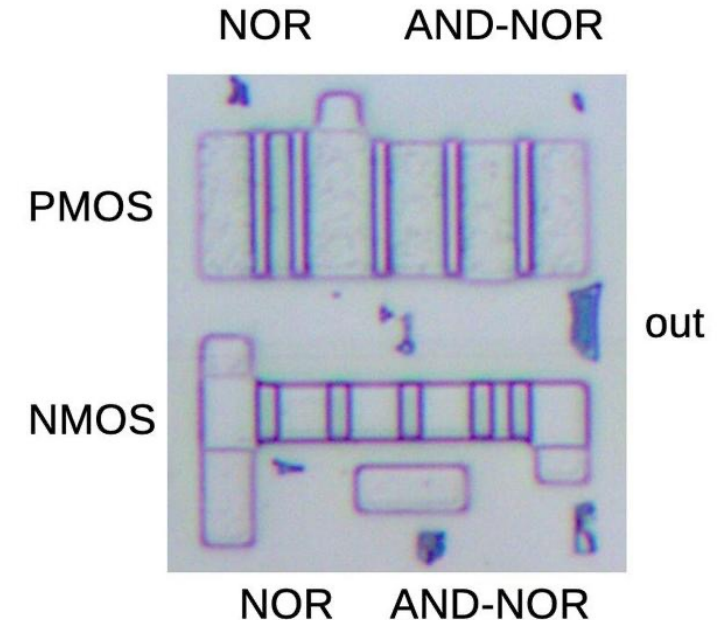


COE COLLEGE®

This is the true logic simplification for IC



This is the actual XOR gate on IC



<https://www.righto.com/2023/12/386-xor-circuits.html>



COE COLLEGE®

Step 1: data manipulation

But we will only pouch base on two:

- Karnaugh map (K-map) (2/2)

| NAND gate truth table | | |
|-----------------------|---|----------|
| Input | | Output |
| A | B | A NAND B |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

K-map of NAND gate

| B \ A | 0 | 1 |
|-------|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 0 |

K-map grouping rules

- 1.No zeros allowed.
- 2.No diagonals.
- 3.Only power of 2 number of cells in each group.
- 4.Groups should be as large as possible.
- 5.Every one must be in at least one group.
- 6.Overlapping allowed.
- 7.Wrap around allowed.
- 8.Fewest number of groups possible.

Step 1: data manipulation

- Karnaugh map (K-map) (2/2), continued

K-map grouping rules

- 1.No zeros allowed.
- 2.No diagonals.
- 3.Only power of 2 number of cells in each group.
- 4.Groups should be as large as possible.
- 5.Every one must be in at least one group.
- 6.Overlapping allowed.
- 7.Wrap around allowed.
- 8.Fewest number of groups possible.

$$(A \text{ AND } B) \text{ OR } (\text{NOT } C \text{ AND } D)$$
$$(A \times B) + (!C \times D)$$

| AB | | 00 | 01 | 11 | 10 |
|-------|----|----|----|----|----|
| CD | | | | | |
| !C×!D | 00 | 0 | 0 | 1 | 0 |
| | 01 | 1 | 1 | 1 | 1 |
| C×D | 11 | 0 | 0 | 1 | 0 |
| C×!D | 10 | 0 | 0 | 1 | 0 |



Step 1: data manipulation

- Karnaugh map (K-map) (2/2), in class practice

K-map grouping rules

- 1.No zeros allowed.
- 2.No diagonals.
- 3.Only power of 2 number of cells in each group.**
- 4.Groups should be as large as possible.
- 5.Every one must be in at least one group.
- 6.Overlapping allowed.
- 7.Wrap around allowed.**
- 8.Fewest number of groups possible.

| AB CD | 00 | 01 | 11 | 10 |
|----------|----|----|----|----|
| 00 | 0 | 0 | 1 | 1 |
| 01 | 1 | 1 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 1 | 1 |

Step 1: data manipulation

- Karnaugh map (K-map) (2/2), in class practice

K-map grouping rules

- 1.No zeros allowed.
- 2.No diagonals.
- 3.Only power of 2 number of cells in each group.**
- 4.Groups should be as large as possible.
- 5.Every one must be in at least one group.
- 6.Overlapping allowed.
- 7.Wrap around allowed.**
- 8.Fewest number of groups possible.

$$(!A \times !C \times D) + (A \times !D)$$

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



Step 1: data manipulation

- Karnaugh map (K-map) (2/2), 1 more in class practice

K-map grouping rules

- 1.No zeros allowed.
- 2.No diagonals.
- 3.Only power of 2 number of cells in each group.**
- 4.Groups should be as large as possible.
- 5.Every one must be in at least one group.
- 6.Overlapping allowed.**
- 7.Wrap around allowed.
- 8.Fewest number of groups possible.

| BC | 00 | 01 | 11 | 10 |
|----|----|----|----|----|
| A | | | | |
| 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

Step 1: data manipulation

- Karnaugh map (K-map) (2/2), 1 more in class practice

K-map grouping rules

- 1.No zeros allowed.
- 2.No diagonals.
- 3.Only power of 2 number of cells in each group.**
- 4.Groups should be as large as possible.
- 5.Every one must be in at least one group.
- 6.Overlapping allowed.**
- 7.Wrap around allowed.
- 8.Fewest number of groups possible.

$$(A \times !B) + (!A \times B) + C$$

| BC | 00 | 01 | 11 | 10 |
|----|----|----|----|----|
| A | | | | |
| 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

Step 1: data manipulation

- Other than K-map, the more mechanical way to turn truth map to Boolean equations:

Step 1

| A | B | C | |
|---|---|---|----------------------|
| 0 | 0 | 1 | $C_0 = !A \times !B$ |
| 0 | 1 | 1 | $C_1 = !A \times B$ |
| 1 | 0 | 1 | $C_2 = A \times !B$ |
| 1 | 1 | 0 | |

Step 2

$$C = C_0 + C_1 + C_2$$

Step 3

$$\begin{aligned}
 C &= !A \times !B + !A \times B + A \times !B \\
 &= !A(!B + B) + A \times !B \quad \text{Complements: } (5) \quad X + X' = 1 \\
 &= !A + A \times !B \\
 &= !A \times (1 + !B) + A \times !B \quad \text{Null Elements: } (2) \quad X + 1 = 1 \\
 &= !A + !A !B + A !B \\
 &= !A + (!A + A) \times !B \\
 &= !A + !B
 \end{aligned}$$

- This expression is also called **Sum of Products (SOP)**.
- There are also **Product of Sums**, for sure.