CISC260 Machine Organization and Assembly Language

Boolean logic, Gates, NAND universality

Any function can be implemented in Boolean logic.

- Function is a mapping from input variables **I** to output value **O**.

  F: I $\rightarrow$ O,

  where I$\in$ {0,1}^{N}, O $\in${0, 1}^{M}.

Inputs ABC	Output XY
000	00
001	01
010	01
011	10
100	01
101	10
110	10
111	11

- Boolean logic
  - Boolean variables
  - Boolean operators

    AND (&), OR (|), NOT(~)

Y = AND (A, B) = A & B

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Y = OR (A, B) = A | B

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Y = NOT(A) = ~A

A	Y
0	1
1	0

Boolean expressions: *made of Boolean variables and Boolean operators much like arithmetic expressions.*

Boolean variables: take values {0, 1} or {F, T}.

Boolean operators:

- (& : .)
- (| : + module 2)
- (~ : . (-1))

E.g., Y=A&B|C&~B

Precedence(high to low): ~, &, |

Meaning of a Boolean Expression

- Each such expression, interpreted as functional mapping, implies a truth table. Boolean expression $\Rightarrow$ truth table.
- *E.g.* Y=A&B|C&~B

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Canonical representation (truth table $\Rightarrow$ Boolean expression)

Every Boolean function (defined in a truth can be expressed using at least one Boolean expression called the canonical representation.

Procedure:

- And together all literals (negate if 0) in each row (conjunct)
- Or together rows that have true output
- Repeat for each output bit of the function.

The Boolean expression thus obtained is called the “sum-of-product” canonical form.

Conclusion: Every Boolean function, no matter how complex, can be expressed using three Boolean operators: AND, OR and NOT.

Boolean Logic Example

Conjuncts:

- ~A&~B&~C
- ~A&~B&C
- ~A&B&~C
- ~A&B&C
- A&~B&~C
- A&~B&C
- A&B&~C
- A&B&C

Inputs ABC	Output XY
000	01
001	00
010	00
011	10
100	10
101	00
110	11
111	00

Boolean Logic Example

Conjuncts:

- ~A&~B&~C
- ~A&~B&C
- ~A&B&~C
- ~A&B&C
- A&~B&~C
- A&~B&C
- A&B&~C
- A&B&C

Inputs ABC	Output XY
000	01
001	00
010	00
011	10
100	10
101	00
110	11
111	00

X=~A&B&C
| A&~B&~C
| A&B&~C

Boolean Logic Example

- Conjuncts:
  - ~A&~B&~C
  - ~A&~B&C
  - ~A&B&~C
  - ~A&B&C
  - A&~B&~C
  - A&~B&C
  - A&B&~C
  - A&B&C

Inputs ABC	Output XY
000	01
001	00
010	00
011	10
100	10
101	00
110	11
111	00

Y=~A&~B&~C
| A&B&~C

X=~A&B&C
| A&~B&~C
| A&B&~C

Boolean Logic Example

sum-of-product

Inputs ABC	Output XY
000	01
001	00
010	00
011	10
100	10
101	00
110	11
111	00

X=~A&B&C | A&~B&~C | A&B&~C

$= \bar{A}BC + A\bar{B}\bar{C} + AB\bar{C}$

Y=~A&~B&~C | A&B&~C

$= \bar{A}\bar{B}\bar{C} + AB\bar{C}$

Alternative Procedure:

OR together all literals (negate if true) in each row
AND together rows that have false output
Repeat for each output bit of the function.

The Boolean expression thus obtained has the so-called the “product-of-sum” form.

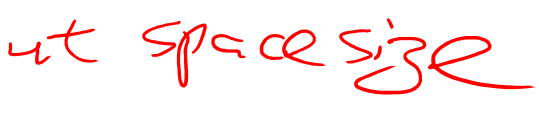
This procedure works for the same reason that the canonical procedure works.

Just look at the truth tables for AND, OR. They are “identical” in the sense that AND to 1 is just like OR to 0, vice versa. Namely, AND has value 1 iff all input values are 1, and OR has value 0 iff all input values are 0.

Therefore, this alternative procedure is to define the output from the zero’s perspective.

NB: equivalence of sum-of-product and product-of-sum can also be proved using DeMorgan’s Law. (though the logic could be the other way around, or circular.)

The number of Boolean functions that can be defined over n Boolean variables is 2^{2^n}

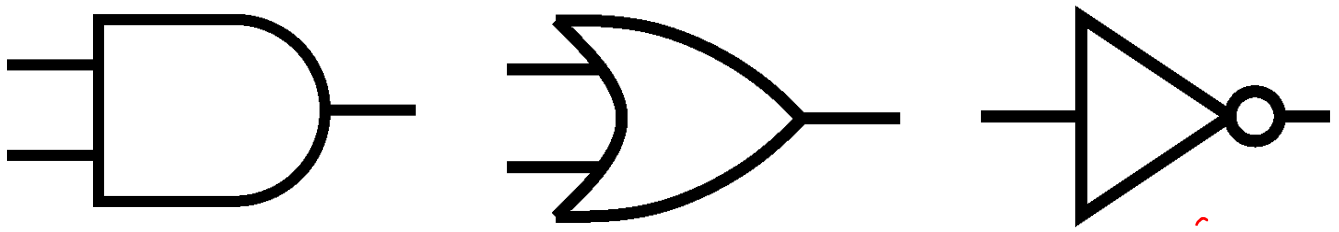


All Boolean functions of 2 variables

Function	x	0	0	1	1
	y	0	1	0	1
Constant 0	0	0	0	0	0
And	$x \cdot y$	0	0	0	1
x And Not y	$x \cdot \overline{y}$	0	0	1	0
x	x	0	0	1	1
Not x And y	$\overline{x} \cdot y$	0	1	0	0
y	y	0	1	0	1
Xor	$x \cdot \overline{y} + \overline{x} \cdot y$	0	1	1	0
Or	$x + y$	0	1	1	1
Nor	$\overline{x + y}$	1	0	0	0
Equivalence	$x \cdot y + \overline{x} \cdot \overline{y}$	1	0	0	1
Not y	$\overline{y}$	1	0	1	0
If y then x	$x + \overline{y}$	1	0	1	1
Not x	$\overline{x}$	1	1	0	0
If x then y	$\overline{x} + y$	1	1	0	1
Nand	$\overline{x \cdot y}$	1	1	1	0
Constant 1	1	1	1	1	1

Gates implement Boolean logic

- Physical, Boolean Operator
  - A physical entity that implements a small truth table
  - *E.g.* AND, OR, NOT



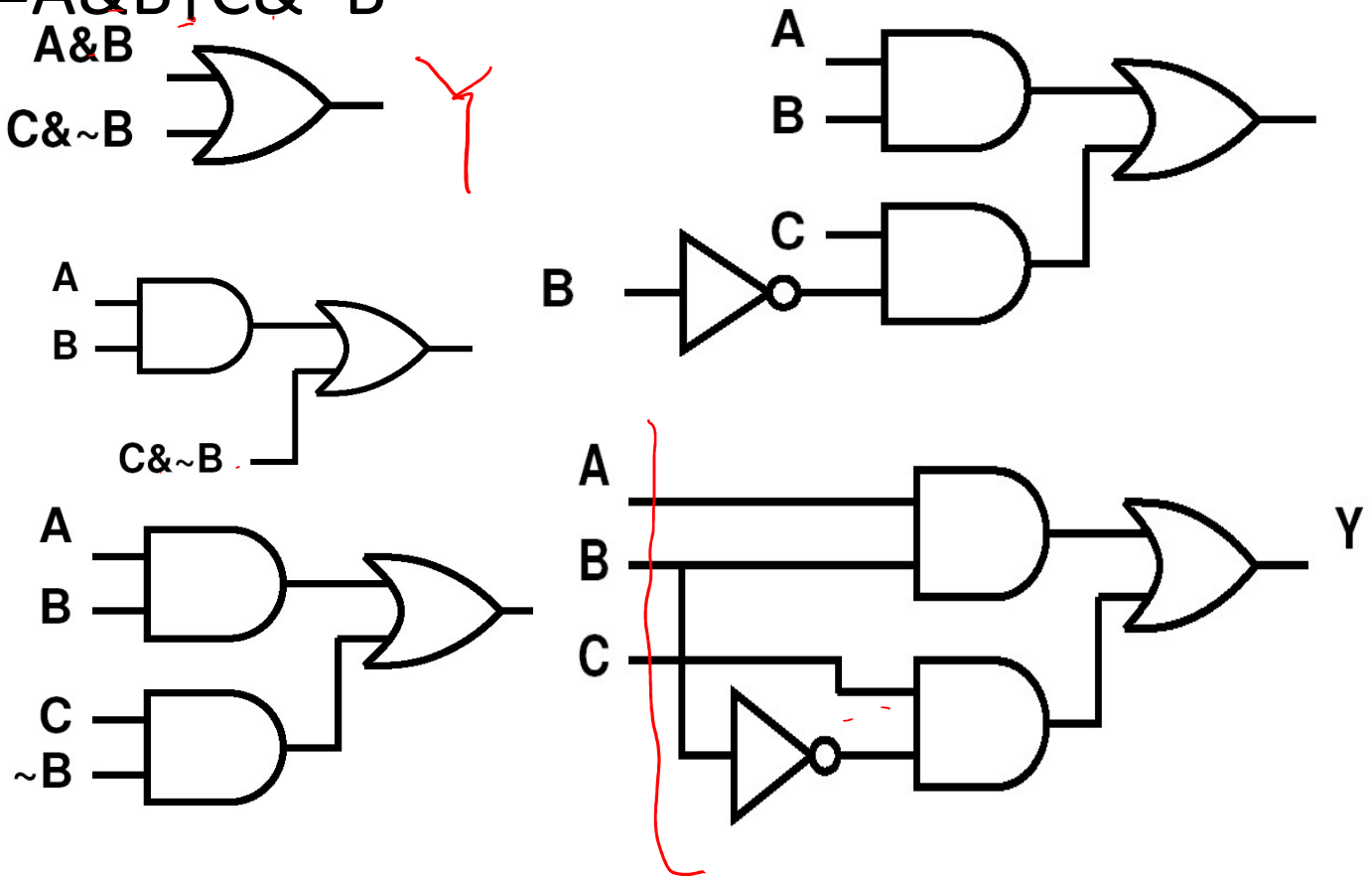
How to wire the gates according to a Boolean expression?

Any Boolean expression can be implemented with a collection of logical gates

1. Find outer-most logical operator
2. Replace with the corresponding gate
3. Work recursively on input functions

Example

- Y=A&B|C&~B



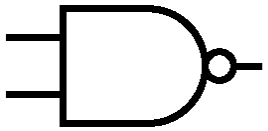
Compute Any Function in Gates

- Combining AND, OR, NOT: we can
  - Express any function with Boolean expressions
  - Implement any Boolean expression with gates

→ Can implement any function with gates

NAND gate universality

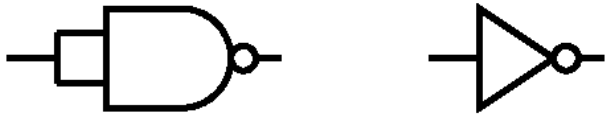
- NAND gate



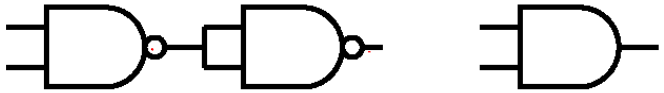
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

- Universality: Any Boolean expression can be implemented using NAND gates only.

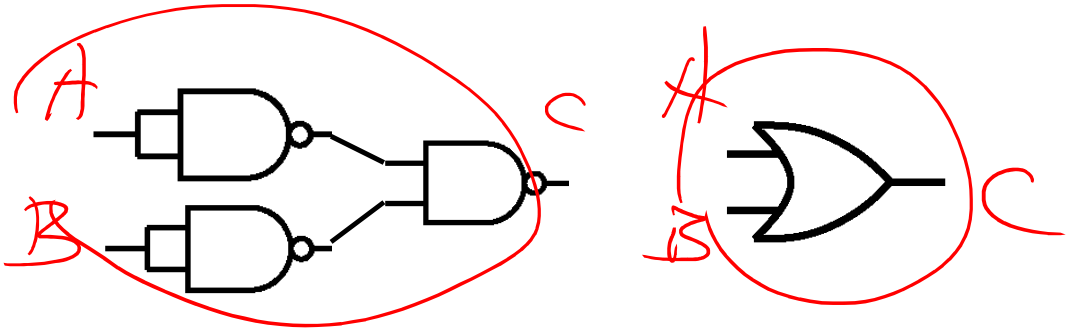
- NOT gate



- AND gate

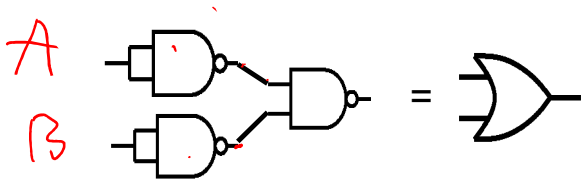


- OR gate



				(~C)|(~D)				~(C&D)
A	B	A&B	~(A&B)	A|B	~A	~B	~A&~B	~(~A & ~B)
0	0	0	1	0	1	1	1	0
0	1	0	1	1	1	0	0	1
1	0	0	1	1	0	1	0	1
1	1	1	0	1	0	0	0	1

From the truth table, we see A|B = ~ (~A & ~B), which means

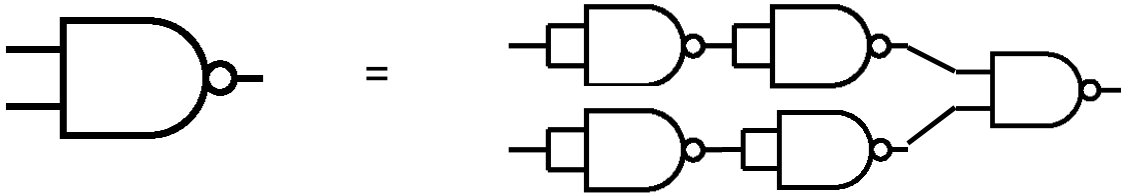


If we let ~A = C and ~B = D, then we have ~(C&D) = (~C)|(~D), which is DeMorgan's Law.

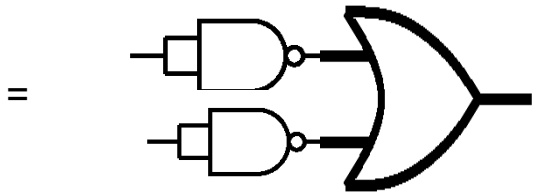
DeMorgan's Law: ~(A&B)=~A|~B

"Not A and B means neither A nor B"

Start with the left hand side, it is a NAND gate with inputs A, B.



Negate the input twice is still equal to the input itself. That is, ~~A = A. Recognize the OR gate implemented by 3 NAND gates.



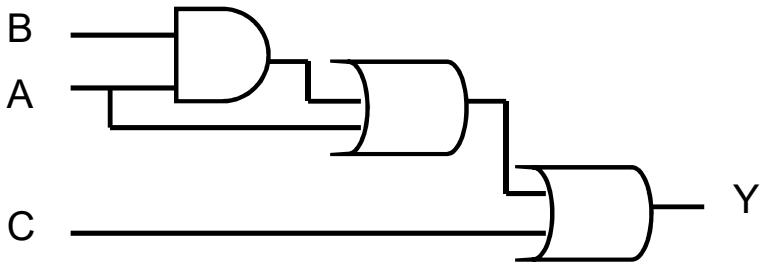
A | False = A

- Identity law: $A + 0 = A$ and $A \cdot 1 = A$.
- Zero and One laws: $A + 1 = 1$ and $A \cdot 0 = 0$.
- Inverse laws: $A + \overline{A} = 1$ and $A \cdot \overline{A} = 0$.
- Commutative laws: $A + B = B + A$ and $A \cdot B = B \cdot A$.
- Associative laws: $A + (B + C) = (A + B) + C$ and $A \cdot (B \cdot C) = (A \cdot B) \cdot C$.
- Distributive laws: $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ and $A + (B \cdot C) = (A + B) \cdot (A + C)$.

$\overline{x \cdot y} = \overline{x} + \overline{y}$ $\overline{(x + y)} = \overline{x} \cdot \overline{y}$ DeMorgan's Law

Simplifying logic

e.g.,



Y = A&B | A | C

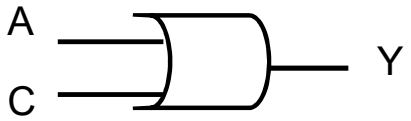
= AB+A +C

= A(B+1) +C distribution, identity

= A(1) + C law of 1's

= A + C identity

= A | C



- Logic optimization
  - Very difficulty: NP-complete
  - Commercial software

Summary

- Any function on binary input/output can be implemented in Boolean logic
- Boolean logic can be implemented by physical devices – gates.
- Logic gates, as an abstraction, hide the physical details of the devices.
- Only need a small number of primitive gates, actually, only a single gate type NAND2 is enough.