Summary

- Last week (review of CH8)
- This week and next week
  - Basic logic gates (Today)
  - Datapath modules (Dec 2nd)
  - Sequential elements & memories (Dec 4th)
  - Transistors (Dec 7th)
- Course wrap-up Dec 9th
- Discussion section Dec 11th
- 4th exam Dec 14th
- NO FINAL EXAM

Today

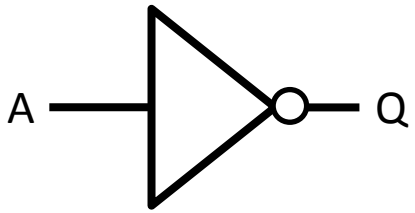
- Abstraction of logic gates
- Notation
- Simple logic gates
- Combining logic gates
- Creating our own circuits

Chapter summary

- Logic gates provide Boolean functionality
- Transistors made of gates, behave as switch controlled by one terminal

Basic Logic Gates

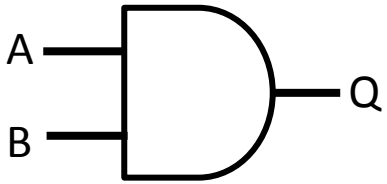
NOT GATE



A	Q
0	1
1	0

Text notation: $\sim$

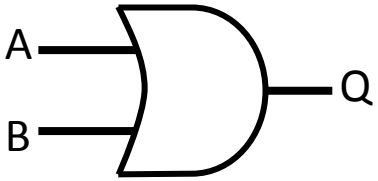
AND GATE



A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

Text notation: $\cdot$

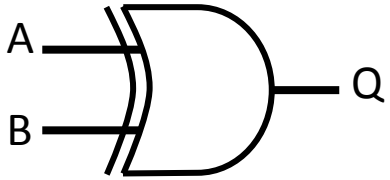
OR GATE



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

Text notation: $+$

XOR GATE

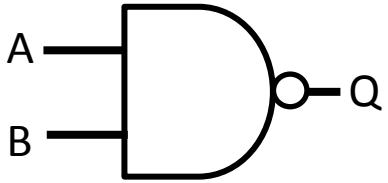


A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

Text notation: $\oplus$

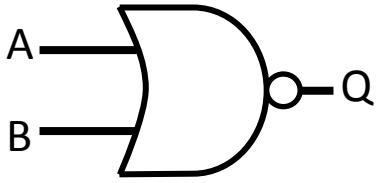
NAND, NOR, XNOR gate

NAND GATE



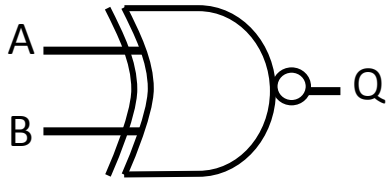
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

NOR GATE



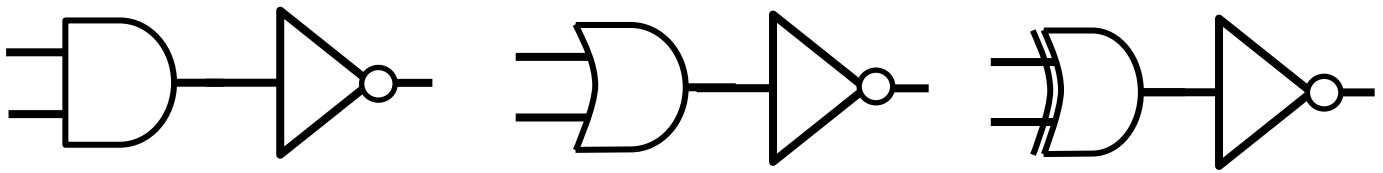
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

XNOR GATE



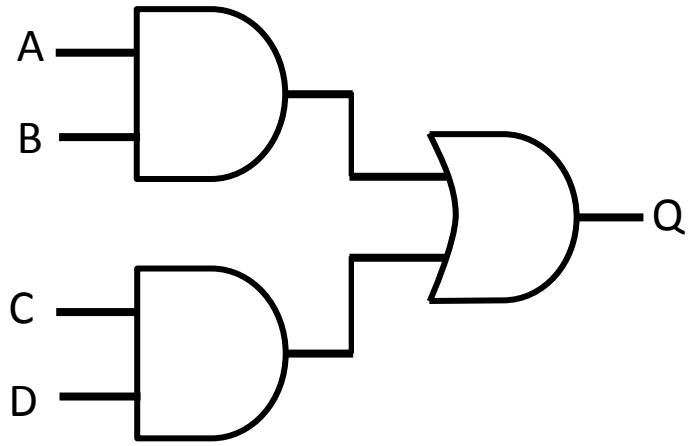
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

Using other gates

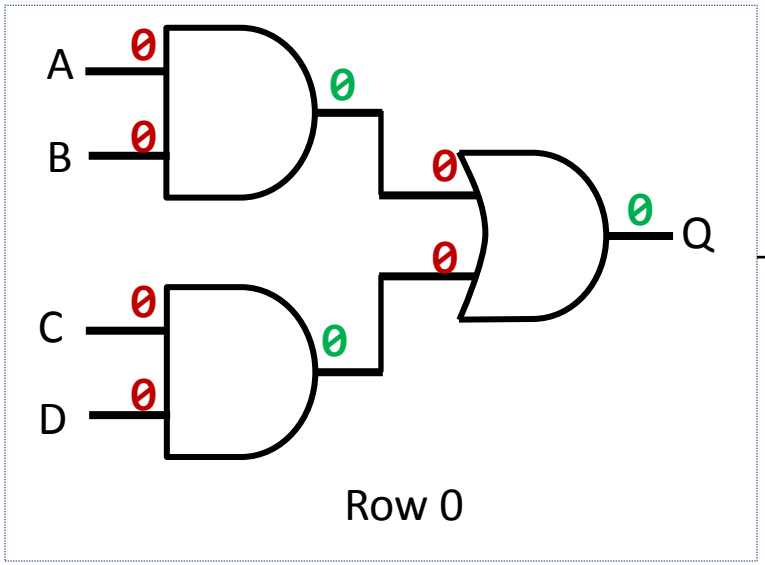


Combining logic gates

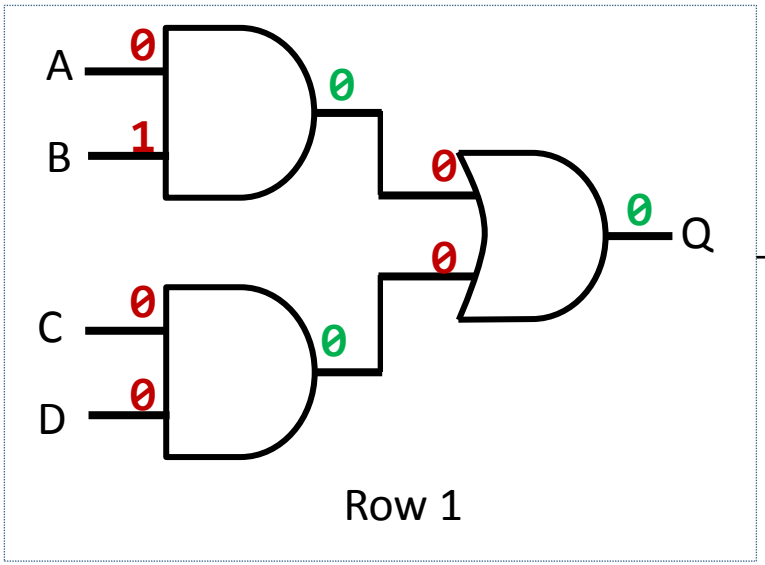
- Output of logic gate can be connected to any number of other inputs of other logic gates



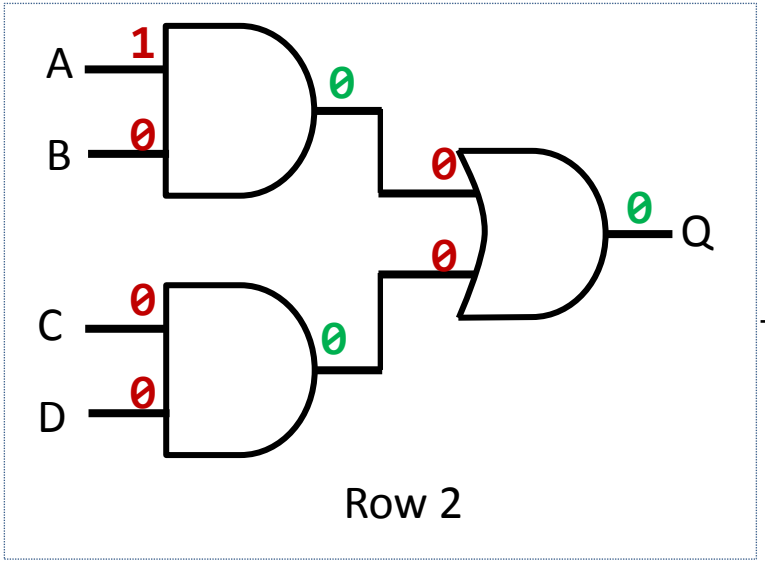
- What does this circuit do?



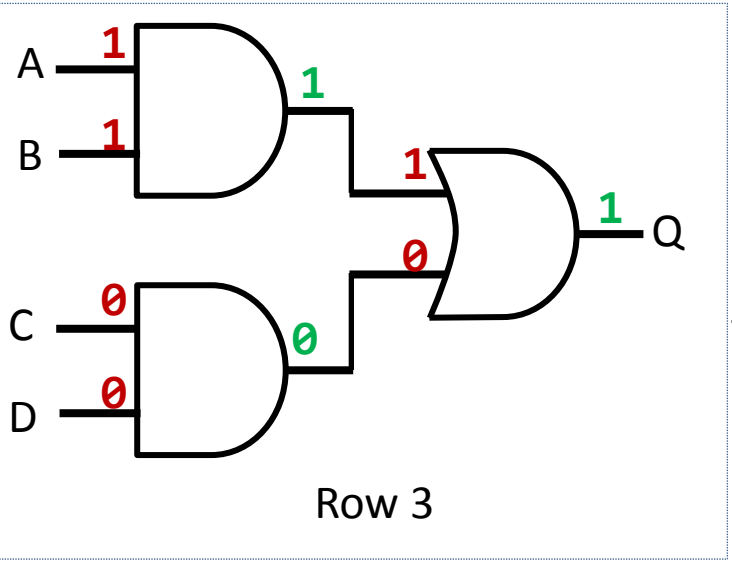
C	D	A	B	Q
0	0	0	0	0



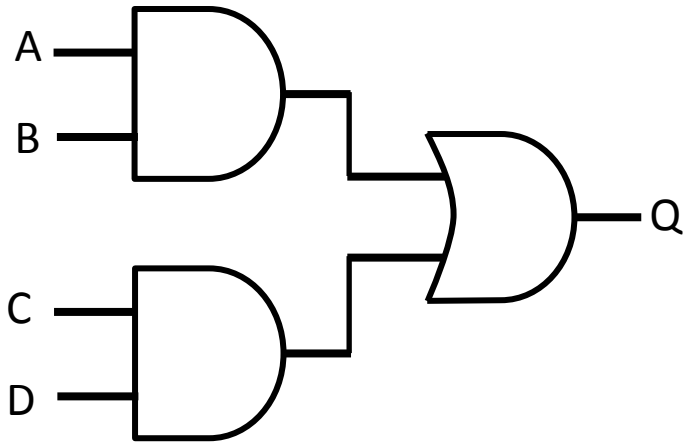
C	D	A	B	Q
0	0	0	0	0
0	0	0	1	0



C	D	A	B	Q
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0



C	D	A	B	Q
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1



C	D	A	B	Q
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Truth table to gates: Sum of Products

A	B	C	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = A'B'C + A'BC' + ABC$$

$$Cout = A'BC + AB'C + ABC' + ABC$$

Truth table to gates: Sum of Products

A	B	C	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = A'B'C + A'BC' + ABC$$

$$Cout = A'BC + AB'C + ABC' + ABC$$

Truth table to gates: Sum of Products

A	B	C	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = A'B'C + A'BC' + ABC$$

$$Cout = A'BC + AB'C + ABC' + ABC$$

Logic minimization (in brief)

$$Cout = A'BC + AB'C + ABC' + ABC$$

Add any number of ABC and still remains the same

$$Cout = A'BC + ABC + AB'C + ABC + ABC' + ABC$$
$$= BC(A' + A) + AC(B' + B) + AB(C' + C)$$
$$= \boldsymbol{BC + AC + AB}$$

NOT ON EXAM

De Morgan's Laws

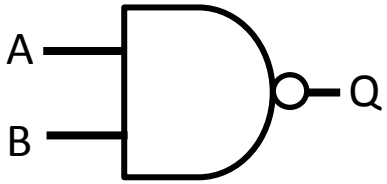
$$\sim(A \text{ AND } B) = \sim A \text{ OR } \sim B$$

$$\sim(A \text{ OR } B) = \sim A \text{ AND } \sim B$$

In practice used for building optimized circuits. Also allows us to understand why NAND gate is a universal gate

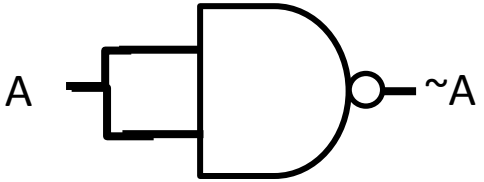
NAND Gate

NAND GATE

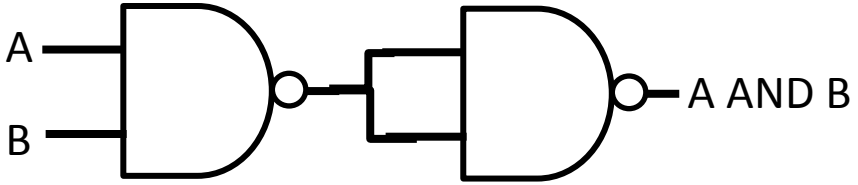


A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

NOT



AND



From De Morgan's Laws

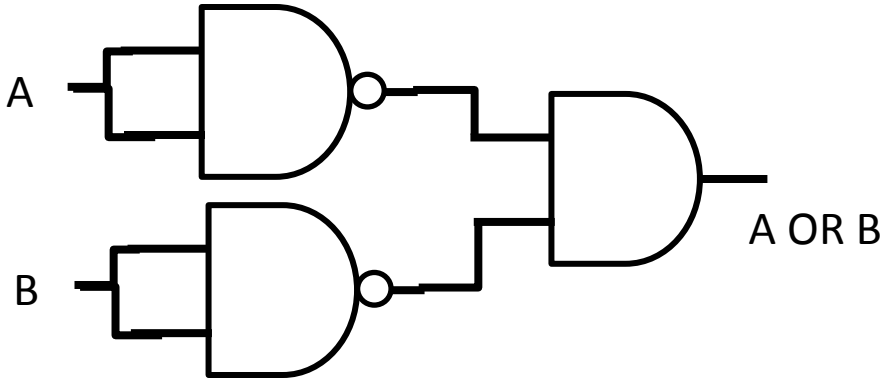
~(A + B) = ~A and ~B

~(~(A + B)) = A + B

= ~ (~A and ~B)

= ~A NAND ~B

OR



NOR gate?

NOT, AND, and OR using NOR gate

Laws of Boolean Algebra

- **Commutative Law**
  (a) $A + B = B + A$
  (b) $A\ B = B\ A$
- **Associate Law**
  (a) $(A + B) + C = A + (B + C)$
  (b) $(A\ B)\ C = A\ (B\ C)$
- **Distributive Law**
  (a) $A\ (B + C) = A\ B + A\ C$
  (b) $A + (B\ C) = (A + B)\ (A + C)$
- **Identity Law**
  (a) $A + A = A$
  (b) $A\ A = A$
- **Redundance Law**
  (a) $A + A\ B = A$
  (b) $A\ (A + B) = A$

Boolean Algebra cheat sheet

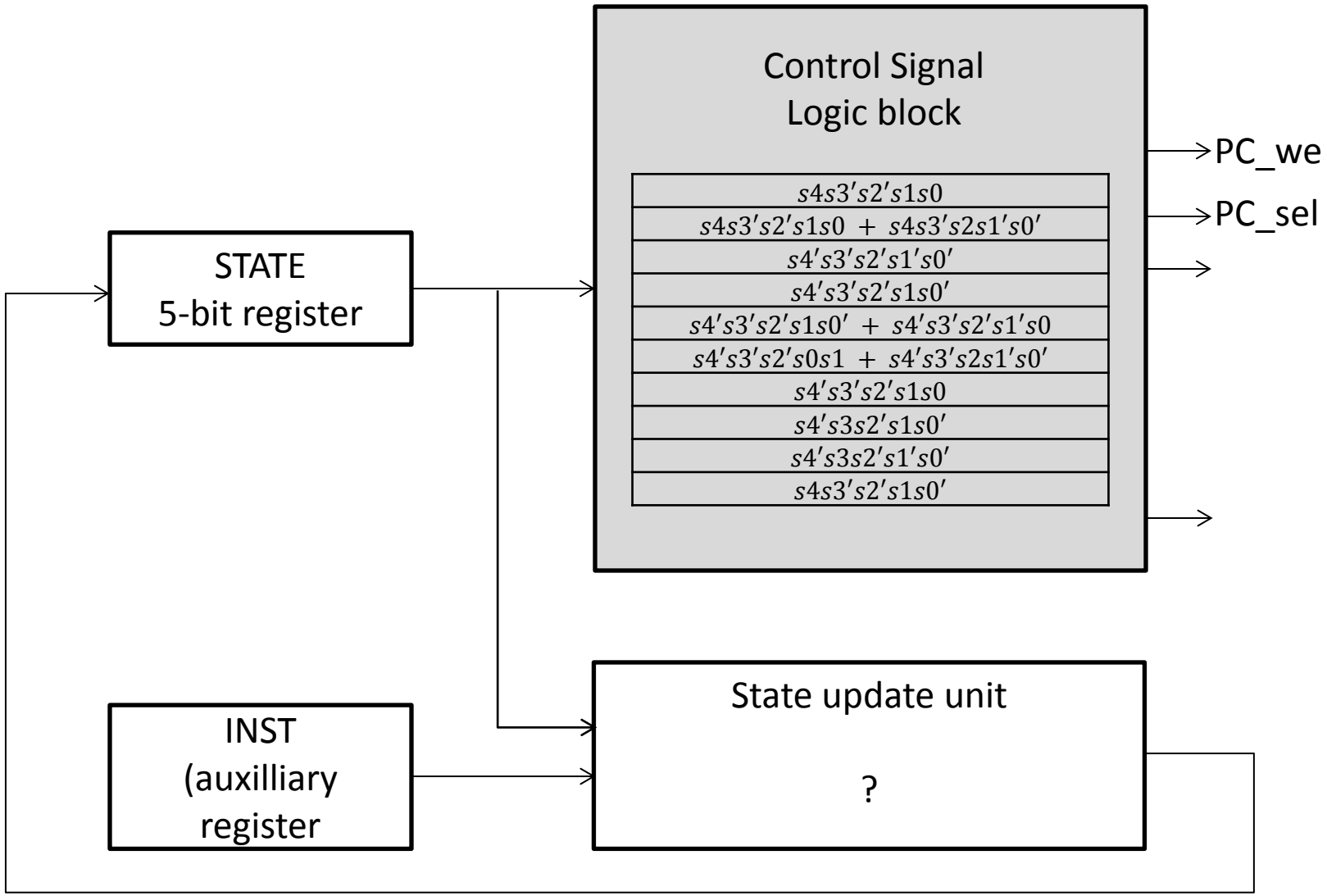
- **Adding 0**
  (a) *0 + A = A*
  (b) *0 A = 0*
- **Adding 1**
  (a) *1 + A = 1*
  (b) *1 A = A*
- AB + AB’ = A
  (A+B)(A+B’) = A
- A’+ A = 1
  A’A = 0
- A+ A’B = A + B
  A(A’ + B) = AB

State transition table

State number $s_4s_3s_2s_1s_0$	State name	PC_sel	PC_we	INST_we	REG_sel	REG_we	OFF_we	OFF_sel	REG2_we	RF_sel	RF_we
00000	INST = PM[PC]		0	1		0	0		0		0
00001	REG = 1,INST[7:4]		0	0	0	1	0		0		0
00010	REG = INST[8:4]		0	0	1	1	0		0		0
00011	OFF = Z?SEXT16(INST[9:3]):0		0	0		0	1	1	0		0
00100	OFF = SEXT16(INST[11:0])		0	0		0	1	0	0		0
00101	VAL = INST[11:8],INST[3:0]		0	0		0	0		0		0
00110	VAL = RF[REG]		0	0		0	0		0	0	0
00111	VAL1 = RF[REG]		0	0		0	0		0	0	0
01000	VAL2 = RF[REG2]		0	0		0	0		0	1	0
01001	ADDR = X		0	0		0	0		0		0
01010	REG2 = INST[9],INST[3:0]		0	0		0	0		1		0
01011	VAL = VAL1 + VAL2		0	0		0	0		0		0
01100	VAL = VAL1 - VAL2		0	0		0	0		0		0
01101	Update SREG		0	0		0	0		0		0
01110	VAL = OFF+PC		0	0		0	0		0		0
01111	VAL2 = INST[11:8],INST[3:0]		0	0		0	0		0		0
10000	VAL = RAM[ADDR]		0	0		0	0		0		0
10001	RAM[ADDR] = VAL		0	0		0	0		0		0
10010	RF[REG] = VAL		0	0		0	0		0	0	1
10011	PC = PC+1	1	1	0		0	0		0		0
10100	PC = VAL	0	1	0		0	0		0		0

PC_sel	$s4s3's2's1s0$
PC_we	$s4s3's2's1s0 + s4s3's2s1's0'$
INST_we	$s4's3's2's1's0'$
REG_sel	$s4's3's2's1s0'$
REG_we	$s4's3's2's1s0' + s4's3's2's1's0$
OFF_we	$s4's3's2's0s1 + s4's3's2s1's0'$
OFF_sel	$s4's3's2's1s0$
REG2_we	$s4's3s2's1s0'$
RF_sel	$s4's3s2's1's0'$
RF_we	$s4s3's2's1s0'$

State machine controller gates



Today

- Basic logic gates (Today)
  - Most of state machine controller using gates
- Next class
  - Datapath modules (Dec 2nd)
  - Complete processor using gates

CS 252
Lecture 30; 2015 Nov 30th; Transcribed Lecture notes
Basic Logic Gates

Announcements

Exam Summary

```
  2 below 50%
  5 between 50% and 70%
 12 between 70% and 80%
 22 between 80% and 90%
132 above 90%
```

You will be able to view your projected grade before exam 4

Your netID will be hashed to send you the message

Outline

Basic logic gates

- Abstraction
- Notation of simple logic gates
- Combining logic gates
- Truth tables to gates
- Logic minimization and boolean algebra (not on exam)
- De Morgan's law and making logic gates
- State transition table

Abstraction of Logic Gates

Logic gates provides boolean functionality.

Made up of transistors.

Notation of Simple Logic Gates

NOT gate (~) is set if the input is low

AND gate ($\cdot$) is set if both inputs are high

OR gate (+) is set if either inputs are high

XOR gate ($\oplus$) is set if exactly 1 input is high

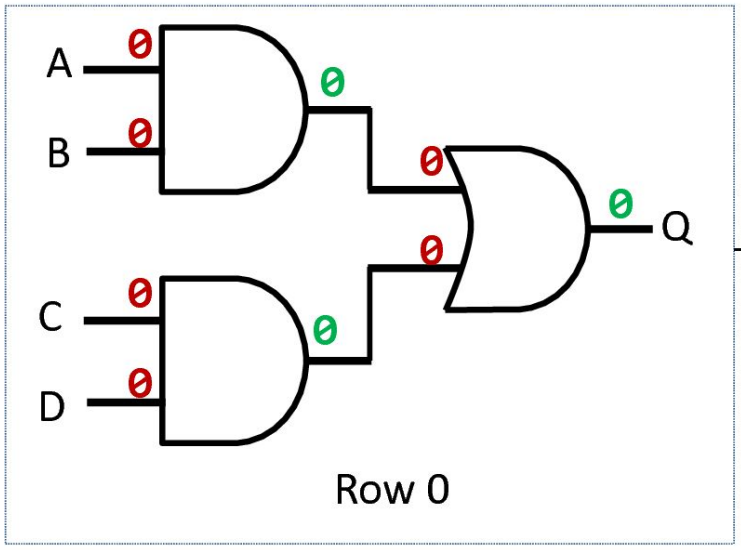
NAND gate is set if either inputs are low

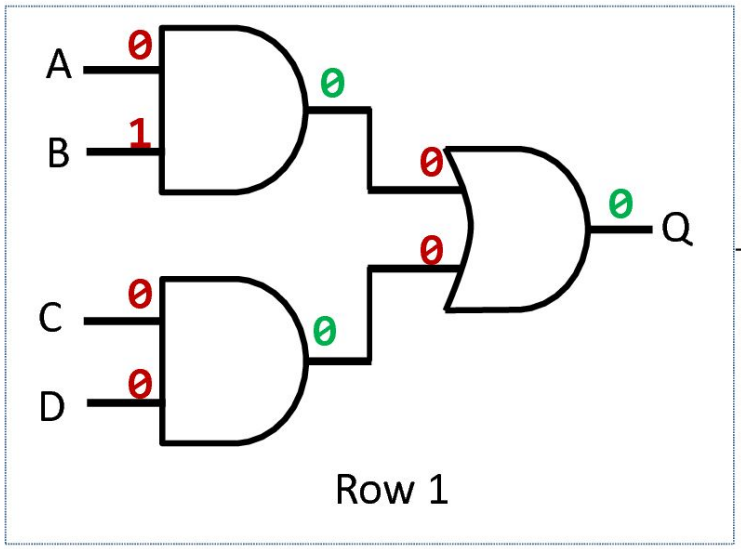
NOR gate is set if both inputs are low

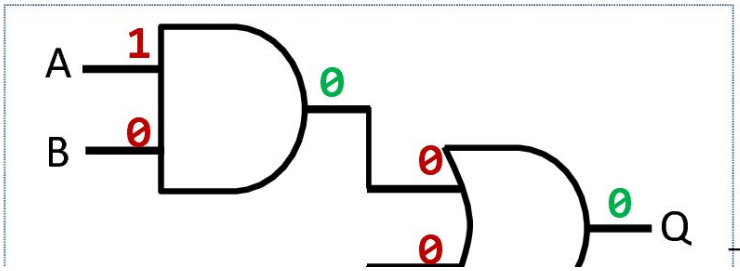
XNOR gate is set if the inputs are the same

Combining Logic Gates

Example below shows the combination of 2 AND gates and 1 OR gate and its truth table







C	D	A	B	Q
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

So far, there are no cycles

cycle is where the output of 1 gate goes into the input of another gate AND vice versa

A cycle can be used as an infinite loop and hold memory

Truth Tables to Gates

Full adder example

A	B	Carry-In	Sum	Carry-Out
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

```
Sum = A'B'C+A'BC'+ABC
Carry_out = A'BC+AB'C+ABC'+ABC
```

Logic minimization (in brief, NOT ON EXAM, covered in 352)

Full adder example

```
Carry_out = A'BC+AB'C+ABC'+ABC
          = A'BC+ABC+AB'C+ABC+ABC'+ABC
          = BC(A'+A)+AC(B'+B)+AB(C'+C)
          = BC+AC+AB
```

Important since fewer gates and faster

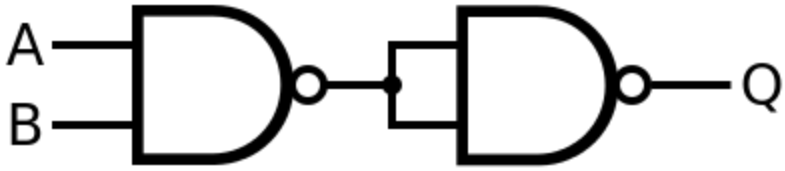
De Morgan's Laws

```
~(A AND B) = ~A OR ~B
~(A OR B) = ~A AND ~B
```

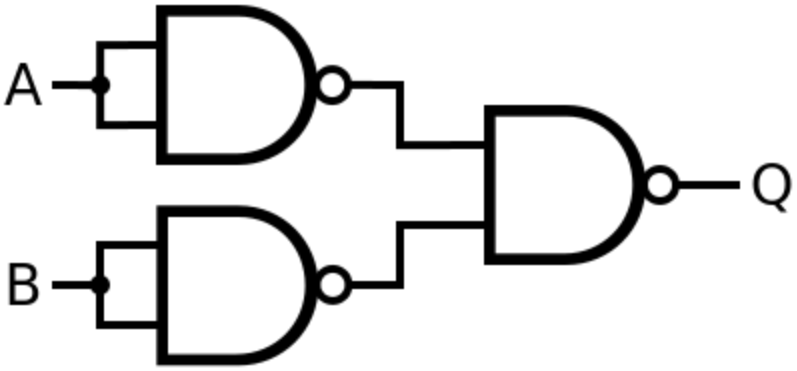
Universal NAND Gate

All the gates can be built with just NAND gates
NAND gates can be built with transistors, so all logic can be built from NAND gate
NOT gate

AND gate



OR gate (A OR B = ~A NAND ~B by De Morgan's Law)



Homework: How can we build NOT, AND, and OR gate using only NOR gates

Boolean Algebra Cheat Sheet

State Transition Table and State Machine Controller

PC_sel = s4s3's2's1s0
PC_we = s4s3's2's1s0 + s4s3's2s1's0'
...

Used to build the state machine controller
    Figuring the next state is yet another truth table