Intro to Digital Systems

Store Format Example

`sw x14, 36(x5)`

31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7	5	5	3	5	7

31 25	24 20	19 15	14 12	11 7	6 0
0000001	01110	00101	010	00100	0100011
7	5	5	3	5	7

36 = 0b $\boxed{01}\boxed{00100}$

Levels of Representation/Interpretation

High Level Language Program (e.g., C)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Compiler

Assembly Language Program (e.g., RISC-V)

```
lw   $t0, 0($2)
lw   $t1, 4($2)
sw   $t1, 0($2)
sw   $t0, 4($2)
```

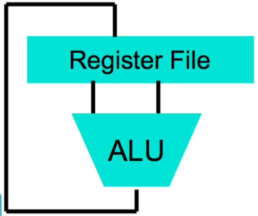
Assembler

Machine Language Program (RISC-V)

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

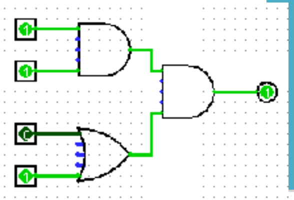
Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)



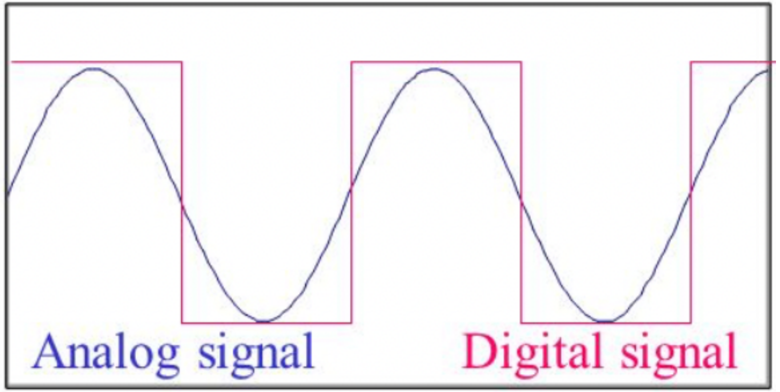
Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)



Digital Systems

- Digital
  - All values are discrete
  - A value can be on (1) or off (0)
- Analog
  - Have a continuous range of values



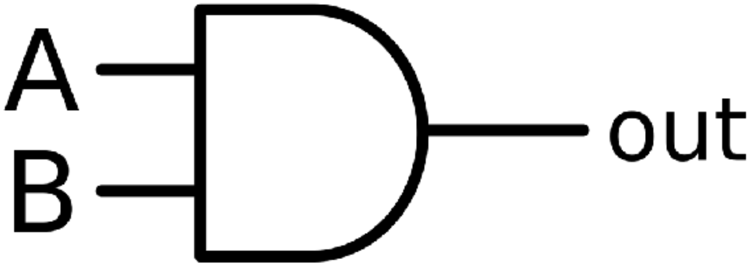
Logic Gates

Logic Gates

- The building block of digital circuits
- Perform logic operations
  - AND
  - OR
  - XOR
  - NOT
  - NAND
  - NOR
  - XNOR

AND

A	B	out
0	0	0
0	1	0
1	0	0
1	1	1

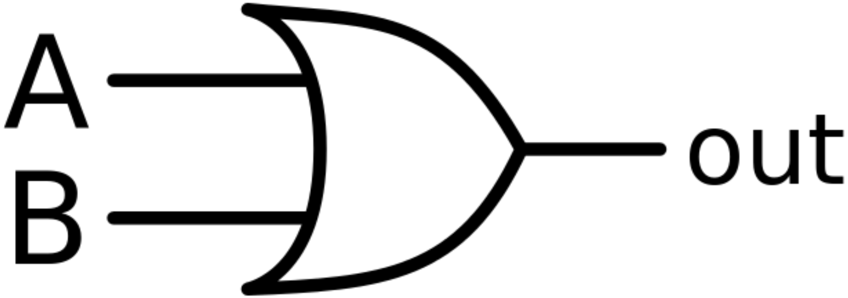


`out = A & B`

C syntax

OR

A	B	out
0	0	0
0	1	1
1	0	1
1	1	1

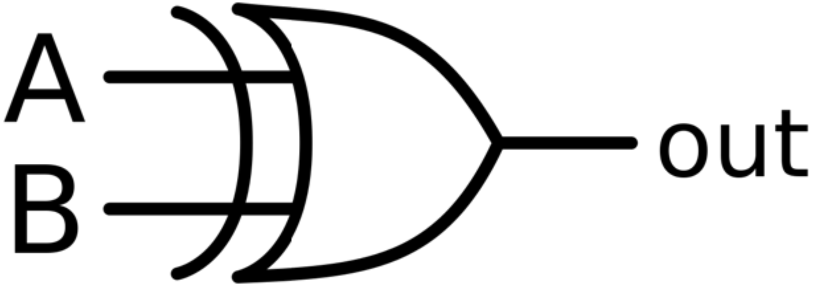


`out = A | B`

C syntax

XOR

A	B	out
0	0	0
0	1	1
1	0	1
1	1	0

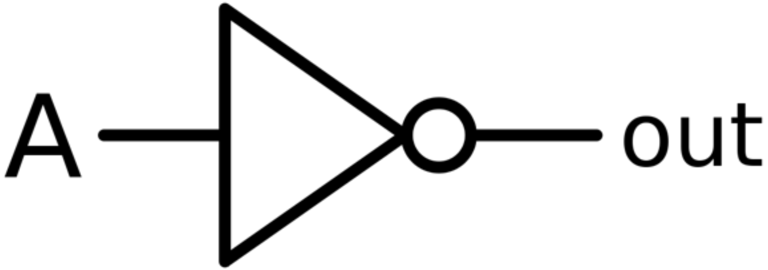


`out = A ^ B`

C syntax

NOT

A	out
0	1
1	0

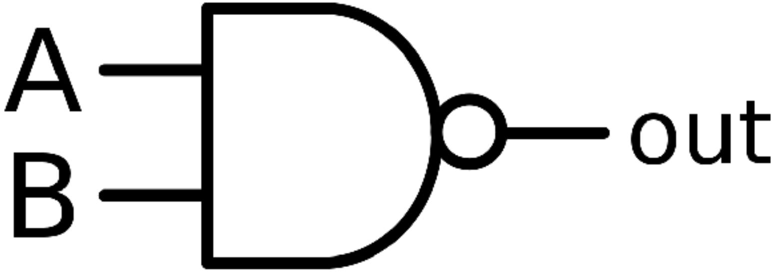



```
out = ~A
```

C syntax

NAND

A	B	out
0	0	1
0	1	1
1	0	1
1	1	0

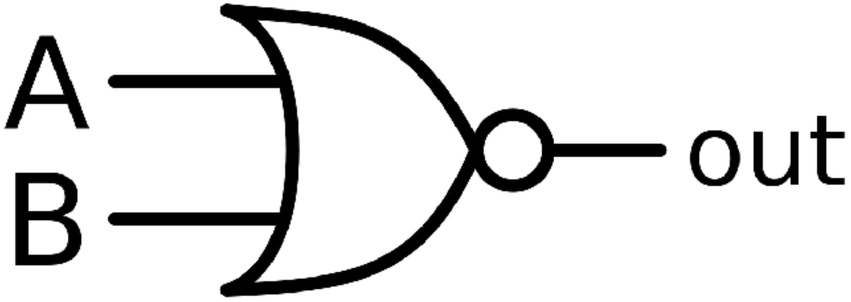


`out = ~(A & B)`

C syntax

NOR

A	B	out
0	0	1
0	1	0
1	0	0
1	1	0

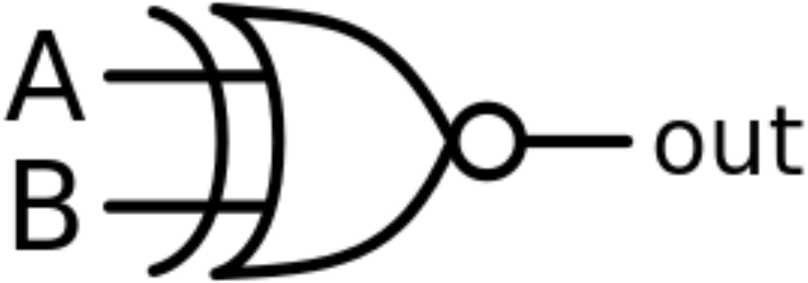


`out = ~(A | B)`

C syntax

XNOR

A	B	out
0	0	1
0	1	0
1	0	0
1	1	1




```
out = ~(A ^ B)
```

C syntax

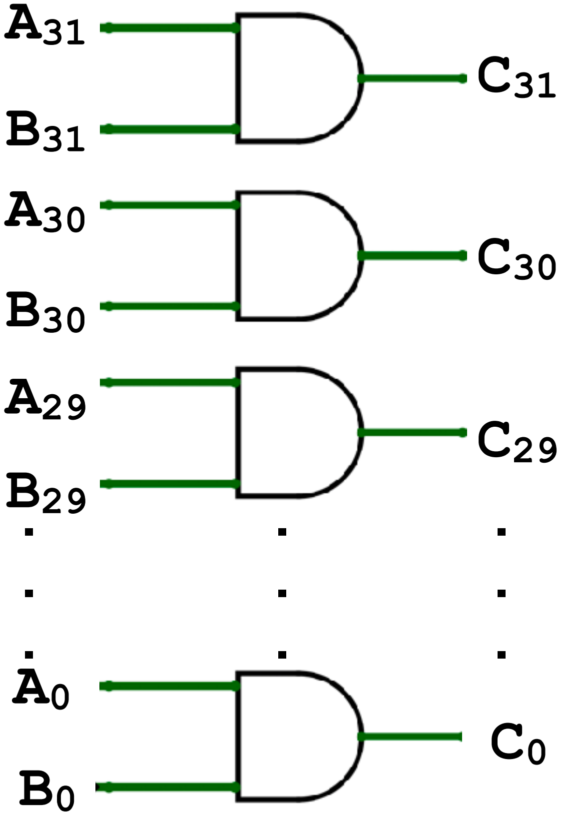
Logic operations on 32 bit numbers

$C = A \& B$

$A = A_{31}\ A_{30}\ A_{29} \ldots A_0$

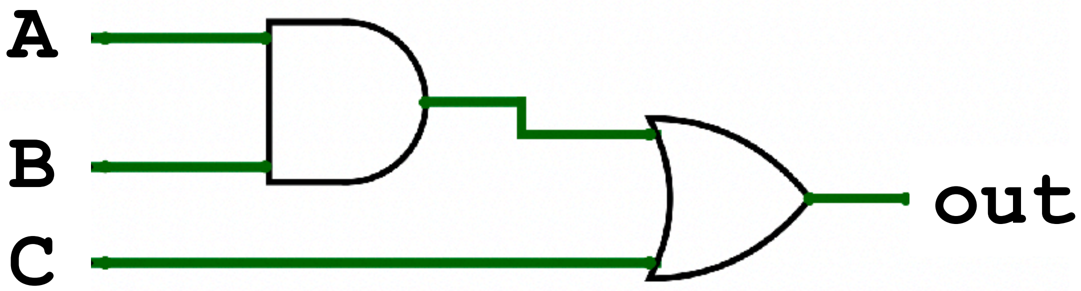
$B = B_{31}\ B_{30}\ B_{29} \ldots B_0$

$C = C_{31}\ C_{30}\ C_{29} \ldots C_0$



Connecting Logic Gates


```
out = (A & B) | C
```



(A, B, and C are one bit each)

Boolean Algebra

Boolean Algebra

- A branch of algebra in which
  - The operands can only be 0 or 1
  - The basic operations are AND, OR, and NOT
  - (NAND, NOR, XOR, and XNOR can be created with a combination of the above operations)

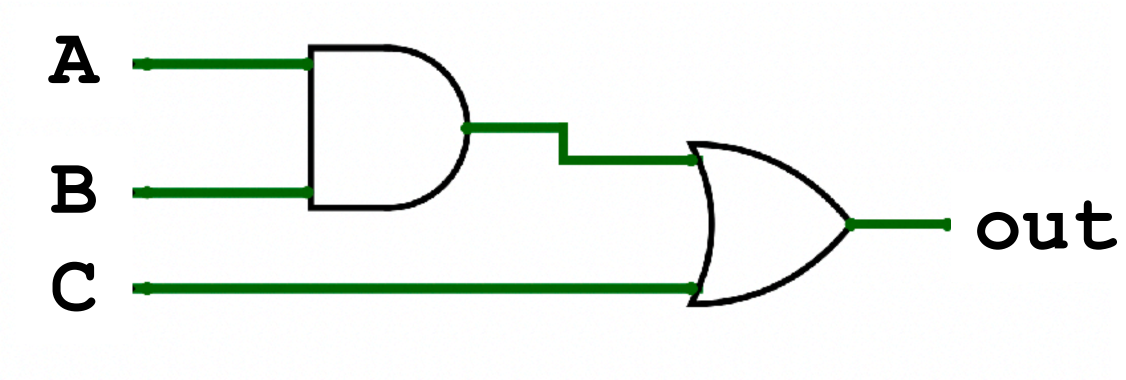
Boolean Notation

Operation	C Notation	Boolean Notation
A AND B	A & B	AB
A OR B	A | B	A + B
NOT A	~A	$\bar{A}$

Boolean Algebra

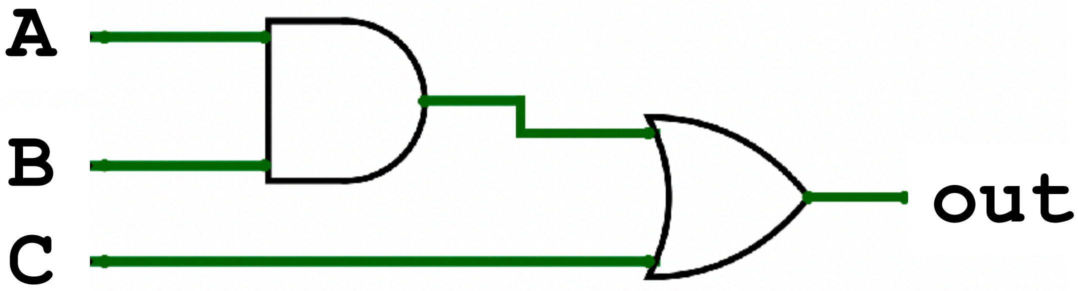
C notation: `out = (A & B) | C`

Boolean notation: `out = AB + C`



Boolean Equation -> Truth Table

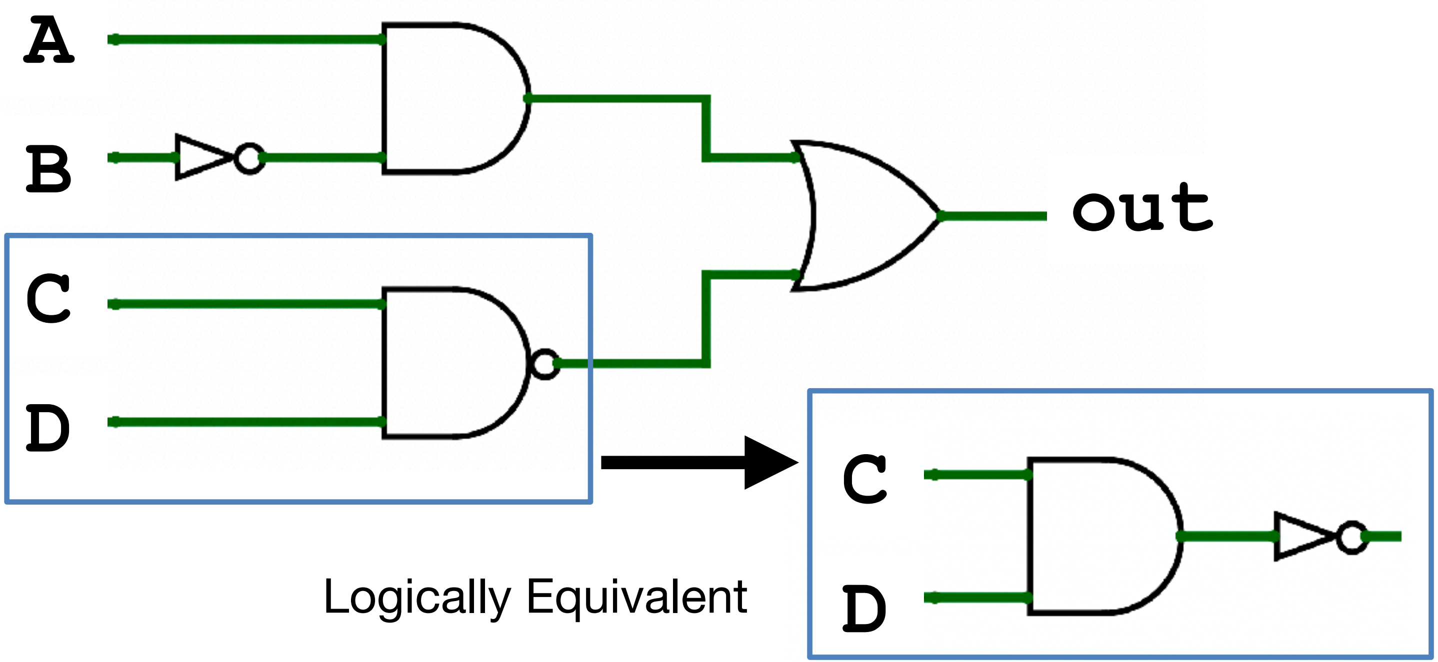
`out = AB + C`



A	B	C	out
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	1
1	1	1	1

Boolean Equation -> Logic Gate Diagram

$$out = A\bar{B} + \overline{CD}$$

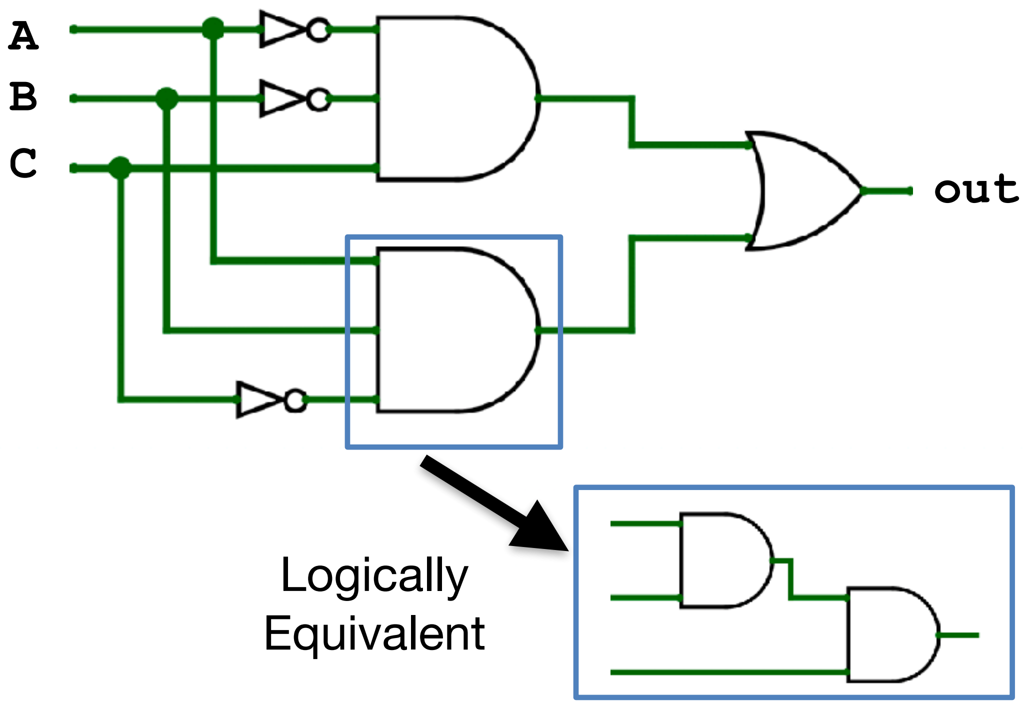


Logically Equivalent

Truth Table to Circuit Diagram

A	B	C	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

$$\texttt{out} = \overline{A}\,\overline{B}C + AB\overline{C}$$



How to Make XOR with AND and OR

A	B	out
0	0	0
0	1	1
1	0	1
1	1	0

$$\texttt{out} = \overline{A}B + A\overline{B}$$

How to Make XNOR with AND and OR

A	B	out
0	0	1
0	1	0
1	0	0
1	1	1

$$\mathtt{out} = \overline{A}\,\overline{B} + AB$$

Laws of Boolean Algebra

Complementary

$$A\bar{A} = 0 \qquad\qquad A + \bar{A} = 1$$

Laws of Boolean Algebra

Laws of 0's and 1's

$$A0 = 0 \qquad\qquad A + 1 = 1$$

Laws of Boolean Algebra

Identities

$$A1 = A \qquad\qquad A + 0 = A$$

Laws of Boolean Algebra

Idempotent

$$AA = A \qquad\qquad A + A = A$$

Laws of Boolean Algebra

Commutativity

AB = BA

A + B = B + A

Laws of Boolean Algebra

Associativity

```
(AB)C = A(BC)
```

```
(A + B) + C = A + (B + C)
```

Laws of Boolean Algebra

Distribution

$$A(B + C) = AB + AC$$

$$A + BC = (A + B)(A + C)$$

Laws of Boolean Algebra

Uniting Theorem

```
AB + A = A          (A + B)A = A
```

Laws of Boolean Algebra

Uniting Theorem #2

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

Laws of Boolean Algebra

Uniting Theorem #2

$$(\overline{A} + B)A = AB$$

Laws of Boolean Algebra

DeMorgan's Law

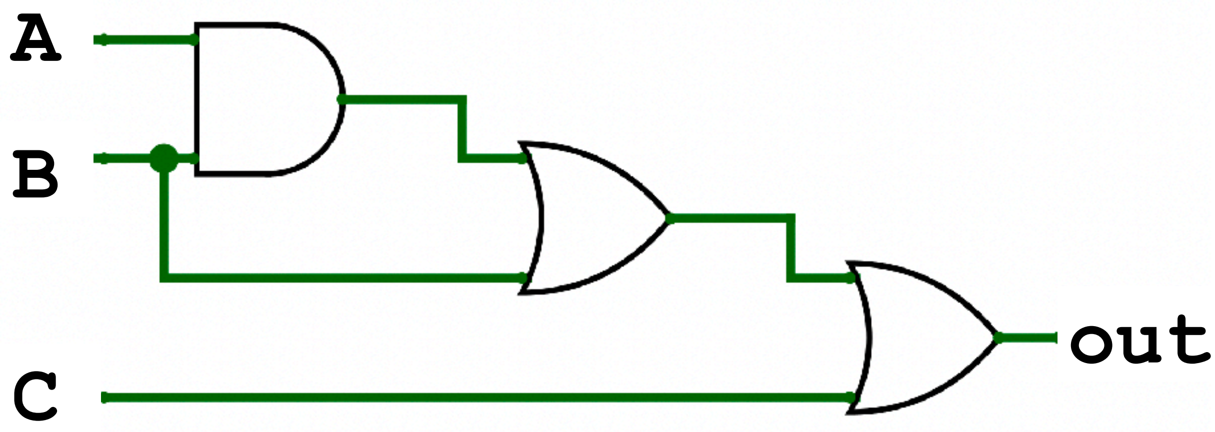
$$\overline{AB} = (\overline{A} + \overline{B})$$

Laws of Boolean Algebra

DeMorgan's Law

$$\overline{(A + B)} = \overline{A}\,\overline{B}$$

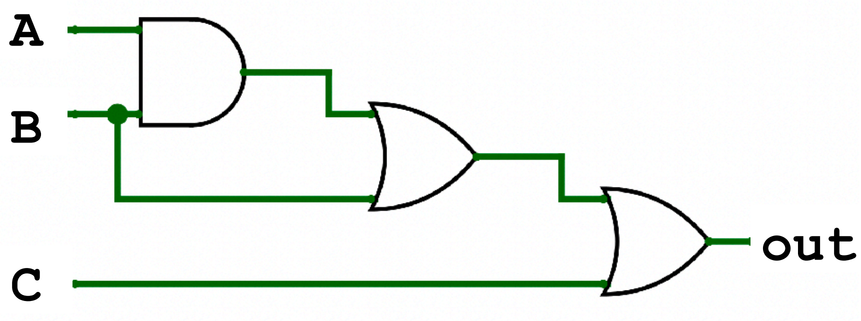
Using Boolean Algebra to Simplify Circuits



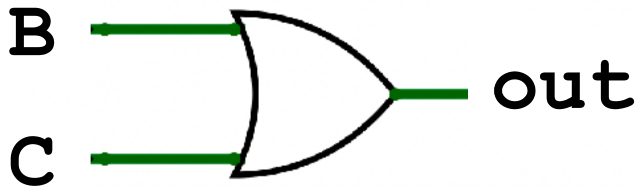

```
out = AB + B + C
out = B(A + 1) + C
out = B + C
```

Using Boolean Algebra to Simplify Circuits

`out = AB + B + C`

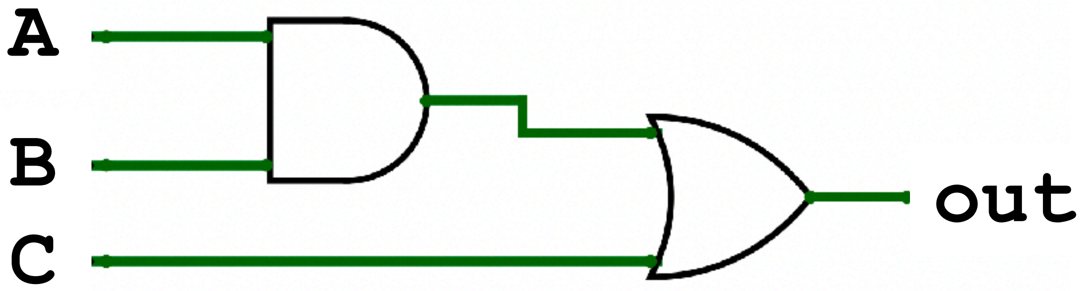


`out = B + C`



Recall: Boolean Equation -> Truth Table

`out = AB + C`



1. Determine when each product will be true
2. Place a one in the corresponding rows
3. Place a zero in the remaining rows

A	B	C	out
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	1
1	1	1	1

Sum of Products

- When two or more products (AND) are summed (OR) together

```
out = AB + AC
```

~~`out = A(B + C)`~~

- When the equation is written in this form, it is easier to convert it to a truth table

Sum of Products


```
out =  (A + B)(A + C)

out = AA + AC + AB + BC

out = A + AC + AB + BC

out = A(1 + C + B) + BC

out = A + BC
```

A	B	C	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

How to Build an Adder

Recall: Binary Addition

Carry out from the second column / Carry into the third column

Carry in to the second column

```
  111
  101
+ 111
-----
  100
```

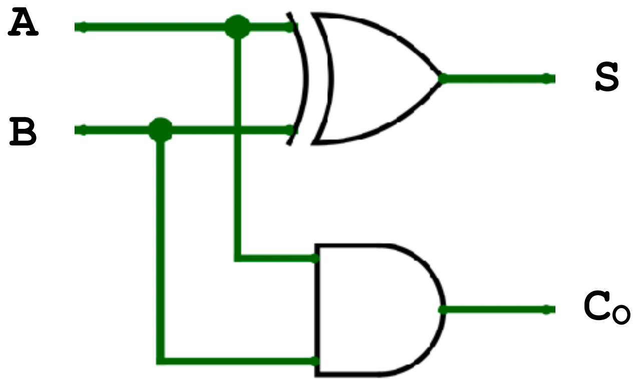
Half Adder

- How to add two bits together using logic gates?

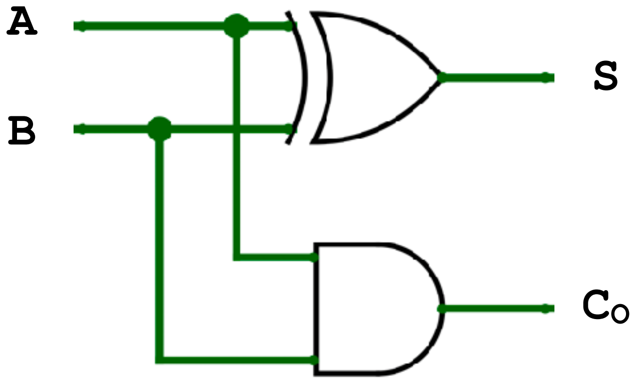
A	B	C_O	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

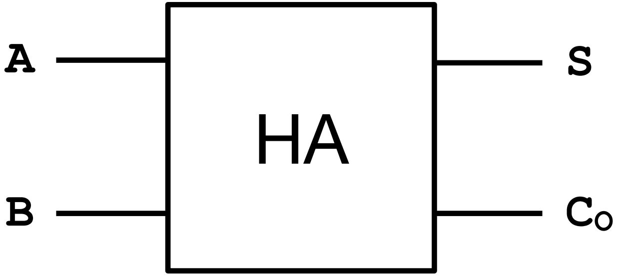
$$S = A \oplus B$$

$$C_O = AB$$



Half Adder Abstraction





Recall: Binary Addition

Carry out from the second column / Carry into the third column

Carry in to the second column

$$
\begin{array}{r}
1\boxed{1}\boxed{1} \\
101 \\
+\;111 \\
\hline
100
\end{array}
$$

Full Adder (single bit)

A	B	C_i	C_o	S
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

S is 1 when 1 or 3 of the bits are 1

$$S = A \oplus B \oplus C_i$$

Full Adder (single bit)

A	B	C_i	C_o	S
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

$$C_o = \overline{A}BC_i + A\overline{B}C_i + AB\overline{C_i} + ABC_i$$

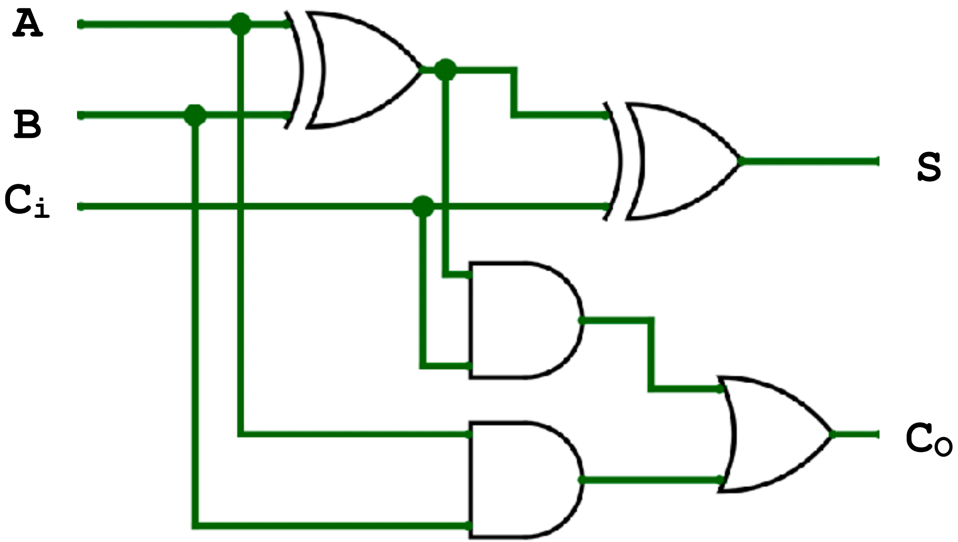
$$C_o = C_i(\overline{A}B + A\overline{B}) + AB(\overline{C_i} + C_i)$$

$$C_o = C_i(A \oplus B) + AB$$

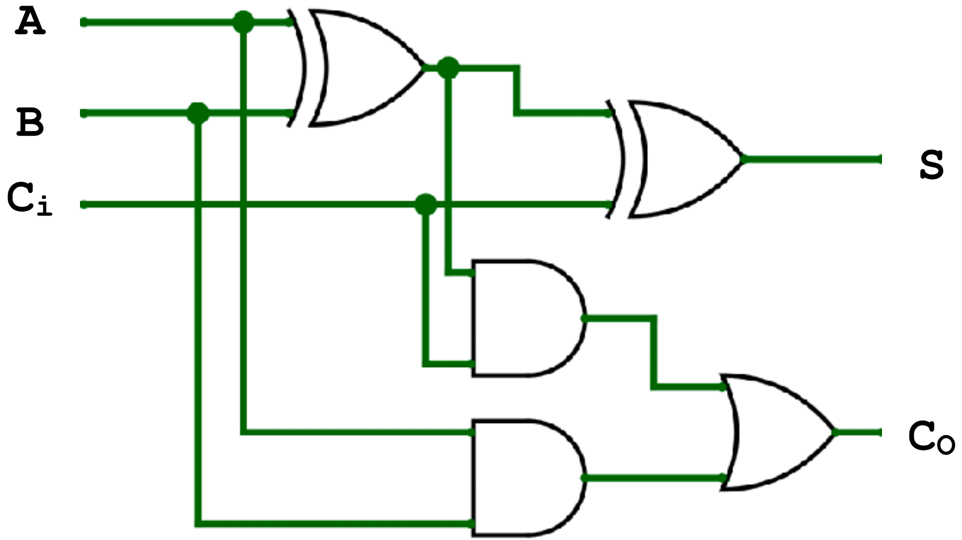
Full Adder

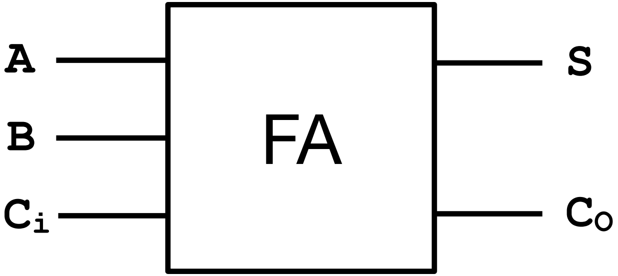
$$S = A \oplus B \oplus C_i$$

$$C_o = AB + C_i(A \oplus B)$$

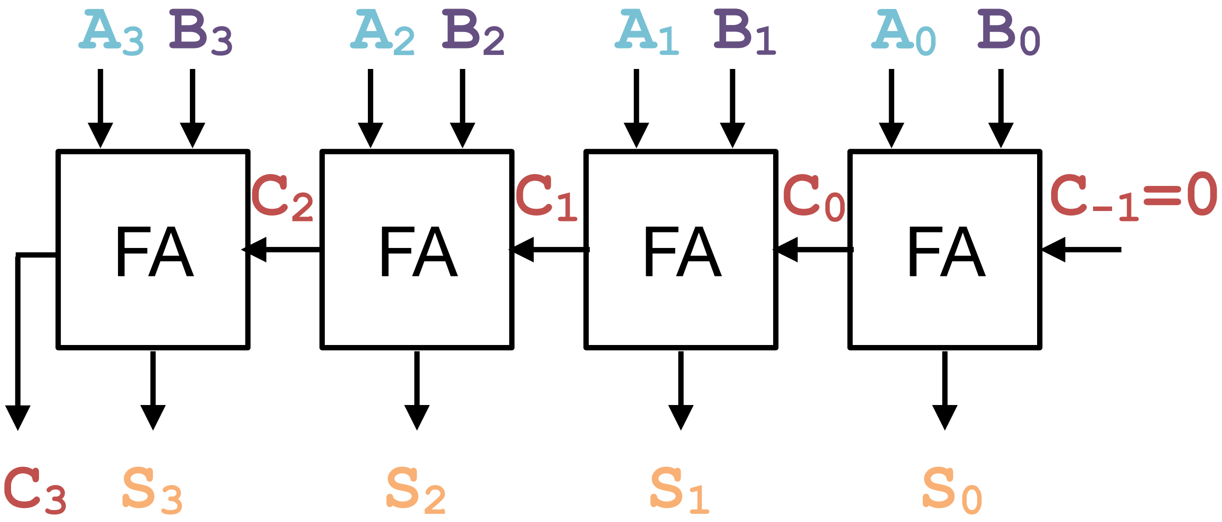


Full Adder Abstraction





4-bit Adder



$$
\begin{array}{ccccc}
C_3 & C_2 & C_1 & C_0 & C_{-1} = 0 \\
 & A_3 & A_2 & A_1 & A_0 \\
 & B_3 & B_2 & B_1 & B_0 \\
\hline
 & S_3 & S_2 & S_1 & S_0
\end{array}
$$

Arithmetic Logic Unit

Arithmetic Logic Unit (ALU)

- Carries out arithmetic and logical operations on integer binary numbers

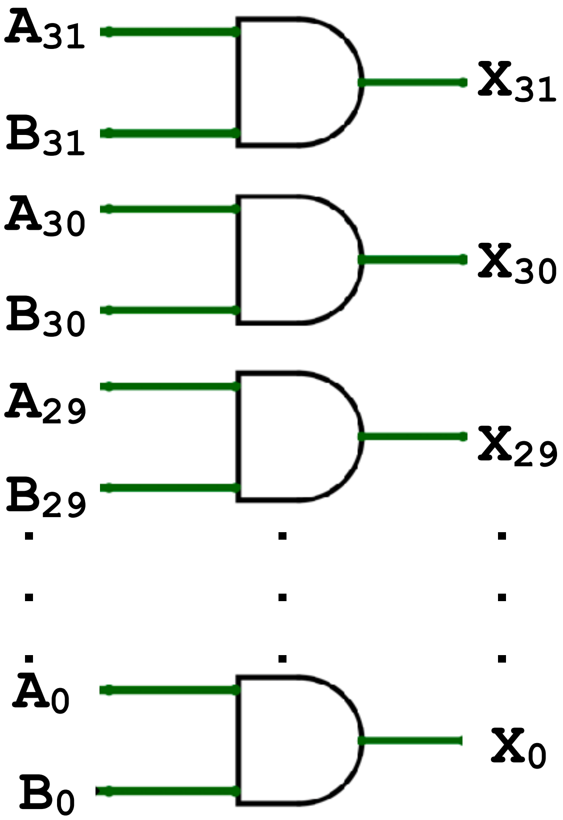
Recall: Logical AND on 32-bit Number

$X = AB$

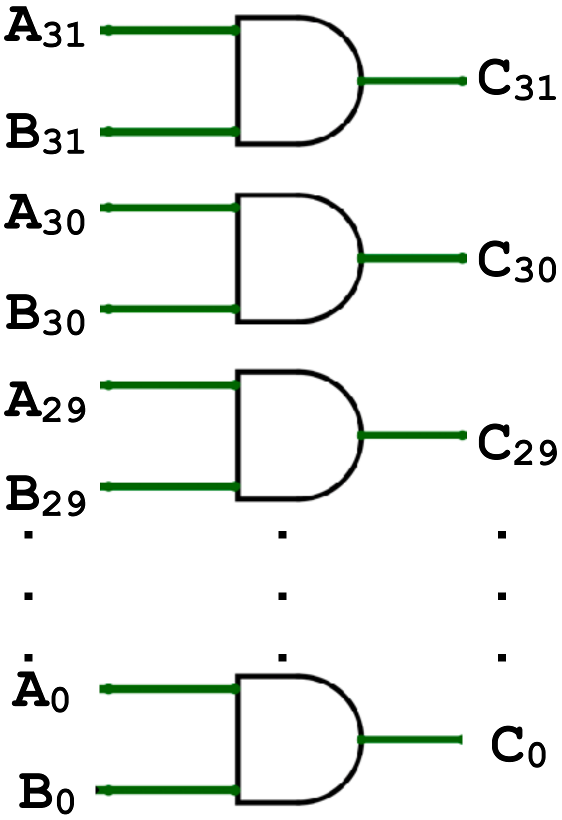
$A = A_{31}\ A_{30}\ A_{29} \ldots A_0$

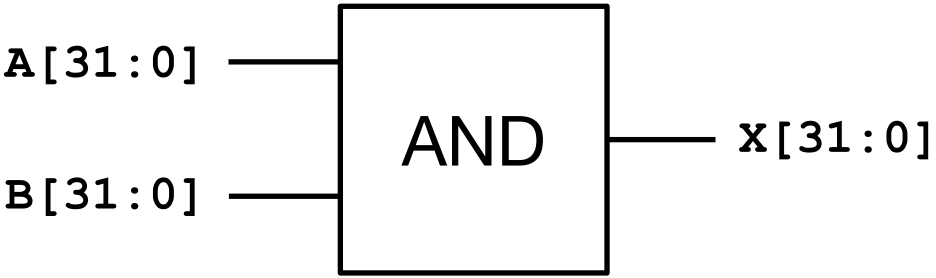
$B = B_{31}\ B_{30}\ B_{29} \ldots B_0$

$X = X_{31}\ X_{30}\ X_{29} \ldots X_0$

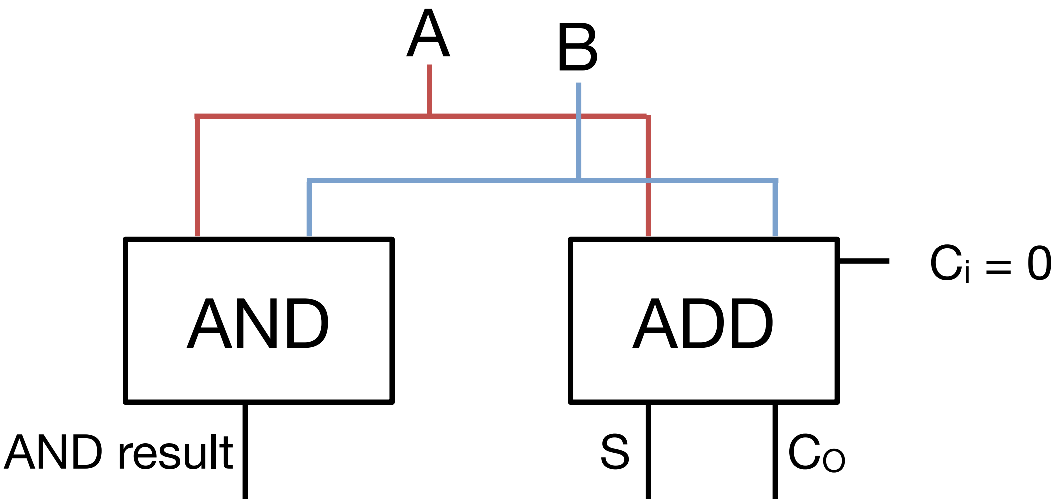


Logical AND on 32-bit Number Abstraction



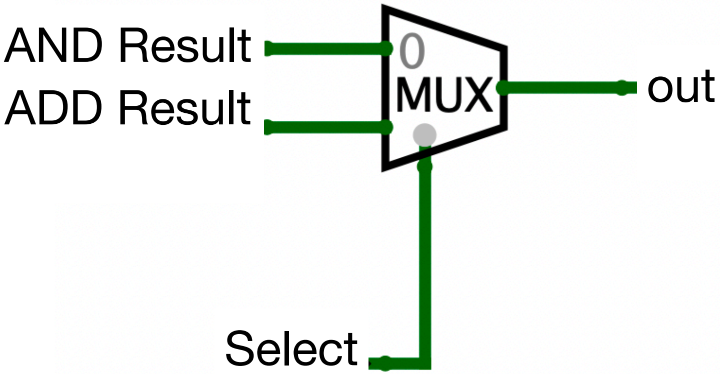


Arithmetic Logic Unit



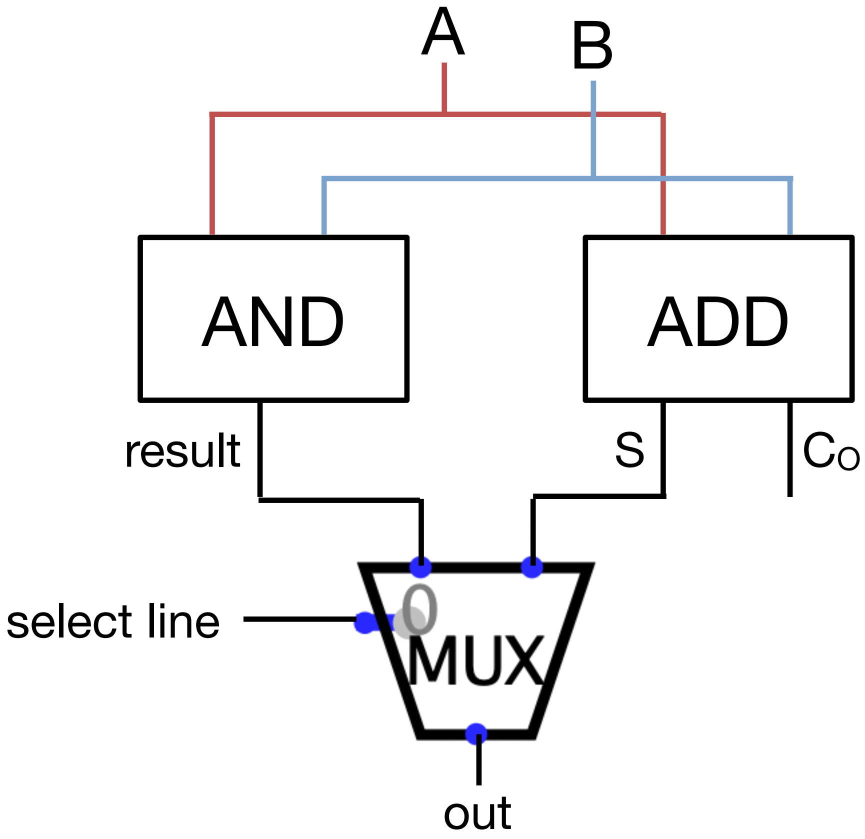
2:1 Multiplexors (Abstraction)

- Selects an input to propagate to the output

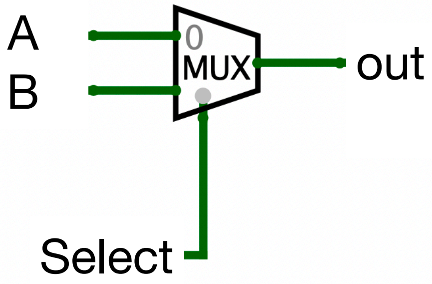


If select = 0, out = AND Result
If select = 1, out = ADD Result

Arithmetic Logic Unit

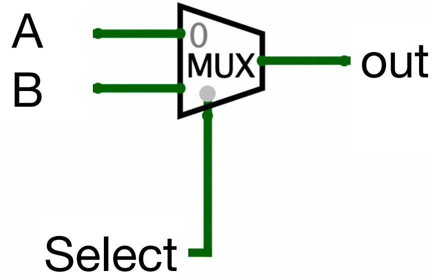


2:1 Mux Implementation

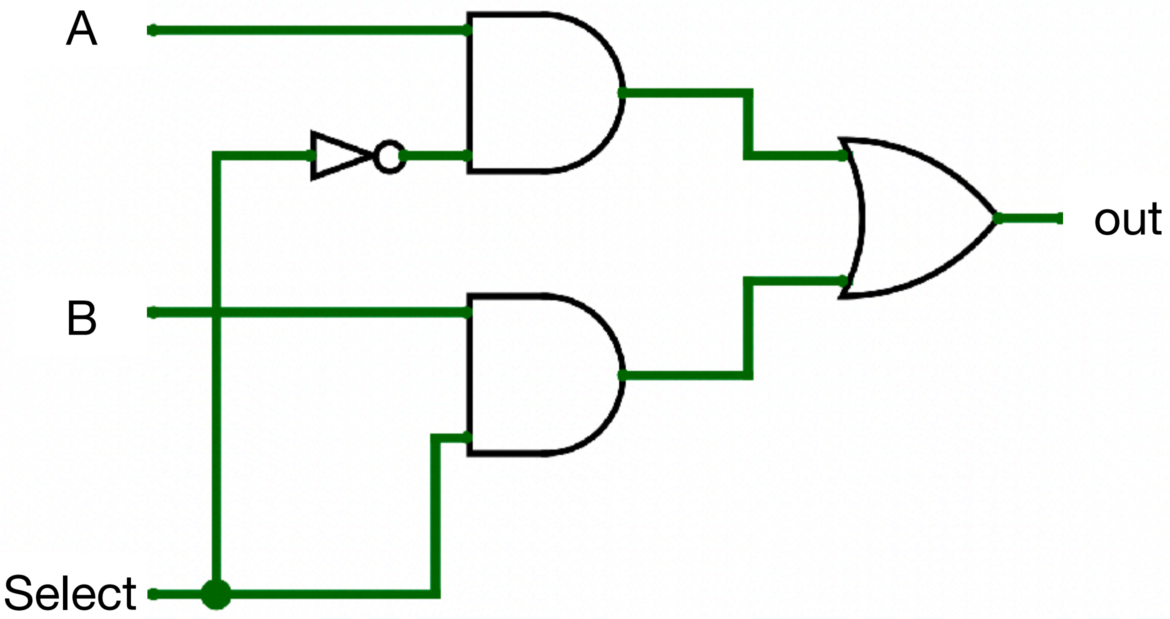


$$\texttt{out} = \mathbf{A}\overline{\mathbf{S}} + \mathbf{BS}$$

2:1 Mux Implementation



$$\texttt{out} = \texttt{A}\overline{\texttt{S}} + \texttt{BS}$$



Combinational Logic

- Everything that we have talked about so far is called combinational logic
- As soon as the inputs are available, the output starts being computed
- Output depends only on the current input
  - We'll see circuits whose outputs depend on more than the current input in the next lecture