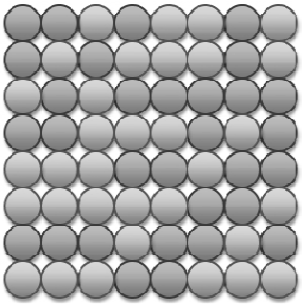


UNIT 8A

Computer Circuitry: Layers of Abstraction

Boolean Logic & Truth Tables

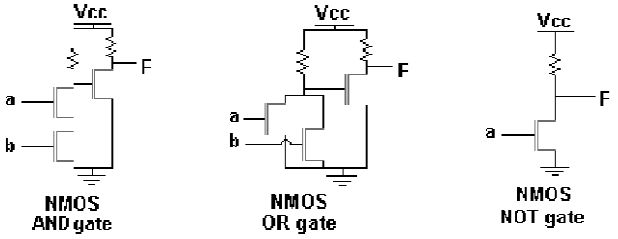
- Computer circuitry works based on Boolean logic: operations on true (1) and false (0) values.

A	B	A ∧ B (A AND B) (Ruby: A && B)	A ∨ B (A OR B) (Ruby: A || B)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

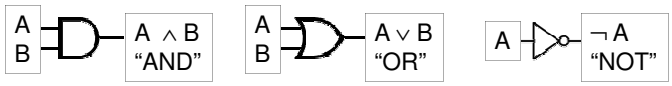
A	¬ A (NOT A) (Ruby: !A)
0	1
1	0

Gates

- A gate is an electronic device that implements a logical function. (Images from Wikipedia)
- Circuit diagram:



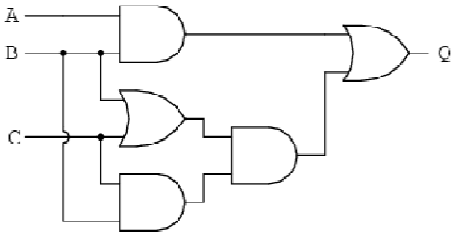
- Abstract Diagram:



Properties

- Commutative: $a \wedge b = b \wedge a$ $a \vee b = b \vee a$
- Associative: $a \wedge b \wedge c = (a \wedge b) \wedge c = a \wedge (b \wedge c)$
  $a \vee b \vee c = (a \vee b) \vee c = a \vee (b \vee c)$
- Distributive: $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$
- Identity: $a \wedge 1 = a$ $a \vee 0 = a$
- Idempotence: $a \wedge a = a$ $a \vee a = a$
- Complementation: $a \wedge \neg a = 0$ $a \vee \neg a = 1$
- Double Negation: $\neg \neg a = a$

Equivalence

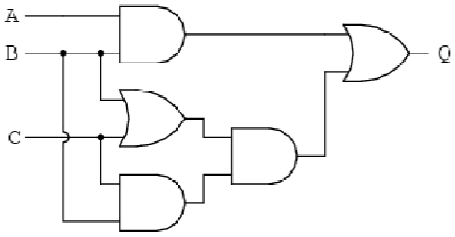


$Q = (A \wedge B) \vee ((B \vee C) \wedge (C \wedge B))$

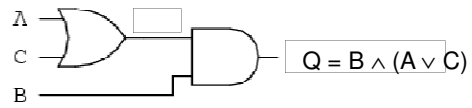
A	B	C	Q
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

http://www.allaboutcircuits.com/vol_4/chpt_7/6.html

Equivalence



$Q = (A \wedge B) \vee ((B \vee C) \wedge (C \wedge B))$



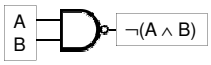
A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

http://www.allaboutcircuits.com/vol_4/chpt_7/6.html

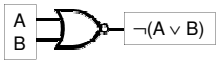
More gates

A	B	A nand B	A nor B	A xor B
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	0	0	0

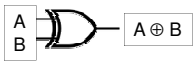
- nand ("not and"): A nand B = not (A and B)



- nor ("not or"): A nor B = not (A or B)



- xor ("exclusive or"):
  A xor B = (A and not B) or (B and not A)



DeMorgan's Law

Nand: $\neg(A \wedge B) = \neg A \vee \neg B$

```
if !((x > 15) && (x < 110)) then ...
```

is logically equivalent to

```
if (x <= 15) || (x >= 110) then ...
```

Nor: $\neg(A \vee B) = \neg A \wedge \neg B$

```
if !((x < 15) || (x > 110)) then ...
```

is logically equivalent to

```
if (x >= 15) && (x <= 110) then ...
```

A Logical Function Using Gates

3-bit odd parity checker

$F = (\neg x \wedge \neg y \wedge z) \vee (\neg x \wedge y \wedge \neg z) \vee (x \wedge \neg y \wedge \neg z) \vee (x \wedge y \wedge z)$

x	y	z	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

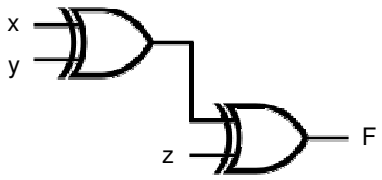
A Logical Function Using Gates

3-bit odd parity checker

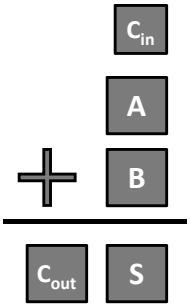
$F = (\neg x \wedge \neg y \wedge z) \vee (\neg x \wedge y \wedge \neg z) \vee (x \wedge \neg y \wedge \neg z) \vee (x \wedge y \wedge z)$

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$F = x \oplus y \oplus z$

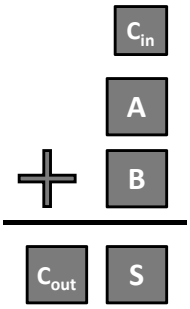


A Full Adder



A	B	C_{in}	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

A Full Adder

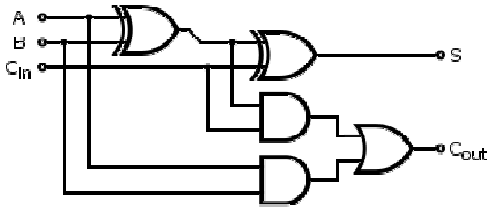


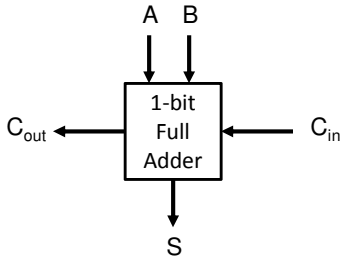
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$S = A \oplus B \oplus C_{in}$

$C_{out} = ((A \oplus B) \wedge C) \vee (A \wedge B)$

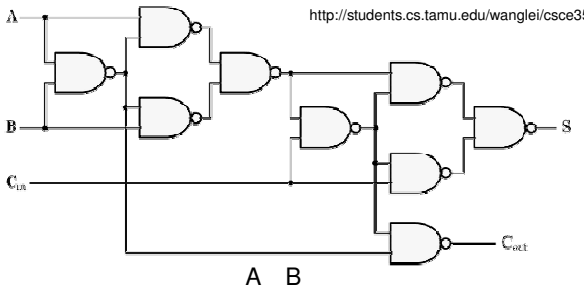
Full Adder (FA)

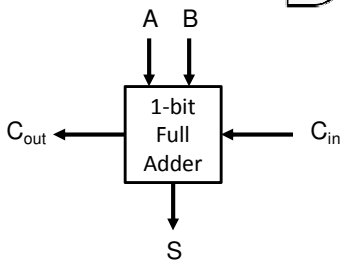




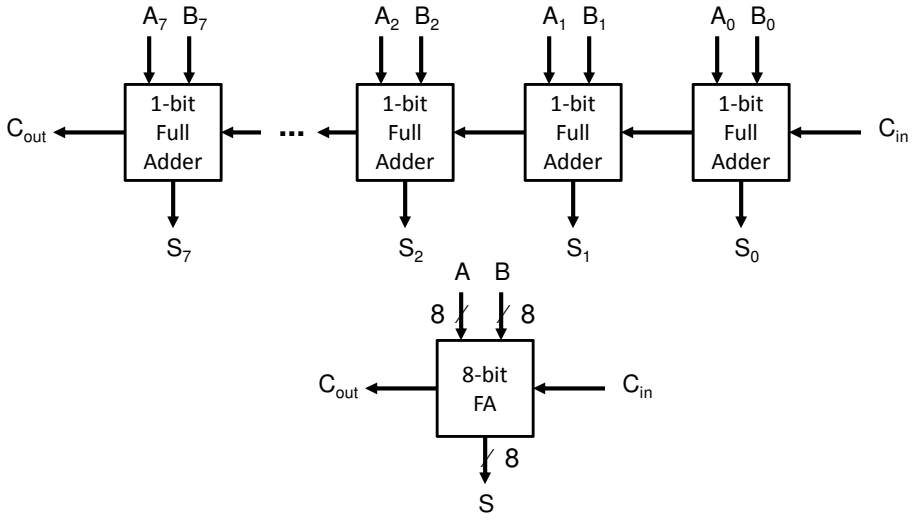
Another Full Adder (FA)

http://students.cs.tamu.edu/wanglei/csce350/handout/lab6.html



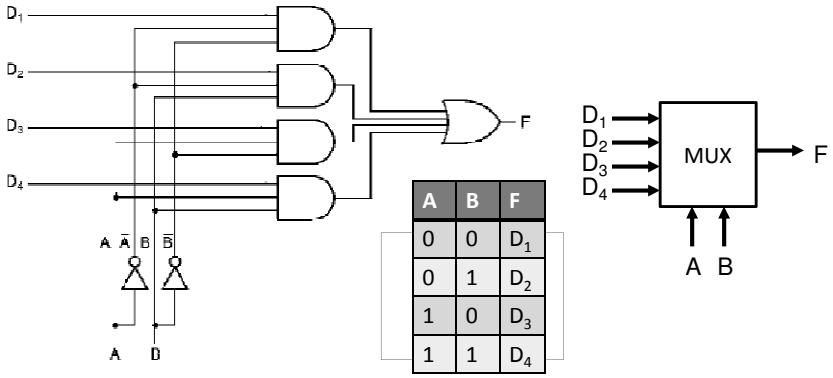


8-bit Full Adder



Multiplexer (MUX)

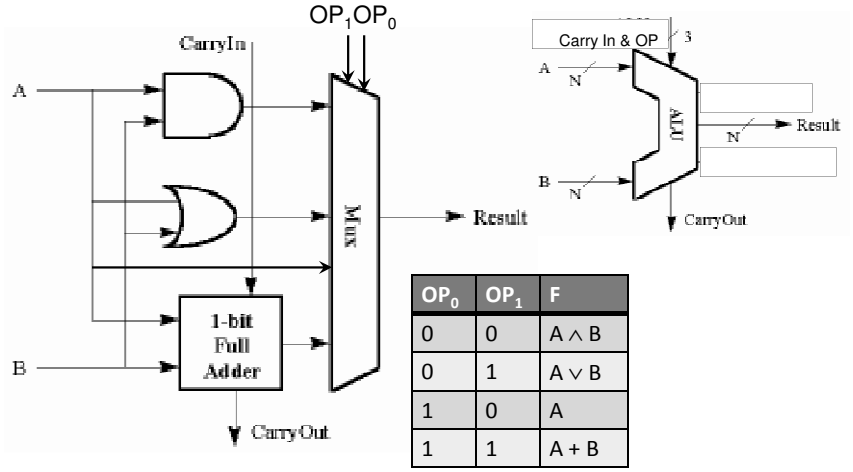
- A multiplexer chooses between a set of inputs.



A	B	F
0	0	D_1
0	1	D_2
1	0	D_3
1	1	D_4

http://www.cise.ufl.edu/~mssz/CompOrg/CDAintro.html

Arithmetic Logic Unit (ALU)

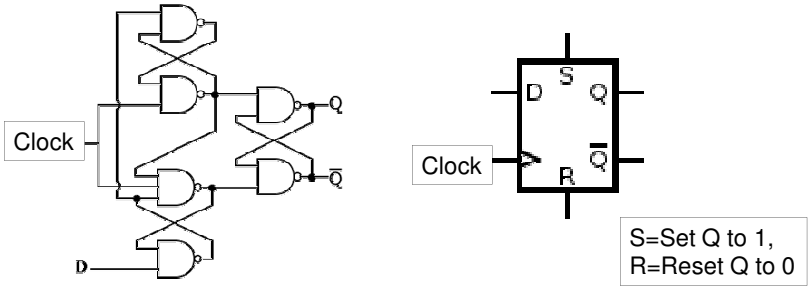


OP_0	OP_1	F
0	0	$A \wedge B$
0	1	$A \vee B$
1	0	A
1	1	A + B

http://cs-alb-pc3.massey.ac.nz/notes/59304/l4.html

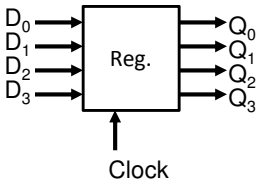
Flip Flop

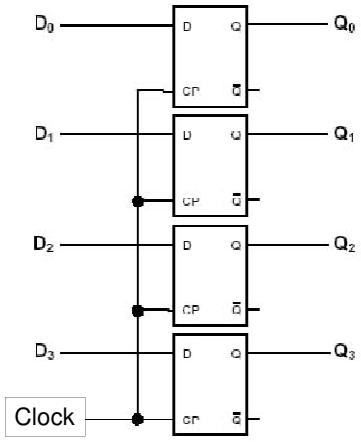
- A flip flop is a sequential circuit that is able to maintain (save) a state.
  - Example: D (Data) Flip-Flop – sets output Q to input D when clock turns on. (Images from Wikipedia)



Registers

- A register is just a set of edge-triggered flip-flops. Registers are triggered by a clock signal.



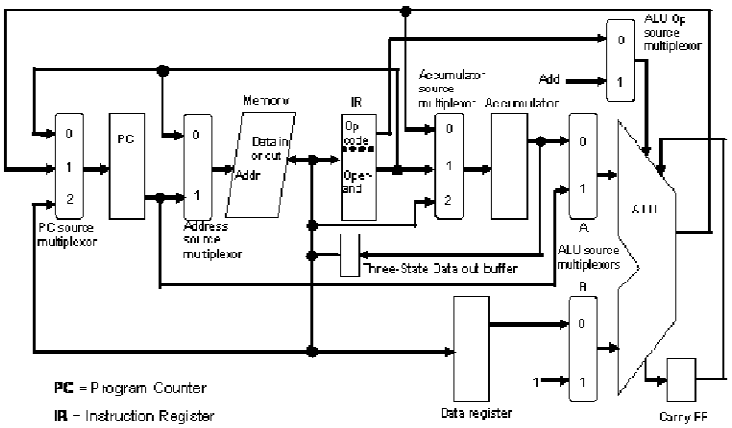


http://cpuville.com/register.htm

Central Processing Unit (CPU)

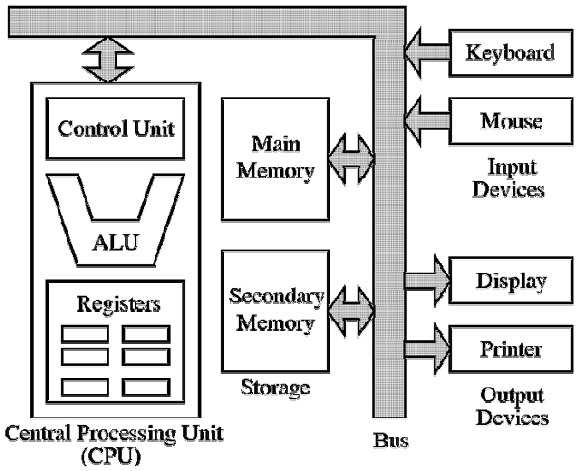
- A CPU contains:
  - Arithmetic Logic Unit to perform computation
  - Registers to hold information
    - Instruction register (current instruction being executed)
    - Program counter (to hold location of next instruction in memory)
    - Accumulator (to hold computation result from ALU)
    - Data register(s) (to hold other important data for future use)
  - Control unit to regulate flow of information and operations that are performed at each instruction step

A sample CPU



http://cpuville.com/main.htm

Computer



http://cse.iitkgp.ac.in/pds/notes/intro.html

Abstraction

- We can use layers of abstraction to hide details of the computer design.
- We can work in any layer, not needing to know how the lower layers work or how the current layer fits into the larger system.

  -> transistors
  -> gates
  -> circuits (adders, multiplexors, flip-flops)
  -> central processing units (ALU, registers, control)
  -> computer