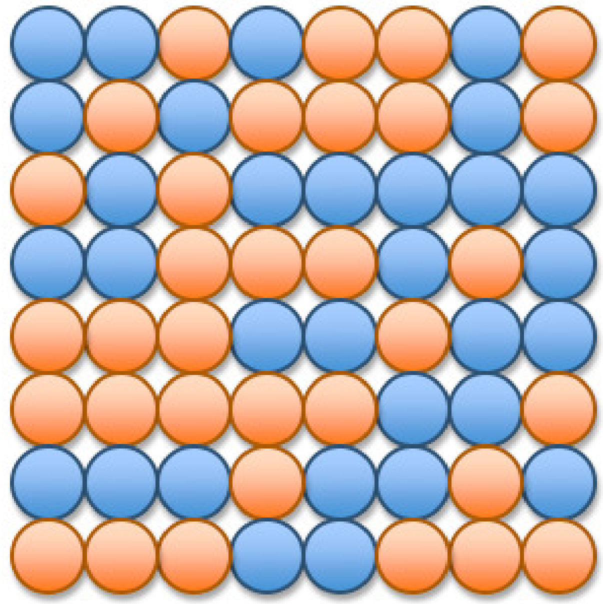


UNIT 8B

Computer Organization: Levels of Abstraction

Announcements

- Midsemester grades will be assigned this weekend
- Ps7 is due Friday March 22 in class
- Pa7 will be assigned shortly (due after break)

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)

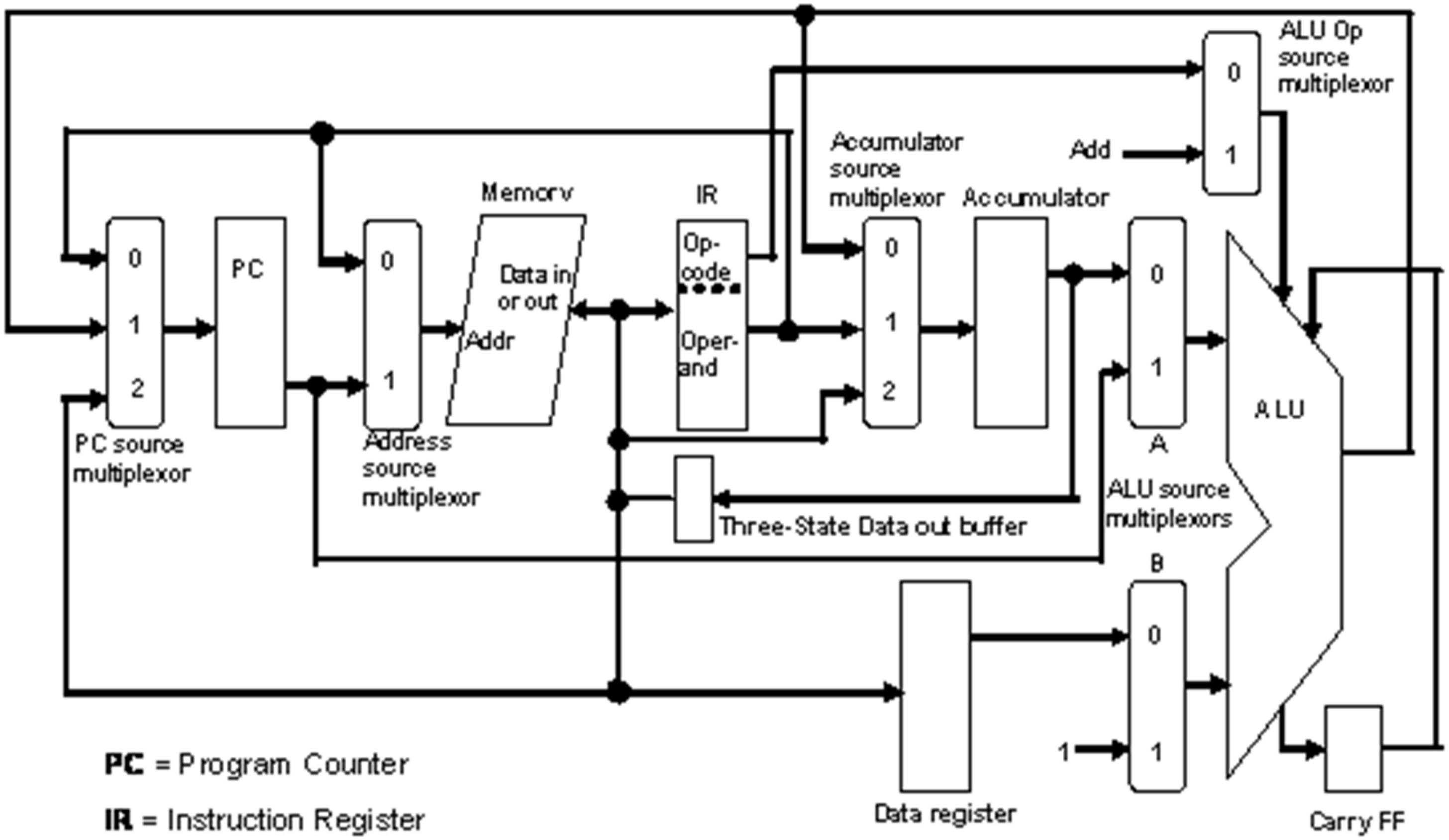
  -> central processing units (ALU, registers, control)

  -> computer

Central Processing Unit (CPU)

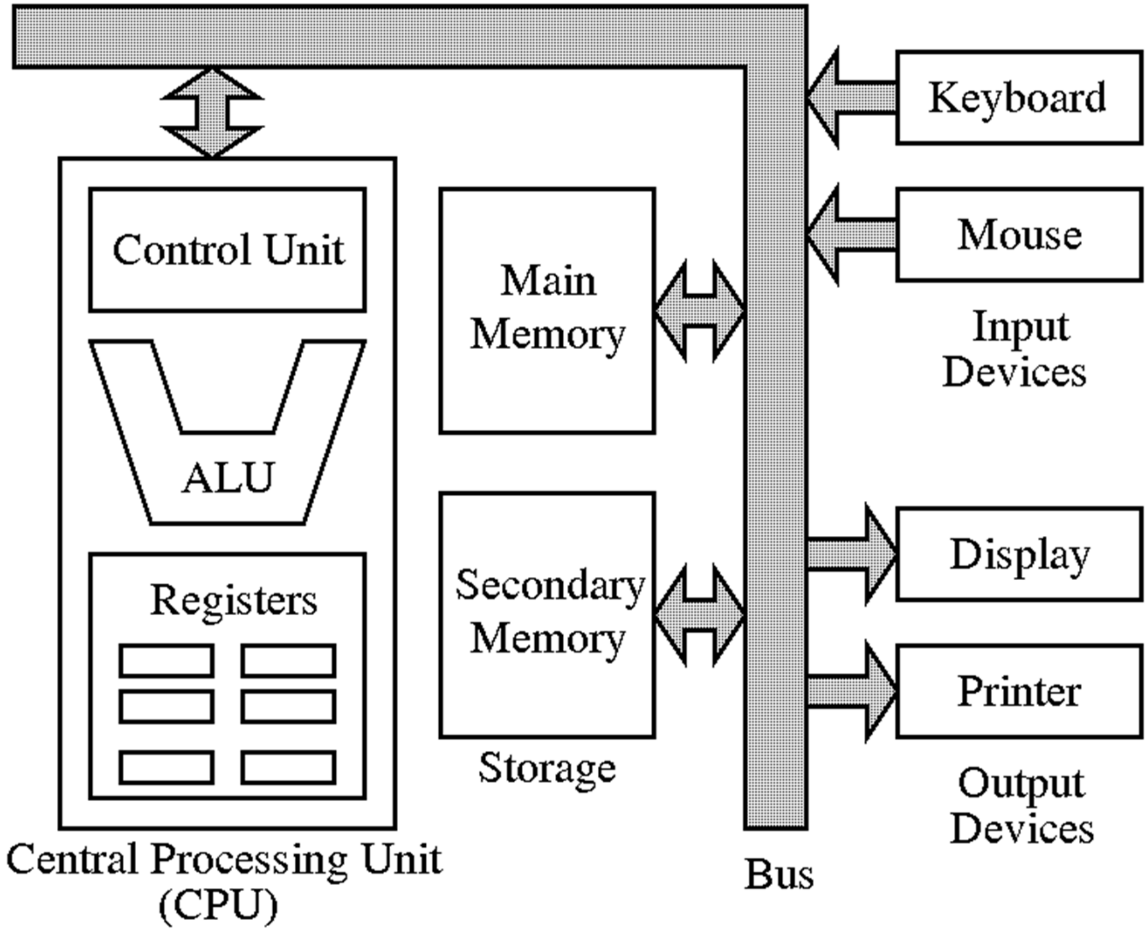
- A CPU contains:
  - Arithmetic Logic Unit to perform computation
  - Registers to hold information
    - Instruction register (current instruction being executed)
    - Program counter (to hold location of next instruction in memory)
    - Accumulator (to hold computation result from ALU)
    - Data 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



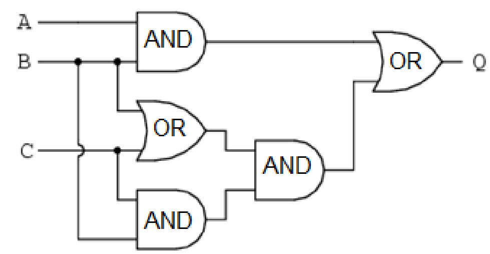
http://cse.iitkgp.ac.in/pds/notes/intro.html

To Gates

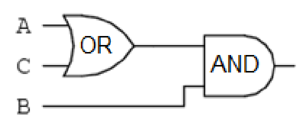
Properties (Similar to × and +)

- Commutative: $a \wedge b = b \wedge a$ $\quad a \vee b = b \vee a$
- 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: $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$
  $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ ← Not true for + and ×
- Identity: $a \wedge 1 = a$ $\quad a \vee 0 = a$
- Dominance: $a \wedge 0 = 0$ $\quad a \vee 1 = 1$
- Idempotence: $a \wedge a = a$ $\quad a \vee a = a$
- Complementation: $a \wedge \neg a = 0$ $\quad a \vee \neg a = 1$
- Double Negation: $\neg \neg a = a$

equivalence



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

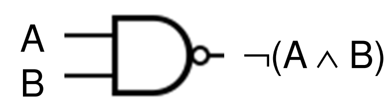


$Q = B \wedge (A \vee C)$

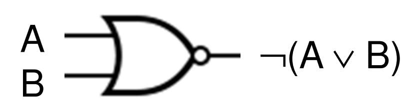
More gates

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

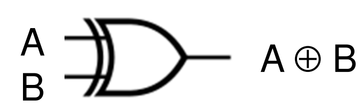
- nand (“not and”): A nand B = not (A and B)



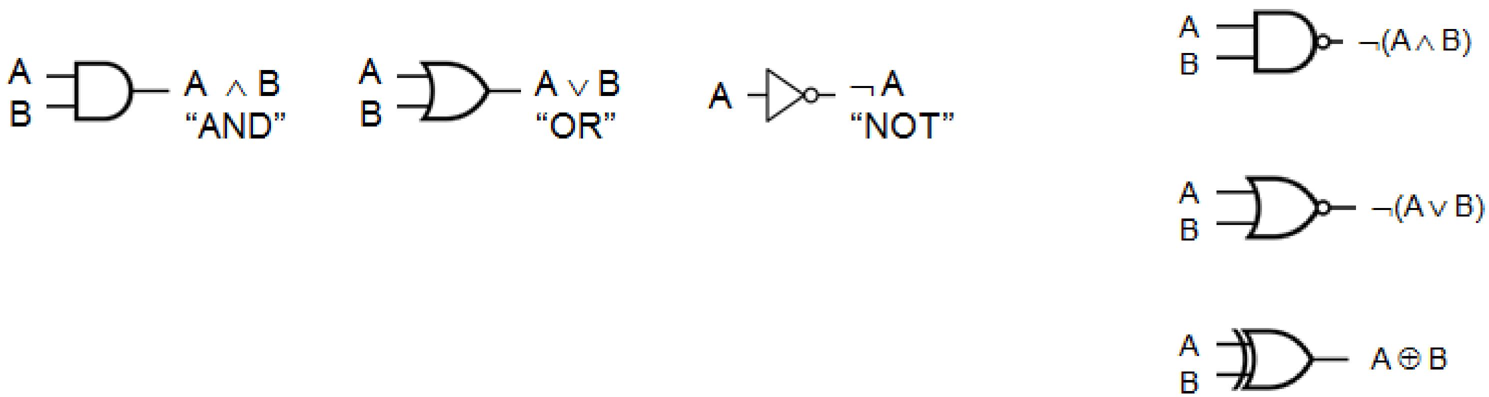
- nor (“not or”): A nor B = not (A or B)



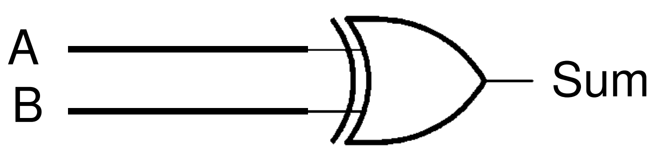
- xor (“exclusive or”):
  A xor B = (A and not B) or (B and not A)



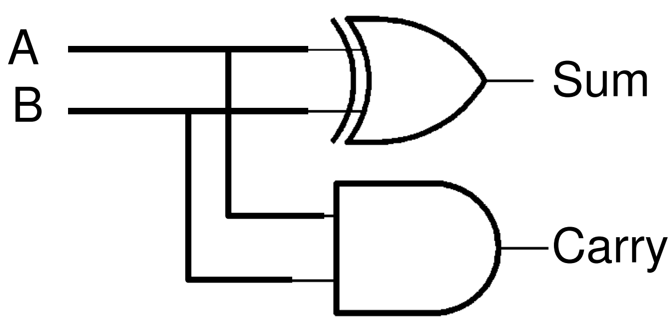
All Gates



Adding Binary Numbers

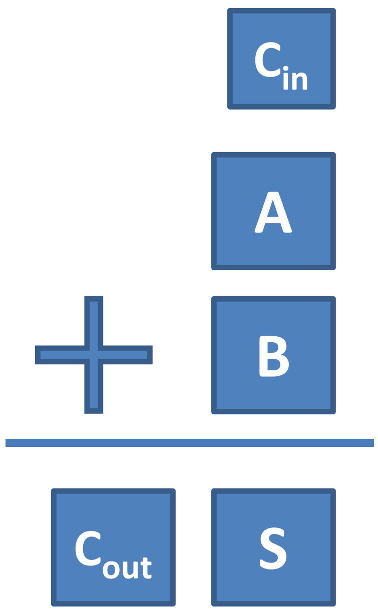


Adding Binary Numbers



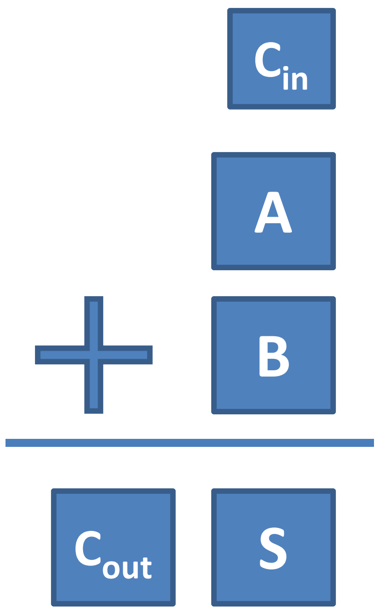
Half Adder

A Full Adder



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

A Full Adder

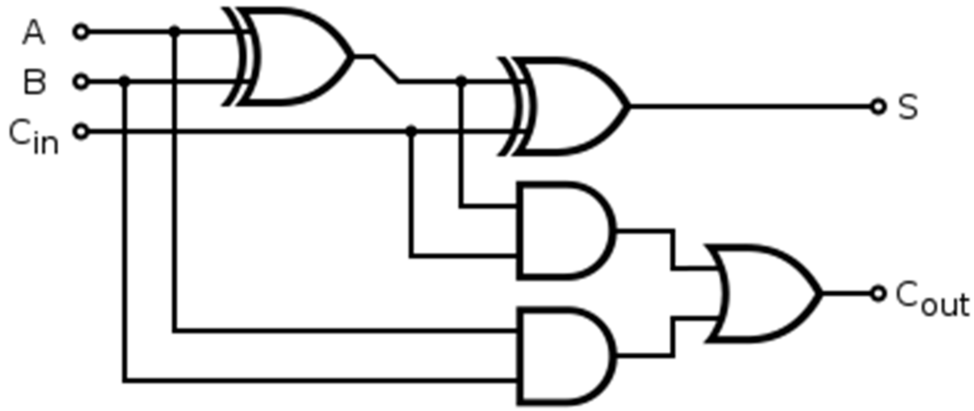


A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

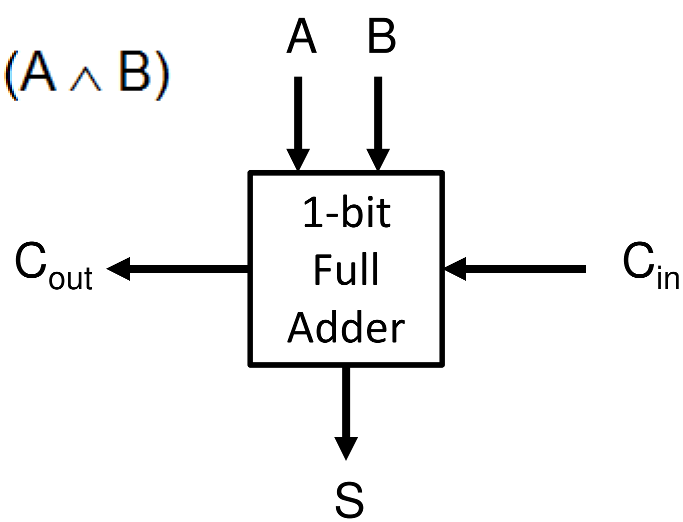
$$S = A \oplus B \oplus C_{in}$$

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

Full Adder (FA)

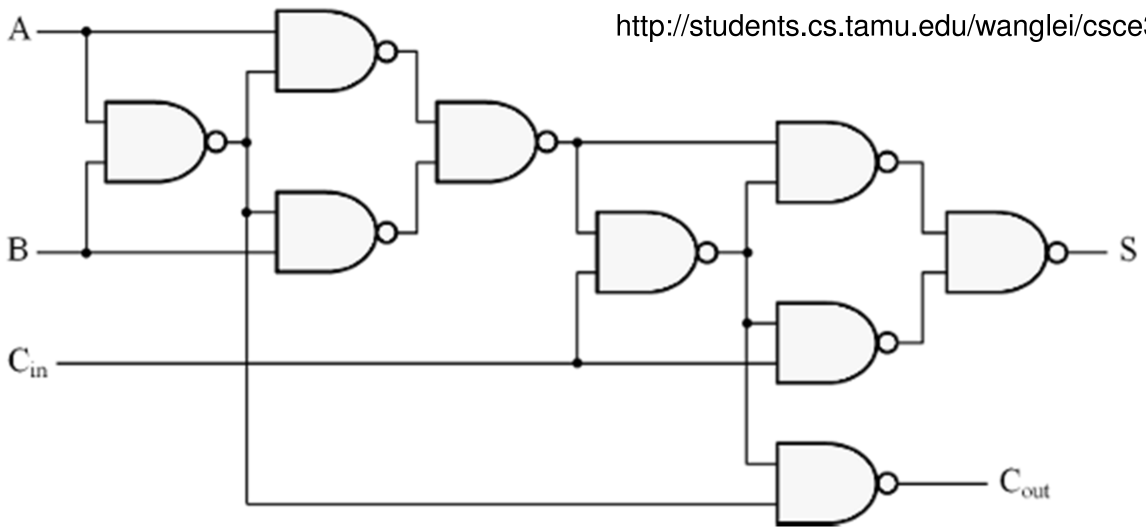


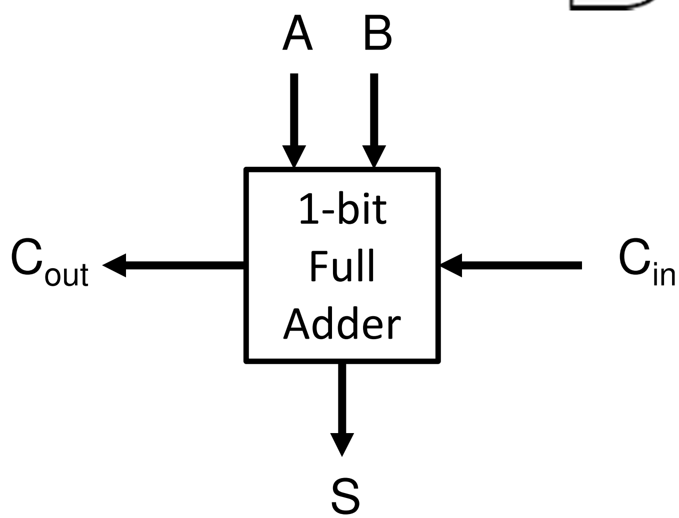
$S = A \oplus B \oplus C_{in}$
$C_{out} = ((A \oplus B) \wedge C) \vee (A \wedge B)$



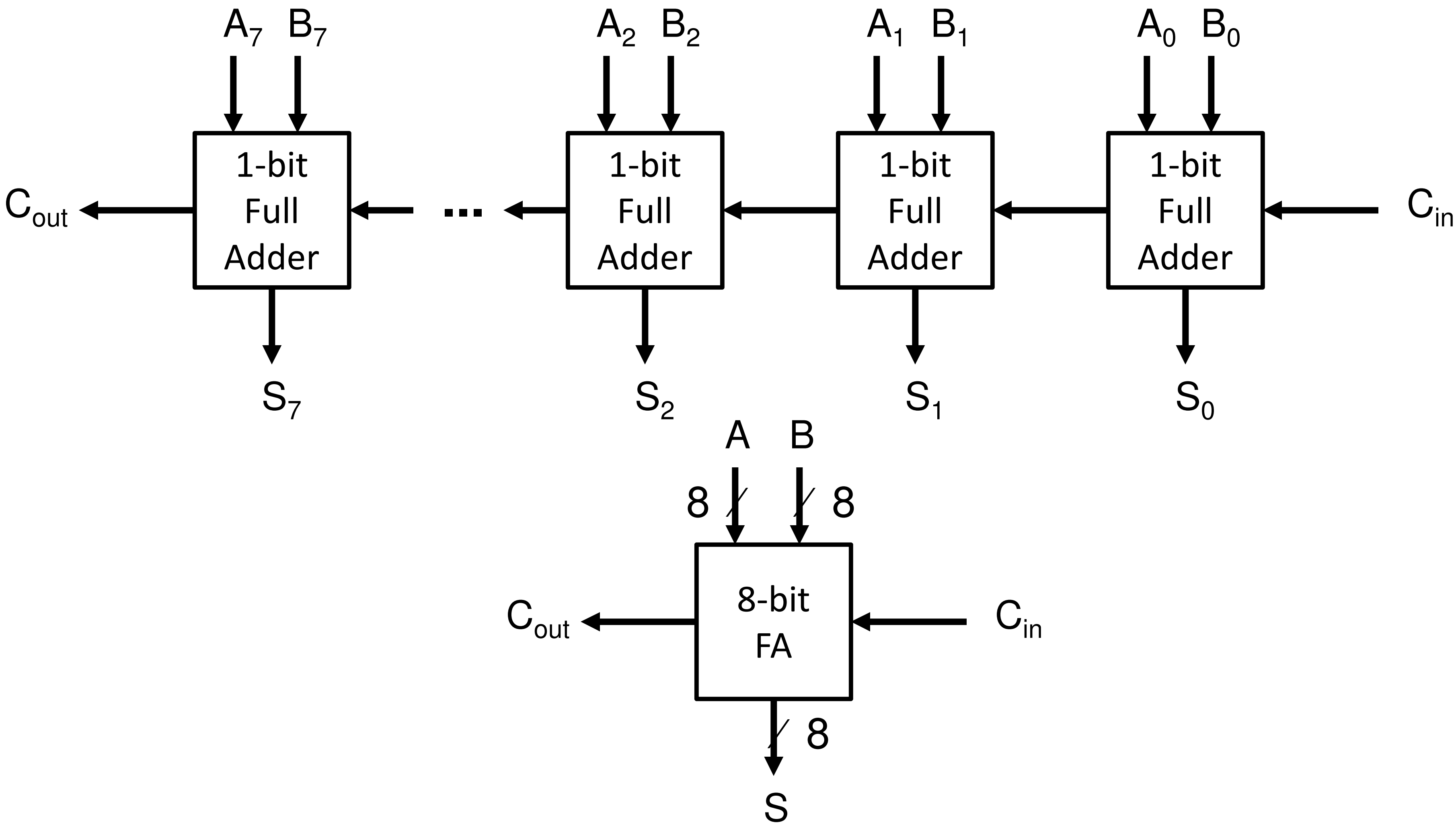
Another Full Adder (FA)

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

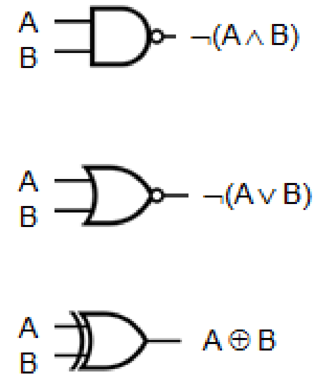




8-bit Full Adder

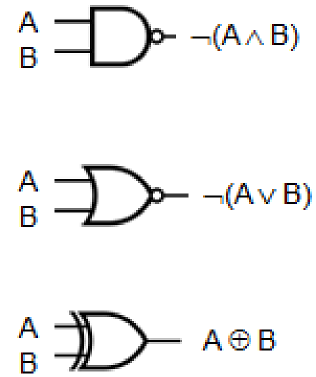


Exercise



- Convert expressions to circuit

Exercise



- Convert circuit to expressions

exercise

- Designing a control system

From Design to circuits

A	B	C	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1