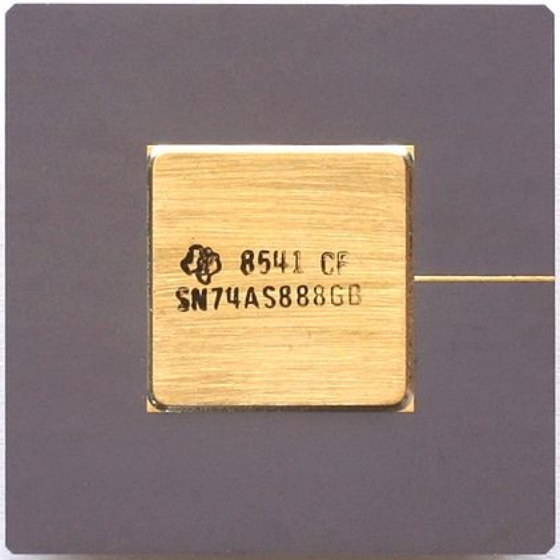
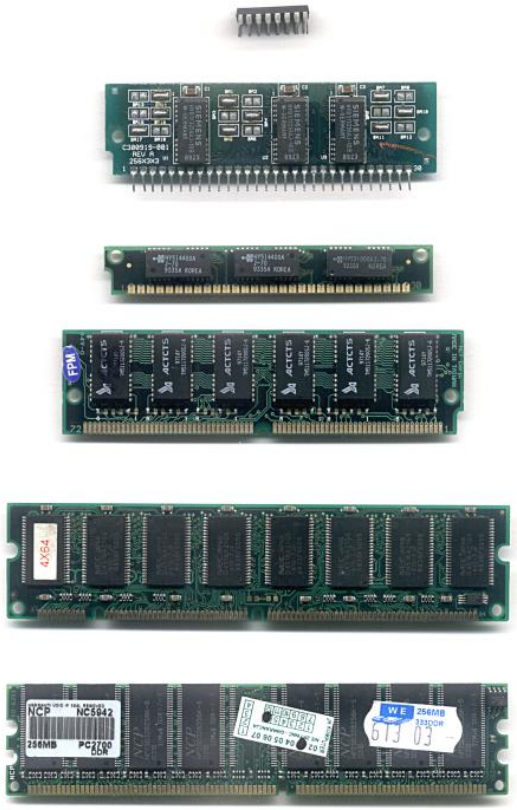
Arithmetic logic units and memory





Layers of abstraction

abstraction	building blocks	examples
computer	**components**	**Macbook Pro**
components	**multiplexer, demultiplexer, adder, flip-flop**	**registers, ALU, counter**
sequential circuits	logic gates, connectors, clock	flip-flop
combinational circuit	logic gates, connectors	multiplexer, demultiplexer, adder
logic gates	switches, connectors	AND, OR, NOT
clock	raw materials	crystal oscillator
connector	raw materials	Wire
switch	raw materials	transistor, relay

ALU

- ALU (Arithmetic Logic Unit)
  - Building block of the CPU
  - Normally at least:
    - Bitwise AND, NOT, OR, XOR
    - Integer addition, subtraction
    - Bit shifting
  - Input from memory register(s)
  - Output to memory register
  - Which operation stored determined by signal from control unit

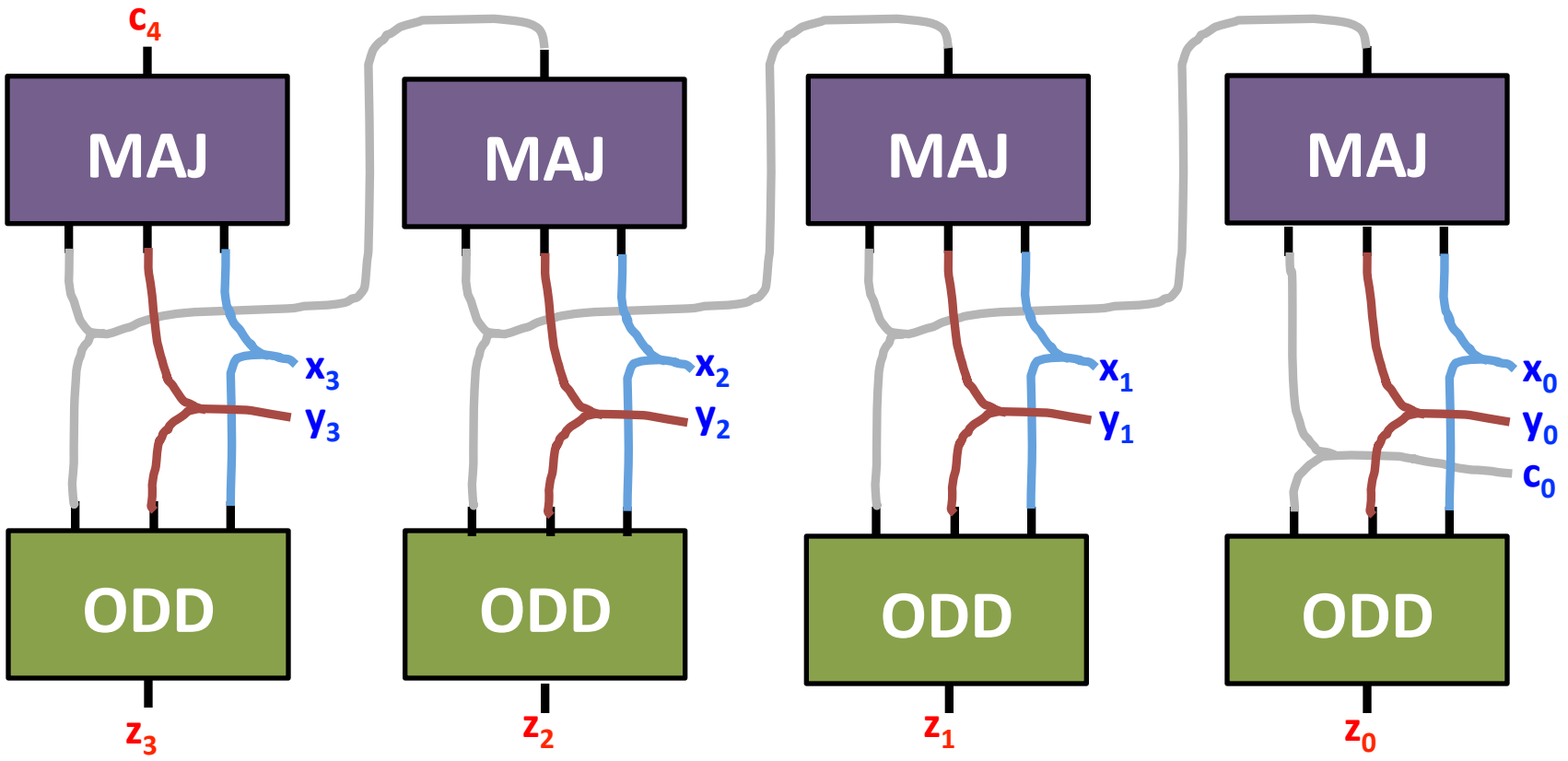
Let's build an ALU

- ALU (Arithmetic Logic Unit)
  - 4 different operations
    - Add, subtract, bitwise AND, bitwise XOR
  - Two 4-bit inputs
  - One 4-bit output
  - ALU performs all 4 operations in parallel
  - 2 selection bits select result put on output wires

4-bit full adder

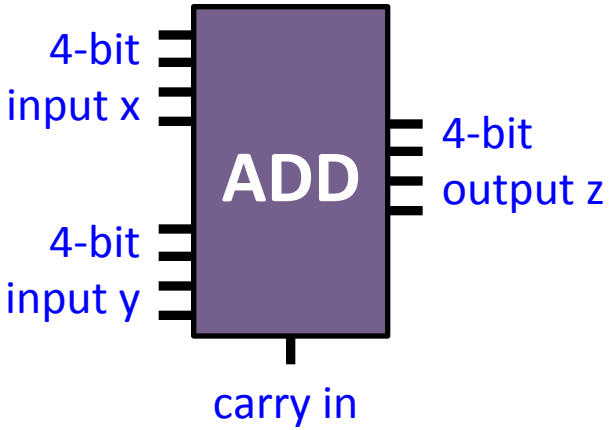
- Goal: x + y = z for 4-bit integers

				c_0
	x_3	x_2	x_1	x_0
+	y_3	y_2	y_1	y_0
c_4	z_3	z_2	z_1	z_0



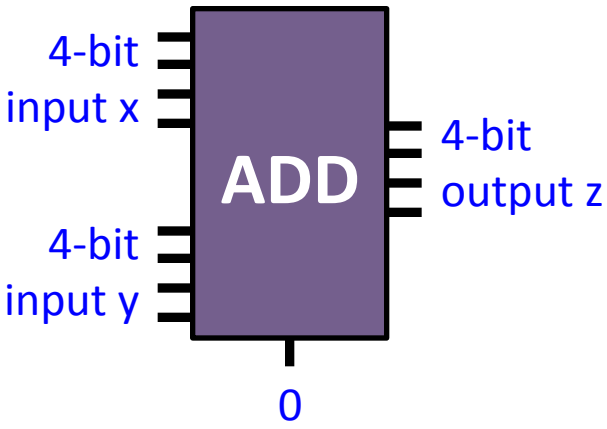
4-bit adder

				c_0
	x_3	x_2	x_1	x_0
+	y_3	y_2	y_1	y_0
c_4	z_3	z_2	z_1	z_0

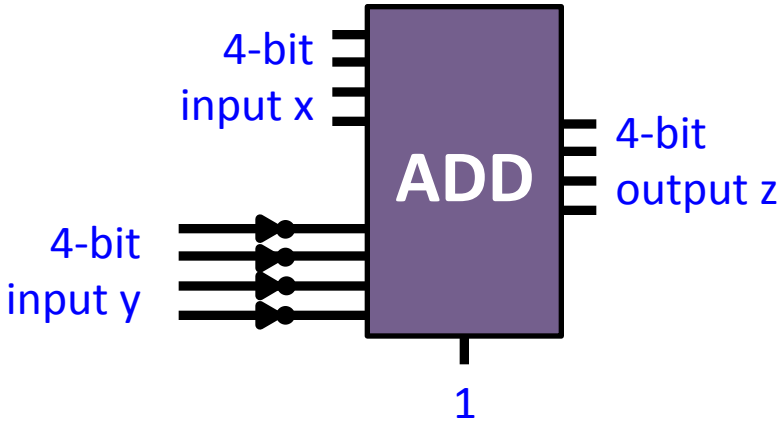


4-bit subtractor

- Goal: x - y = z for 4-bit integers
  - One approach: design like adder circuit
  - Better approach: reuse the adder
  - Two's complement, invert bits and add 1

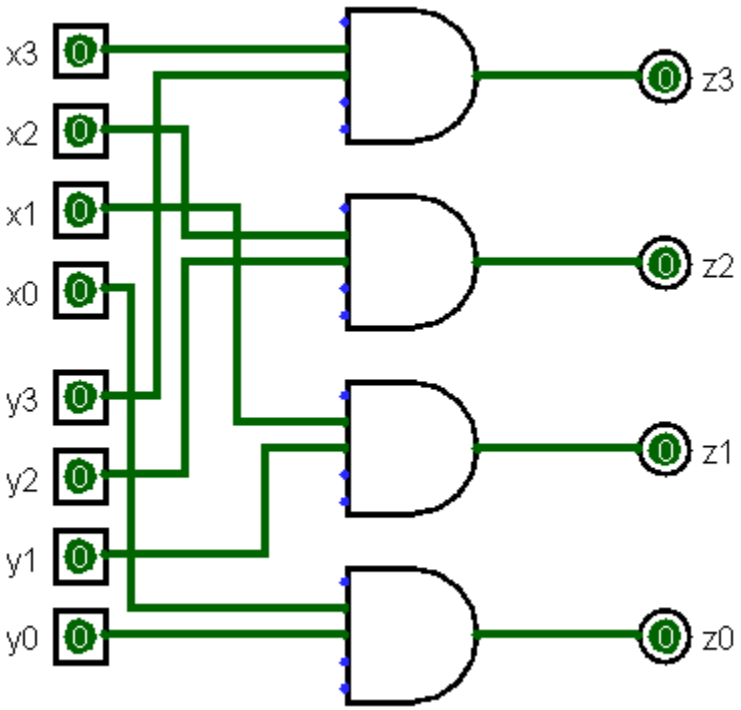


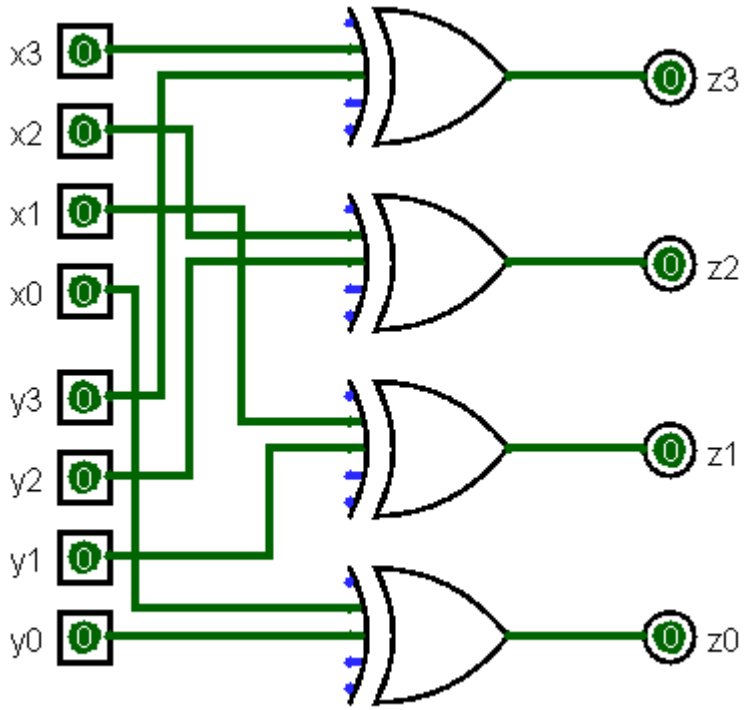
Circuit doing addition

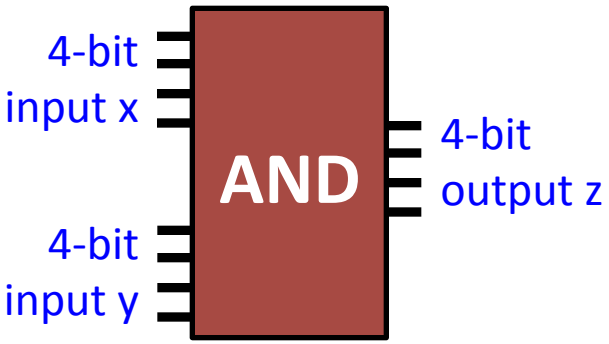


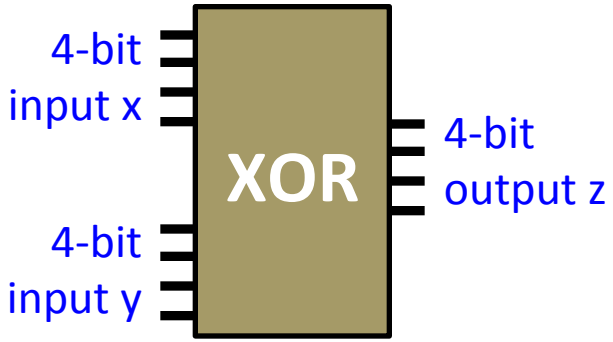
Circuit doing subtraction

Bitwise AND, XOR

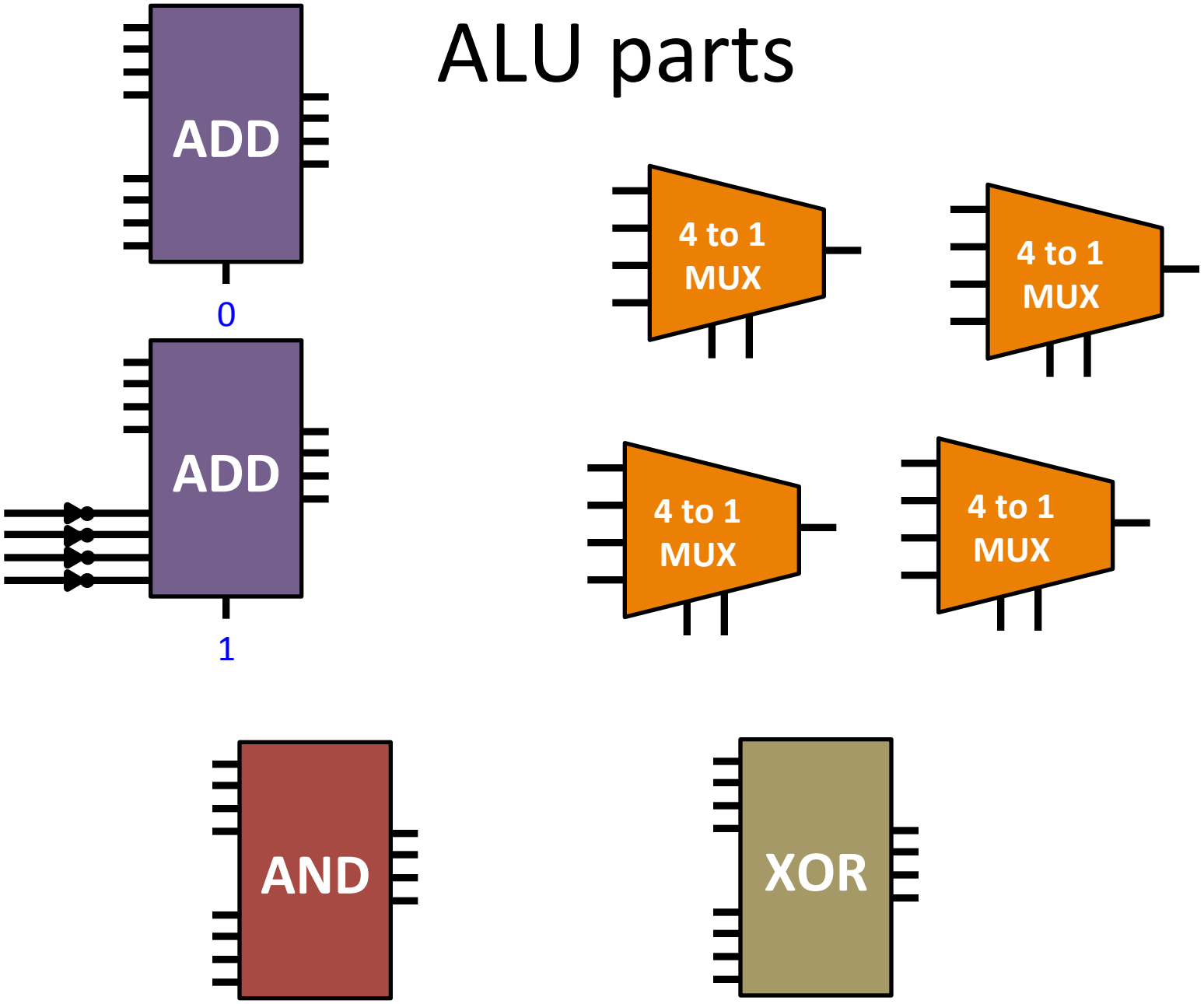




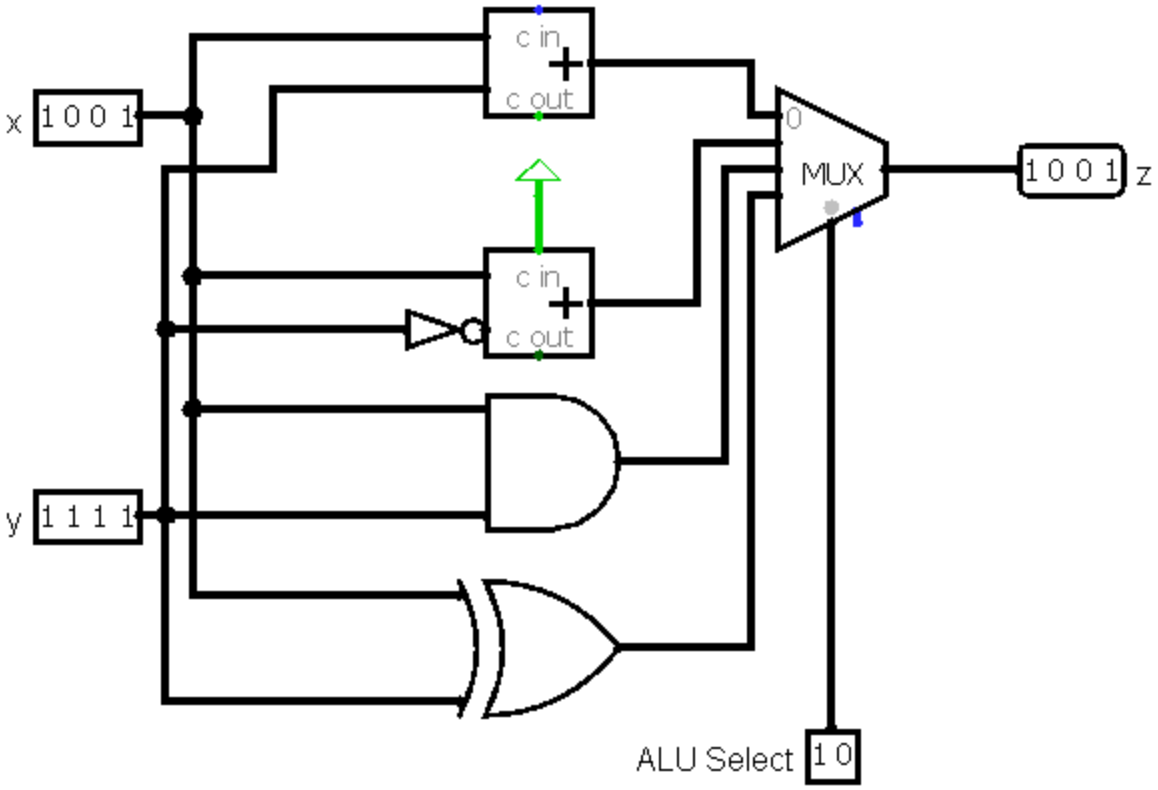




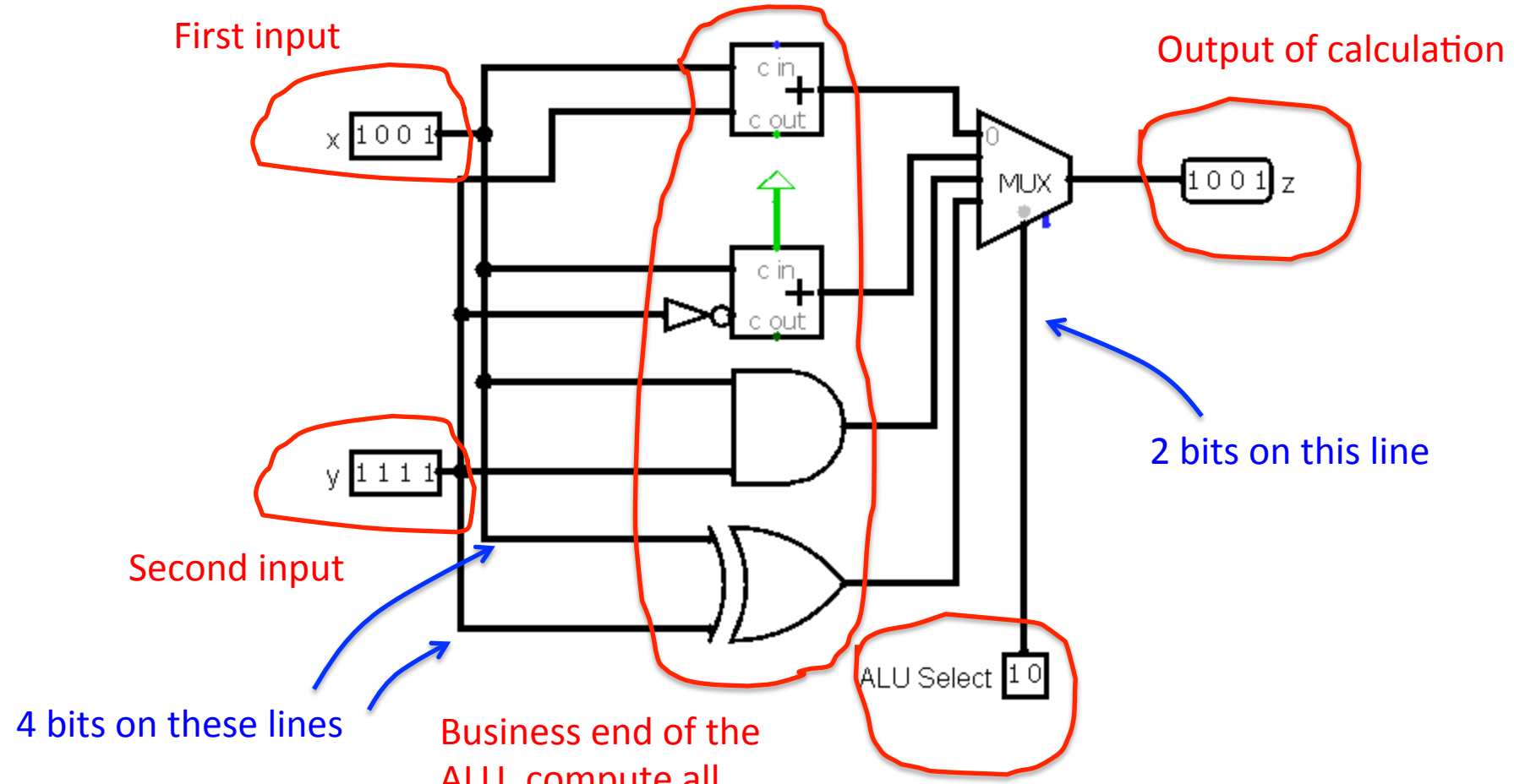
ALU parts



ALU with 4-bit lines



Our simple ALU

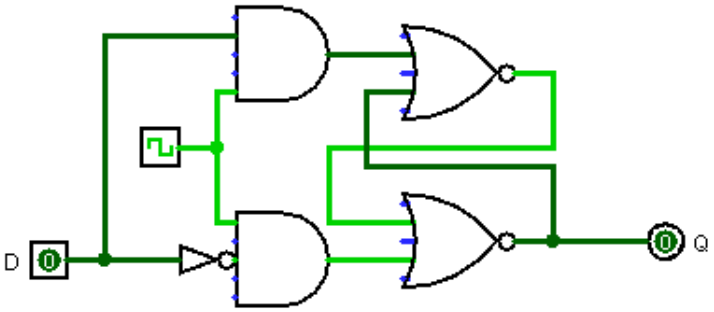


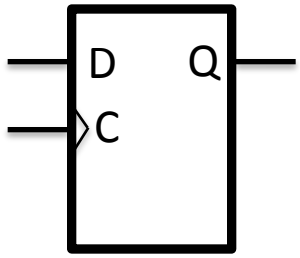
Let's build a register

- n-bit register
  - Group of n flip-flops storing n bits
  - Includes combinational gates
  - Provides output of its n stored bits
  - Can be told to memorize n bits on input lines
    - Someone sets load line high

Clocked D flip-flop

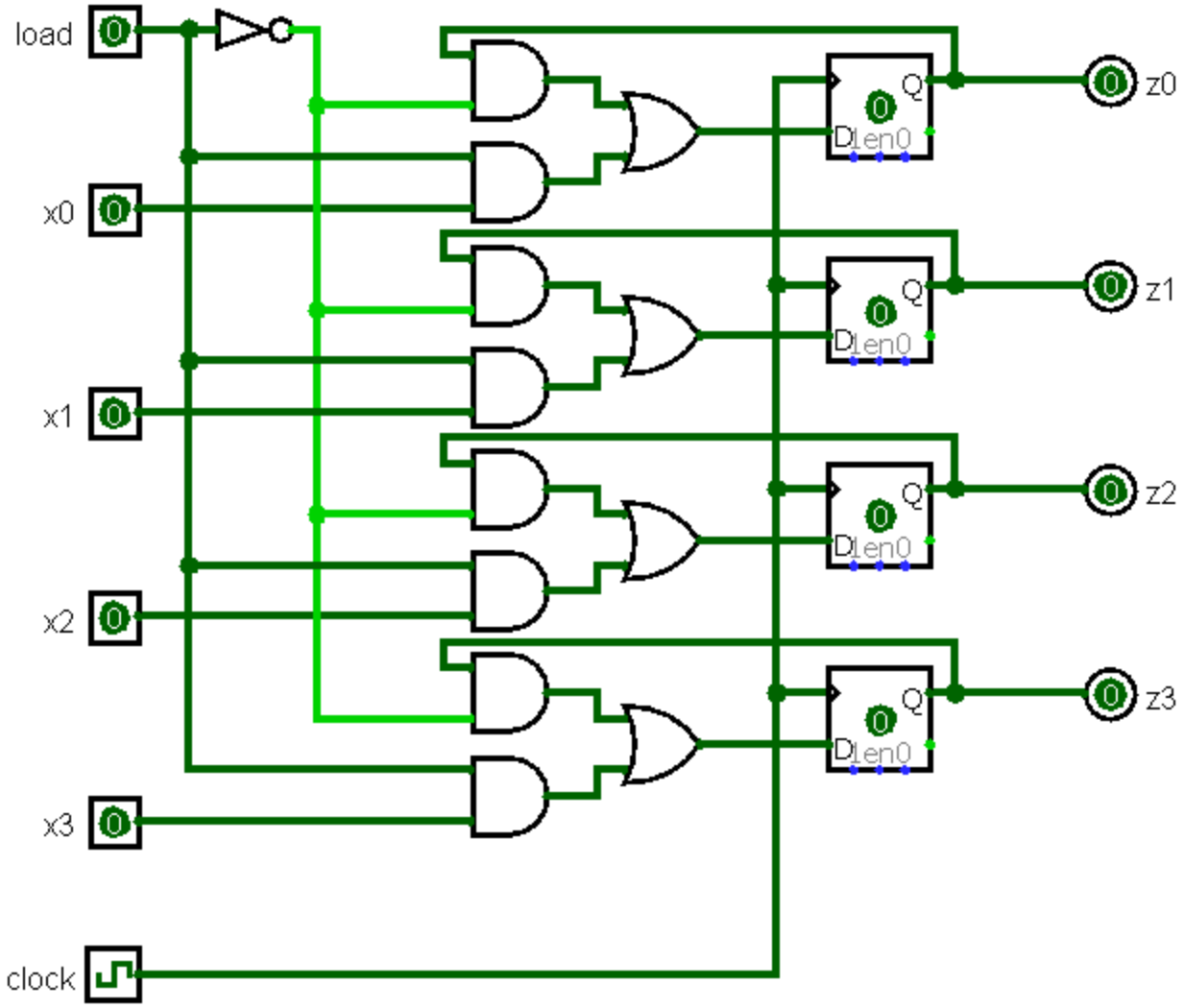
- D flip-flop
  - Hook the enable line to a clock
  - Clocked latch = flip-flop
  - State change has to wait for next clock cycle



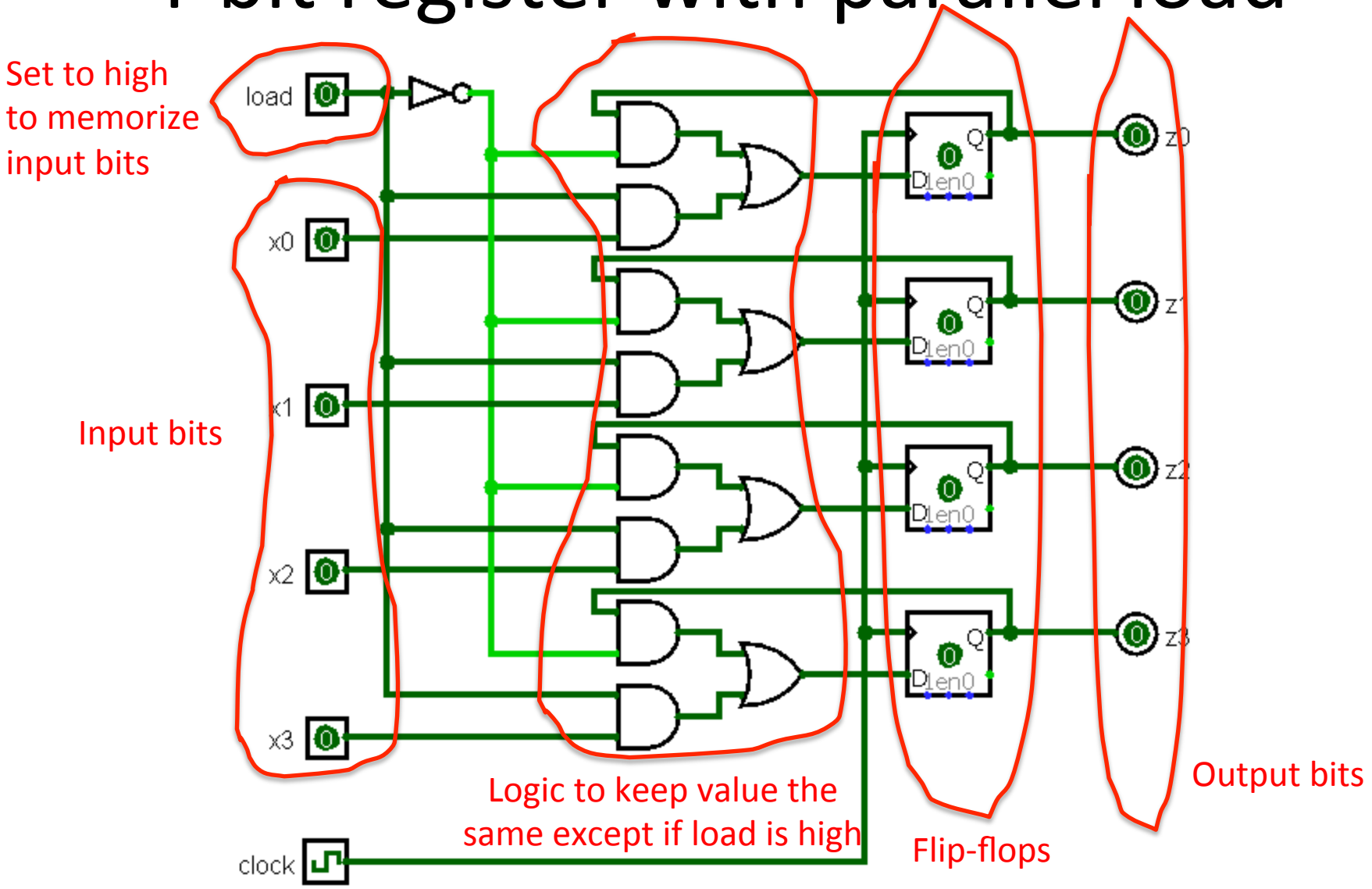


D (data)	C (clock)	Q(t)	Q(t+1)
0	high	x	0
1	high	x	1

4-bit register with parallel load



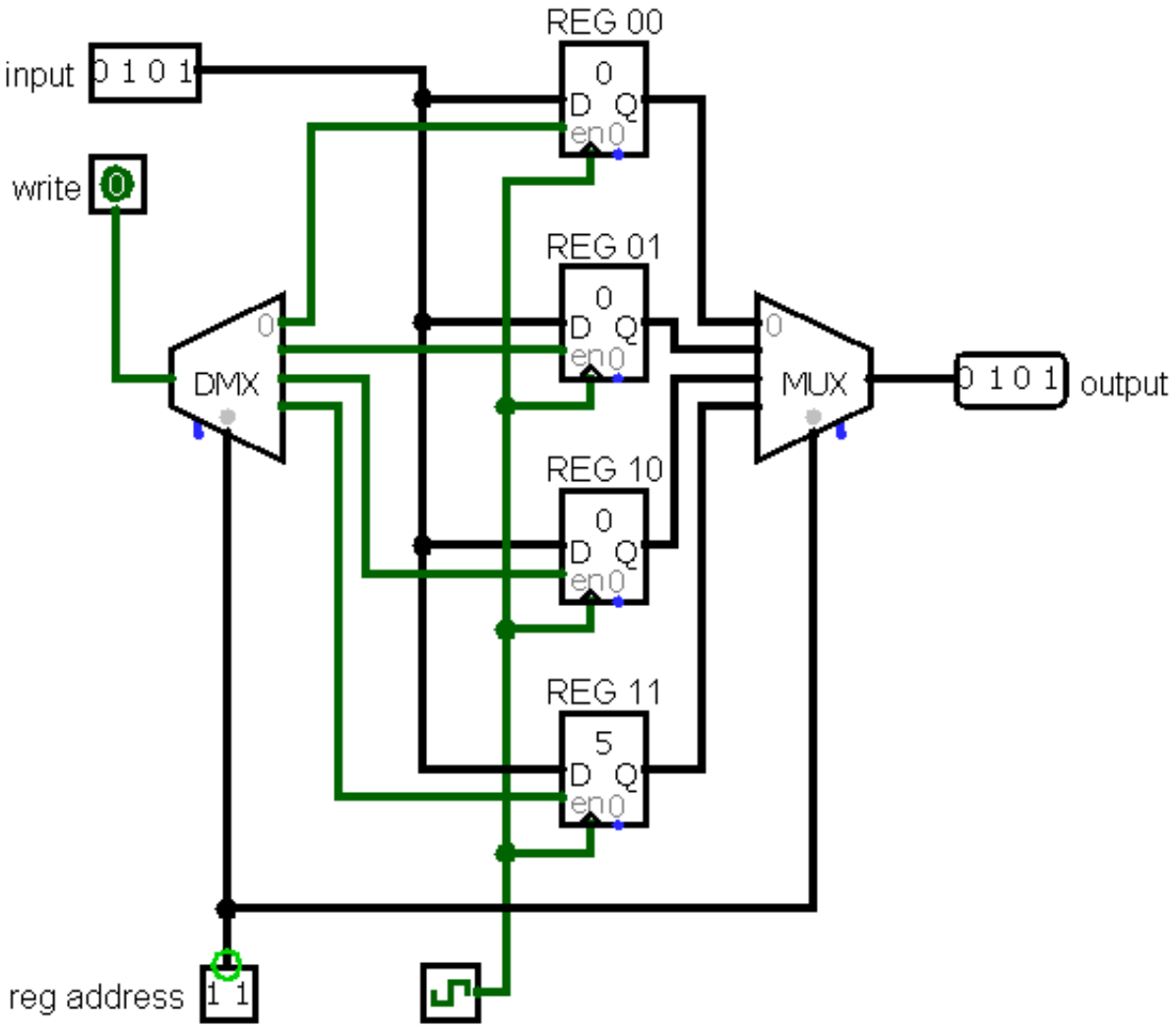
4-bit register with parallel load



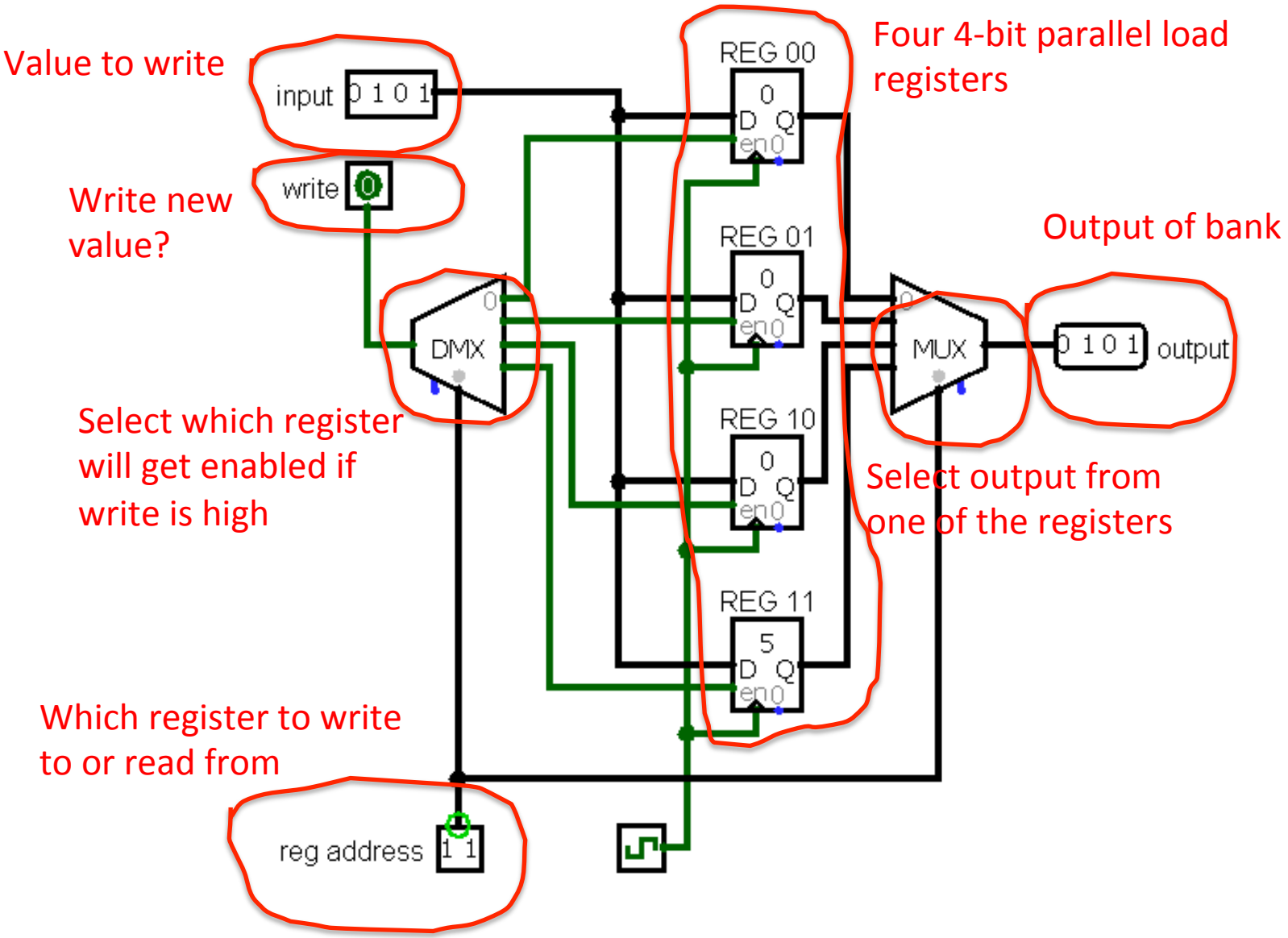
Let's build a register bank

- We need to store multiple variables
- Use four 4-bit registers
  - Someone sets target register number 0-3
  - Put target register's stored value on output line
  - If write bit enabled, make target register memorize data on input line

Four 4-bit register bank



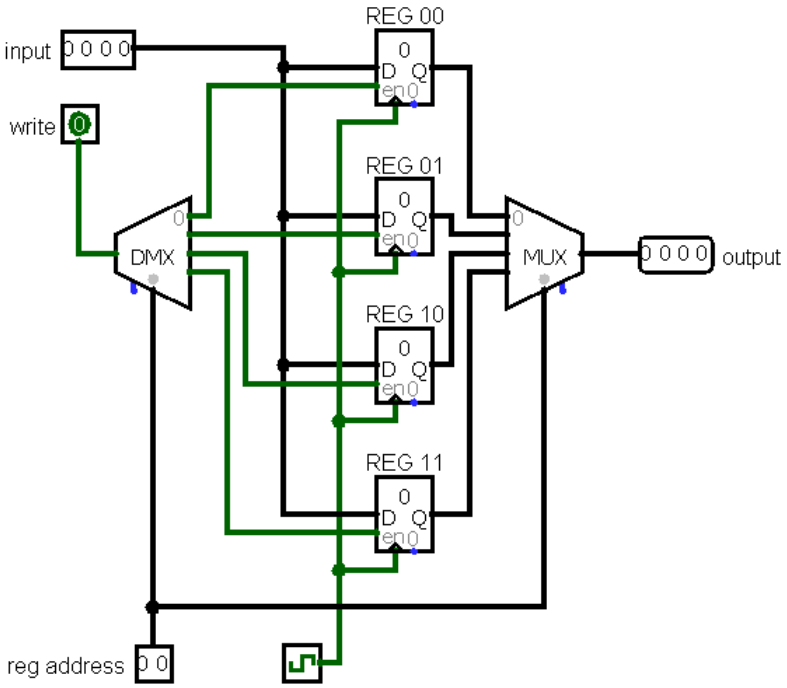
Four 4-bit register bank



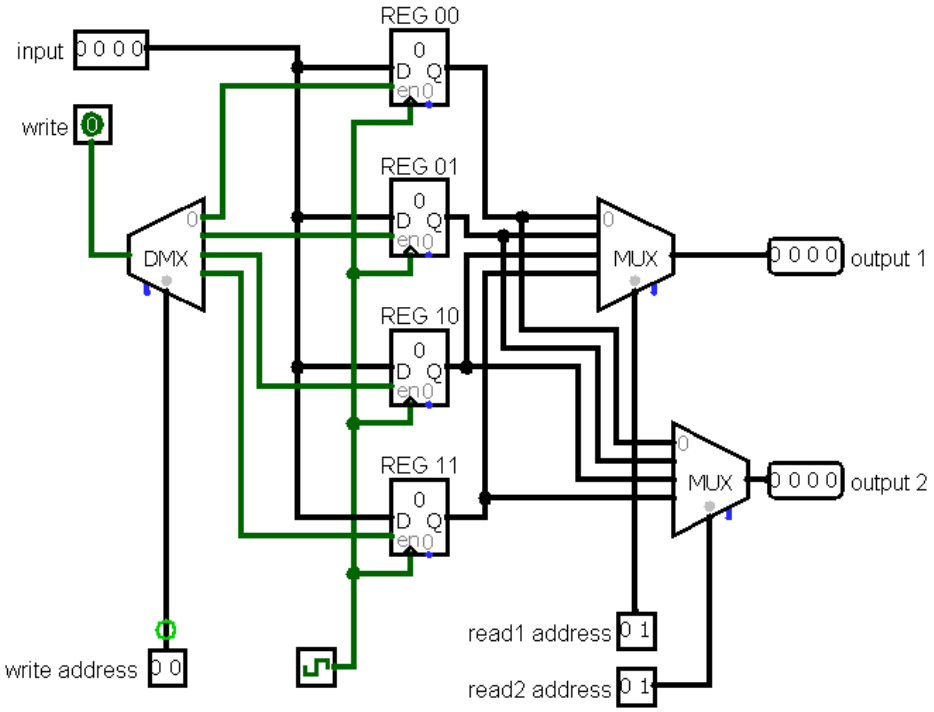
Let's build a computer (sort of)

- So far we've built:
  - ALU
    - Computes 4 operations on two 4-bit numbers
  - Register bank
    - Select one of four 4-bit numbers
- Let's hook them together!
  - Problem: ALU needs two inputs and one output

Multiport register bank

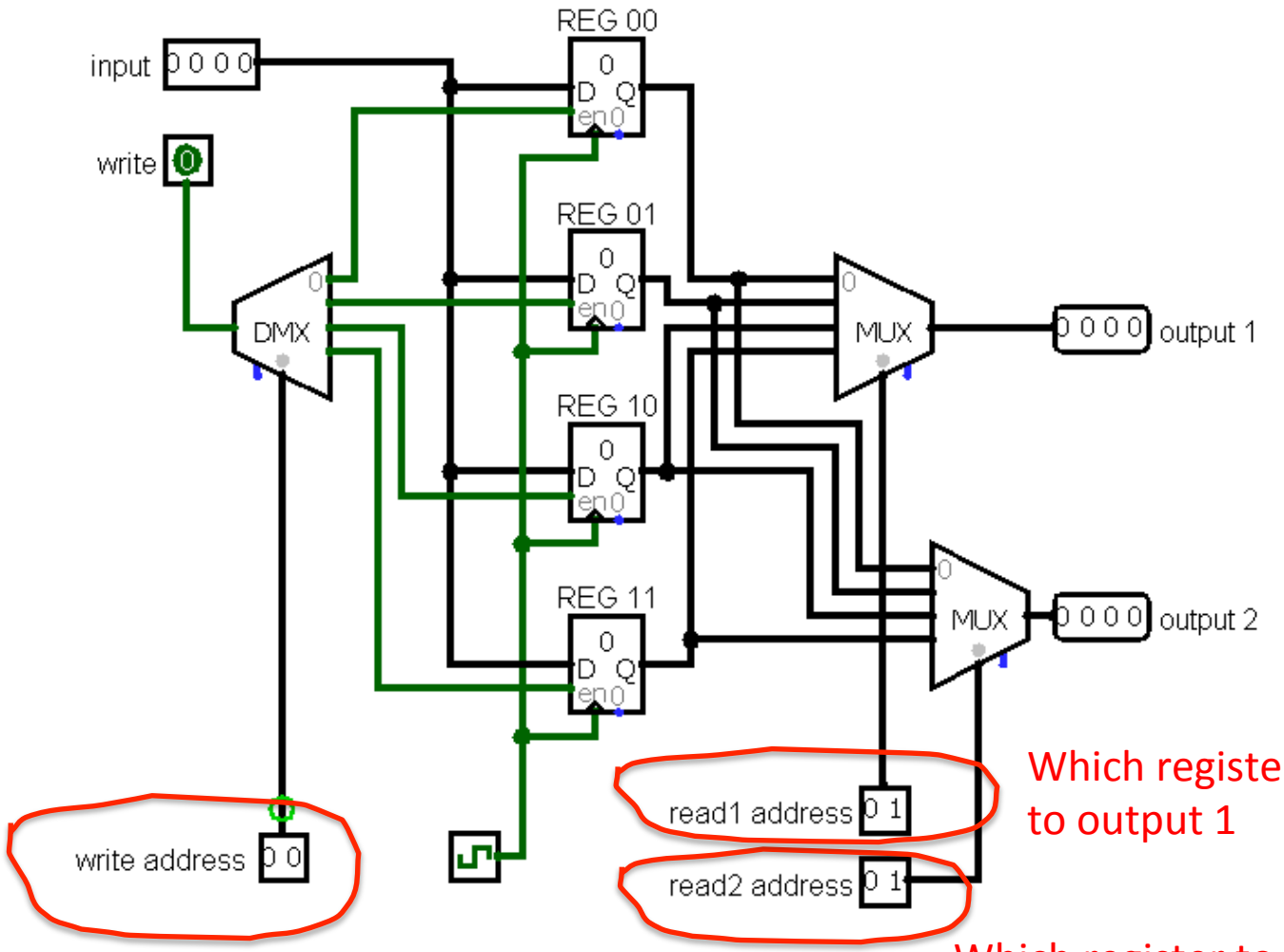


Original design of register bank



New design, can write to one address while reading from two different addresses

Multiport register bank

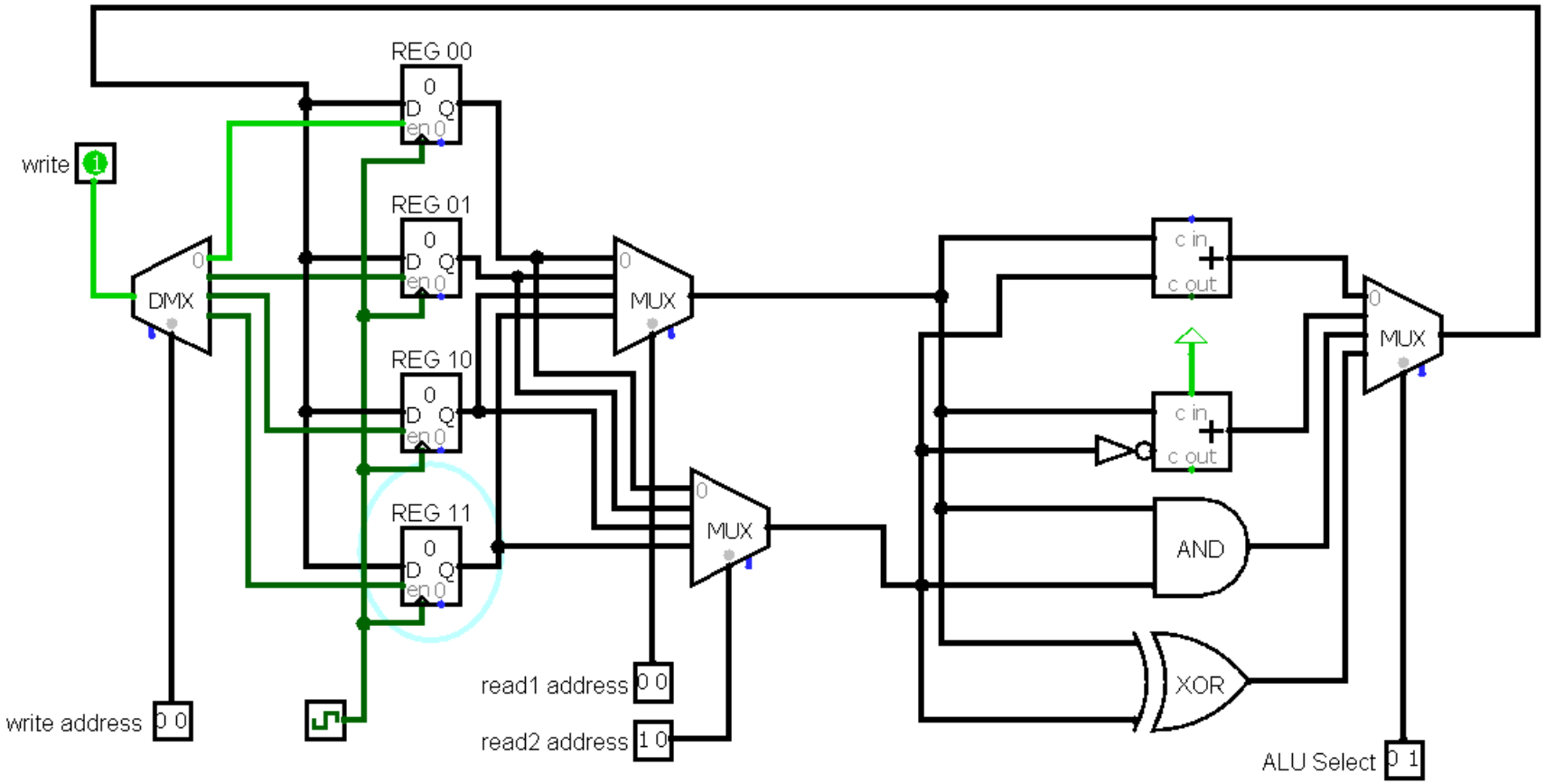


Which register to write input 1 to (if write is high)

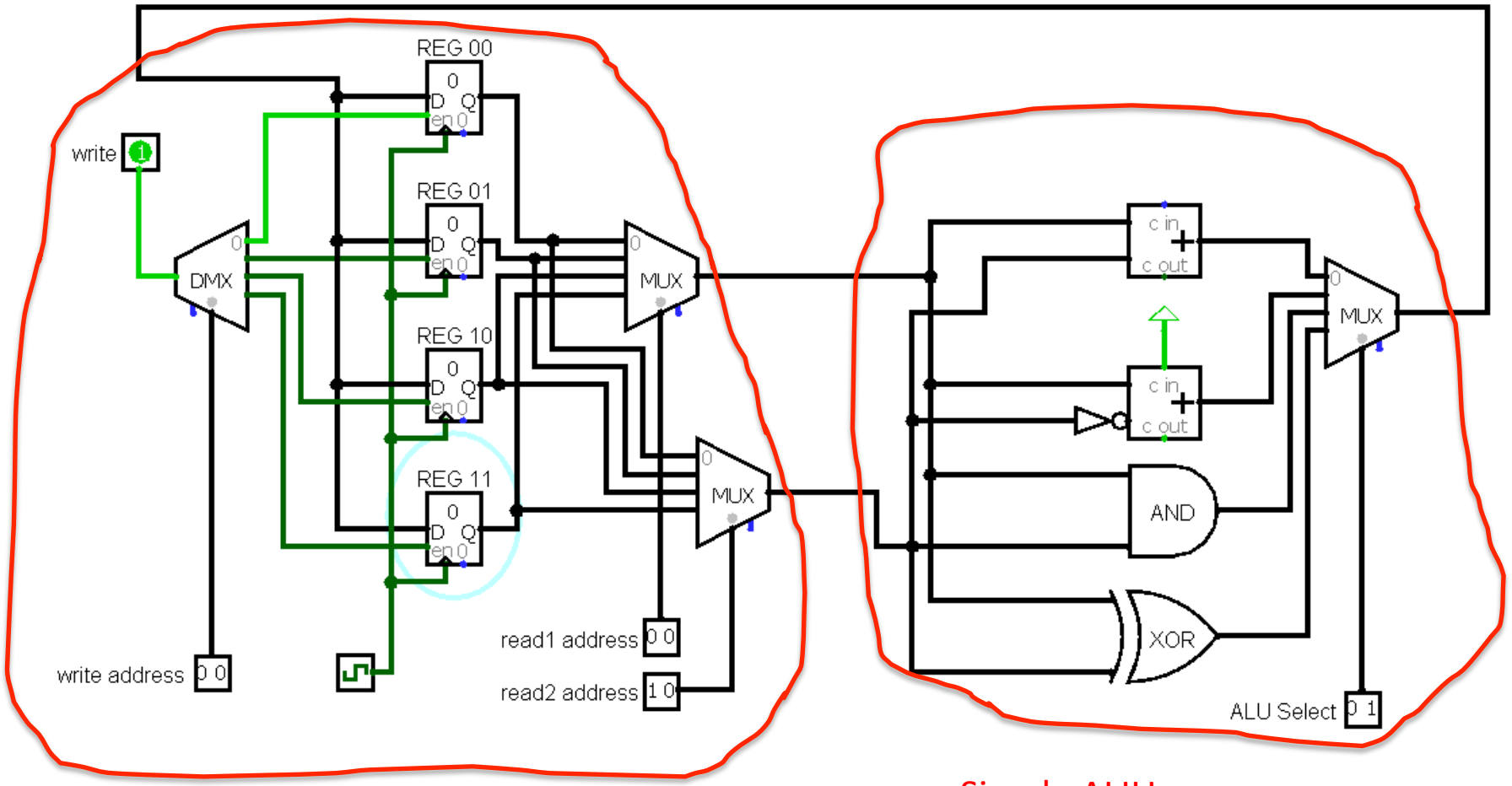
Which register to send to output 1

Which register to send to output 2

Hello computer



Hello computer



Multiport register bank

Simple ALIU

Let's build a counter

- 4-bit synchronous counter
  - Count from 0000 up to 1111
  - Roll back to 0
  - Increment on each clock cycle
- Working on an algorithm
  - Which bit always flips?
  - When do other bits flip?

Counter
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

Let's build a counter

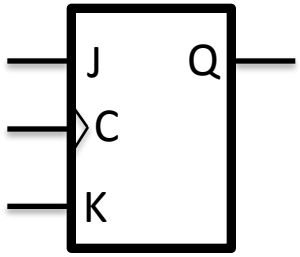
- 4-bit synchronous counter
  - Count from 0000 up to 1111
  - Roll back to 0
  - Increment on each clock cycle
- Working on an algorithm
  - Which bit always flips?
    - Least significant bit always flips
  - When do other bits flip?
    - Other bits flip when all bits to right are 1

Counter
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

JK flip-flop

- JK flip-flop
  - Refinement of SR flip-flop
  - SET: J=1, K=0
  - RESET: J=0, K=1
  - TOGGLE: J=1, K=1 (flips the stored bit)

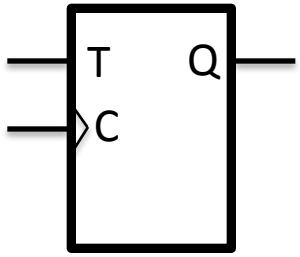
J	K	C (clock)	Q(t)	Q(t+1)
0	0	high	x	x
0	1	high	x	0
1	0	high	x	1
1	1	high	x	x’



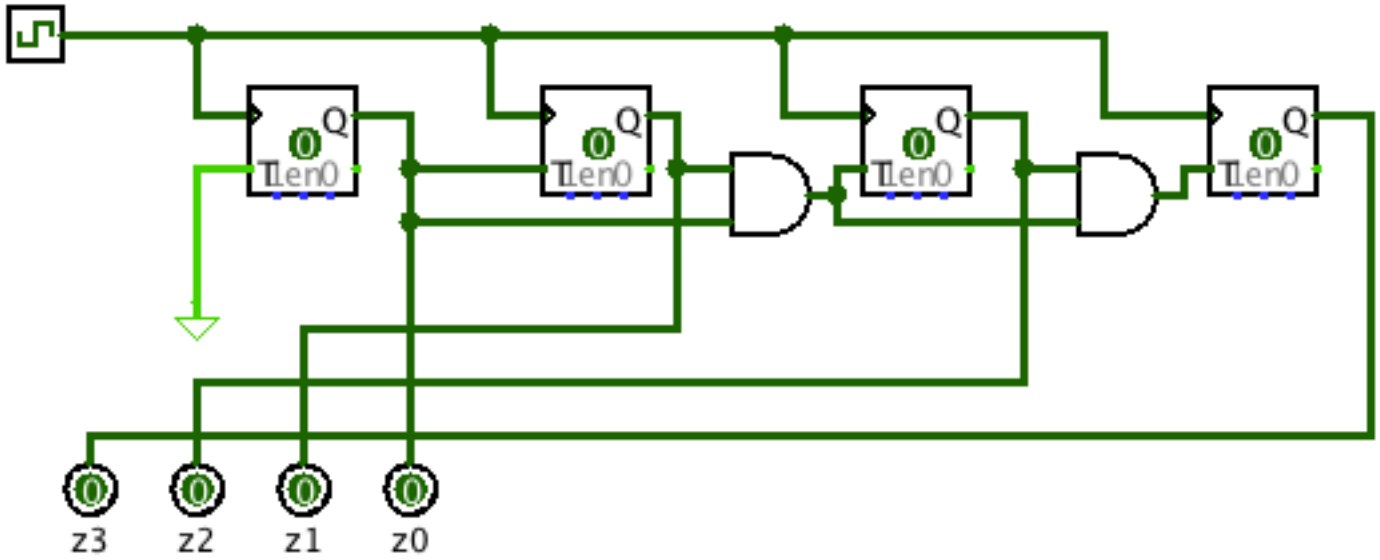
T flip-flop

- T flip-flop "toggle"
  - Like JK but connect J and K together
  - LEAVE: T=0
  - TOGGLE: T=1 (flips the stored bit)

T	C (clock)	Q(t)	Q(t+1)
0	high	x	x
1	high	x	x'

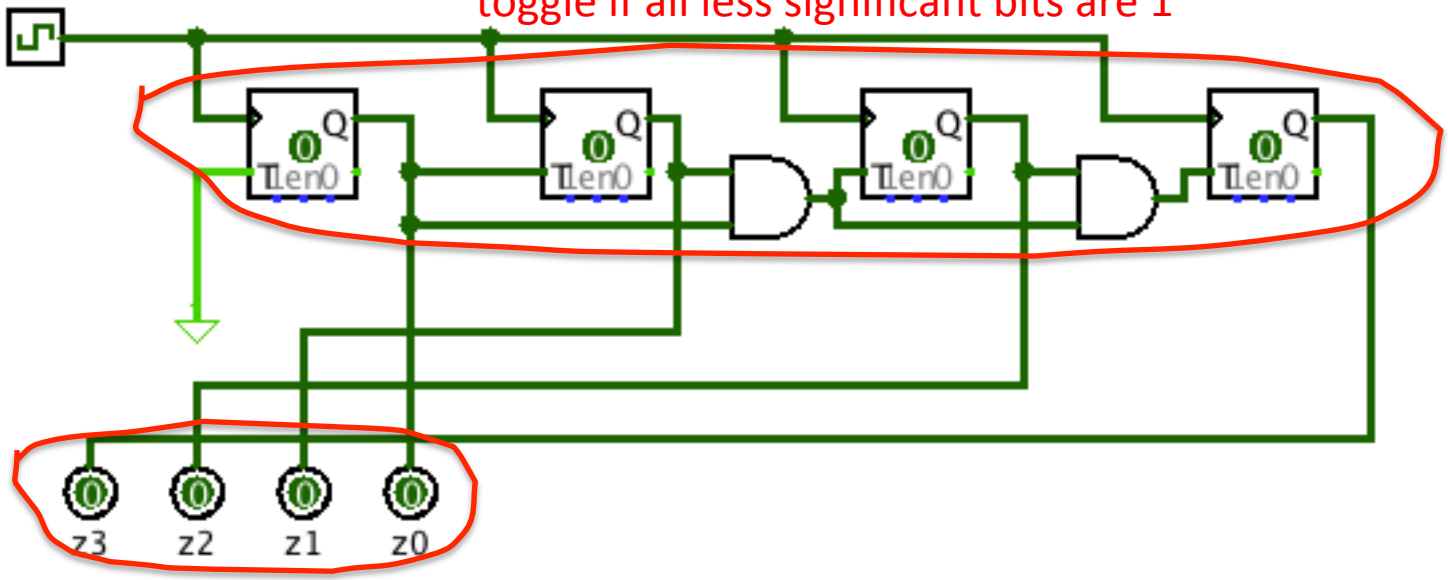


4-bit synchronous counter



4-bit synchronous counter

4 T flip-flops plus logic so they only toggle if all less significant bits are 1



4-bit output of counter

Summary

- Built a simple ALU
- Used flip-flops to build a register
- Created a register bank
- Created a multiported register bank
- Made a simple computer
- Made a counter