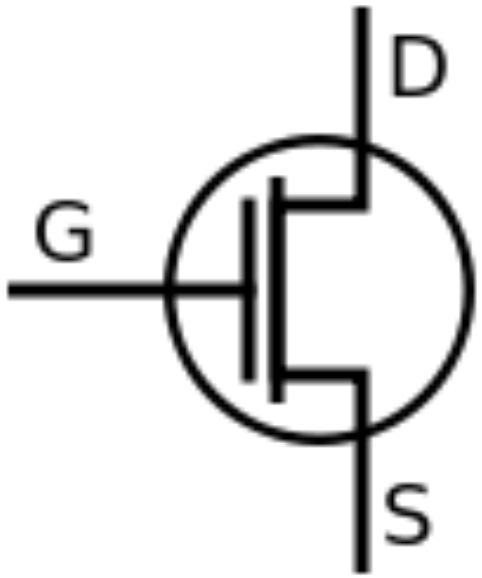
From Transistors to Computers

On voltage and current

Metaphor: Electricity is air.

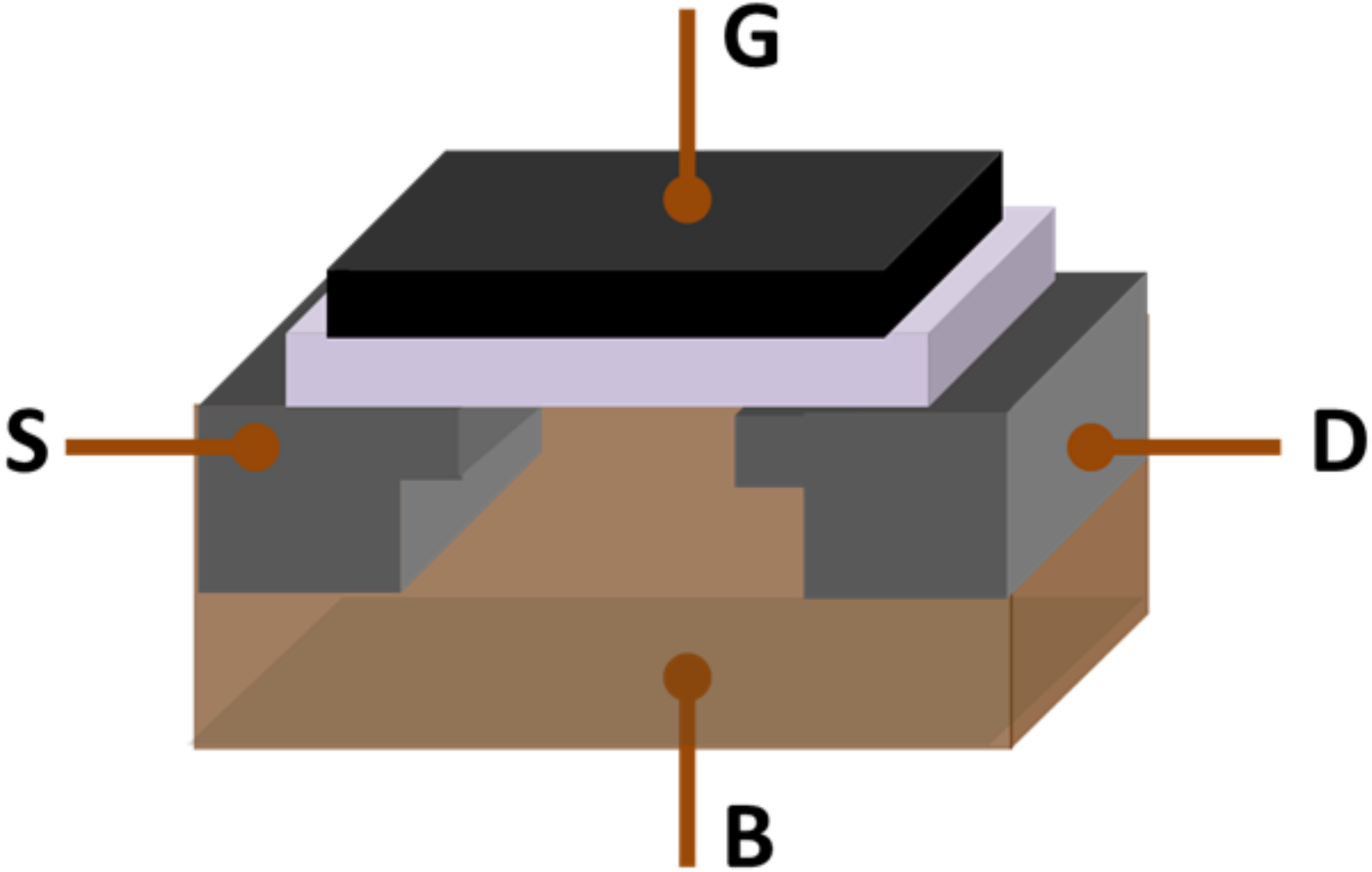
- **Voltage**: pressure. Wants to flow from high pressure to low pressure.
- **Current**: flow. The amount of (air, charge) going past
- **Resistance**: narrow tube. Restricts the amount of flow that can go through.

Transistors ("n-FET")

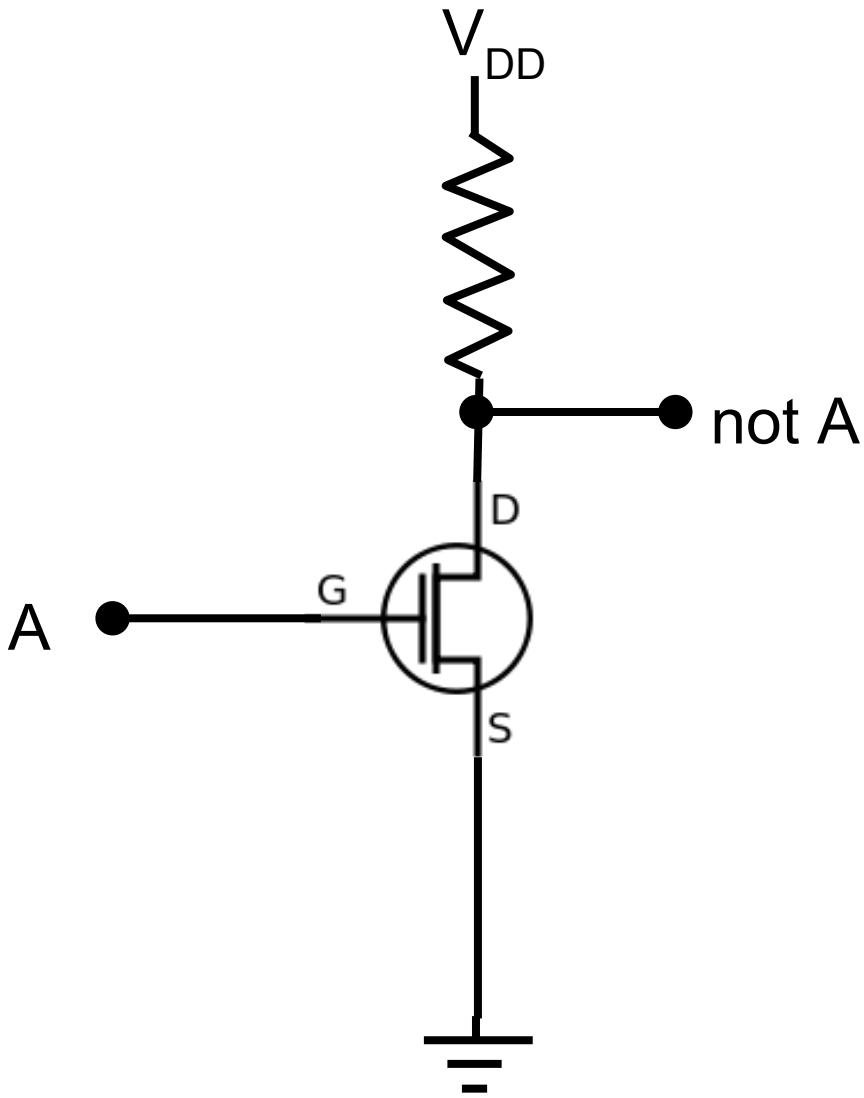


Current can flow this way from **D**rain to **S**ource.

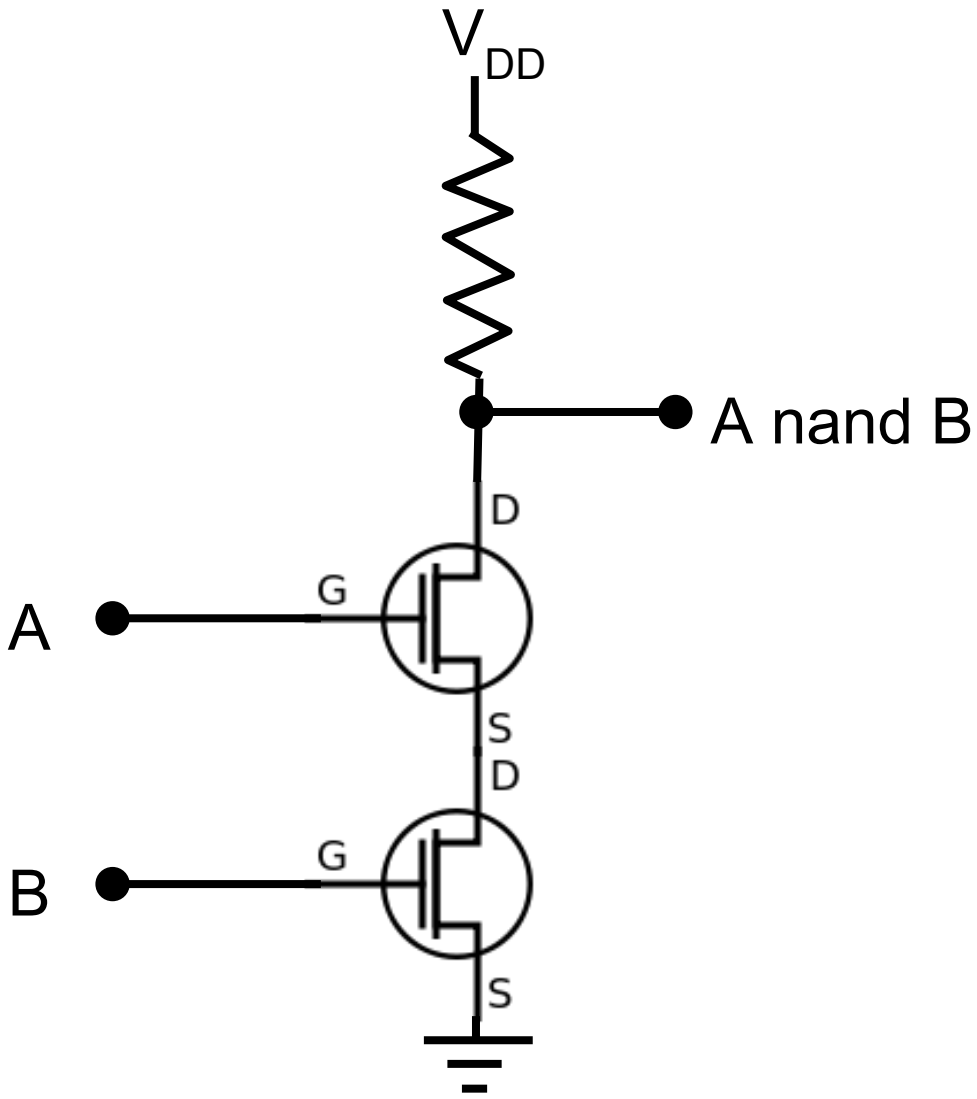
When the **G**ate has sufficiently **higher** voltage than the **S**ource


G
S
D
B

Making a NOT gate



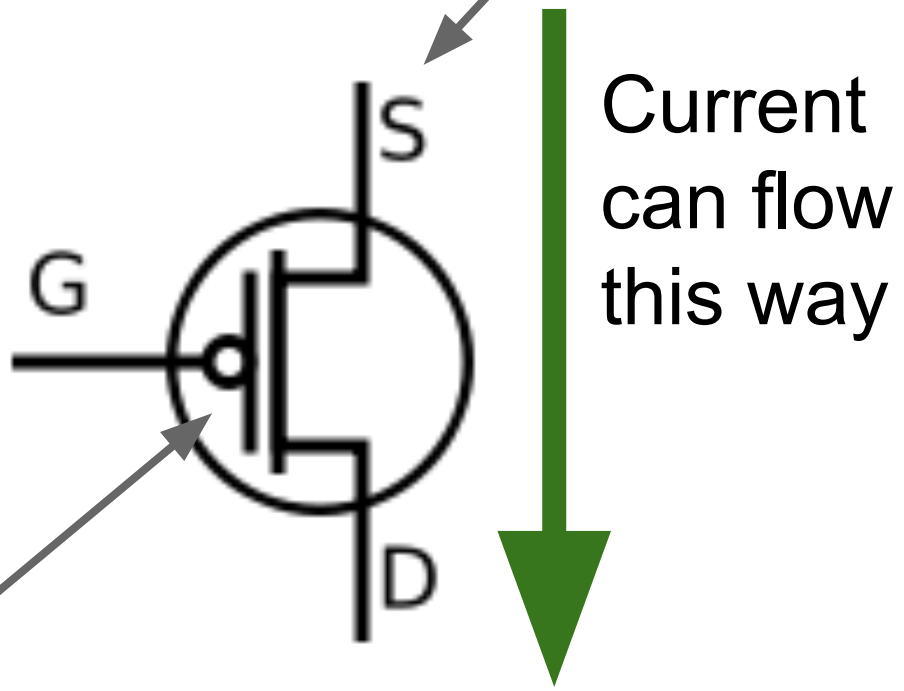
Making a NAND gate



Transistors ("p-FET")

When the **G**ate has sufficiently **lower** voltage than the **S**ource

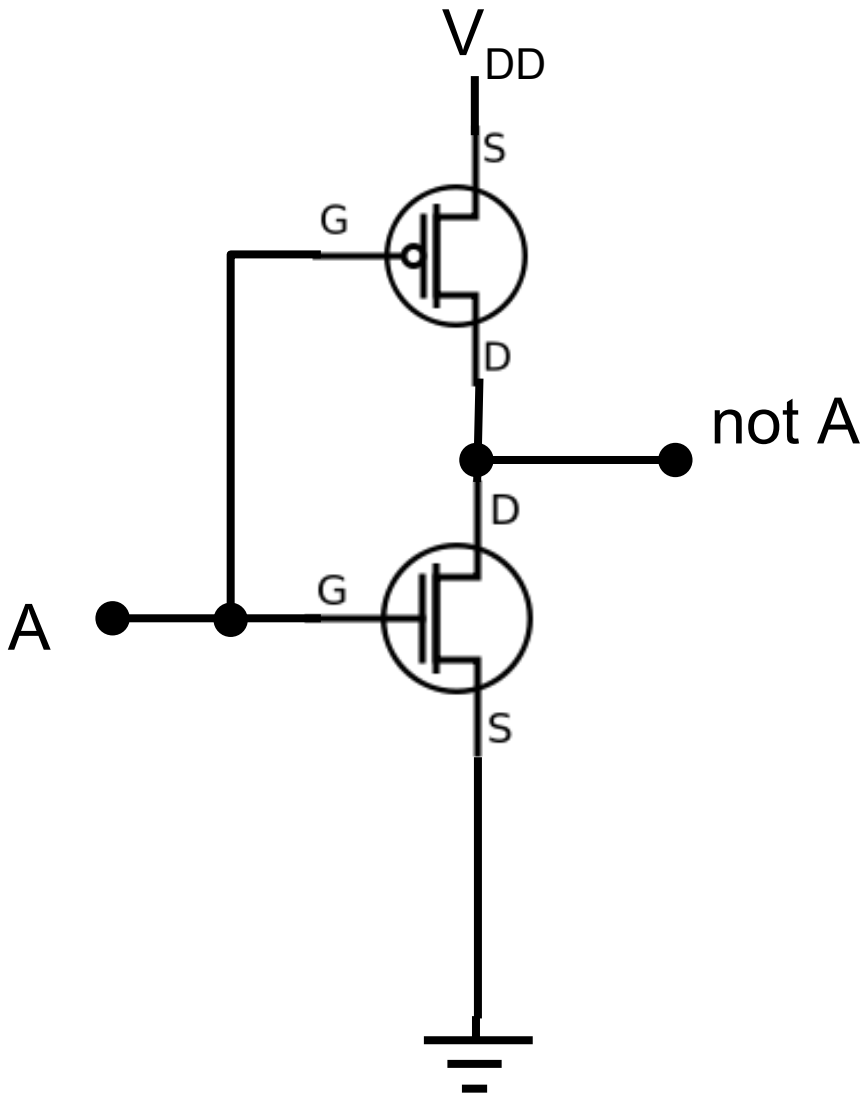
Note the source is at the top now.



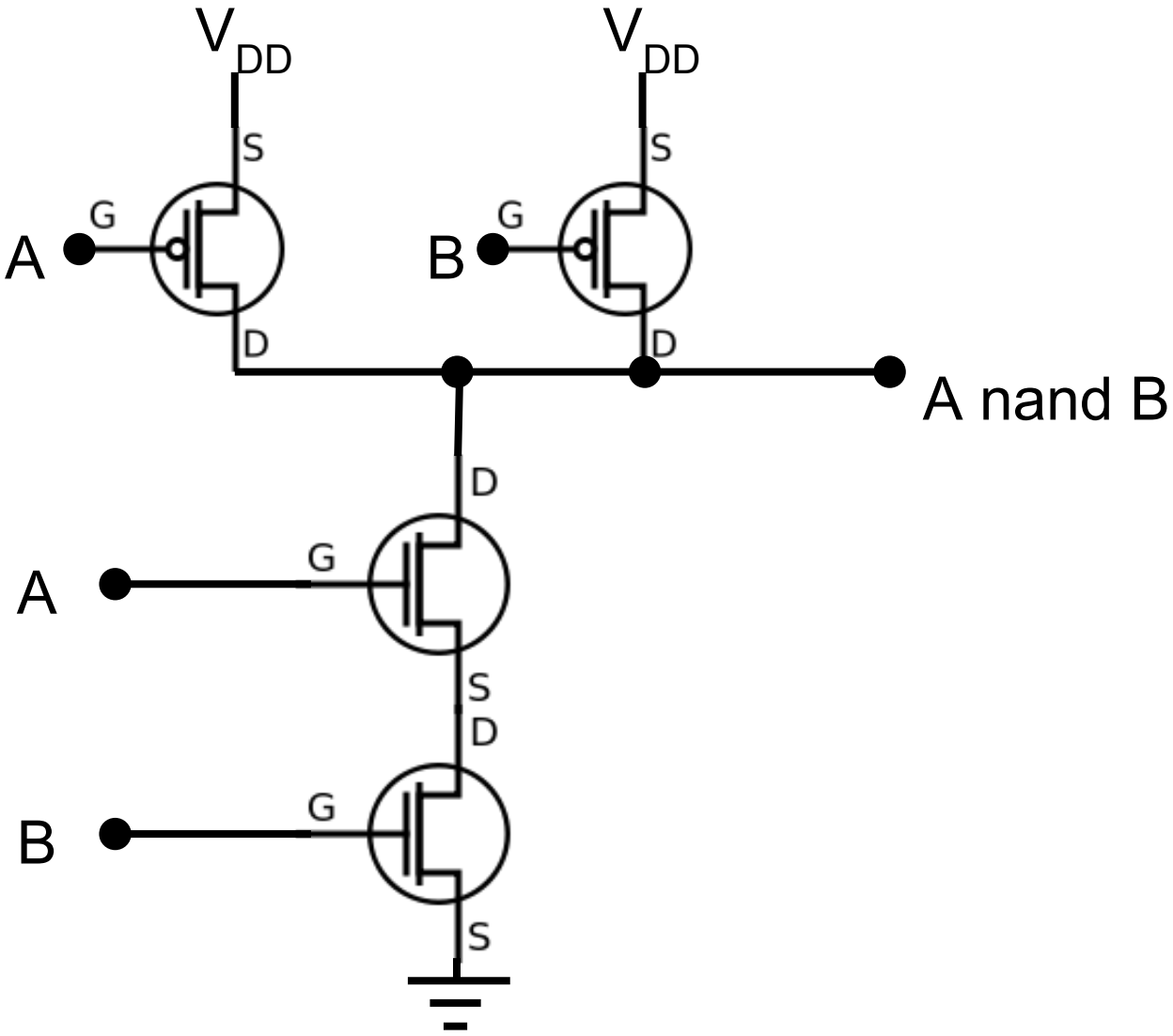
The bubble is the difference in the diagram.


GATE
SOURCE
DRAIN
BOBA FET

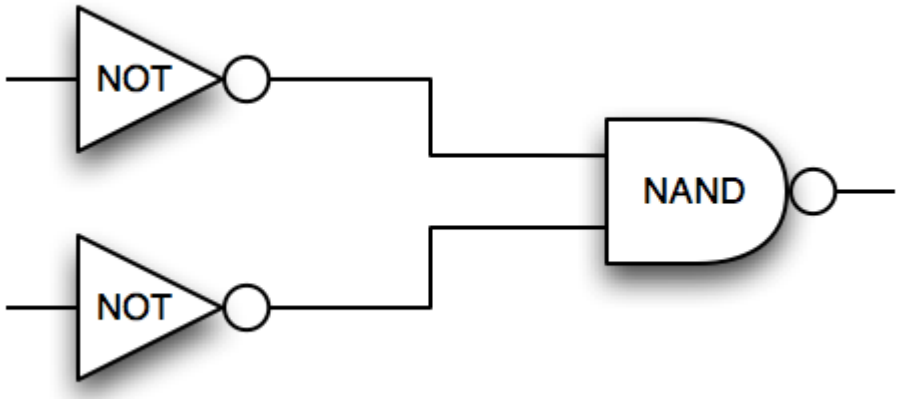
Making a NOT gate better using CMOS

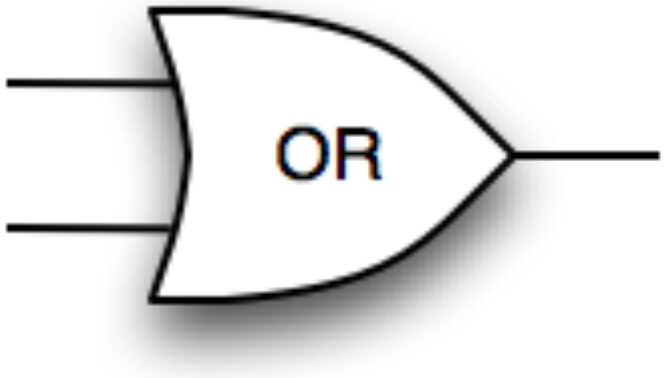


Making a NAND gate better using CMOS



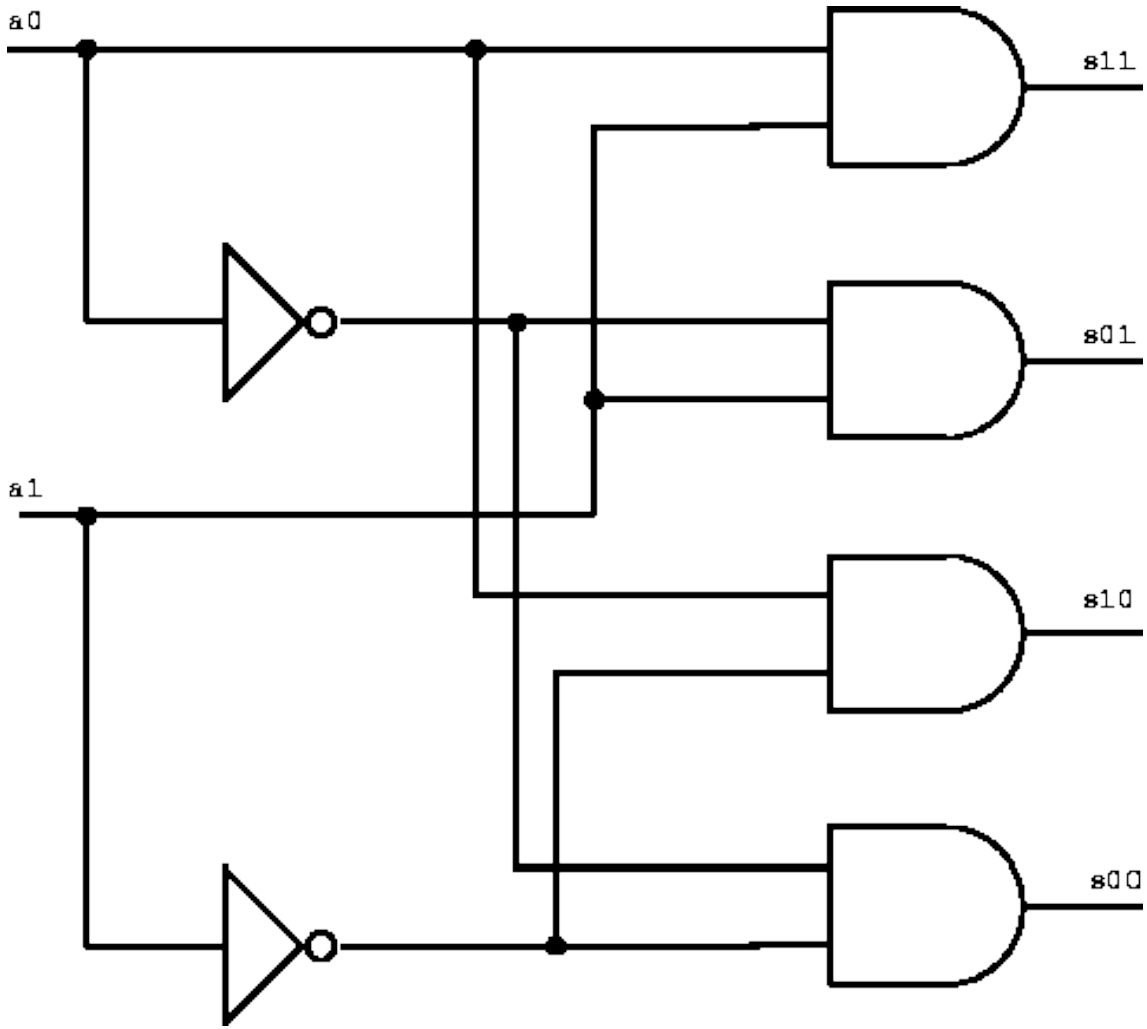
You can build logic gates out of other logic gates.



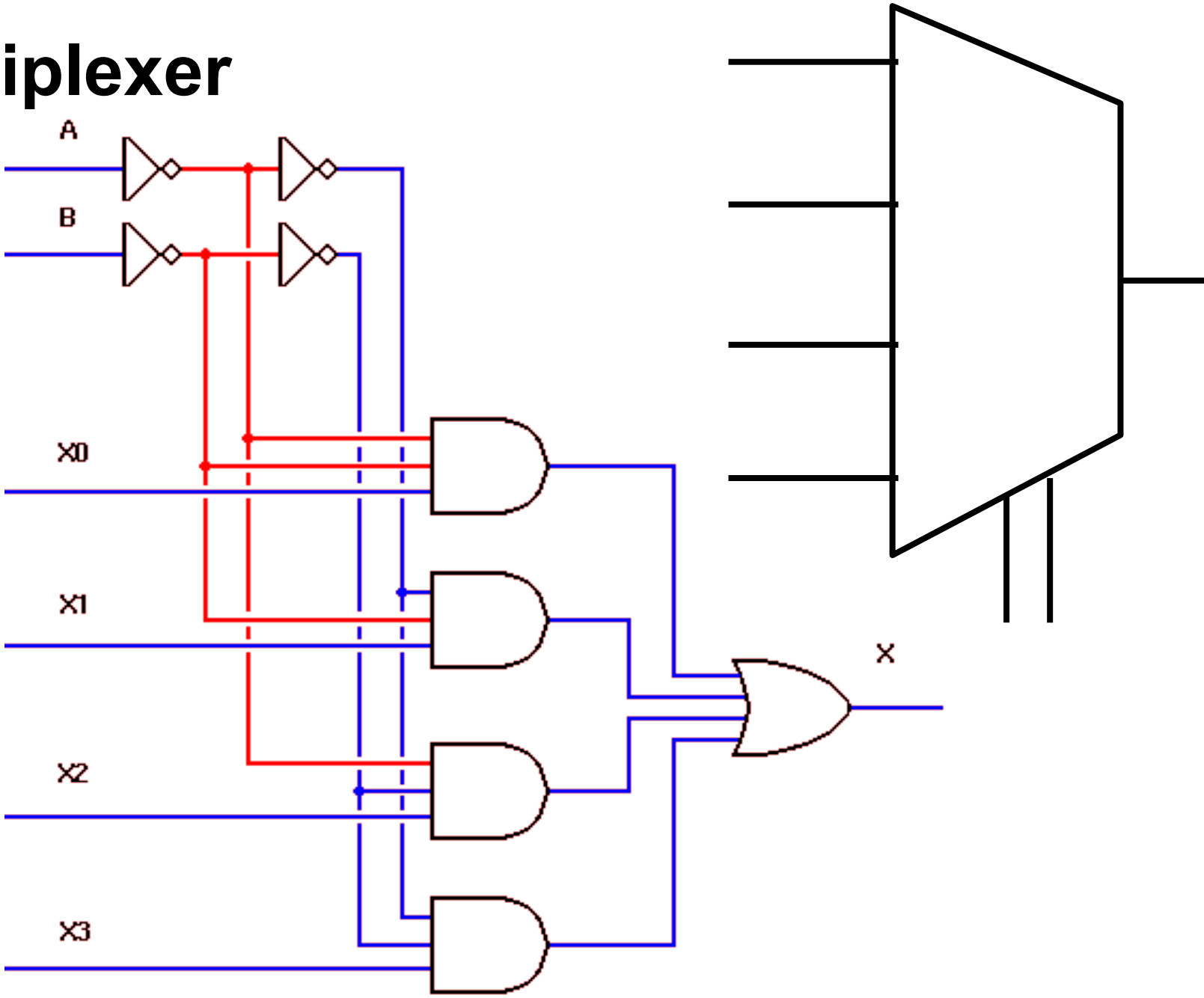


Some useful pieces

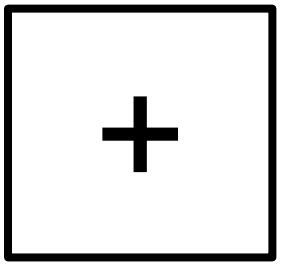
Decoder

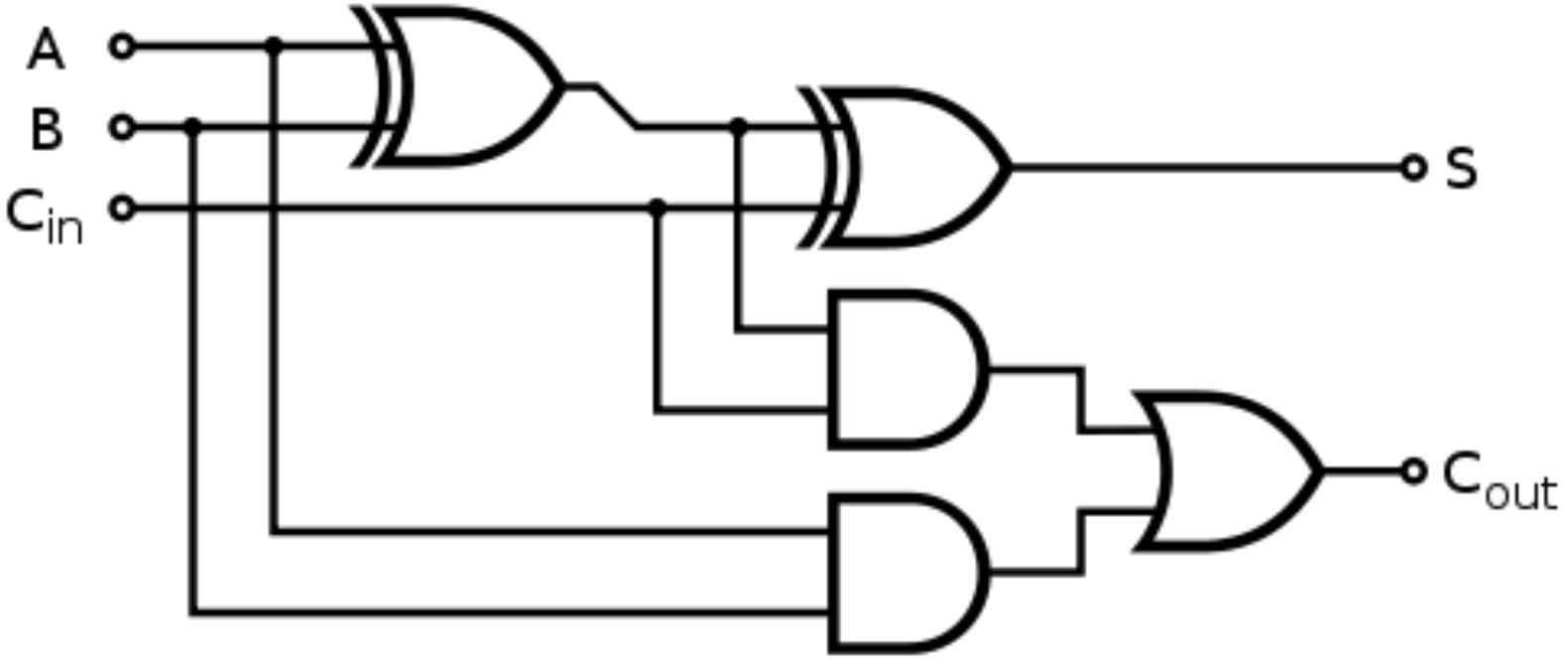


Multiplexer



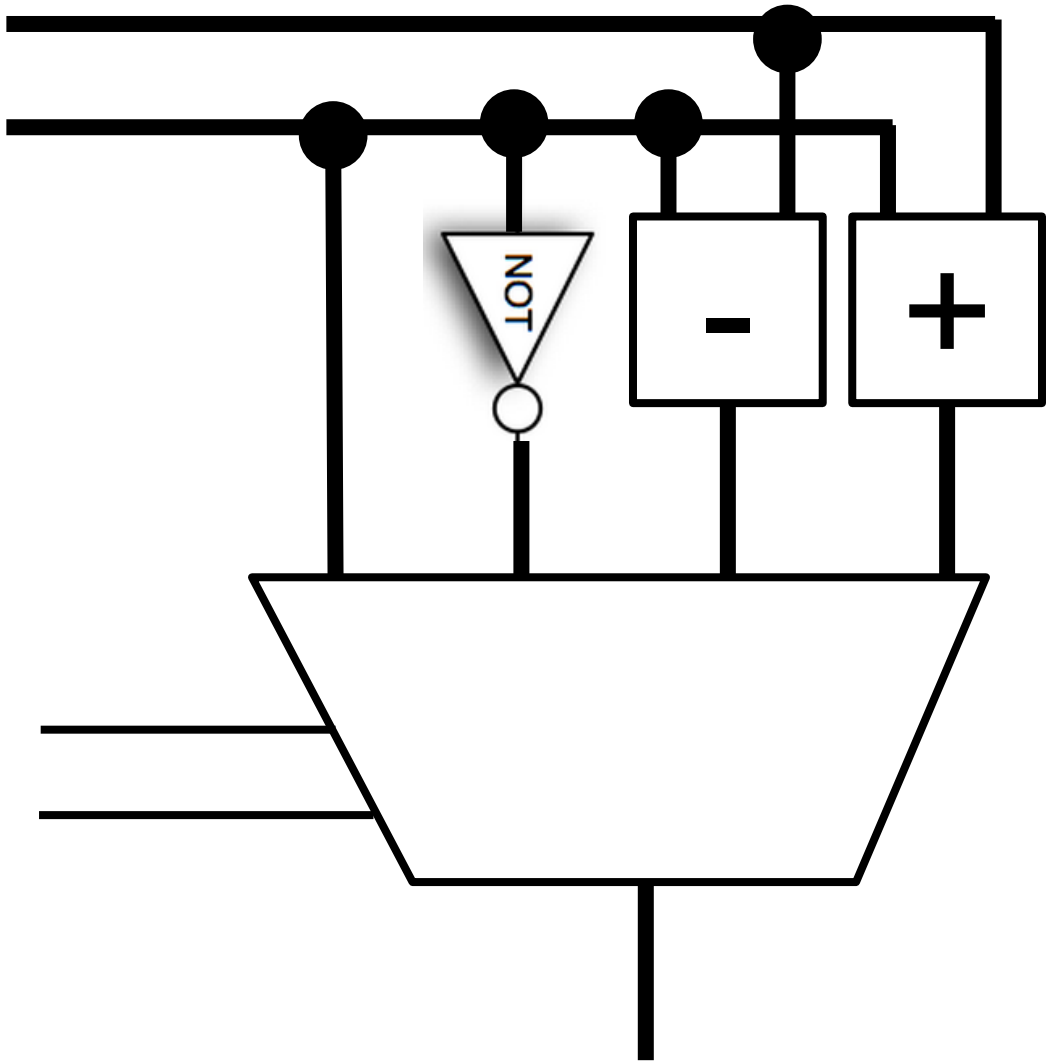
Full Adder



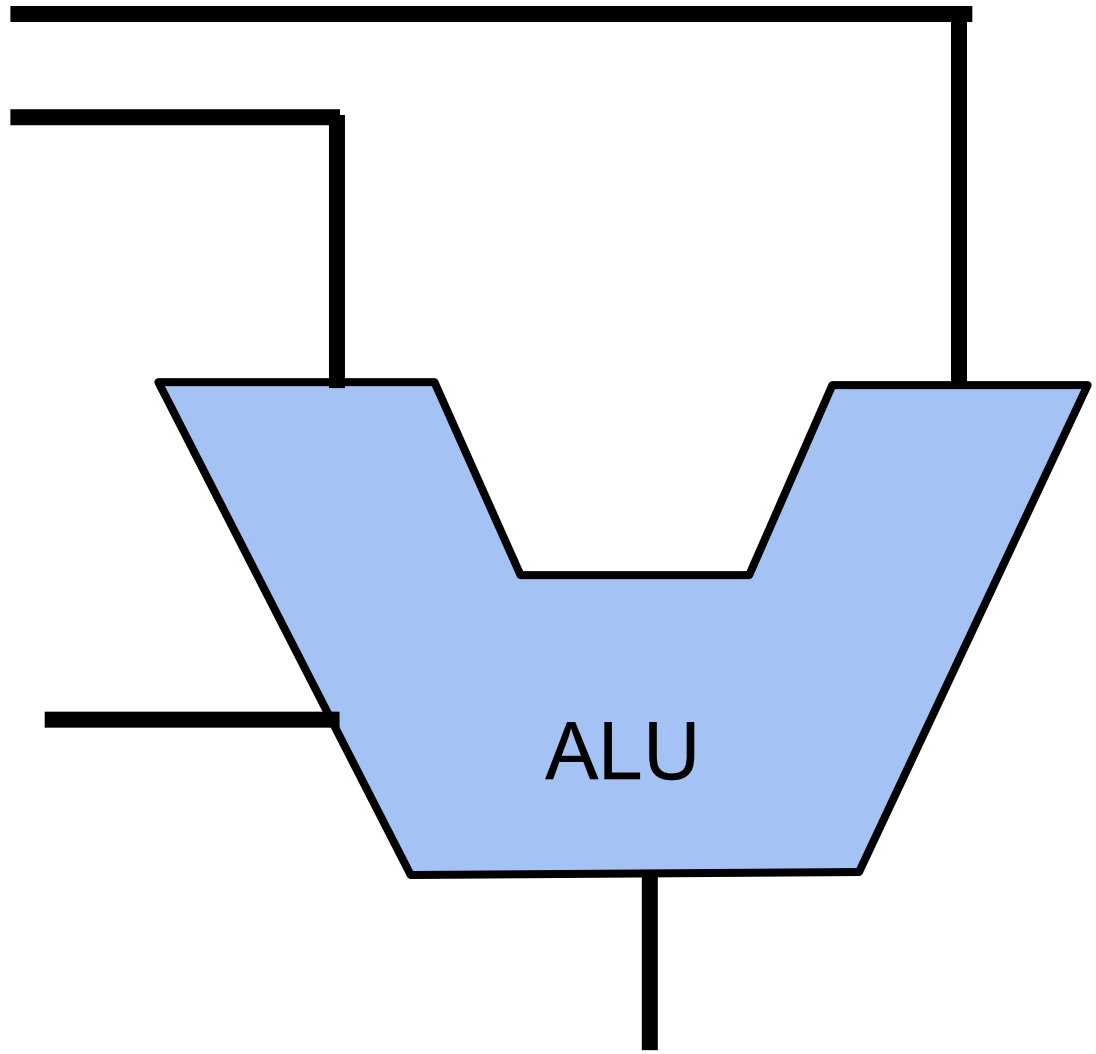


Square Wave Source

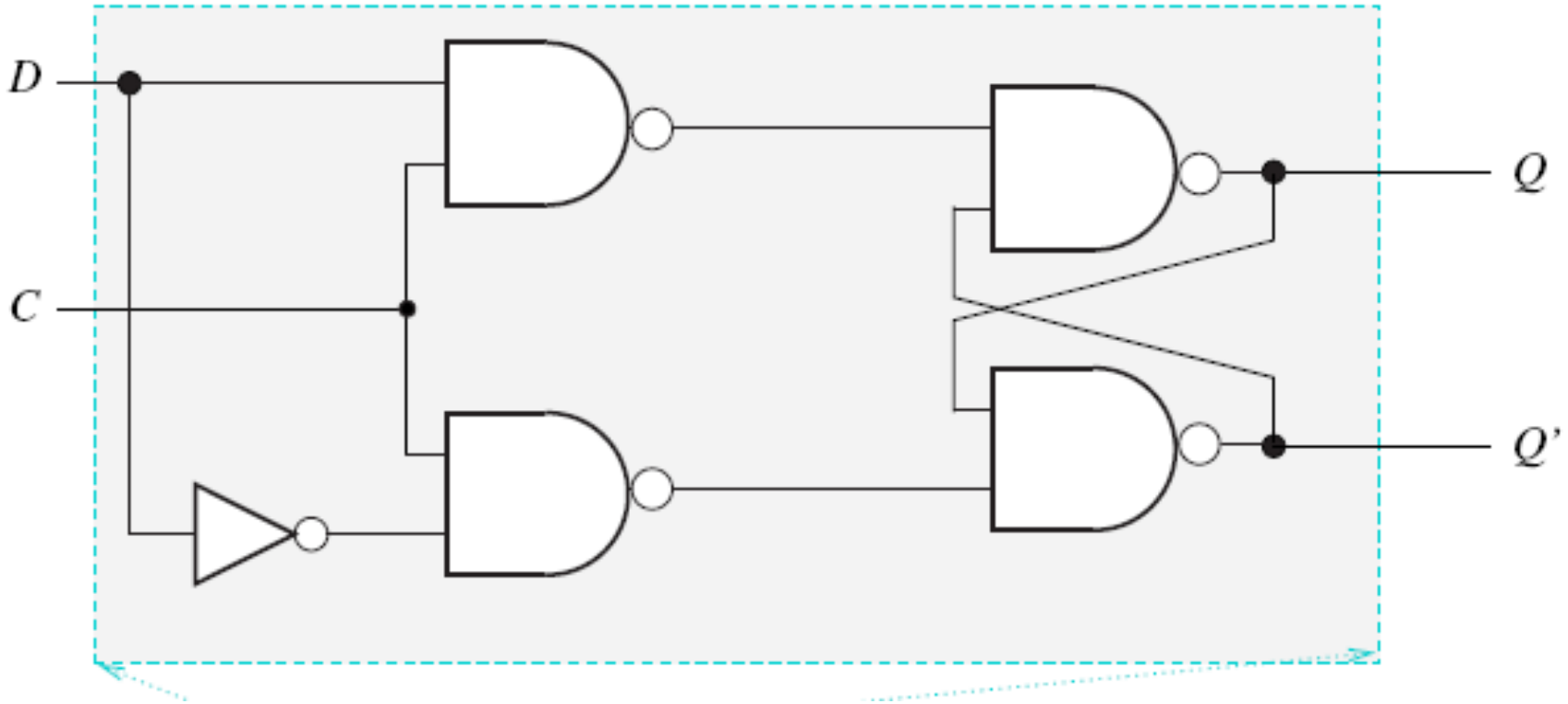
Arithmetic Logic Unit



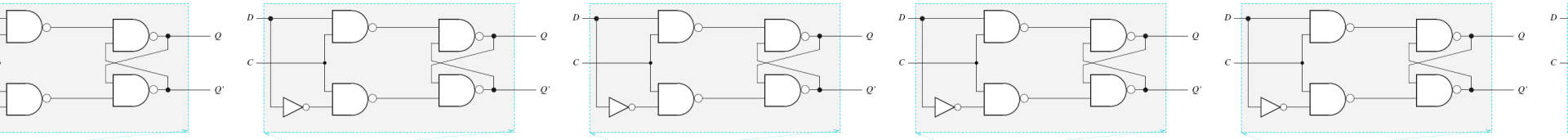
Arithmetic Logic Unit



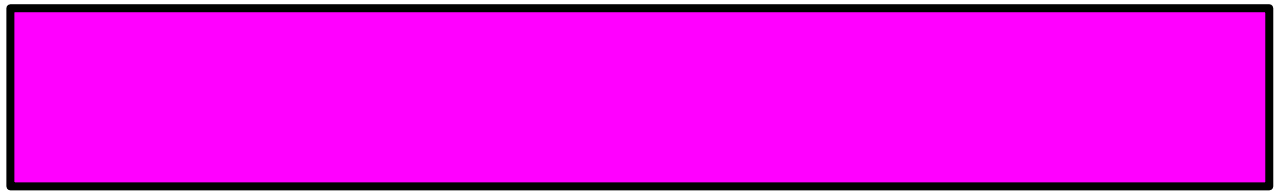
Latch



Register



Register



Instructions!

Structure of an instruction

Structure of an instruction

Opcodes

- Some to do math
  - **ADD** - A + B
  - **AND** - bitwise A and B
  - **OR** - bitwise A or B
  - **XOR** - bitwise A xor B
  - **NEG** - negative A
  - **SHL** - shift A left by 1 bit
  - **SHR** - shift A right by 1 bit
  - **INV** - bitwise invert A

Opcodes

- Some interact with RAM
  - **LD** - load address A from RAM
  - **STO** - store B to RAM at address A
- Some manage flow control
  - **JNZ** - jump to B if A not zero
  - **JZ** - jump to B if A zero

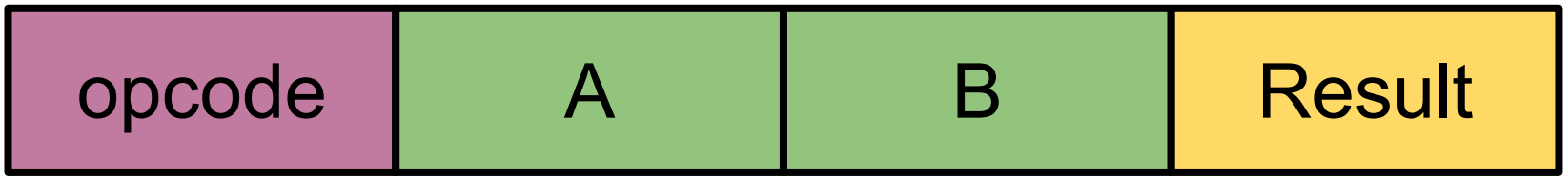
Opcodes to Binary!

- **ADD** - 0000
- **AND** - 0001
- **OR** - 0010
- **XOR** - 0011
- **NEG** - 0100
- **SHL** - 0101
- **SHR** - 0110
- **INV** - 0111
- **??** - 1000
- **??** - 1001
- **??** - 1010
- **??** - 1011
- **LD** - 1100
- **STO** - 1101
- **JZ** - 1110
- **JNZ** - 1111

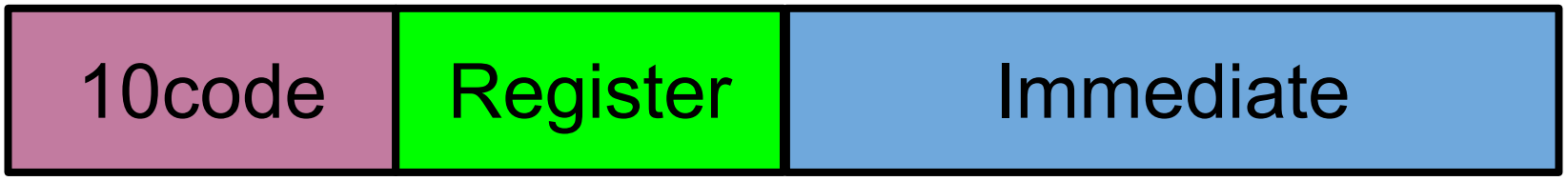
Memory in our computer

- 2^{16} bytes of RAM (64 kilobytes)
- 16 registers (scratch space)
- Register 0 is always 0. Writes to it do nothing.

Structure of an instruction



Structure of an instruction



Opcodes to Binary!

- **ADD** - 0000
- **AND** - 0001
- **OR** - 0010
- **XOR** - 0011
- **NEG** - 0100
- **SHL** - 0101
- **SHR** - 0110
- **INV** - 0111
- **ADDI** - 1000
- **ANDI** - 1001
- **ORI** - 1010
- **XORI** - 1011
- **LD** - 1100
- **STO** - 1101
- **JZ** - 1110
- **JNZ** - 1111

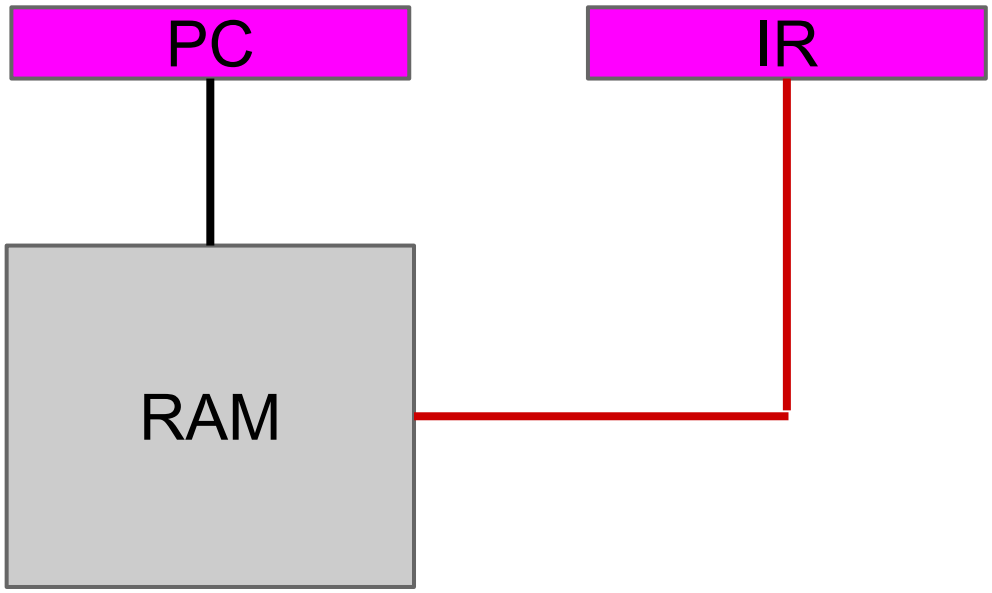
We're ready to build our computer!

Plan: 3 Phases

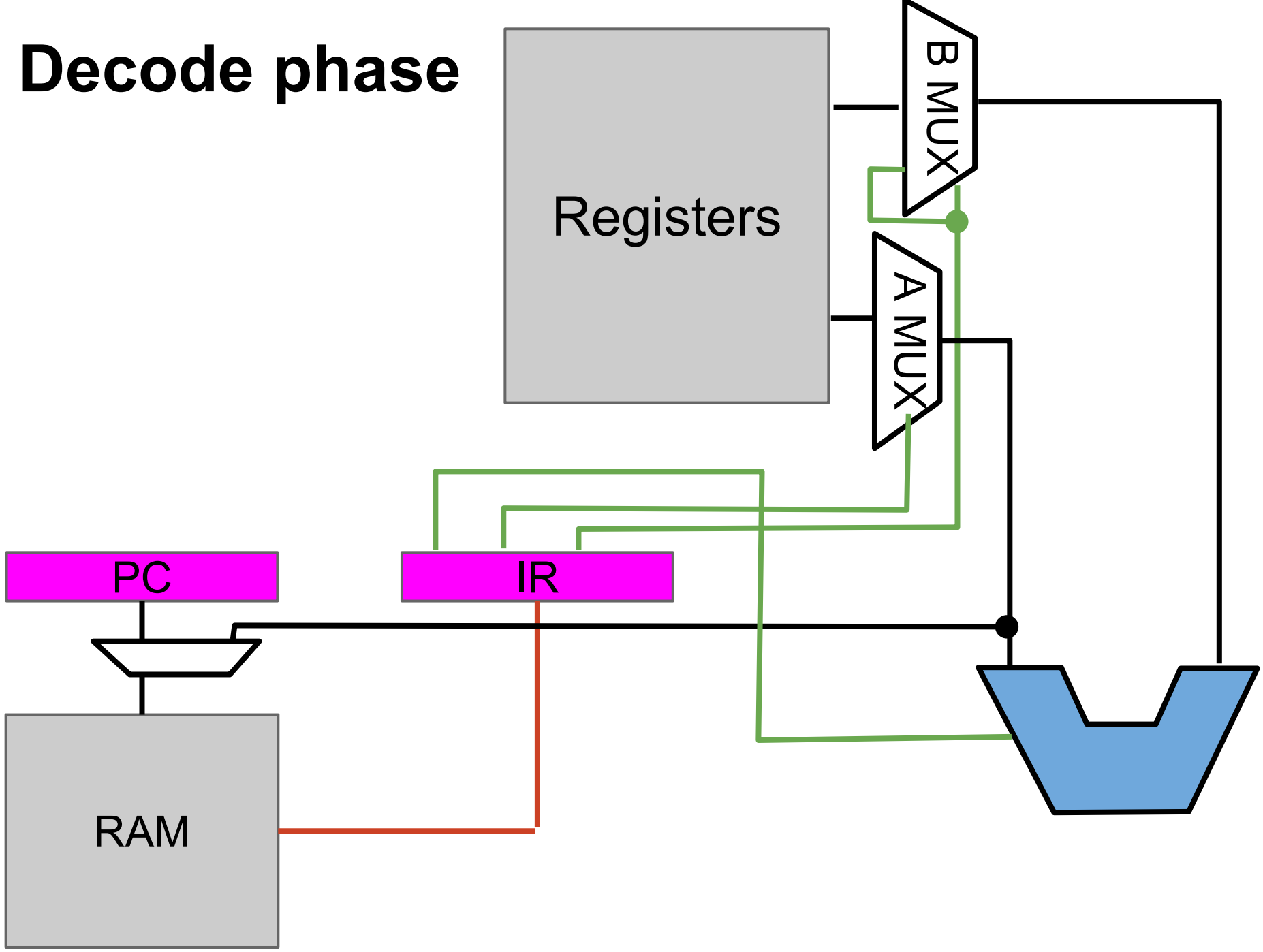
- **Fetch**: Get the current instruction from memory
- **Decode**: Make the calculations based on the current instruction
- **Execute**: Actually store the results, and calculate the location of the next instruction

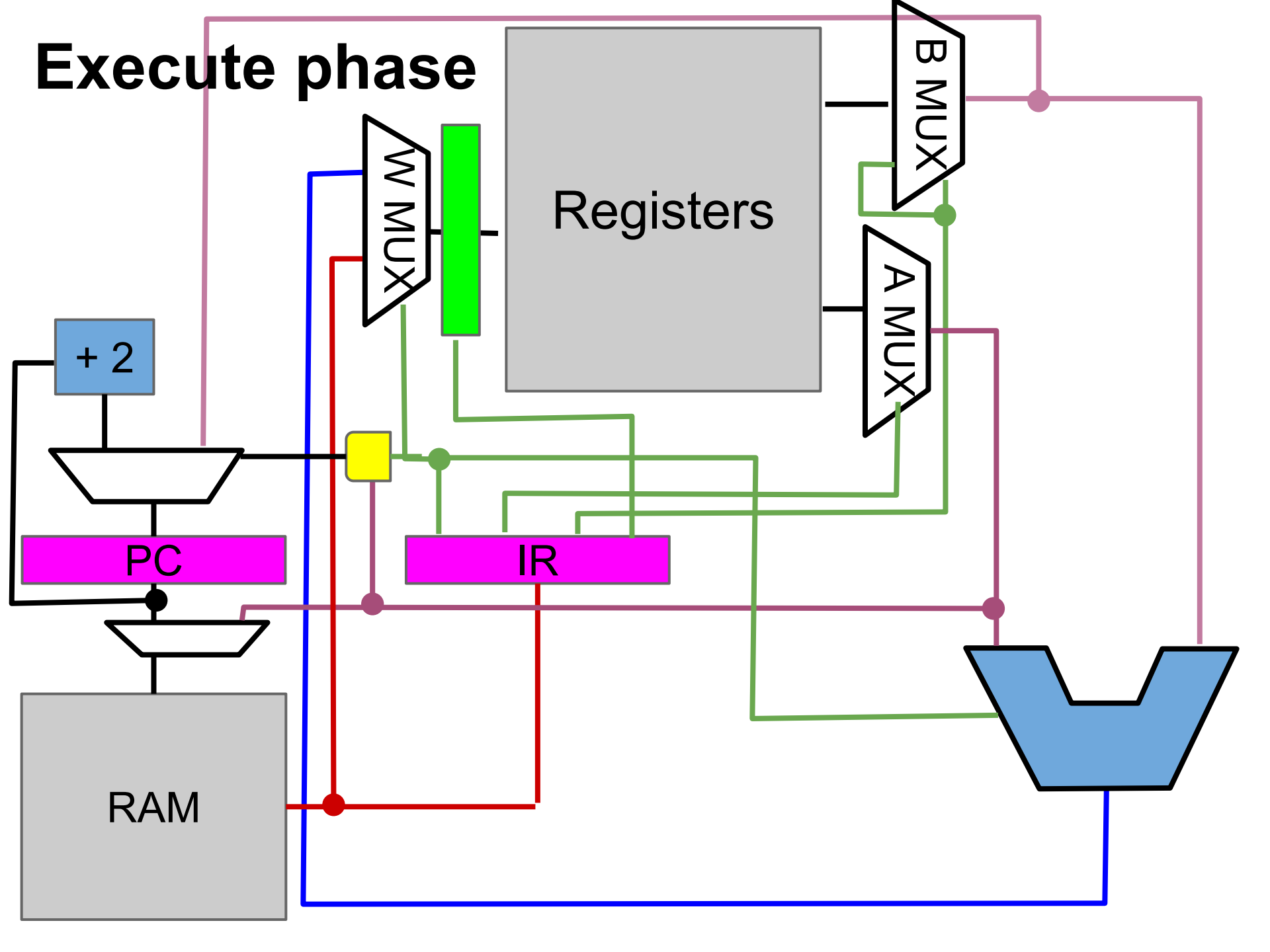
All these phases are managed by the ticking of square waves (not shown in the upcoming slides: imagine them)

Fetch Phase



Decode phase




Execute phase
W MUX
Registers
B MUX
A MUX
+ 2
PC
IR
RAM