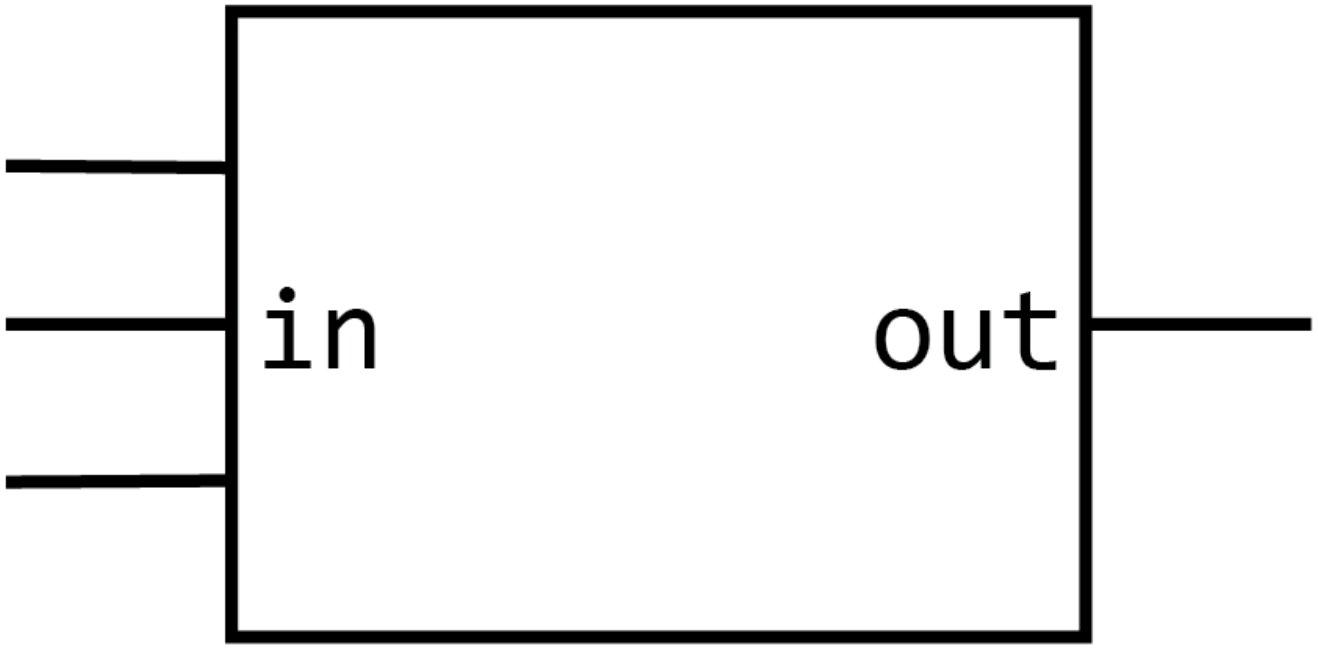
Circuit components

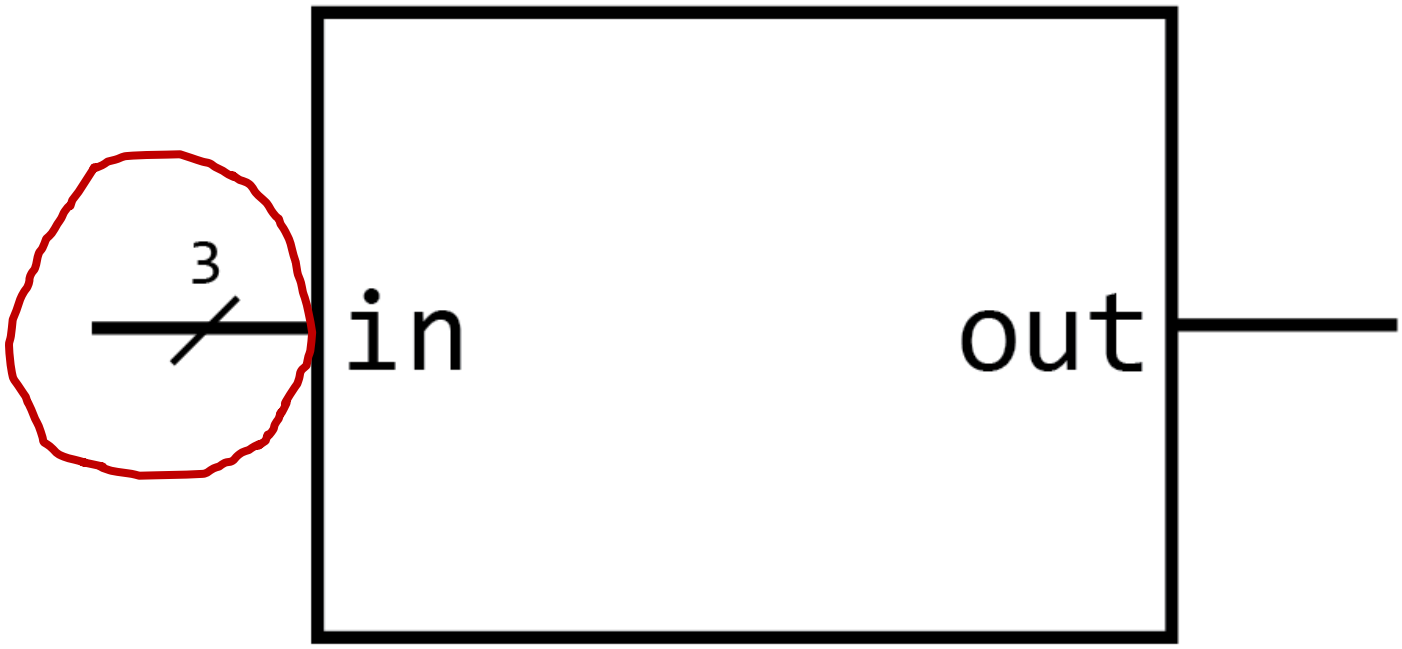
- Circuits are hardware blocks that use gates and physically use electric signals to implement abstraction of boolean representation
- We will look at some abstract notations first

Basic circuit block



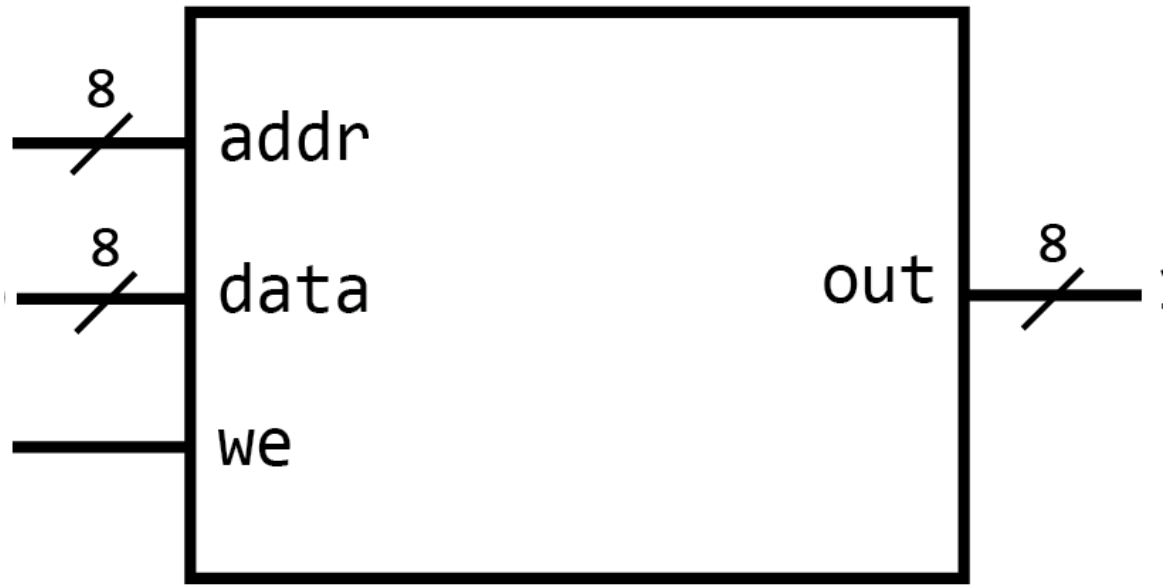
- 3 input bits
- 1 output bit
- Something happens inside that applies an arbitrary boolean function on the input bits

Basic circuit block



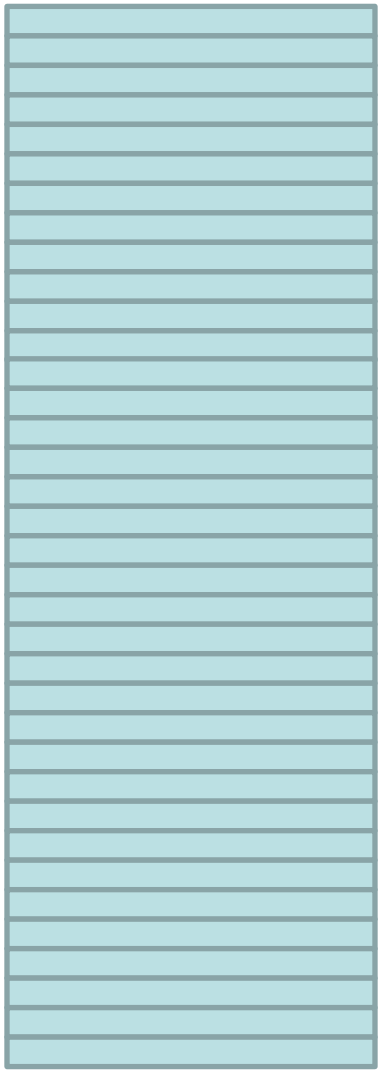
- Shorthand for 3 input bits

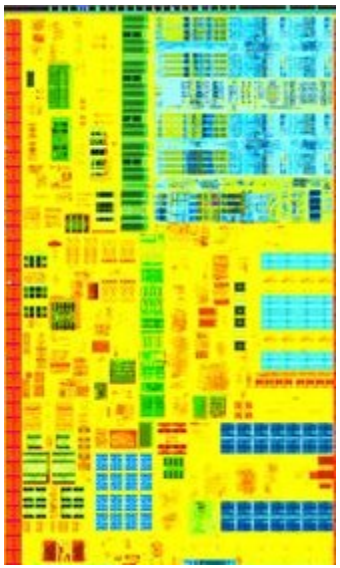
Memory



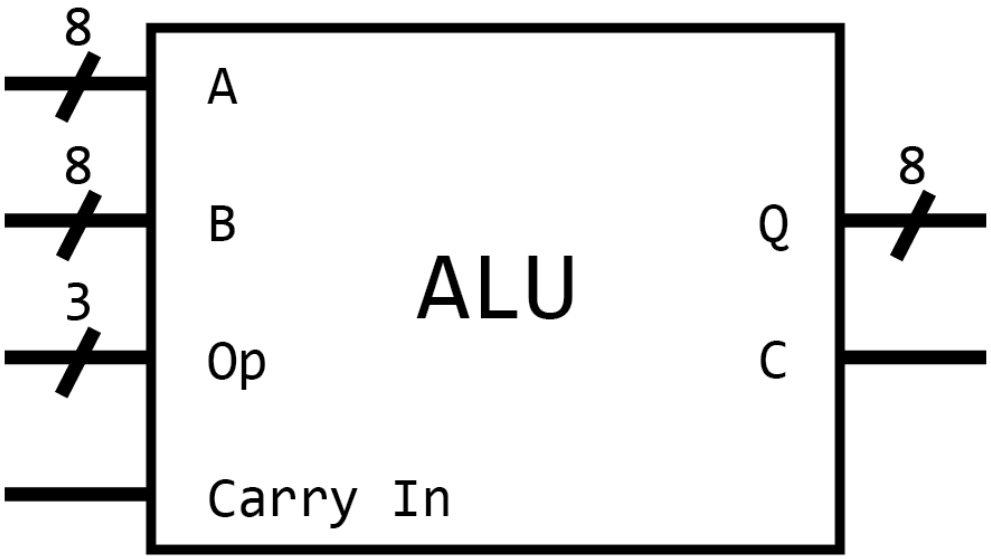
- When we is 0
  - Read what is in location **addr**, and place it in **out;** ignore data
- When we is 1
  - Read what is in **data**, and write it into memory location **addr**

Memory (internals)



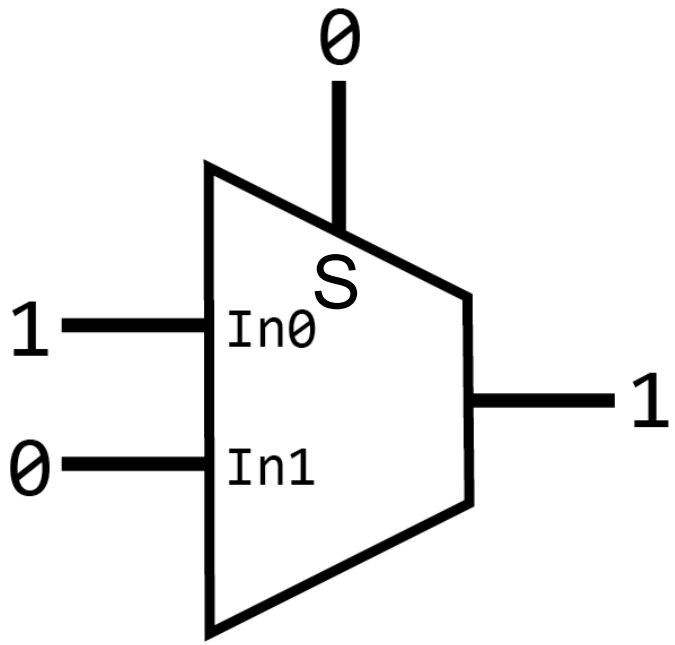


ALU



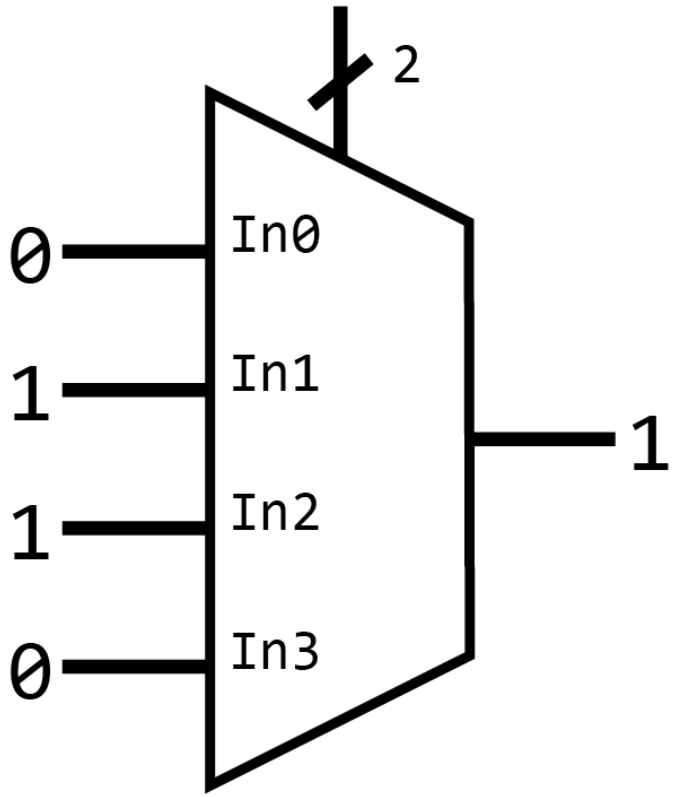
Number	Operation
0	Addition
1	Addition with carry
2	Subtraction
3	Subtraction with carry
4	Binary and
5	Binary or
6	Binary xor
7	Binary nor

Multiplexor (allows selection)



- If S = 1
  - Output = In0
- If S = 0
  - Output = In1

How do we select one thing out of 4 things?



Lets do two mind blowing things now!

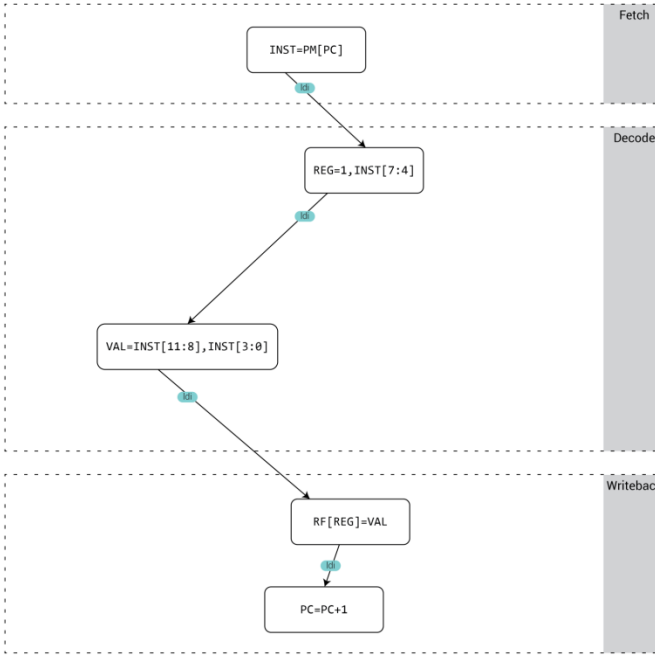
How do we build the 4-way MUX using only 2-way MUXes!

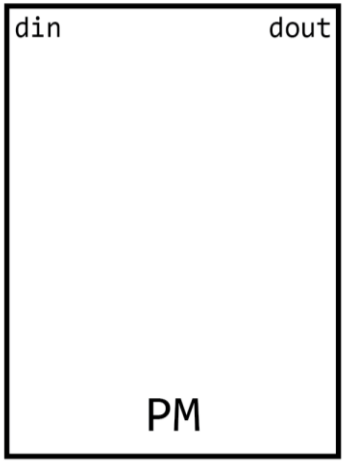
How do we build the 2-way mux using gates

Two simple rules of circuit blocks

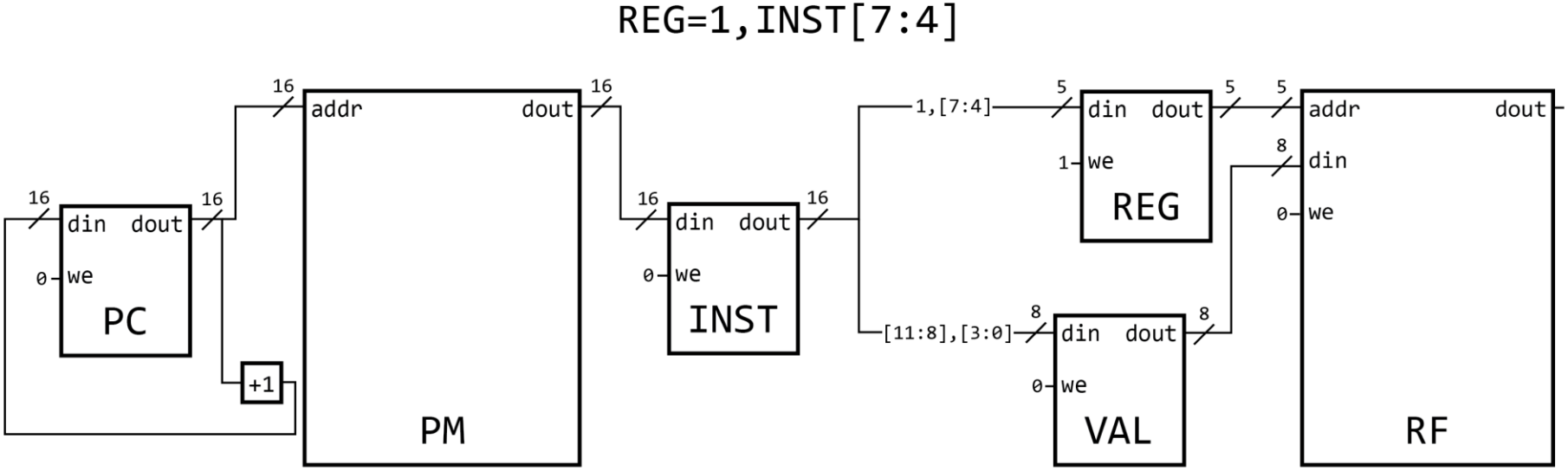
- Bits can be arbitrarily split out of output “ports”
- Blocks can be arbitrarily connected to each other to create cool functionality

The LDI computer!





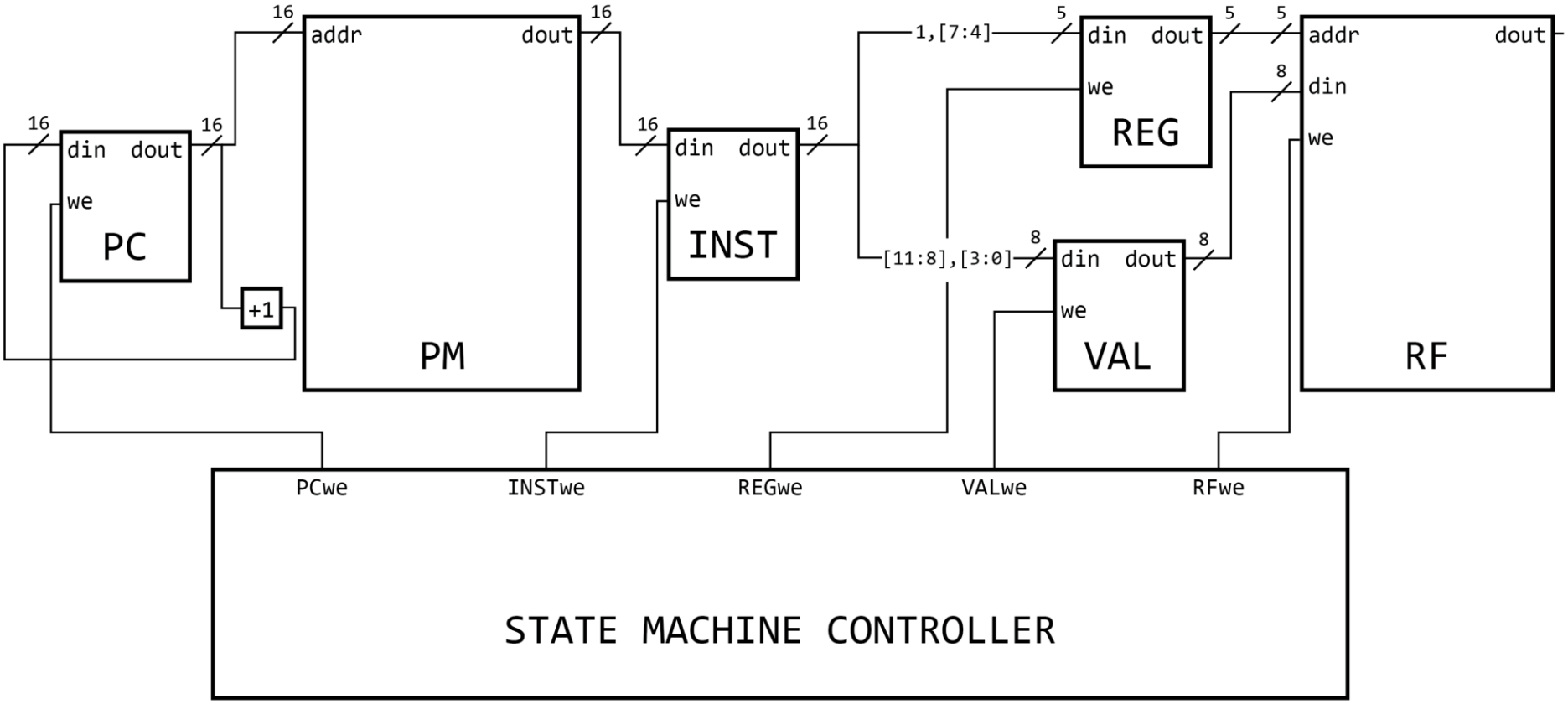
LDI Computer



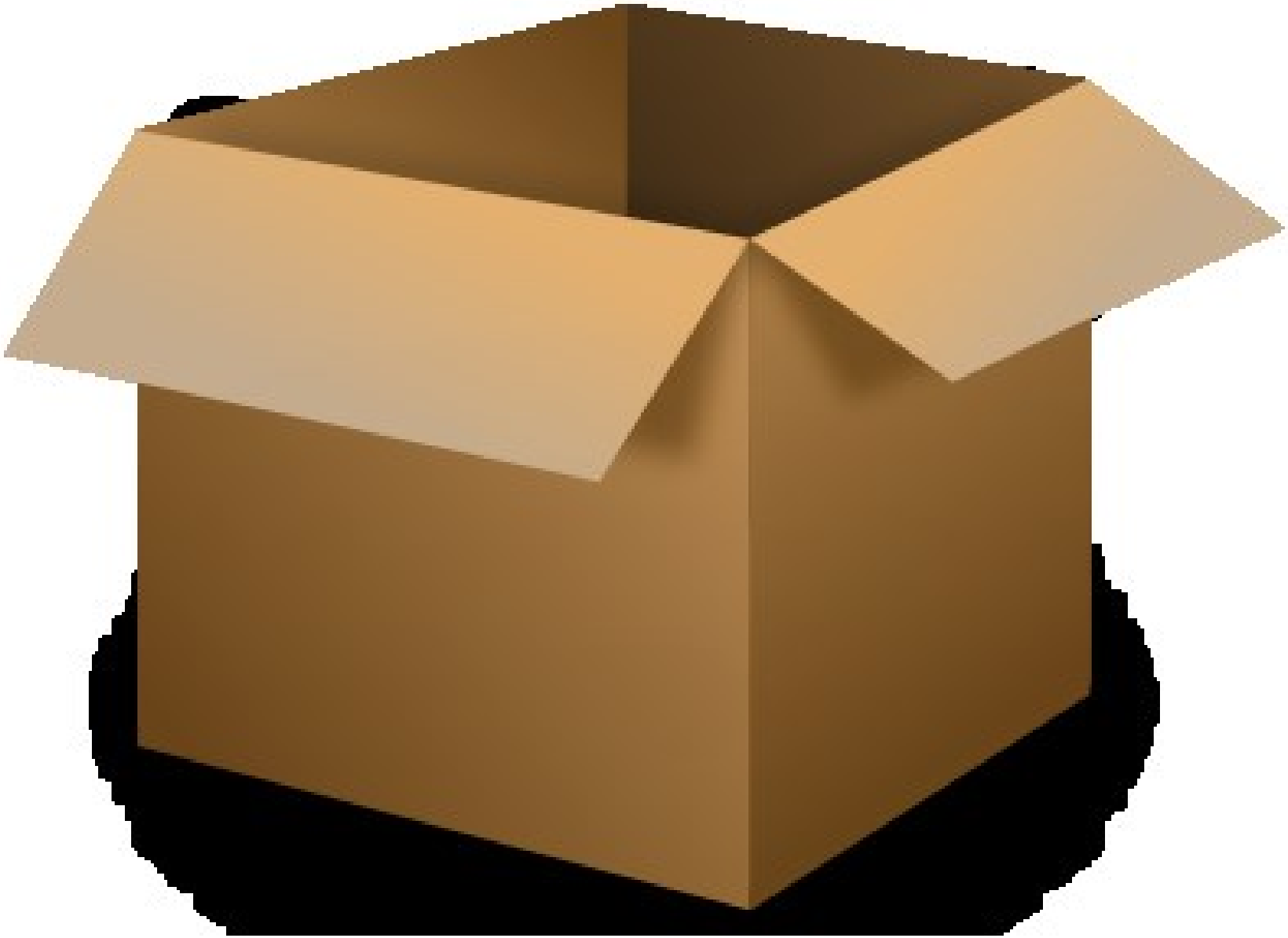
What is missing?

A for anyone who can answer this if you haven't read the book – but that would incentivize not reading the book ☹

LDI Computer



That is just a box!



It is a magic box!

Seriously what is in it?

I will really given an A to anyone who can answer this!

The box implements the state transition table

	PC_we	INST_we	REG_we	VAL_we	RF_we
INST=PM[PC]	0	1	0	0	0
REG=1,INST[7:4]	0	0	1	0	0
VAL=INST[11:8],INST[3:0]	0	0	0	1	0
RF[REG]=VAL	0	0	0	0	1
PC=PC+1	1	0	0	0	0

Summary

- State machine
- Circuit blocks
- LDI computer!

October 30, 2015
Overview of physical implementation of computer
Current point: take assembly instructions, come up with ISA, can specify an abstract state machine with two instructions (ldi, ld)

Circuits: abstract representation of the physical entity that is actually used to build a computer. These blocks are built out of logic gates, a single component may be as simple as just two logic gates, or may connect hundreds of components together. Thousands of these make up “memory”.

We're going to use abstract representations to build complex machines! Whoo!

Physically, circuits take electric signals (hi 1, lo 0) and output electrical signals to perform boolean logic.

Basic circuit block. The lines represents “ports”. The name you see next to it is the name of the port. Each wire drawn like this represents a single bit.

The example on the slide: this will take 3 input bits and produce 1 output bit.

Different notation: a slash in a port with a 3 above it represents a three-bit signal. It's identical to the previous slide, just notational shorthand.

Right now we're still talking about notation, not how we build them!

Memory:

Three types of input:

- address (this has nothing to do with auxiliary registers), data.
- One output port with 8-bits.
- We → “write-enable”.

When we = 0, you *read* what is in the memory. This will read the memory location @ addr, place the values stored there into out, and ignore whatever is sent thru data.

When we = 1, you *store* into the meomry. You take the mem location of addr and store data into it. The “out” data is then undefined.

Internally, memory just consists of many locations.

Chips: the actual silicon chips representing what has been physically constructed! The blue things on the chip are the actual physical memory locations constructed out of transistors. To build a single bit of memory, it takes 6 transistors! Memory on a chip is rows and rows of 6 transistors.

In future lectures, we'll get to the gate and transistor level of memory and talk about how this is actually built.

ALU (Arithmetic and Logic Unit):

Combining many individual circuit components to build one large thing.

A typical ALU has three inputs: A and B (the operands that need to be fed to perform the execution of any instruction), operand, and carry in (for adc). Output: q and c (carry output).

A table for the ALU we are using in the slide is provided on the slide. The number corresponds to the operation for this particular ALU.

So far: two circuit blocks, memory and ALU.

Multiplexor (allows selection):

Circuit block that allows you to do selection.
Special notation: port on top, the S port which decides which input you'll use.
If S =1; the output = input0.
If S = 0; the output = input1.

How do we select out of *four* things? A 2-bit select port that picks among four things.

If select = 00; input0
select = 01; input1
select = 10; input2
select = 11; input3

Two rules of circuit blocks: output ports are essentially eletrical wires that can be divided in whatever you want. You can connect these in arbitrary matters for cool effects.

Modern computers are basically *well-designed* connections of circuit blocks.

How do we build a 4-way MUX using only 2-way MUXes?

You connect them! But how?

See diagram. (Will be passed out in a later class). The essential principle is that you use first level to select between top two or bottom two inputs in two MUXes, and then you use the second selector to choose which of the remaining two inputs you want.

Gates → transistors → silicon building pieces → computer!

We'll get into MUXes in more detail on Wednesday.

We are on our way now to build a computer! We know how to build a memory, an ALU. These things can be connected to each other by physical wires. This is created out of copper wire (used to be aluminum). State machine at top is an abstract representation.

Whenever memory is not multiple entries deep, we'll just have din/dout as opposd to addr. There's no selection of which entry to write to so we don't need addr.

Components :

PC: Program counter, 16 bits with input, output, write-enable
PM: Program memory, instructions stored at address
INST: aux reg to store the instructions
REG, VAL: registers
RF: Register file.

We have our building blocks, but we need to connect them now!

Let's look at the different states in an ldi machine.
First state: instr = program memory, which is accessed using program counter
Second state: reg = takes as input something from inst register (see chart)

Val: takes something from inst register as well (8-11, 0-3)

Third state: take output from reg as addr, take output from val as din
Fourth state: pc = pc +1 (use ALU for this)

Not really the most efficient use of these circuit blocks, but it gets the job done in our rough

machine to get our LDI computer.

What is missing though???
There are some input values that are disconnected from the rest of the machine! We need a state machine controller whose job is to feed all the control signals. The box has numbers in it which correspond to different states you want to sequence the machine to. They're not fixed numbers, because we want to get a *programmable* computer. This state machine controller sends out an implementation of the state transition diagram which sequences the machine from one state to another. Based on what state it's in, it will produce values based on the state transition table.

~*~Profound question~*~: **How do you know when you are finished with a state?** How do you know which state you are in?

Asynchronous logic: You can monitor the wires to see where output is coming from in individual components. (No one does this.)

Synchronous logic: assumptions that everything takes the same amount of time. You figure out the time taken by the slowest thing in the system, set the clock frequency to that thing, and whenever you finish a clock cycle you transition to the next state.