Lecture 5: Hardware & Machine Organization

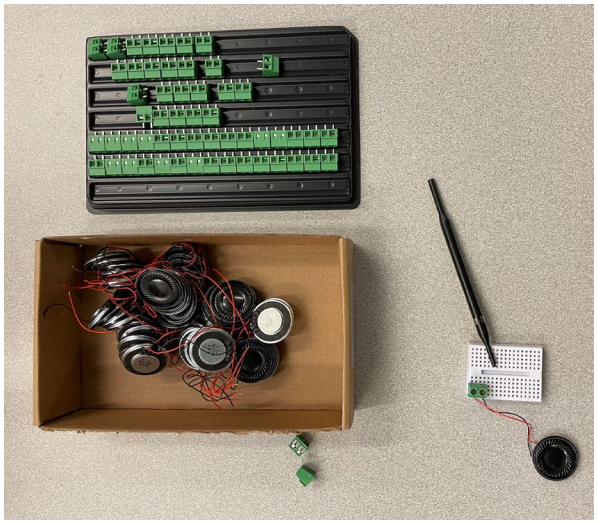
Vikram Iyer

Adapted from material by Blake Hannaford and Justin Hsia

Administrative

- HW1 Autograder
  - If you get an error we will grade manually and give partial credit

- Lab 1- speakers in the lab at the TA desk



Administrative

October				
Monday	Tuesday	Wednesday	Thursday	Friday
07 14:30-15:50 Lecture MOR 220 *Lecture 4: Number Representation and Bitwise Operators* slides, slides (annotated) 16:00-17:00 OH (Vikram) ECE 345	08 12:00-14:00 OH (Yousef) ECE 345 14:00-16:00 OH (Kinner) ECE 345	09 12:30-14:30 OH (Kinner) ECE 345 14:30-15:50 Lecture MOR 220 *Lecture 5: Hardware and Machine Organization* 16:00-17:00 OH (Vikram) ECE 345	10 12:00-14:00 OH (Kurt) ECE 345 16:00-17:00 OH (Yousef) ECE 345	11 11:00-13:00 OH (Kurt) ECE 345 15:00-16:00 OH (Yousef) ECE 345 23:59 C Programming 1 due
14 14:30-15:50 Lecture MOR 220 *Lecture 6: Working with Registers and IMU Demo* 16:00-17:00 OH (Vikram) ECE 345	15 12:00-14:00 OH (Yousef) ECE 345 14:00-16:00 OH (Kinner) ECE 345	16 14:30-15:50 Lecture MOR 220 *Lecture 7: ATMega 2560 Datasheet and Timers* 16:00-17:00 OH (Vikram) ECE 345	17 12:00-14:00 OH (Kurt) ECE 345 14:00-16:00 OH (Kinner) ECE 345 16:00-17:00 OH (Yousef) ECE 345 23:59 Lab 1 due	18 11:00-13:00 OH (Kurt) ECE 345 15:00-16:00 OH (Yousef) ECE 345
21 14:30-15:50 Lecture MOR 220 *Lecture 8: Tasks, Threads, Scheduling I, Interrupts* 16:00-17:00 OH (Vikram) ECE 345	22 12:00-14:00 OH (Yousef) ECE 345 14:00-16:00 OH (Kinner) ECE 345	23 14:30-15:50 Lecture MOR 220 *Lecture 9: Reading Analog Data and Intro to Interrupts* 16:00-17:00 OH (Vikram) ECE 345 23:59 C Programming 2 due	24 12:00-14:00 OH (Kurt) ECE 345 14:00-16:00 OH (Kinner) ECE 345 16:00-17:00 OH (Yousef) ECE 345	25 11:00-13:00 OH (Kurt) ECE 345 15:00-16:00 OH (Yousef) ECE 345

(tentative, posted soon)

Last time

- Allocating memory
- Number representation
  - Overview of Binary and Hex
  - Byte ordering
  - Endianness
  - Encoding integers
- Logical operators
  - Boolean logic
  - Bitwise operators

Plan for today

- Intro to Lab 1
- Hardware architecture- what's inside a processor?
  - Registers
  - Data bus and signaling
  - How a processor works
  - Example of code execution

Lab Assignment 1 CSE474 Autumn 2024

Prof. Vikram Iyer[1] University of Washington

Getting Started with the Arduino Mega

Learning Objectives

With successful completion of this lab, the student will be able to

- Install and set up Arduino IDE
- Build and run a basic sketch (program) using the Arduino Libraries
- Modify and demonstrate blinking light code and speaker output tone.
- Learn to use an oscilloscope for debugging

Lab Overview and Policies CSE/ECE 474

University of Washington 24-Sept-2024

Spring 2024

Vikram Iyer[1]
Kurt Gu
Kinner Parikh
Yousef Gomaa

Introduction to the Labs

The Lab assignments are (see 474 main spreadsheet for links):

- Lab 1 Getting Started with the Arduino Mega
- Lab 2 Digital I/O and timing of outputs
- Lab 3 Round Robin Scheduling and multitasking
- Lab 4 FreeRTOS and Project

This document contains general advice and policy for the labs.

LAB ASSIGNMENTS

The lab projects are a significant part of your grade in the course. Each lab **builds on the previous lab**, so it's important that you keep up with assignments and also ensure that your designs are robust and well tested.

Lab report template

[Your Last Names] CSE 474

Title of your work

[Your Name] [Student #] [Turn-in Date]
Assignment: [Assignment Name]

Page length guidance for written documents refers to single-spaced, 11 or 12 point font.

Content ….

Example: (top of first page:)

Lab 1: Getting Started with the Arduino Mega

Kyle Johnson, 12345678 12-Apr-2023
Vicente Arroyos, 2345678
Assignment: ECE474 Lab 1

… your work …

CSE/ECE 474 Code standard

Vikram Iyer
Kurt Gu
Kinner Parikh
Yousef Gomaa

Rev. Sept 2024

This document describes the formatting requirements for C source code in ECE474.

Libraries and online code

You are welcome to use example code online either directly or as a guide. For any such code you use, you **must** cite the source. This must be included in your source code. This is an easy step that you should get in the habit of doing.

Constants

All constants should be in **symbolic** form, defined at the top of the code, or in a `.h` file included at the top of your code. This is an important feature of maintainable code. All constant names and variable names must be descriptive. Descriptive names are names that tell you something about what the variable or constant is used for. Example:

```
if (a > 31) { ...
```

This is a dangerous piece of code. First, it is hard to understand. Why is 31 important? Why do we care if `a` is greater than 31? Second, it is hard to modify. Usually, this constant must appear in more than one place. Suppose we need to scale up the software to solve a bigger problem? How do we change all of the 31s to 301? Sure you can use a global search and replace, but what if there are other 31s (like `x=247314`) that we do not want to change? Finally, what the heck is "`a`" anyway? The variable should have a **name that means something** in the

CSE/ECE 474 Resource Guide:

Basics of electronics hardware and breadboarding

University of Washington **V 0.1 30-Mar-2023**

Learning Objectives:
After completing this unit, students will be able to

- Visually identify basic electronic components by site and explain their high level uses.
- Visually identify the basic electronics tools required in 474
- Demonstrate the use of the basic electronics tools used in 474
- Build an LED controlled by a power source, switch, and a current limiting resistor using a solderless breadboard.
- Use a digital multimeter (DMM) to make Voltage, Current, and Resistance Measurements.

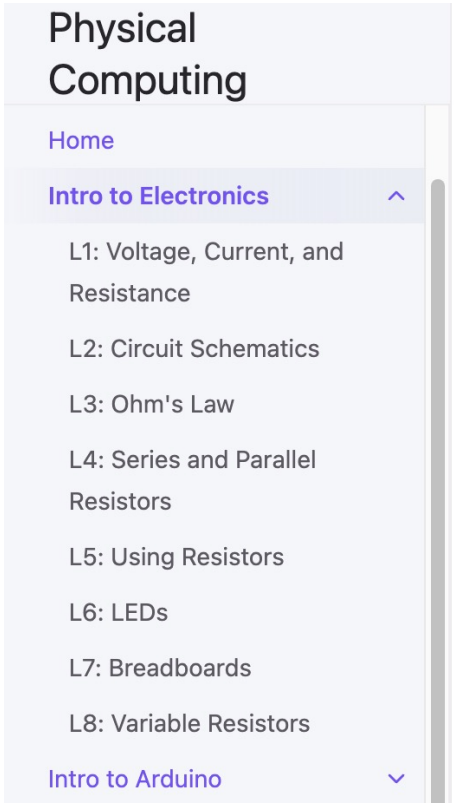
Materials:
These are materials we have reviewed. Criteria for inclusion:

- SHORT videos that are to the point
- CORRECT information
- APPLICABLE to ECE474

Unit 1: Components, Tools, and Basics

The common parts used in almost all electrical circuits are: resistors, capacitors, LEDs, inductors, and transistors. In this course, we will mainly be working with resistors, capacitors, LEDs. You can find more information about all these parts on this website. [LINK]

Intro to electronics resources



Lesson 1: Voltage, current, and resistance

Introduces three key electricity concepts, current, voltage, and resistance, which form the foundation of electronics and circuits.

Lesson 2: Circuit Schematics

Introduces a visual language for describing circuits called circuit schematics, which are used in component datasheets, CAD programs (*e.g.,* circuit simulators, PCB layout software), and circuit analyses. Also includes an activity using Fritzing to build your own schematics.

Lesson 3: Ohm's Law

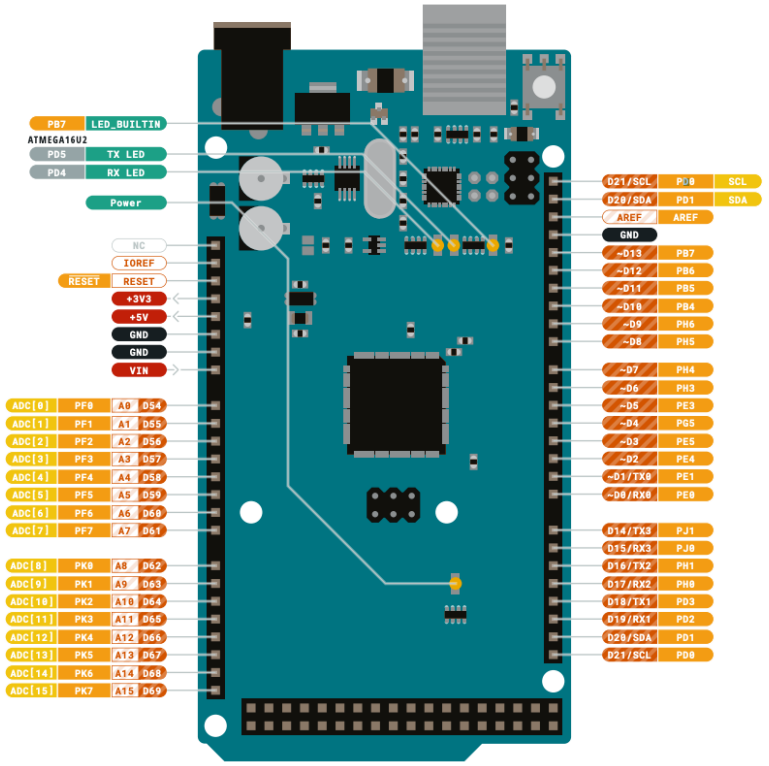
Introduces Ohm's Law, one of the most important empirical laws in electrical circuits that describes how current, voltage, and resistance relate together. Also includes an activity to build and explore resistive circuits in CircuitJS.

Lesson 4: Series vs. Parallel Resistors

https://makeabilitylab.github.io/physcomp/

ARDUINO

ARDUINO
MEGA 2560 REV3
STORE.ARDUINO.CC/MEGA-2560-REV3



Ground
Power
LED
Internal Pin
SWD Pin

Digital Pin
Analog Pin
Other Pin
Microcontroller's Port
Default

MAXIMUM current per I/O pin is 20mA

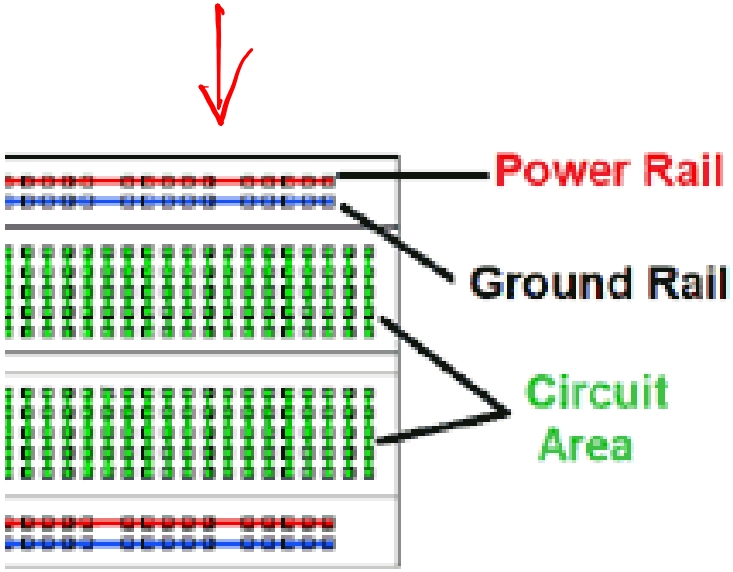
MAXIMUM current per +3.3V pin is 50mA

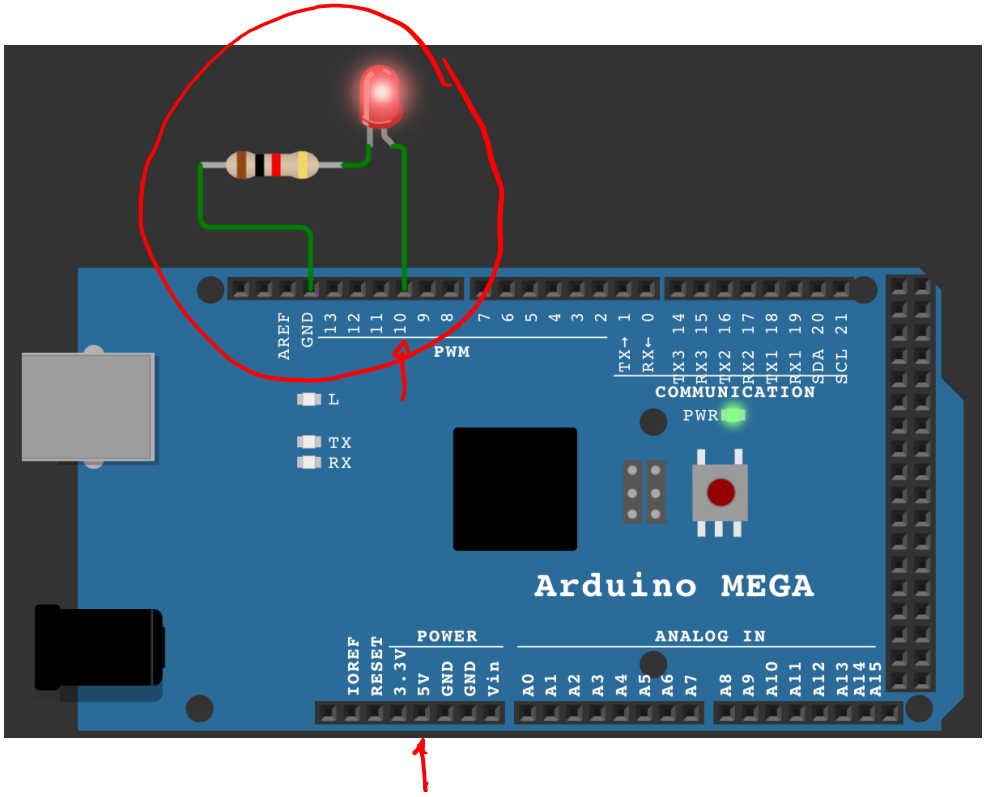
VIN 6-20 V input to the board.

ARDUINO.CC

Last update: 16/12/2020

Setting up Arduino


Power Rail
Ground Rail
Circuit Area

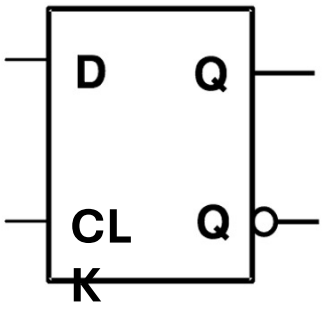

AREF
GND
13
12
11
10
9
8
7
6
5
4
3
2
PWM
TX→ 1
RX← 0
14
15
16
17
18
19
20
21
COMMUNICATION
SDA
SCL
L
TX
RX
PWR
Arduino MEGA
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15

Register

Register- an array of D flip-flops which can store a collection of bits

An *n* bit register has *n* inputs, *n* outputs, and one clock line

Flip flop: Memory element that stores 1 bit

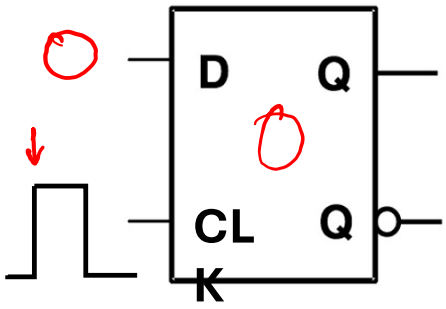


Register

Register- an array of D flip-flops which can store a collection of bits

An *n* bit register has *n* inputs, *n* outputs, and one clock line

Flip flop: Memory element that stores 1 bit

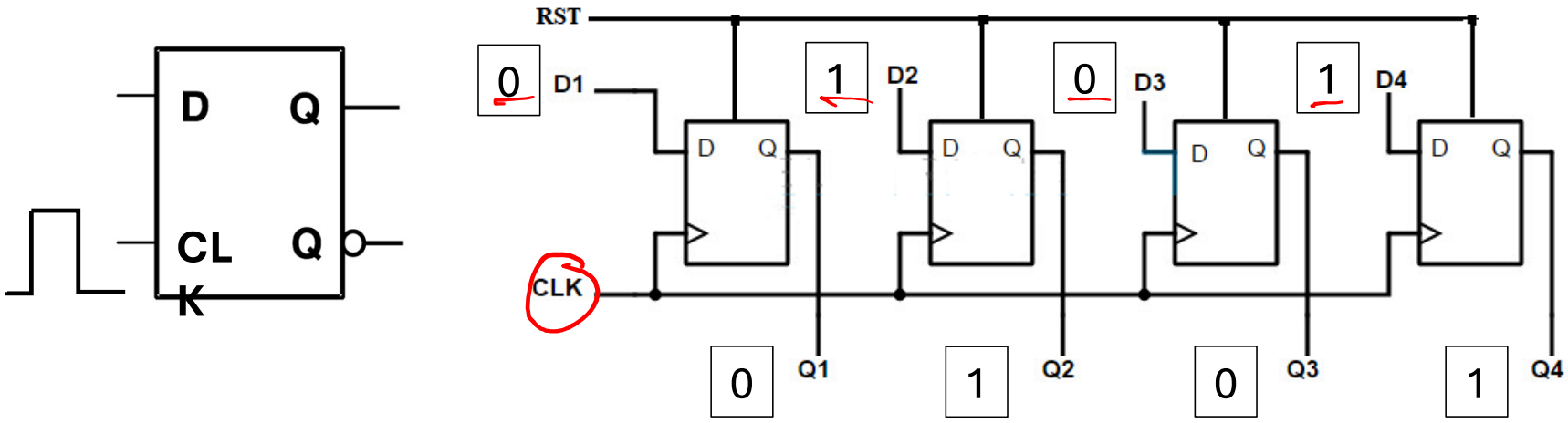


Register

Register- an array of D flip-flops which can store a collection of bits

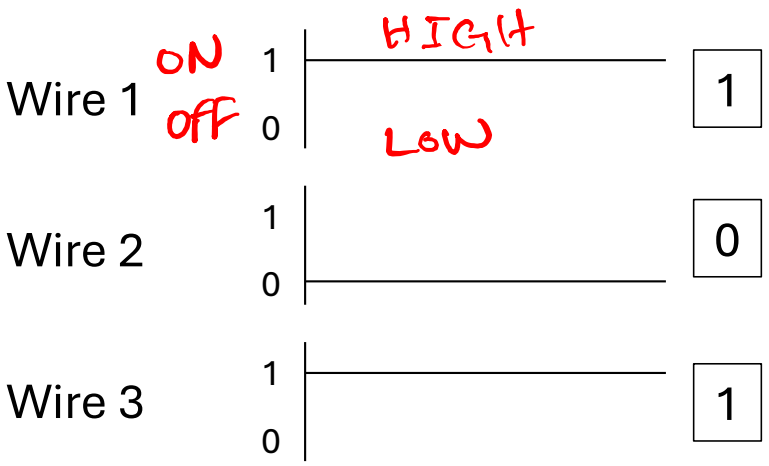
An *n* bit register has *n* inputs, *n* outputs, and one clock line

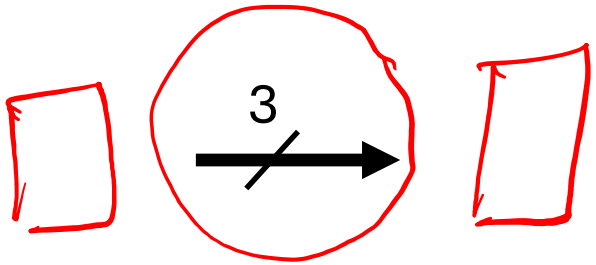
Flip flop: Memory element that stores 1 bit



Buses

Bus- A parallel, bi-directional datapath

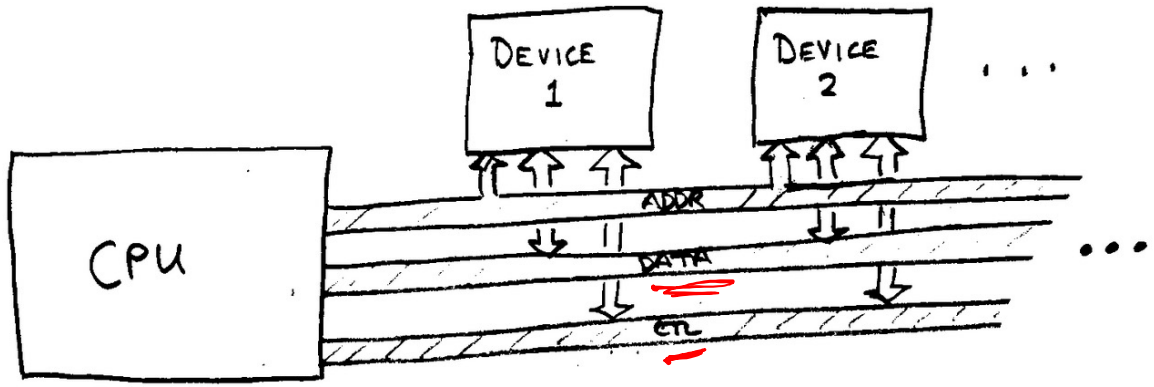




Buses

Bus- A parallel, bi-directional datapath

Multiple devices can send and receive data on one bus



- Address: N bits which specify which location.
- Data: Contents to/from memory or I/O device
- Control: 'Traffic signals'

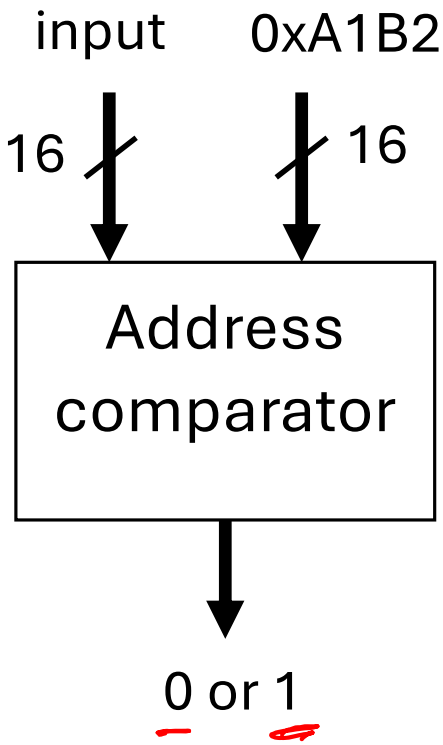
Detailed view

Address Comparator

- A combinatorial logic circuit at each device
- 2 n-bit binary inputs:

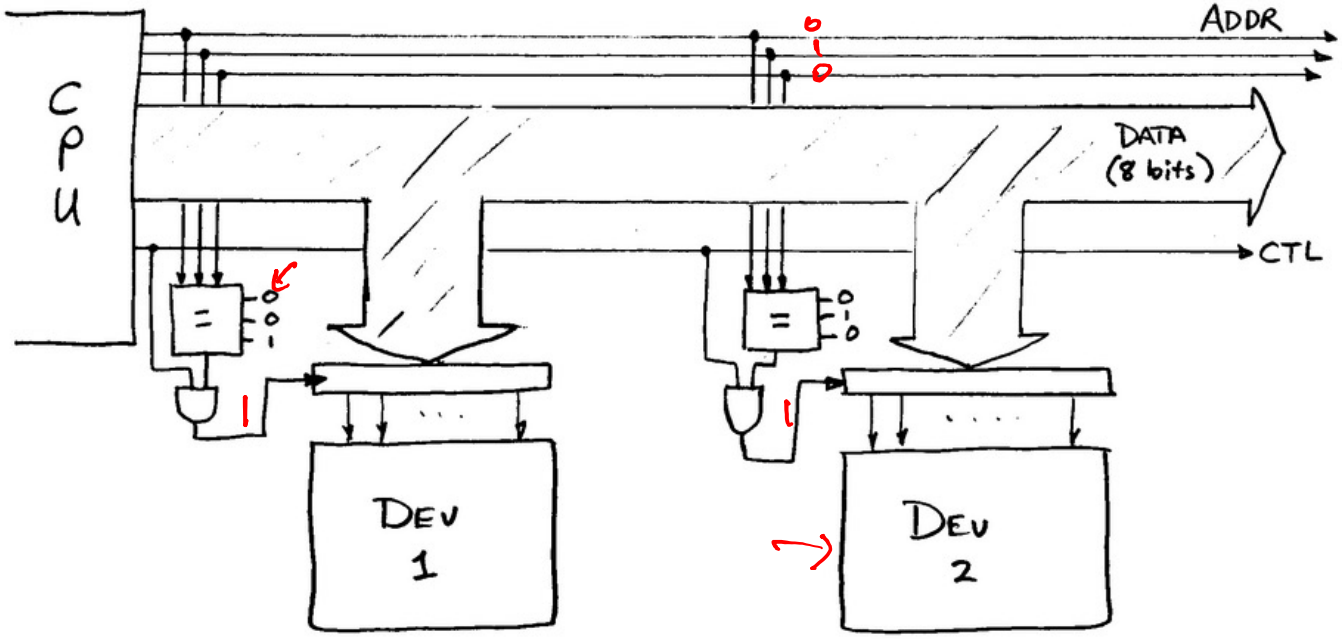
1. ADDR bus (changing)
2. Device address (fixed)

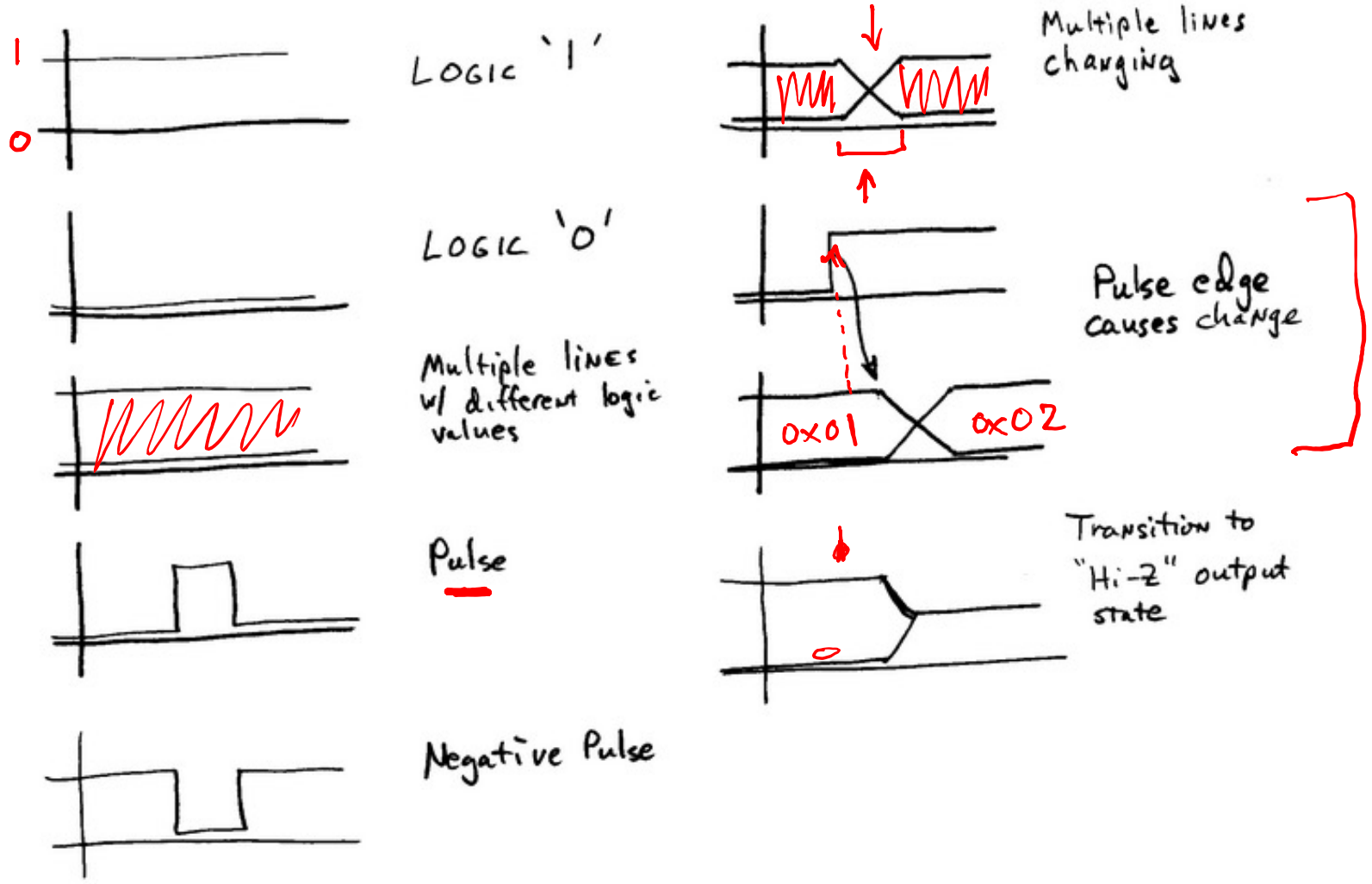
- Control input
- Output = 1 if input1 == input2



Detailed view

- Control bus is gated by ADDR comparator
- Device register is strobed by CTL pulse only if addresses match.

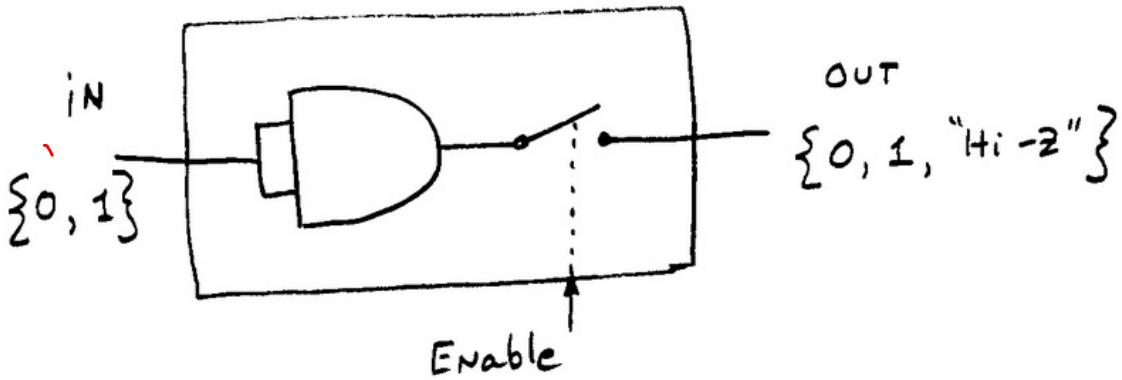



1
0
LOGIC '1'
LOGIC '0'
Multiple lines w/ different logic values
Pulse
Negative Pulse
Multiple lines changing
Pulse edge causes change
0x01
0x02
Transition to "Hi-Z" output state

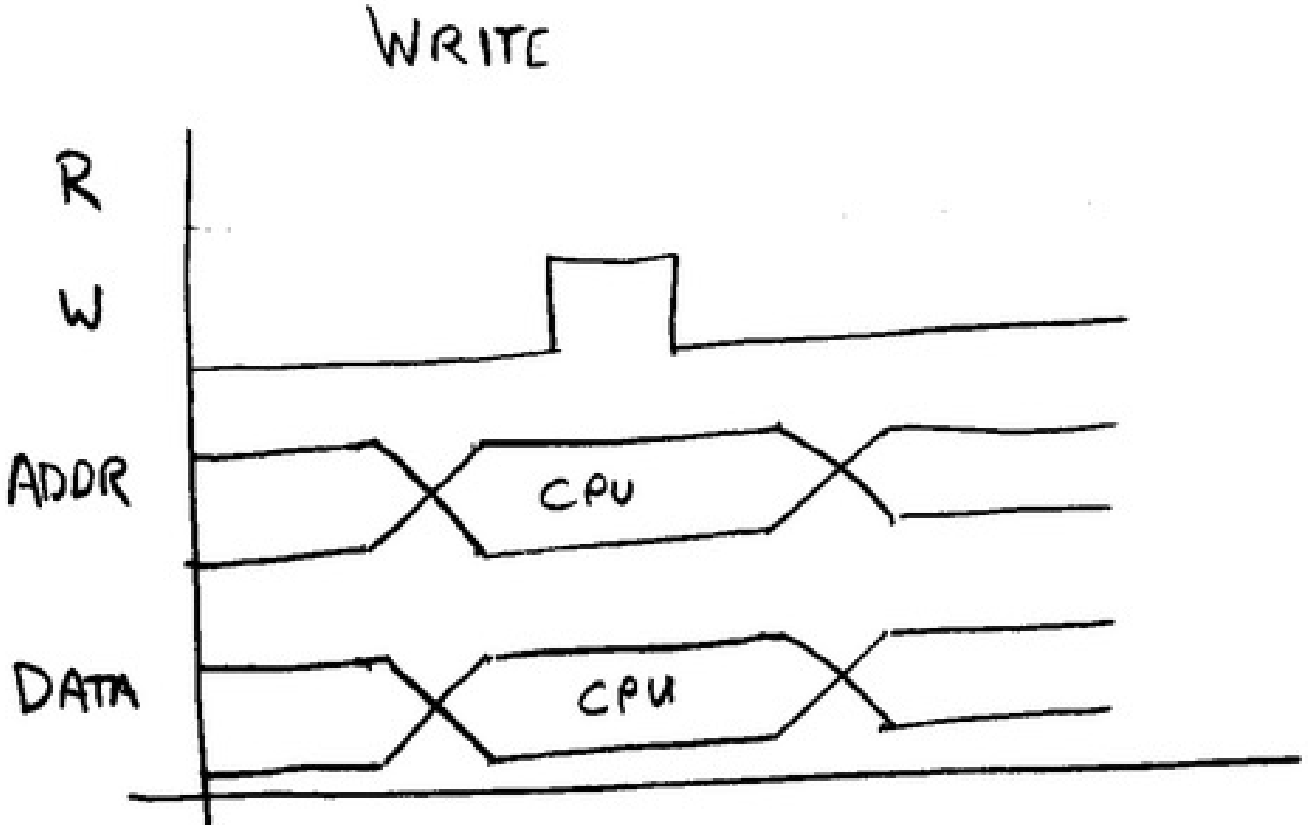
How do devices share a bus?

Solution: Tri-state logic

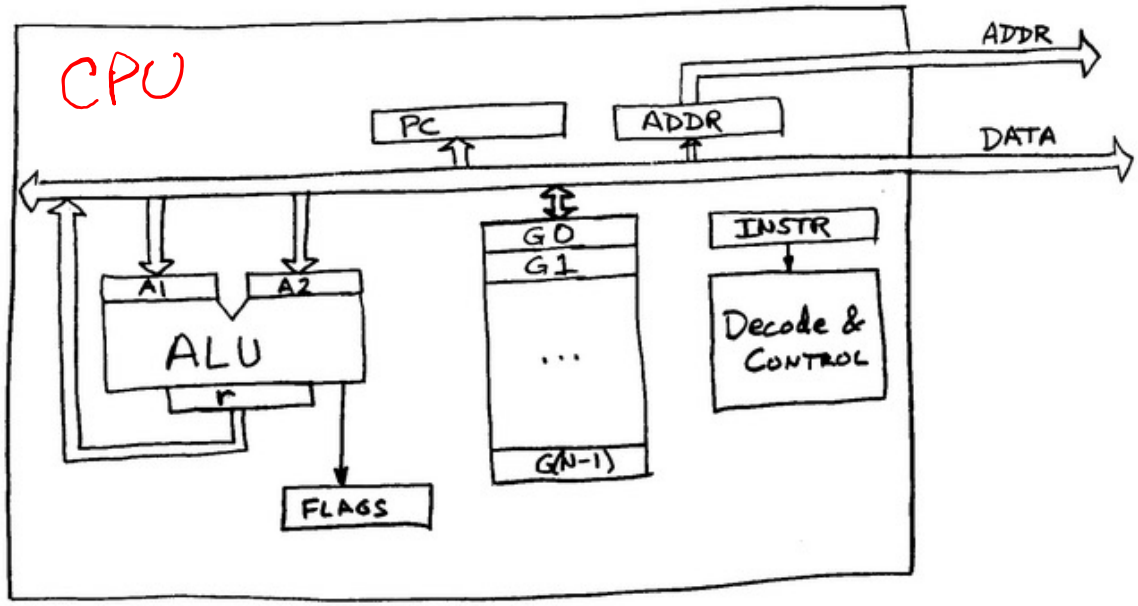
- States: Logic 1, Logic 0, switch OFF
- CPU and all devices can “talk” on data bus.
- Only ONE device my have the tri-state switch closed at any time!



Bus timing: Write cycle

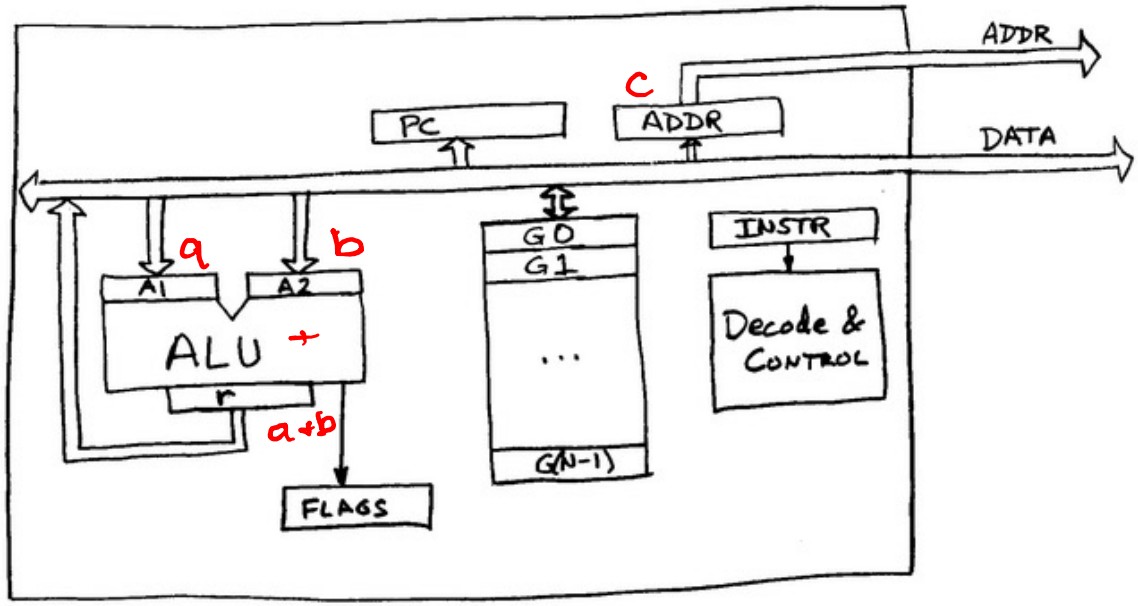


What happens inside our processor?



- ALU: Arithmetic & Logic Unit
- PC: Program Counter. Holds Address of next instruction.
- IR: Instruction Register. Holds current instruction.
- ADDR: Address Register. Holds address of next bus access.
- GP#: General Purpose Registers. Hold intermediate results.
- A1, A2: ALU Arguments. Hold inputs to an ALU operation.
- r: 'Result', holds result of ALU operation.
- FLAGS: a set of bits which tell things like zero/non-zero, negative, etc about the last Result.

Example: Adding two numbers



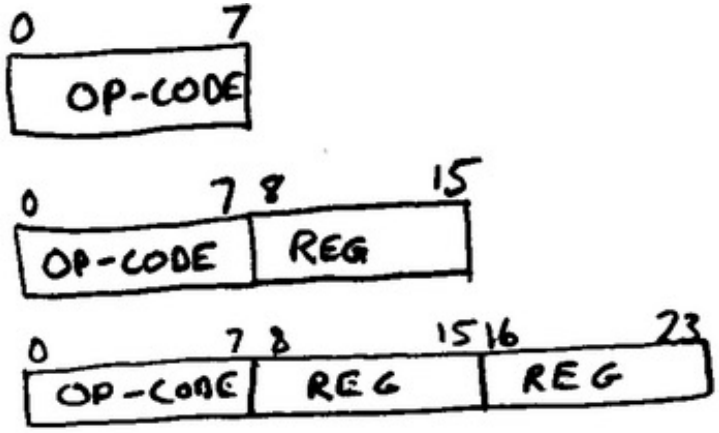
C statement: `c = a + b ;`

1. Load addr of `a` into ADDR
2. Wait for data from memory
3. Clock data into A1
4. Load addr of b into ADDR
5. Wait for data from memory
6. Clock data into A2
7. Send ADD command to ALU
8. Load addr of `c`
9. Transfer RES to data bus
10. Wait for data write to memory.

Machine Instructions

The processor is controlled by machine instructions. A machine instruction is binary data typically broken up into fields:

- The Operation Code (Op-Code)
- One, two, or three Operands.
- Each instruction is typically between one and eight bytes



Example: Add two numbers

Note: All assembler below is pseudo-code!

C code: `c = a + b ;`

Assembler output

mem addr	instruction	comment
0xA000	MOV a, A1	move mem location a to ALU Arg 1
0xA002	MOV b, A2	move mem location b to ALU Arg 2
0xA004	ADD	a one-byte instruction
0xA005	MOV r, c	move ALU result to mem location c

Atmel I/O Instructions

Example: AtMega2560 Chip (Arduino Mega)

- “Memory Mapped” I/O
- Devices and Memory share the same address space
- Device registers are just like memory locations / variables.
- Arduino libraries predeclare correct memory address for each register you need.