Lecture 5: Hardware & Machine Organization

Vikram Iyer

Adapted from material by Blake Hannaford and Justin Hsia

Administrative

April

Monday	Tuesday	Wednesday	Thursday	Friday
08 9:30-11:30 OH (Deeksha) ECE 345 11:30-13:30 OH (Alex) ECE 345	09 13:00-15:00 OH (Zach) ECE 345	10 9:30-11:30 OH (Deeksha) ECE 345 12:30-14:20 Lecture MOR 230 *Lecture 5: Hardware and Machine Organization* 14:00-15:00 OH (Vikram) ECE 345	11 10:00-12:00 OH (Zach) ECE 345 12:30-14:30 OH (Alex) ECE 345	12 12:30-14:20 Lecture MOR 230 *Lecture 6: Working with Registers and IMU Demo* 14:00-15:00 OH (Vikram) ECE 345 23:59 C Programming 1 due
15 9:30-11:30 OH (Deeksha) ECE 345 11:30-13:30 OH (Alex) ECE 345	16 13:00-15:00 OH (Zach) ECE 345	17 9:30-11:30 OH (Deeksha) ECE 345 12:00-13:00 Remote OH (Vikram) Zoom	18 10:00-12:00 OH (Zach) ECE 345 12:30-14:30 OH (Alex) ECE 345 23:59 Lab 1 due	19 12:30-14:20 Lecture MOR 230 *Lecture 7: ATMega 2560 Datasheet and Timers* 14:00-15:00 OH (Vikram) ECE 345
22 9:30-11:30 OH (Deeksha) ECE 345 11:30-13:30 OH (Alex) ECE 345	23 13:00-15:00 OH (Zach) ECE 345	24 9:30-11:30 OH (Deeksha) ECE 345 12:30-14:20 Lecture MOR 230 *Lecture 8: Reading Analog Data and Intro to Interrupts* 14:00-15:00 OH (Vikram) ECE 345	25 10:00-12:00 OH (Zach) ECE 345 12:30-14:30 OH (Alex) ECE 345 23:59 C Programming 2 due	26 12:30-14:20 Lecture MOR 230 *Lecture 9: Tasks, Threads, Scheduling I, Interrupts* 14:00-15:00 OH (Vikram) ECE 345

CSE/ECE 474 C-Programming Assignment 2: C Programming with bit manipulation and structs

Updated: April 8, 2024

In this assignment we will learn how to inspect and modify individual bits in memory. Remember that everything in computers is represented by just a series of 0's and 1's. This means that the software has to remember how each part of memory is converted from 0's and 1's to something else. Here's a simple example: Suppose the memory location 0xAFF5C3 contains the following:

0xAFF5C3 →

0	0	1	0	1	0	1	0

We will focus on the exact bits in memory and the powerful (but yes, low-level) C-functions which allow you to work with them. This 8-bit segment of memory could be a `uint8_t` type (an 8-bit integer between 0-255). In that case, 00101010 equates to 2+8+32 = 42. However, if it is a `char` (character), the same pattern of 00101010 equates to '*' (the asterisk character).

Working with bits is important because hardware control bits need to be set or cleared in order to control a computer chips. You could turn on or off on-chip peripheral devices to save battery life for example.

Instructions

This is an individual assignment

We have provided template C files, **`c_prog2.c`** and **`c_prog2.h`** which contain comments indicating where to put each part of your code as well as directions and sample outputs. **These are the two files you will turn in on canvas.** In addition to this we have provided the file **`c_prog2_arduino.ino`** contains a set of function calls in the **`setup()`** function that will run your code. A sample output is provided in **`sample_output.txt`**.

Lab Assignment 1 CSE474 Spring 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 27-Mar-2024

Spring 2024

Vikram Iyer[1]
Alex Ching
Zachary Englhardt
Deeksha Prabhu

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
Alex Ching
Zachary Englhardt
Deeksha Prabhu

Rev. March 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

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.

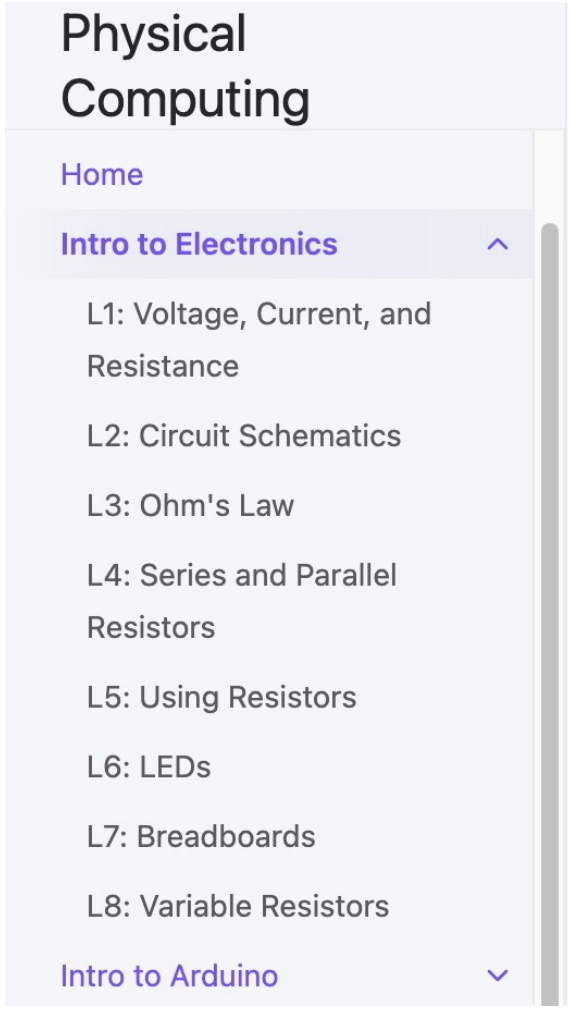
Materials:
These are materials we have reviewed. Criteria for inclusion:

- 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/

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

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

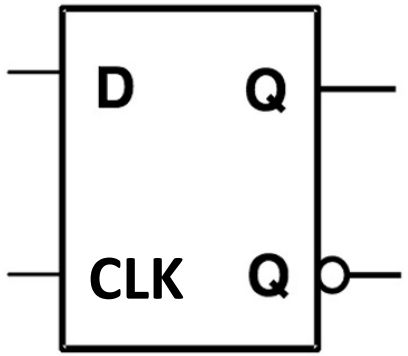
- Getting started with Arduino

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

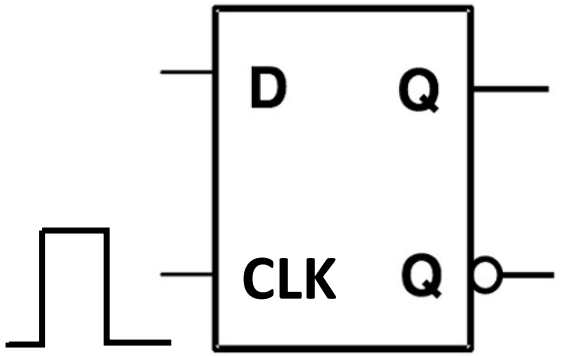


Register

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

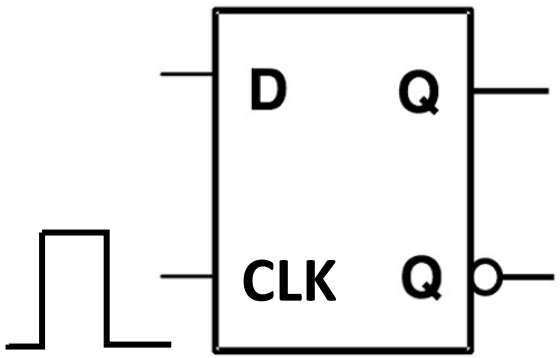


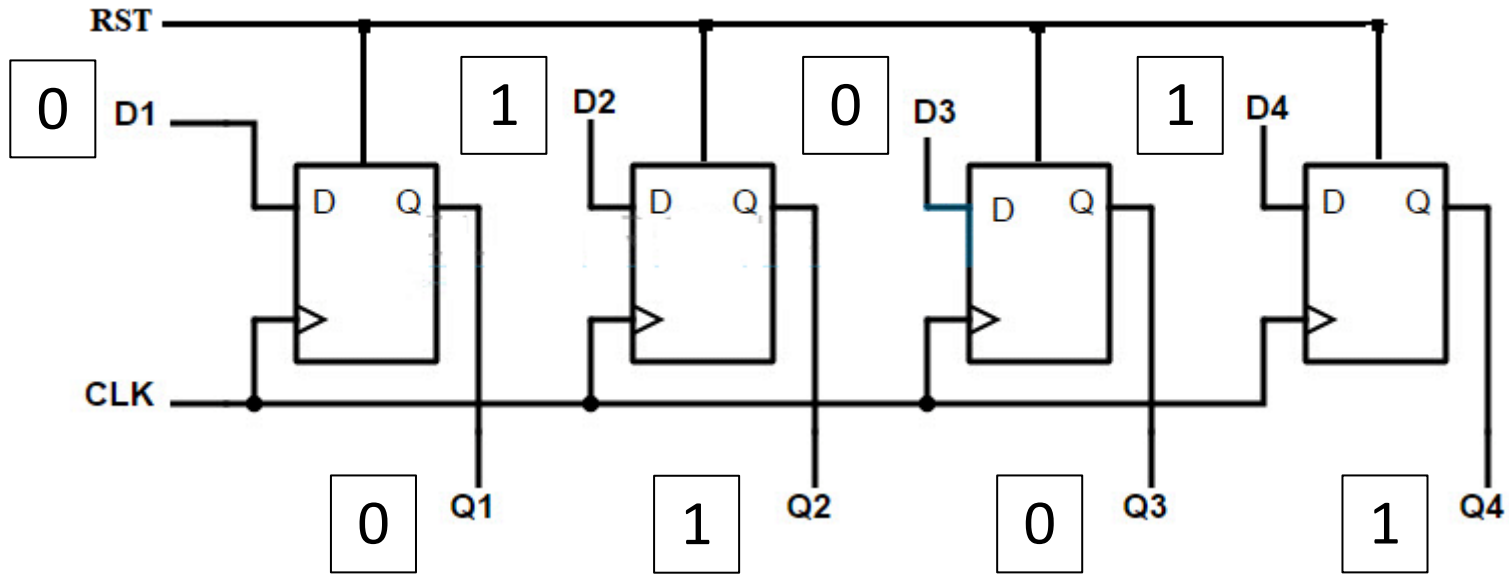
Register

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

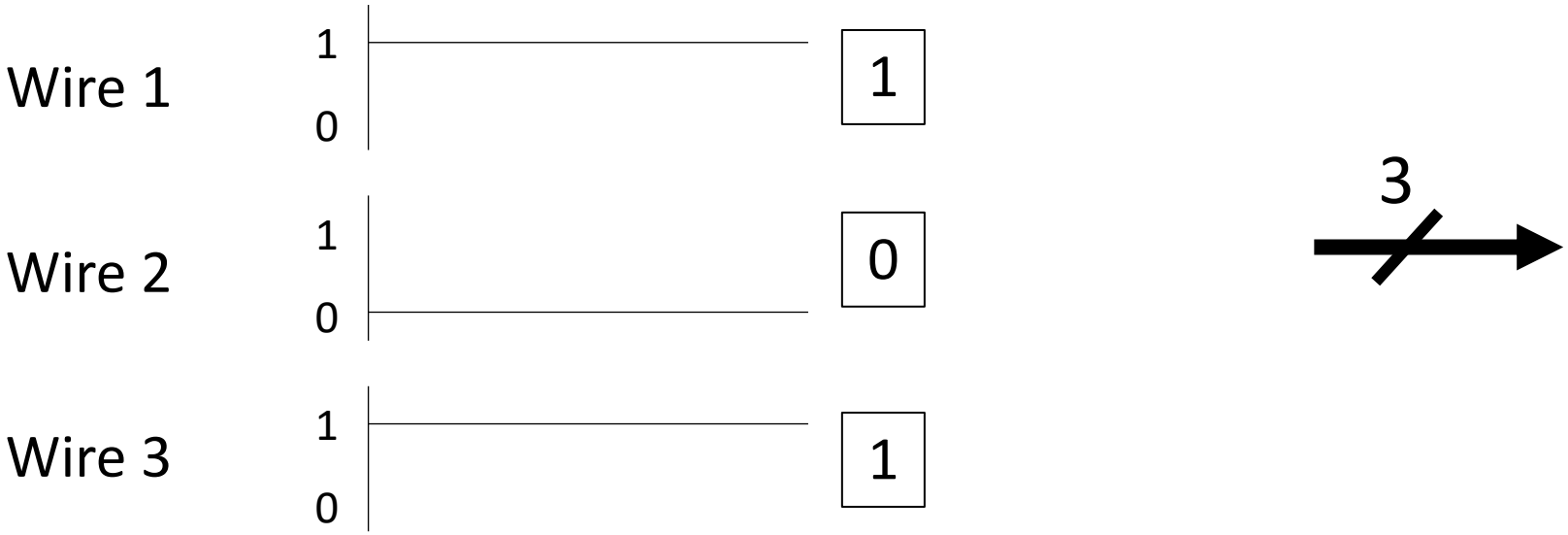
Flip flop: Memory element that stores 1 bit





Buses

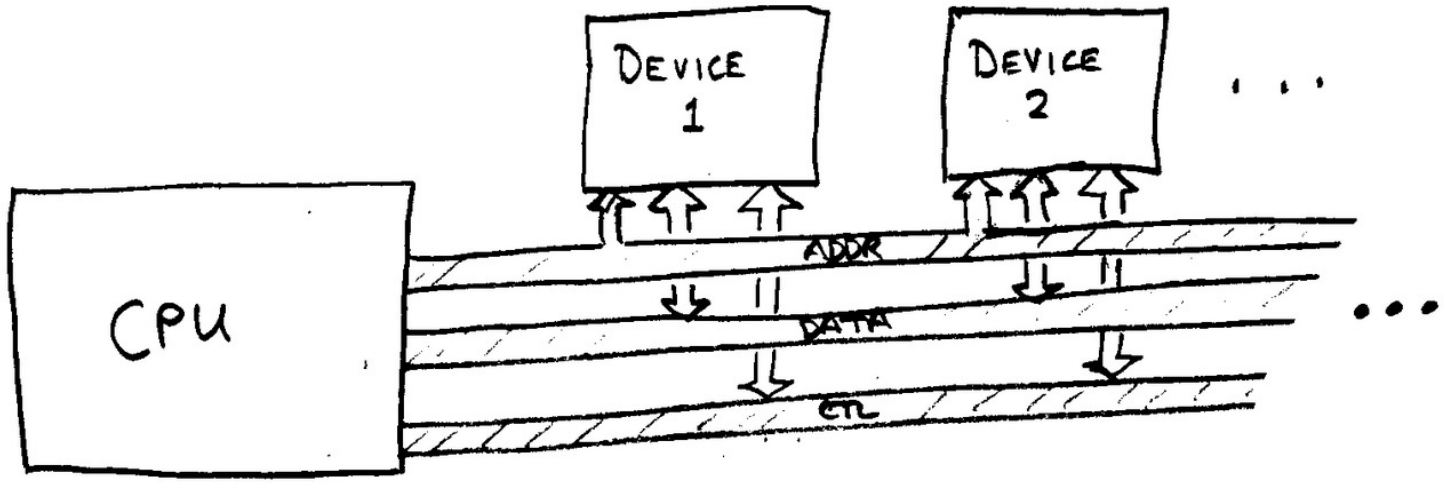
Bus- A parallel, bi-directional datapath



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'

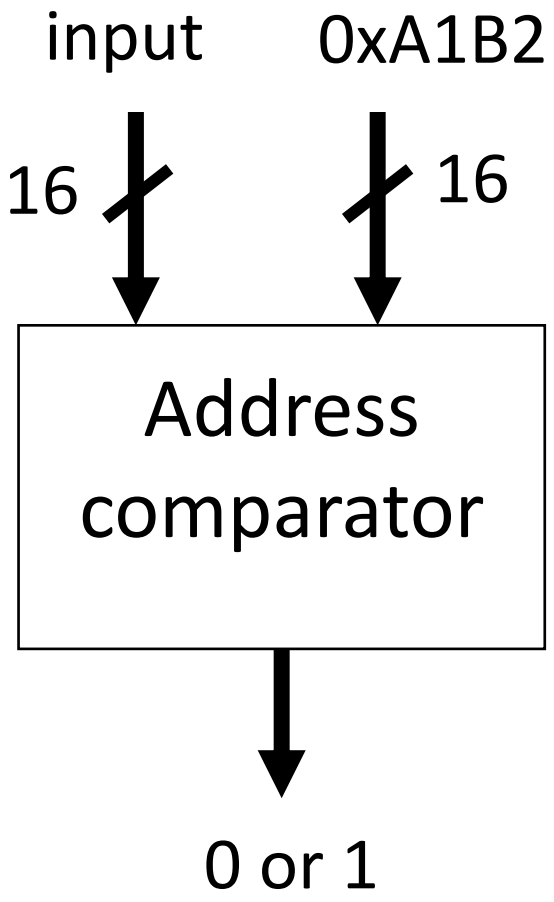
Detailed view

Address Comparator

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

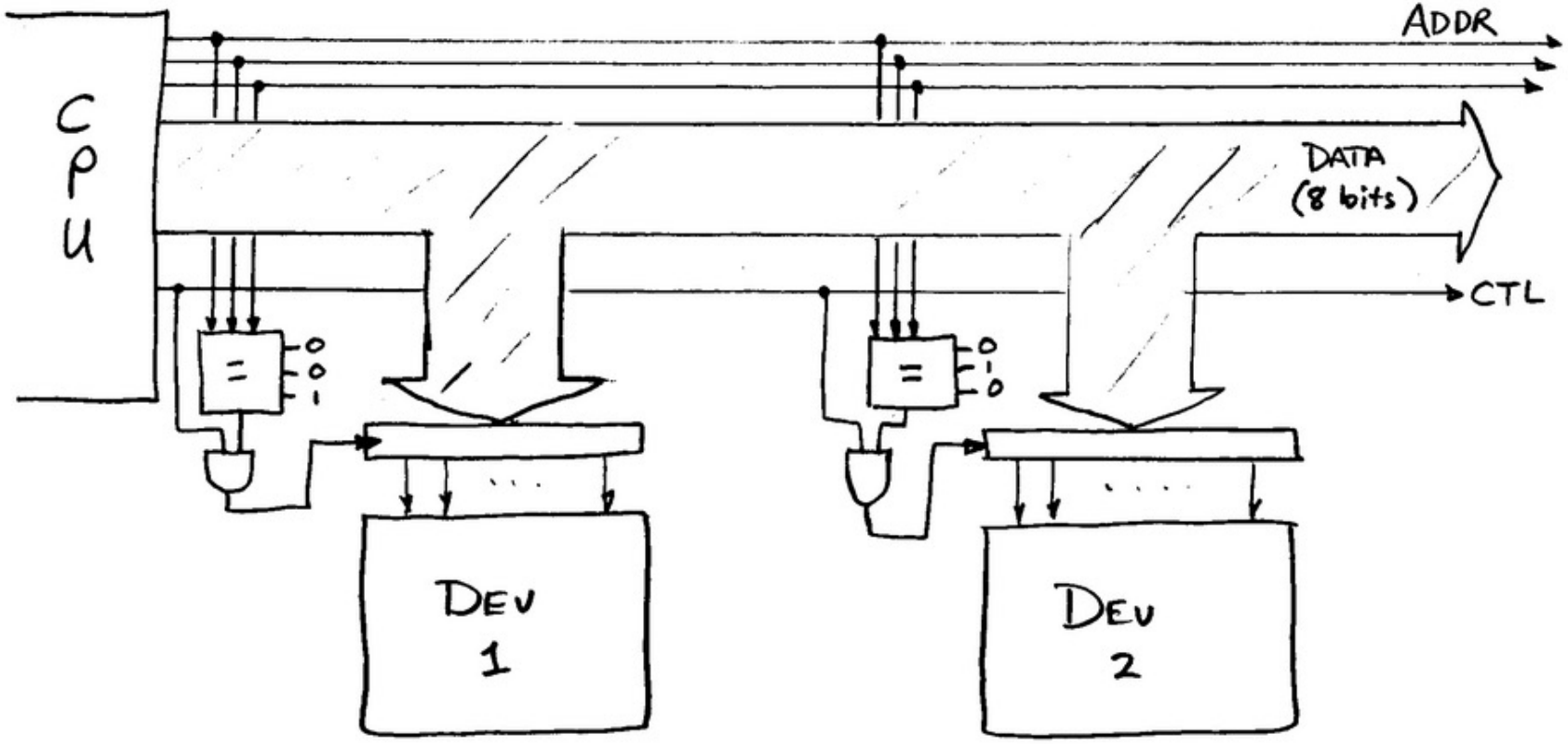
1. ADDR bus (changing)
2. Device address (fixed)

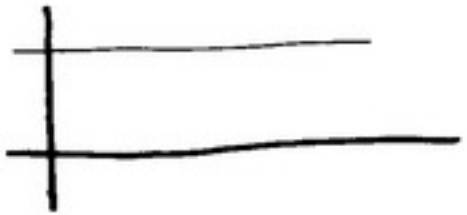
- Control input
- Output = 1 if input1 == input2



Detailed view

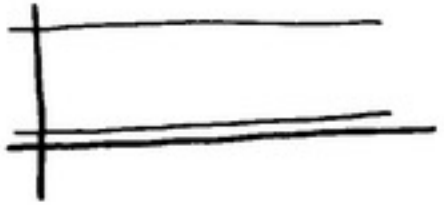
- Control bus is gated by ADDR comparator
- Device register is strobed by CTL pulse only if addresses match.



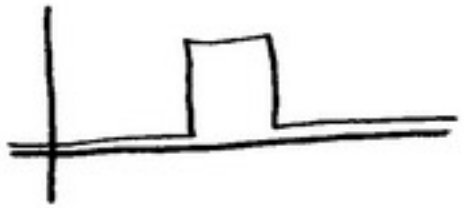


LOGIC '1'

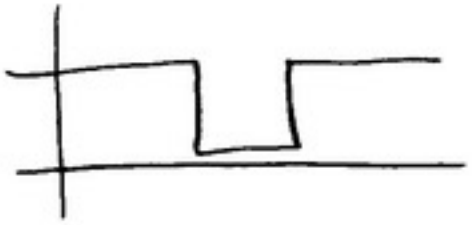
LOGIC '0'



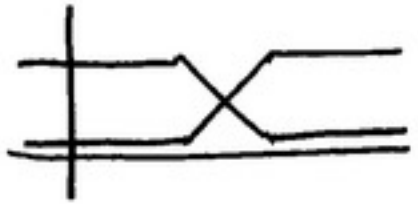
Multiple lines w/ different logic values



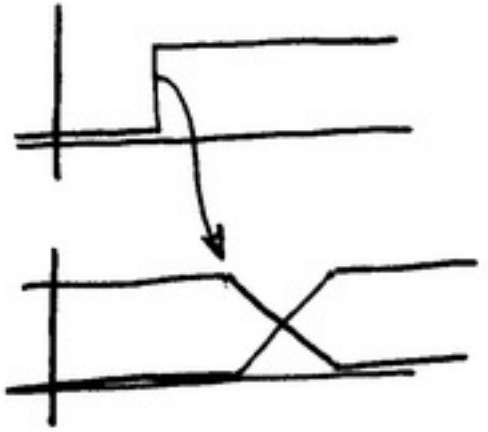
Pulse



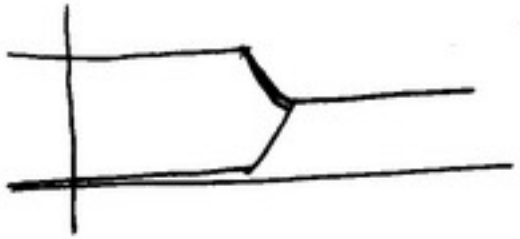
Negative Pulse



Multiple lines changing



Pulse edge causes change

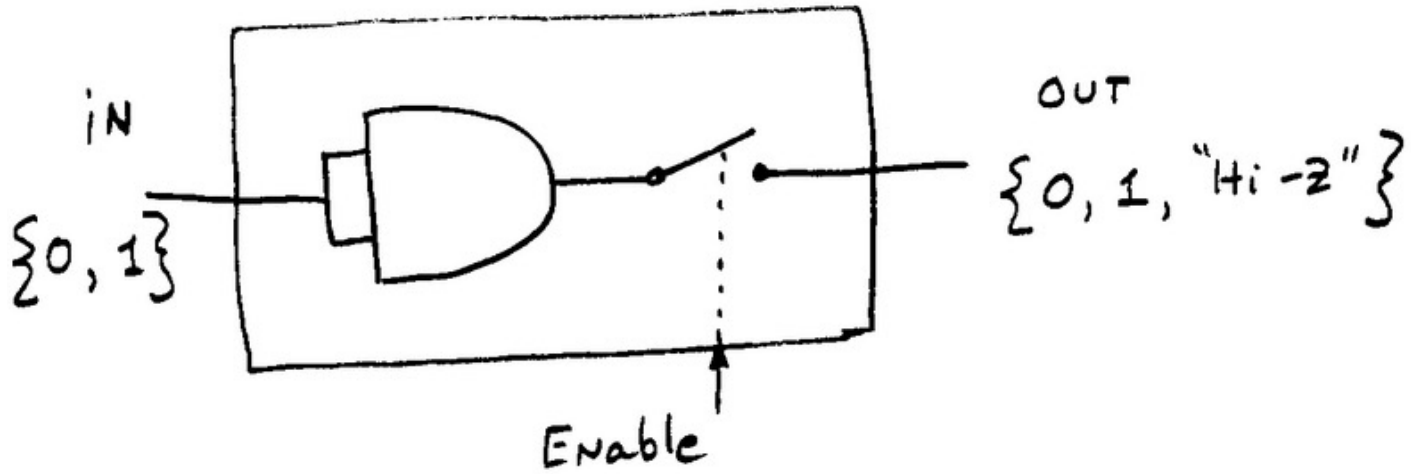


Transition to "Hi-Z" output state

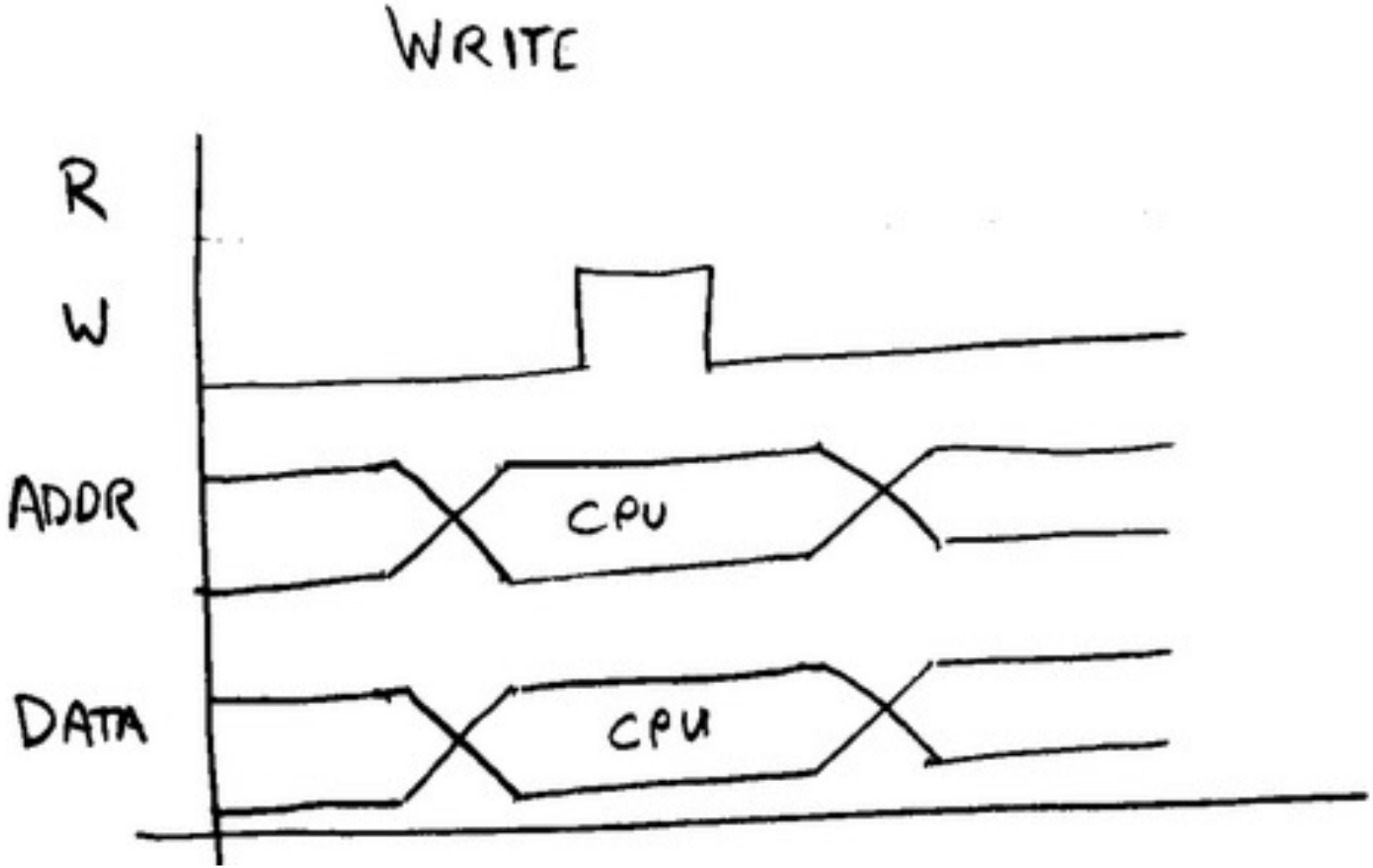
How do devices share a bus?

Solution: Tri-state logic

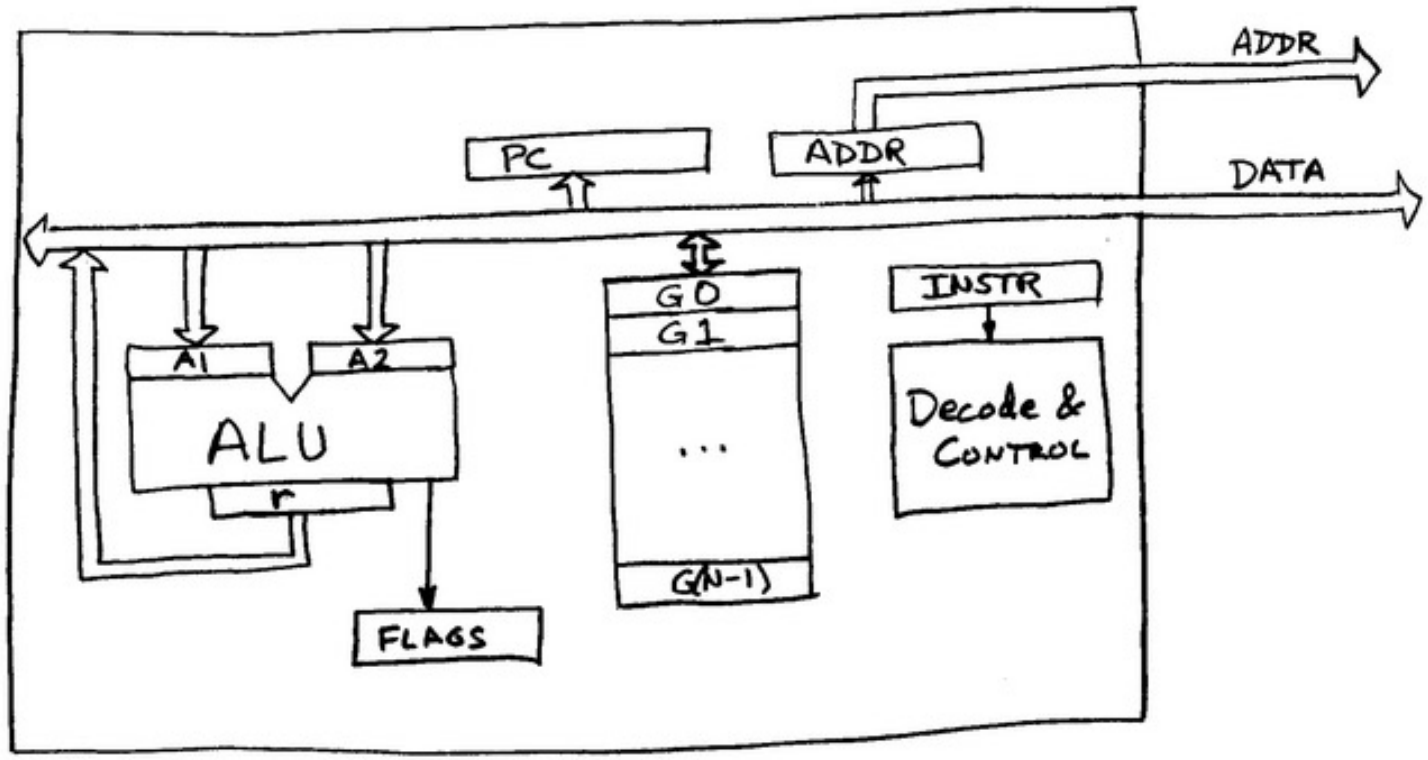
- States: Logic 1, Logic 0, switch OFF
- CPU and all devices can “talk” on data bus.
- Only ONE device my have the tri-state switch closed at any time!



Bus timing: Write cycle

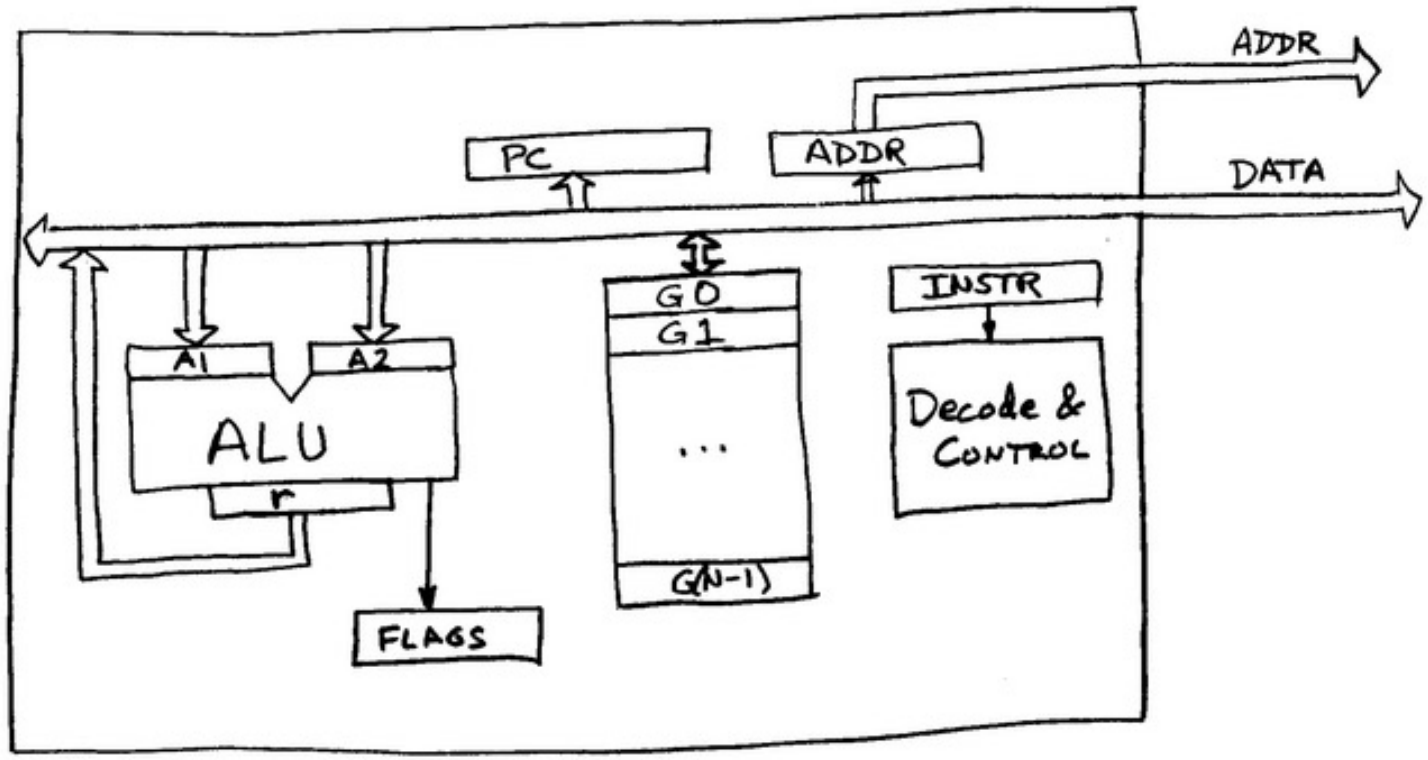


What happens inside our processor?



- 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



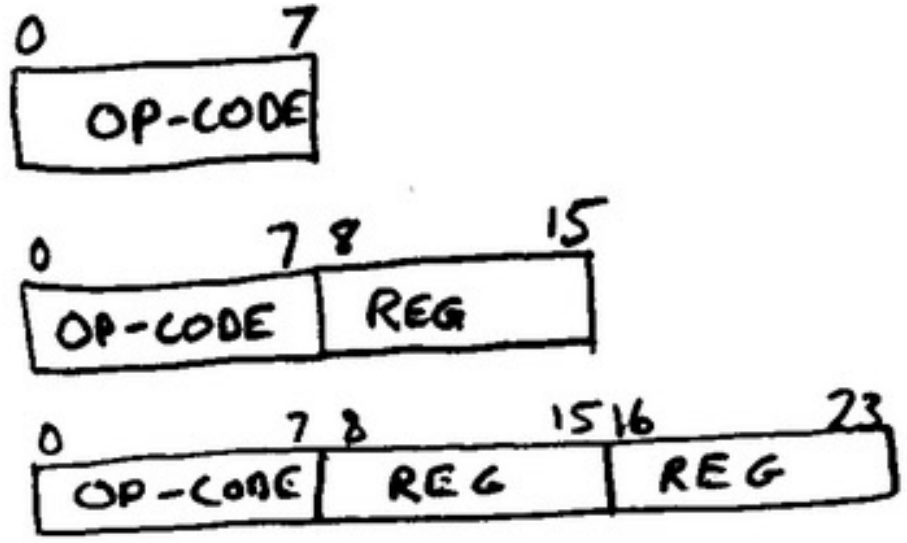
C statement: `c = a + b ;`

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

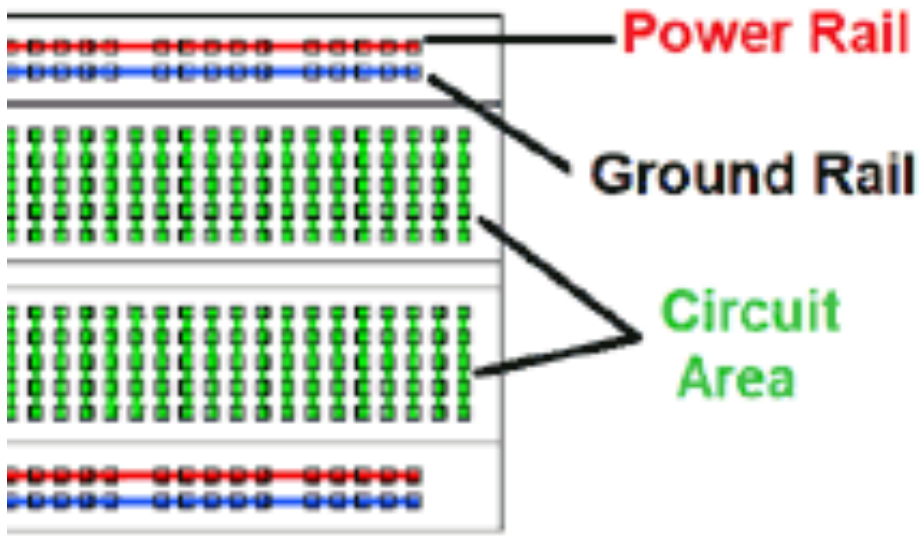
mem addr	instruction	comment
0xA000	MOV a, A1	move mem location a to ALU Arg 1
0xA002	MOV b, A2	move mem location b to ALU Arg 2
0xA004	ADD	a one-byte instruction
0xA005	MOV r, c	move ALU result to mem location c

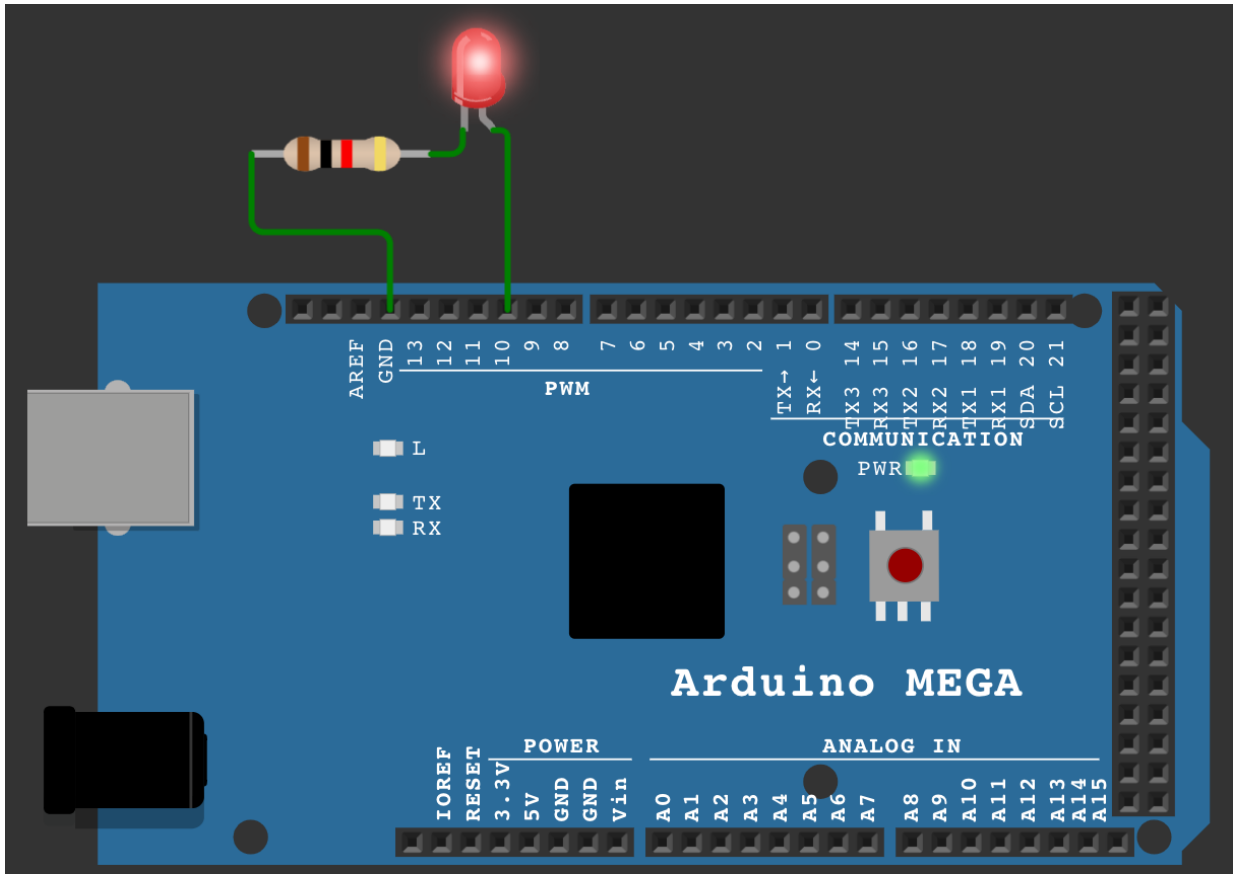
Atmel I/O Instructions

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.

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
TX3 14
RX3 15
TX2 16
RX2 17
TX1 18
RX1 19
SDA 20
SCL 21
COMMUNICATION
L
TX
RX
PWR
Arduino MEGA
POWER
IOREF
RESET
3.3V
5V
GND
GND
Vin
ANALOG IN
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15