Programmering af Mobile Robotter

RB1-PMR – Module 1: Introduction to the course

Jes Hundevadt Jepsen | jegj@mmmi.sdu.dk
SDU Drone Center, MMMI, University of Southern Denmark, Odense, Denmark

Agenda

- Who am I?
- About the course
- My expectations? Your expectations? Your experience?
- Introduction to microcontrollers
- Number systems
- Micro-architecture
- Exercises (Little Man Computer)

Who am I?

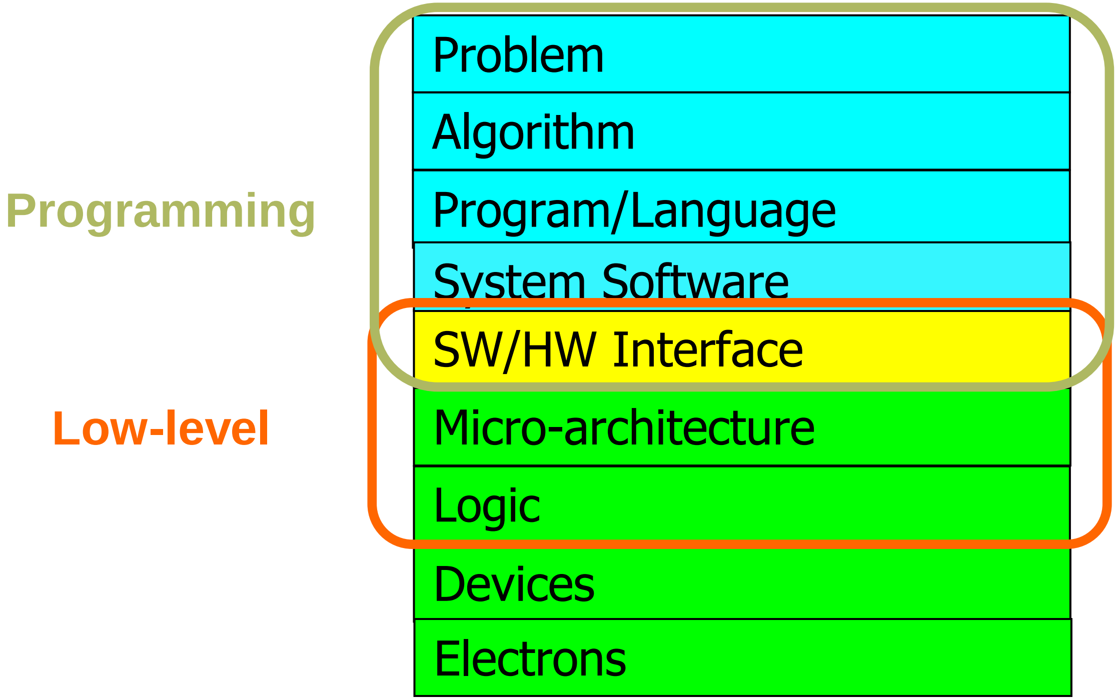
- *Jes Hundevadt Jepsen (jegj@mmmi.sdu.dk)*
- *Background*
  - *BSc, Robot Systems Engineering, SDU*
  - *MSc, Robot Systems Engineering, SDU*
  - *Robotics Developer, Blue Ocean Robotics*
  - *Research Assistant, SDU UAS Centre*
  - *(present) PhD student, SDU UAS Centre*
- *Current research*
  - *Unmanned Aerial System (UAS)*
  - *Traffic management / U-Space*
  - *Communication (5G)*
- *80% Software / 20% Hardware*
- *Practical > Theoretical / High-level > Low-level*
- *Linux and ROS **(C++, Python)***

My expectations?

challenge
MakeAGIF.com

MakeAGIF.com

Our goal?
The Transformation Hierarchy / Computing Stack



Programming

Low-level

Problem

Algorithm

Program/Language

System Software

SW/HW Interface

Micro-architecture

Logic

Devices

Electrons

Problem to solve

Problem can be transformed into an algorithm

The algorithm is written in a programming language

…and executed in a OS as instruction

…that are handled by the HW/SW interface

The lowel architecture handles the instruction, one-by-one

Actives the logic gates

…which are build by transistors, and

Converted it into motion.

Link: https://www.youtube.com/watch?v=cpXdE3HwvK0&t=2200s

About the Course - Content (Indhold)

*“Dette kursus tilbyder en **introduktion til microcontrolleren** med fokus på at anvende den i en **mobil robot-relateret kontekst**. Kurset indeholder både grundlæggende programmering af en microcontroller samt hvordan man kan anvende dens inputs/outputs til at interface sensorer og aktuatorer på en mobil robot. Gennem kurset vil den studerende opbygge en **portefølje af relevante emner og værktøjer.** I slutningen af kurset skal denne portefølje anvendes til at styre en mobil robot ved brug af en microcontroller.”*

Worth discussing / mentioning?

- Portfolio?
- Extra Credit Activities / “Pointgivende aktiviteter”?
- Exam?

Source: https://odin.sdu.dk/sitecore/index.php?a=fagbesk&id=139103&lang=da

Knowledge (Viden)

Efter gennemførelse af kurset har den succesfulde studerende viden om:

- **grundlæggende opbygning** og brug af en microcontroller.
- grundlæggende seriel kommunikation.
- brugen af en microcontrollers input/output i en mobil robot-relateret kontekst.
- **interface sensorer** på en mobil robot med en microcontroller.
- brugen af en microcontroller til indsamling samt **behandling af data.**
- **styre aktuatorer** på en mobil robot med en microcontroller.

Skills (Færdigheder)

Efter gennemførelse af kurset kan den succesfulde studerende:

- udvise grundlæggende forståelse for opbygningen samt **programmering af en microcontroller.**
- styre en mobil robot ved at skrive mindre programmer og eksekvere dem på en microcontroller.
- kombinere elektroniske kredsløb med mikrocontroller, for eksempel som interface til sensorer og aktuatorer på en mobil robot.
- anvende microcontroller til at indsamle og behandle målinger fra sensorer som findes på en mobil robot platform.
- **anvende en microcontroller til at styre aktuatorer på en mobil robot baseret på input, enten fra en bruger eller en sensorer.**
- anvende debugging værktøjer til at teste og fejlfinde mindre programmer.

Competences (kompetencer)

Efter gennemførelse af kurset kan den succesfulde studerende:

- styre en mobil robot ved at programmere en microcontroller og anvende microcontrollerens input/output hertil.
- analysere og udvikle mindre programmer til løsning af et simpelt problem.

Summary….

- Identify a problem / task
- Write a program (in MicroPython)
- Interface sensors
- Control actuators

Course plan?

SDU

Source: https://odin.sdu.dk/sitecore/index.php?a=fagbesk&id=139103&lang=da

Course plan

This plan is tentative, please expect some content changes

Week	Date	Schedule	Location	Module	Content
36	04-09-2024	08:15 - 12:00	U180	1	Introduction to the course
37	09-09-2024	08:15 - 12:00	U167	2	Basic programming concepts using Python (PA 1)
38	16-09-2024	08:15 - 12:00	U167	3	Basic IO programming using MicroPython (Portfolio 1)
39	KiCad course (Semesterprojekt i grundlæggende styring af robotter)				
40	30-09-2024	08:15 - 12:00	U167	4	Actuator interface and Object-Oriented Programming (PA 2)
41	07-10-2024	08:15 - 12:00	U167	5	Drive System designs and Multitasking (Portfolio 2)
42	**Autumn vacation**				
43	21-10-2024	08:15 - 12:00	U167	6	Sensors and Robot Behaviors (Portfolio 3)
44	28-10-2024	08:15 - 12:00	U167	7	Logic Gates
45	04-11-2024	08:15 - 12:00	U167	8	[TBC] Debugging, data collection, and visualization (PA 3)
46	11-11-2024	08:00 - 12:00	U167	9	[TBC] Communication (Wifi/UART/I2C/SPI) and Mini project introduction (Portfolio 4)
47	18-11-2024	08:00 - 12:00	U167	10	Mini project work (and Q/A)
48	25-11-2024	08:00 - 12:00	U167	11	Mini project work (and Q/A)
50	02-12-2024	08:00 - 12:00	U181	12	Mini project presentation (PA 4)

Course plan

This plan is tentative, please expect some content changes

Week	Date	Schedule	Location	Module	Content
36	04-09-2024	08:15 - 12:00	U180	1	Introduction to the course
37	09-09-2024	08:15 - 12:00	U167	2	Basic programming concepts using Python (PA 1)
					asic IO programming using MicroPython (Portfolio 1)
					ggende styring af robotter)
					ctuator interface and Object-Oriented Programming (PA 2)
					rive System designs and Multitasking (Portfolio 2)
					on
					ensors and Robot Behaviors (Portfolio 3)
					ogic Gates
					BC] Debugging, data collection, and visualization (PA 3)
					BC] Communication (Wifi/UART/I2C/SPI) and Mini project troduction (Portfolio 4)
47	18-11-2024	08:00 - 12:00	U167	10	Mini project work (and Q/A)
48	25-11-2024	08:00 - 12:00	U167	11	Mini project work (and Q/A)
50	02-12-2024	08:00 - 12:00	U181	12	Mini project presentation (PA 4)

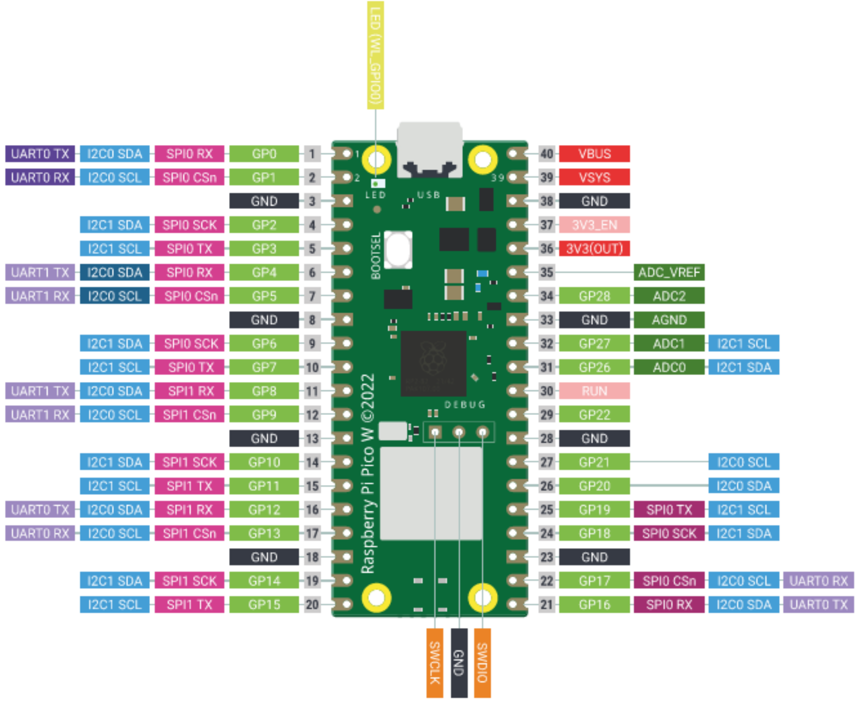
Communication
Either in-class or ItsLearning

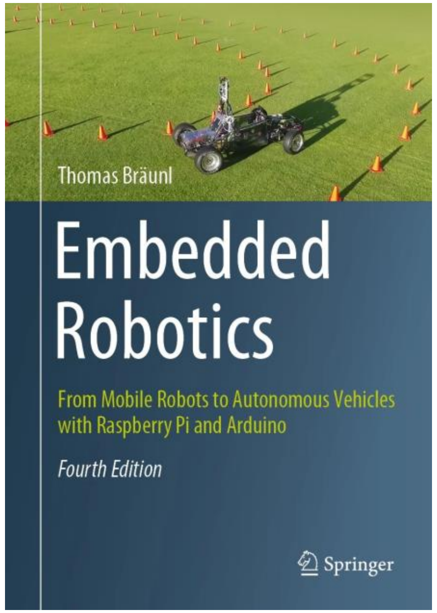
First time running the course…
…so things might need a bit of polishing (or even a byte…)

Responsible for your own learning
…but I will do my best to support and facilitate your progress.

References

- **MicroPython**
  - Documentation (library)
  - Get started with MicroPython on Raspberry Pi Pico
  - Cheatsheets
- **RP2040**
  - Datasheet
  - Assembly Language Programming
- **Raspberry Pi Pico / Pico W**
  - Datasheet
  - Python SDK
  - Documentation (library)
- **eBooks**
  - From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino
  - The Architecture of Computer Hardware and Systems Software


LED (WL_GPIO0)
UART0 TX | I2C0 SDA | SPI0 RX | GP0 | 1
UART0 RX | I2C0 SCL | SPI0 CSn | GP1 | 2
GND | 3
I2C1 SDA | SPI0 SCK | GP2 | 4
I2C1 SCL | SPI0 TX | GP3 | 5
UART1 TX | I2C0 SDA | SPI0 RX | GP4 | 6
UART1 RX | I2C0 SCL | SPI0 CSn | GP5 | 7
GND | 8
I2C1 SDA | SPI0 SCK | GP6 | 9
I2C1 SCL | SPI0 TX | GP7 | 10
UART1 TX | I2C0 SDA | SPI1 RX | GP8 | 11
UART1 RX | I2C0 SCL | SPI1 CSn | GP9 | 12
GND | 13
I2C1 SDA | SPI1 SCK | GP10 | 14
I2C1 SCL | SPI1 TX | GP11 | 15
UART0 TX | I2C0 SDA | SPI1 RX | GP12 | 16
UART0 RX | I2C0 SCL | SPI1 CSn | GP13 | 17
GND | 18
I2C1 SDA | SPI1 SCK | GP14 | 19
I2C1 SCL | SPI1 TX | GP15 | 20
40 | VBUS
39 | VSYS
38 | GND
37 | 3V3_EN
36 | 3V3(OUT)
35 | ADC_VREF
34 | GP28 | ADC2
33 | GND | AGND
32 | GP27 | ADC1 | I2C1 SCL
31 | GP26 | ADC0 | I2C1 SDA
30 | RUN
29 | GP22
28 | GND
27 | GP21 | I2C0 SCL
26 | GP20 | I2C0 SDA
25 | GP19 | SPI0 TX | I2C1 SCL
24 | GP18 | SPI0 SCK | I2C1 SDA
23 | GND
22 | GP17 | SPI0 CSn | I2C0 SCL | UART0 RX
21 | GP16 | SPI0 RX | I2C0 SDA | UART0 TX
LED
USB
BOOTSEL
DEBUG
Raspberry Pi Pico W ©2022
SWCLK
GND
SWDIO

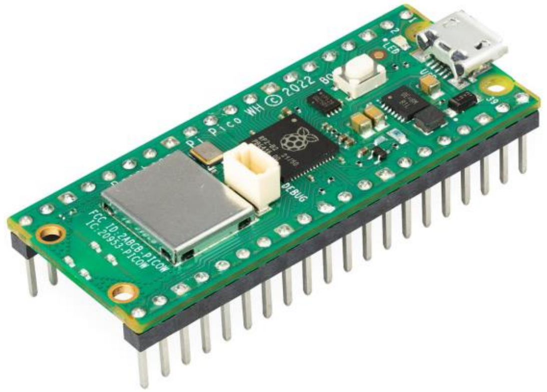

Thomas Bräunl
Embedded Robotics
From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino
Fourth Edition
Springer

(Documents are available through ItsLearning,
more might be added during the course)

Thonny
Python IDE for beginners

Software

- **Thonny** (Open-source Python IDE)
  - Windows / Linux / Mac
- **Alternatives:** VS Code?

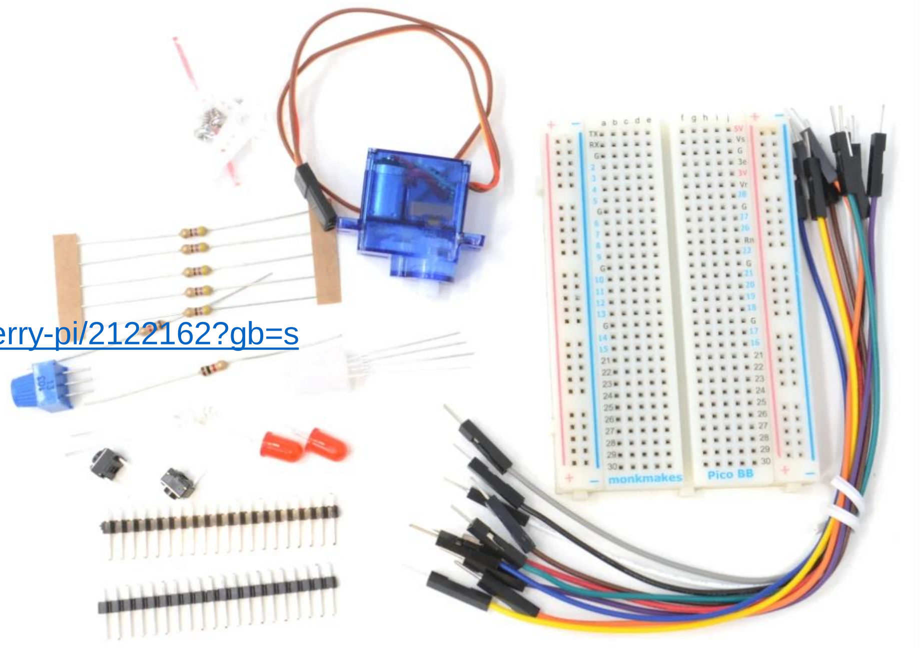


Hardware

- **Raspberry Pi Pico / Pico W**
  - (Recommended) Raspberry Pi Pico WH: https://thepihut.com/products/raspberry-pi-pico-w
  - Raspberry Pi Pico H: https://thepihut.com/products/raspberry-pi-pico

- **Monk Makes Electronics Kit 1 for Pico (lite edition)**
  - https://thepihut.com/products/electronics-kit-for-pico-lite-edition

- **Alternatives**
  - Pico / Pico W
    - https://raspberrypi.dk/en/product/raspberry-pi-pico-wh/
    - https://raspberrypi.dk/en/product/raspberry-pi-pico-h/
    - (not H) https://dk.rs-online.com/web/p/raspberry-pi/2122162?gb=s
  - Monk Makes
    - https://dk.rs-online.com/web/p/raspberry-pi-hats-og-add-ons/2222161



Your experience?
Your expectations?

Number systems

In-Class Q/A:
“What kinds do you know?”
“*Why do we need them?”*

Numbering systems

- Human use **base 10**
  - …decimal system
  - It is said to be of base 10 because it uses 10 digits
    - (0, 1, 2, … , 9)

$$7392_{\mathbf{10}} = 7 \times \mathbf{10}^3 + 3 \times \mathbf{10}^2 + 9 \times \mathbf{10}^1 + 2 \times \mathbf{10}^0$$

Base

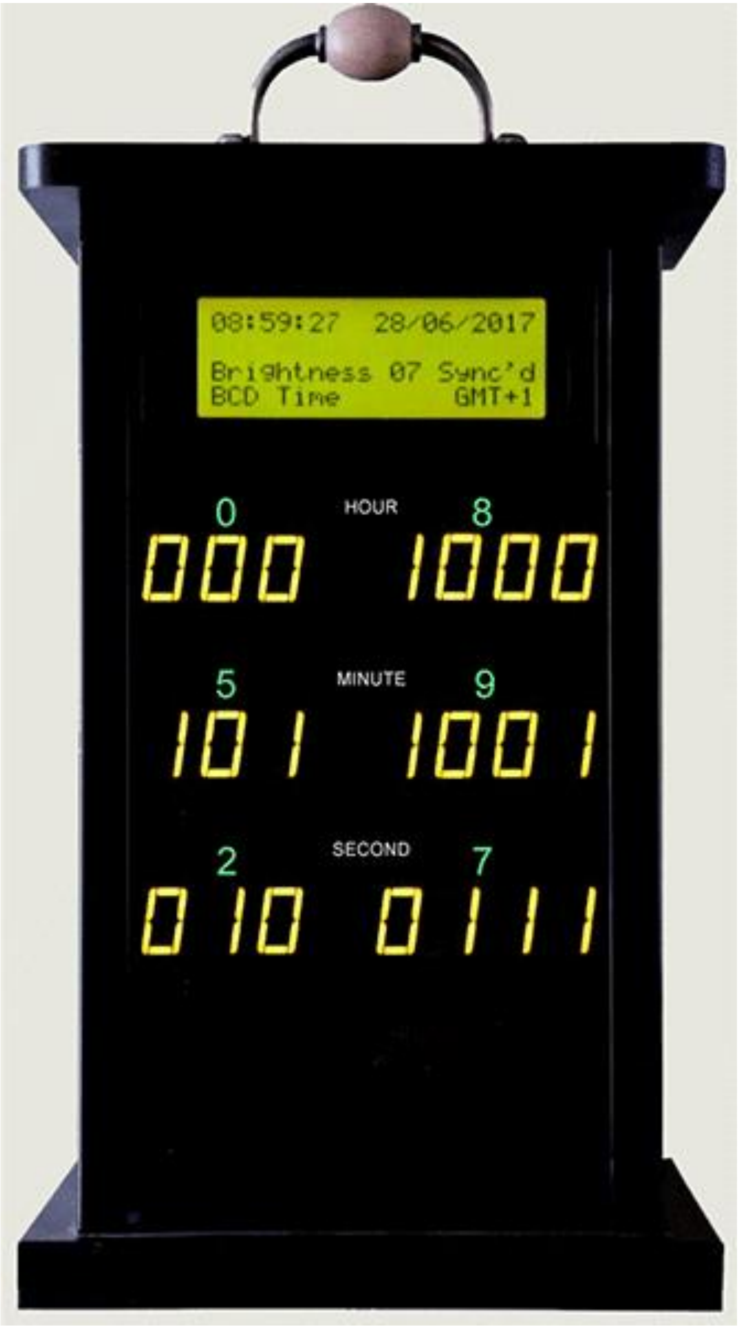
(7 thousands, plus 3 hundreds, plus 9 tens, plus 2 units)

- Machines use **Base 2**
  - …Binary system: it uses 0 or 1 (a bit)

$$7392_{\mathbf{10}} = 1110011100000_{\mathbf{2}}$$

Base

But why use binary numbers then?



Communication

“The process of exchanging information, ideas, thoughts, or messages between individuals, groups, or systems through a shared medium”

*= requires a **sender**, a message, and a **receiver** using the same <u>common understanding</u>*

Humans?

- Language, writing/symbols, appearance/body language, etc.

Machines?

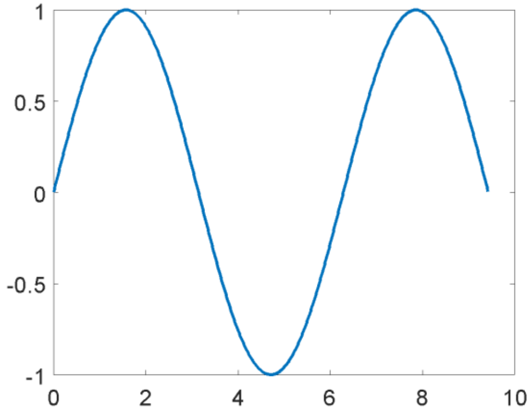
- Protocols, writing/symbols, data, etc.
  - → **signals**

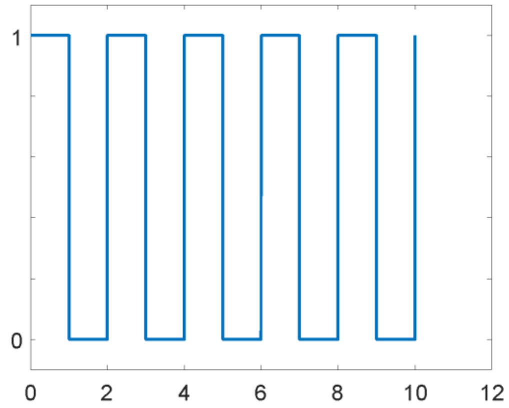
Signals

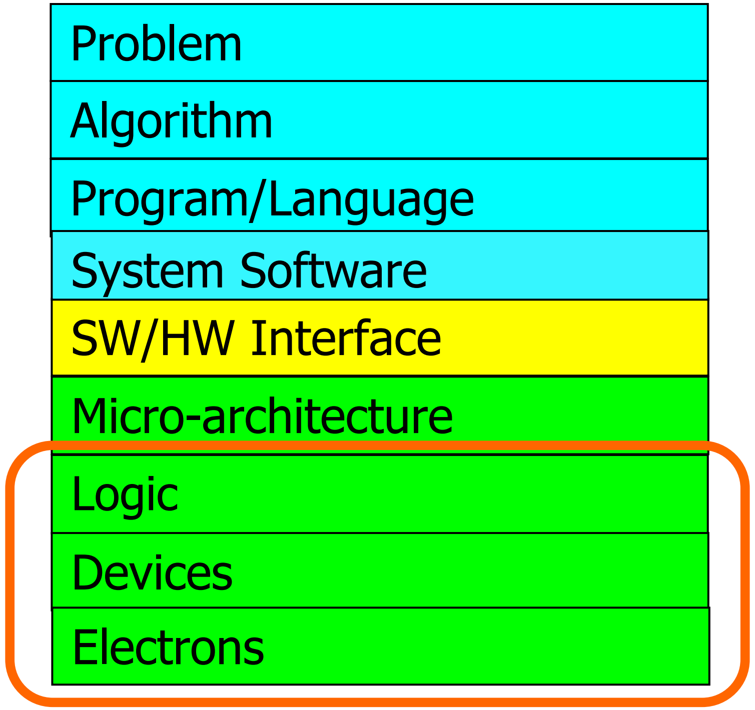
“An electrical or electromagnetic current that is used for carrying data from one device or network to another”

- There are two main types of signal:

Analog	Digital
Analog signal is continuous in time and amplitude.	Digital signals are discrete in time and amplitude.
A continuous range of values with an infinite number of possible values.	Binary values (**high (1)** and **low (0)**) to represent information.
Examples: Analog sensors (temperature, light, etc.), radio signals,	**Examples:** Computer data, CDs, DVDs







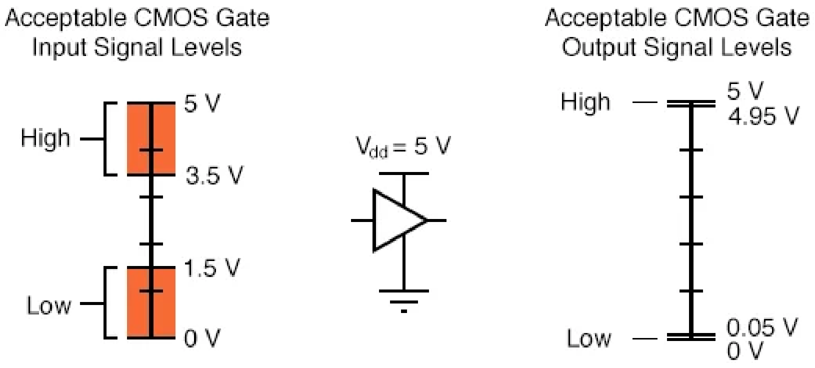
In principle analog could be fine… **but it is not…**

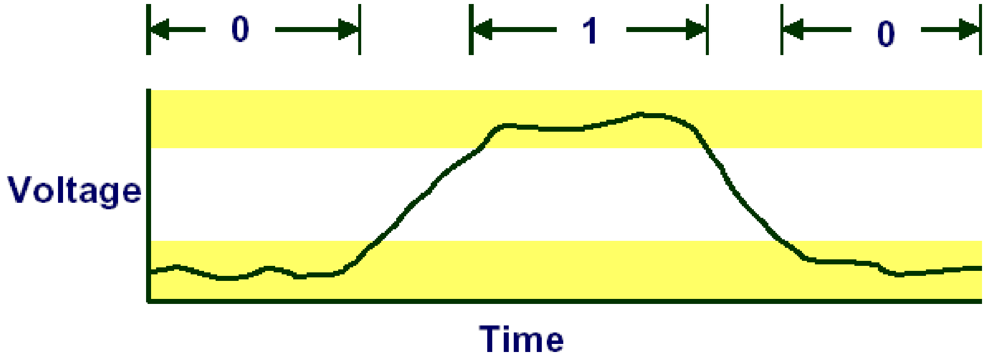
Logic Signal Voltage Levels

- Logic gate circuits are designed to input and output only two types of signals: **High (1) / Low (0)**

- **Example:** “Acceptable” signal voltages range
  - From 0 volts to < 0.8 - 1.5 volts for a “low” logic state
  - From > 2 - 3.5 volts to 5 volts for a “high” logic state

- Ranging between low and high is considered as uncertain


Acceptable CMOS Gate Input Signal Levels
High
5 V
3.5 V
1.5 V
Low
0 V
$V_{dd} = 5$ V
Acceptable CMOS Gate Output Signal Levels
High
5 V
4.95 V
Low
0.05 V
0 V

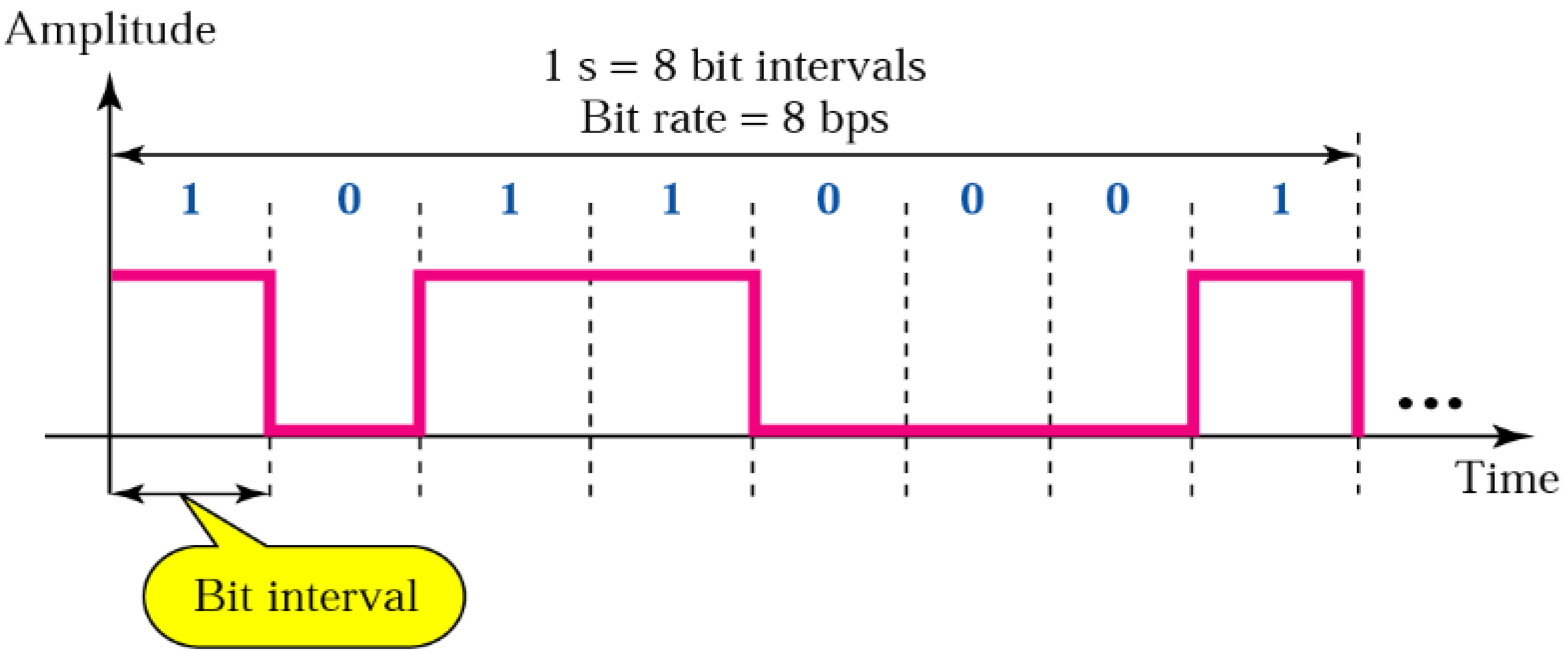


Images: https://www.allaboutcircuits.com/textbook/digital/chpt-3/logic-signal-voltage-levels/
https://people.engr.tamu.edu/djimenez/classes/312/lecture2.html

Data

“A sequence of bits, either 0 or 1, bits arranged in complex or specific patterns”

Hard for the human to read…



= ?

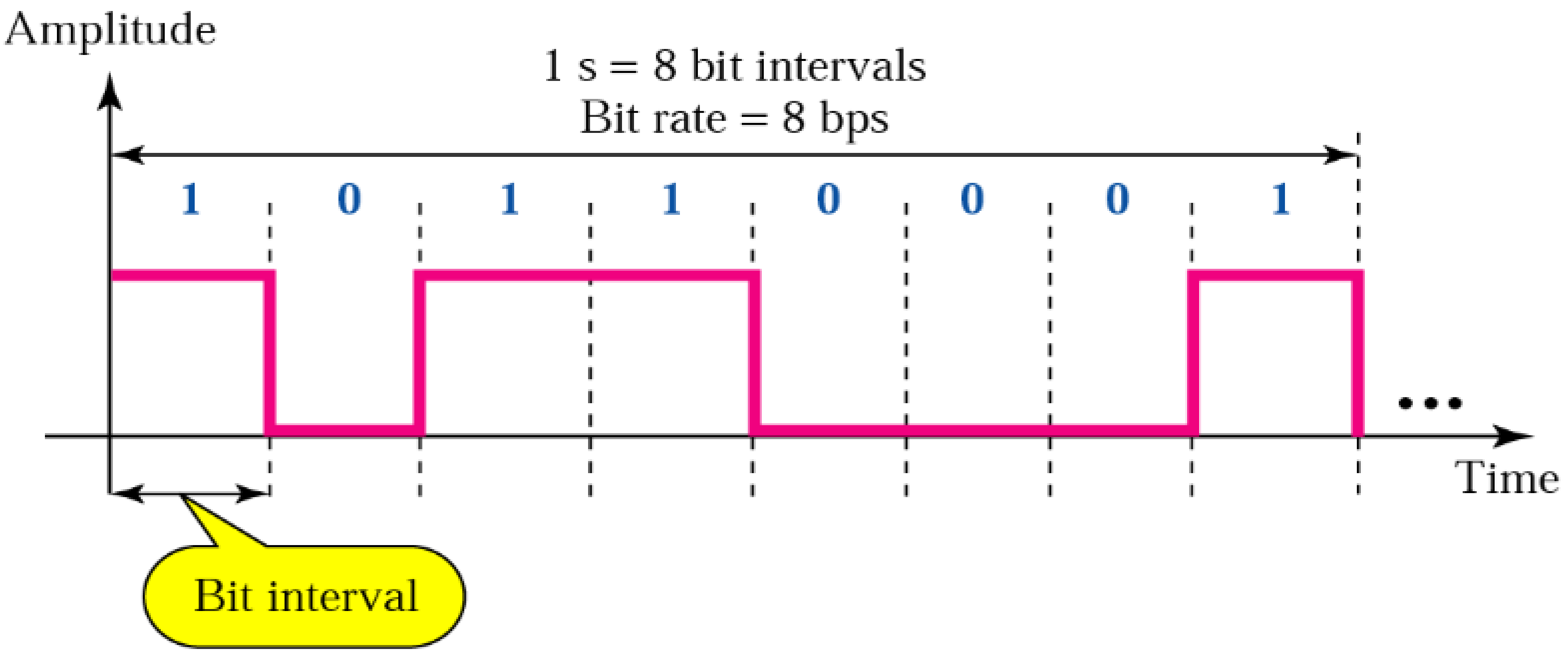
* bits per second(bps)

Images: https://bmsce.ac.in/Content/CS/unit1_CN_LJJ.pdf

Data

“A sequence of bits, either 0 or 1, bits arranged in complex or specific patterns”

Hard for the human to read…
We need to convert it to understand it (faster)!



= 177

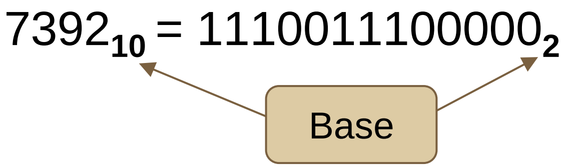
(when converted…)

* bits per second(bps)

Images: https://bmsce.ac.in/Content/CS/unit1_CN_LJJ.pdf

Convert <u>to</u> decimal from binary

- Other systems with different bases
  - **Base 2** (Binary): 0 or 1 (a bit)



$$7392_{\mathbf{10}} = 1110011100000_{\mathbf{2}}$$

Convert <u>to</u> decimal from binary

- Other systems with different bases
  - **Base 2** (Binary): 0 or 1 (a bit)

7392_{10} = 0001 1100 1110 **0000**$_2$

Convert using a table?

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Convert <u>to</u> decimal from binary

- Other systems with different bases
  - Base 2 (Binary): 0 or 1 (a bit)

MSB $7392_{10} = 1110011100000_2$ LSB

Convert using a table?

Or

$$= \mathbf{1 \times 2^{12} + 1 \times 2^{11} + 1 \times 2^{10} + 0 \times 2^{9} + 0 \times 2^{8} + 1 \times 2^{7} + 1 \times 2^{6} + 1 \times 2^{5} + 0 \times 2^{4} + 0 \times 2^{3} + 0 \times 2^{2} + 0 \times 2^{1} + 0 \times 2^{0}}$$

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Convert <u>to</u> decimal from binary/Octo/Hexa

- Other systems with different bases
  - Base 2 (Binary): 0 or 1 (a bit)

$$7392_{10} = 1110011100000_2$$

$$\mathbf{= 1 \times 2^{12} + 1 \times 2^{11} + 1 \times 2^{10} + 0 \times 2^9 + 0 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5}$$
$$\mathbf{+ 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0}$$

  - Base 8 (Octodecimal): 0,1,2,...,7

$$7392_{10} = 16340_8$$

$$\mathbf{= 1 \times 8^4 + 6 \times 8^3 + 3 \times 8^2 + 4 \times 8^1 + 0 \times 8^0}$$

  - Base 16 (Hexadecimal): 0, 1, 2, ..., 9, A, B, C, D, E, F

$$7392_{10} = 1CE0_{16}$$

$$\mathbf{= 1 \times 16^3 + 12 \times 16^2 + 14 \times 16^1 + 0 \times 16^0}$$

(Compared to binary numbers, they are more “easy” to read)

Decimal	Binary	Octal	Hexa
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Convert <u>to</u> decimal from binary/Octo/Hexa

- Other systems with different bases
  - Base 2 (Binary): 0 or 1 (a bit)

$$7392_{10} = 1110011100000_2$$

$$= 1 \times 2^{12} + 1 \times 2^{11} + 1 \times 2^{10} + 0 \times 2^{9} + 0 \times 2^{8} + 1 \times 2^{7} + 1 \times 2^{6} + 1 \times 2^{5}$$

$$+ 0 \times 2^{4} + 0 \times 2^{3} + 0 \times 2^{2} + 0 \times 2^{1} + 0 \times 2^{0}$$

  - Base 8 (Octodecimal): 0,1,2,...,7

$$7392_{10} = 16340_8$$

$$= 1 \times 8^{4} + 6 \times 8^{3} + 3 \times 8^{2} + 4 \times 8^{1} + 0 \times 8^{0}$$

  - **Base 16 (Hexadecimal)**: 0, 1, 2, ..., 9, A, B, C, D, E, F

$$7392_{10} = 1\mathbf{C}E0_{16}$$

$$= 1 \times 16^{3} + \mathbf{12} \times 16^{2} + 14 \times 16^{1} + 0 \times 16^{0}$$

(Compared to binary numbers, they are more “easy” to read)

Decimal	Binary	Octal	Hexa
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	**1100**	**14**	**C**
13	1101	15	D
14	1110	16	E
15	1111	17	F

Convert to decimal from binary/Octo/Hexa

- Other systems with different bases
  - Base 2 (Binary): 0 or 1 (a bit)

$$7392_{10} = 1110011100000_2$$

$$= 1 \times 2^{12} + 1 \times 2^{11} + 1 \times 2^{10} + 0 \times 2^{9} + 0 \times 2^{8} + 1 \times 2^{7} + 1 \times 2^{6} + 1 \times 2^{5} + 0 \times 2^{4} + 0 \times 2^{3} + 0 \times 2^{2} + 0 \times 2^{1} + 0 \times 2^{0}$$

(Compared to binary numbers, they are more "easy" to read)

- 1 byte = 8 bits = 2 hexadecimals

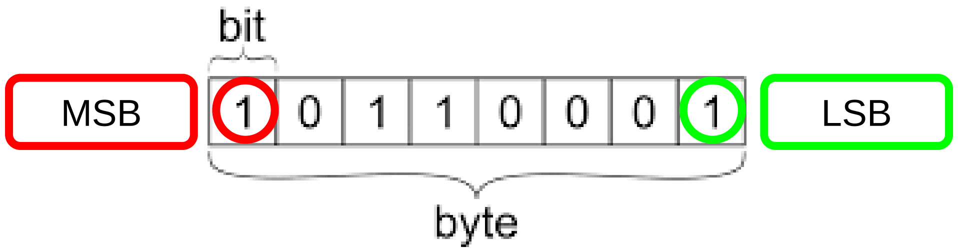


Illustration: https://learnworthy.net/the-amazing-history-of-the-data-byte/

Decimal	Binary	Octal	Hexa
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Convert from decimal to binary

- Divide by the base to get the digits from the remainders
  - Base 2 (Binary): $7392_{10} = 1110011100000_2$

Division by 2	Remainder	Remainder (Digit)	Bit #
(7392)/2	3696	0 **(LSB)**	0

Convert from decimal to binary

- Divide by the base to get the digits from the remainders
  - Base 2 (Binary): $7392_{10} = 1110011100000_2$

Division by 2	Remainder	Remainder (Digit)	Bit #
(7392)/2	3696	0 **(LSB)**	0
(3696)/2	1848	0	1

Convert <u>from</u> decimal to binary

- Divide by the base to get the digits from the remainders
    - Base 2 (Binary): $7392_{10} = 1110011100000_2$

Division by 2	Remainder	Remainder (Digit)	Bit #
(7392)/2	3696	0 **(LSB)**	0
(3696)/2	1848	0	1
(1848)/2	924	0	2

Convert <u>from</u> decimal to binary

- Divide by the base to get the digits from the remainders
  - Base 2 (Binary): $7392_{10} = 1110011100000_2$

Division by 2	**Remainder**	**Remainder (Digit)**	**Bit #**
(7392)/2	3696	0 **(LSB)**	0
(3696)/2	1848	0	1
(1848)/2	924	0	2
(924)/2	462	0	3

Convert from decimal to binary

- Divide by the base to get the digits from the remainders
  - Base 2 (Binary): $7392_{10} = 1110011100000_2$

Division by 2	Remainder	Remainder (Digit)	Bit #
(7392)/2	3696	0 **(LSB)**	0
(3696)/2	1848	0	1
(1848)/2	924	0	2
(924)/2	462	0	3
(462)/2	231	0	4

Convert <u>from</u> decimal to binary

- Divide by the base to get the digits from the remainders
  - Base 2 (Binary): $7392_{10} = 1110011100000_2$

Division by 2	Remainder	Remainder (Digit)	Bit #
(7392)/2	3696	0 **(LSB)**	0
(3696)/2	1848	0	1
(1848)/2	924	0	2
(924)/2	462	0	3
(462)/2	231	0	4
(231)/2	115	(0.5 * 2) 1	5

Convert <u>from</u> decimal to binary

- Divide by the base to get the digits from the remainders
  - Base 2 (Binary): $7392_{10} = 1110011100000_2$

Division by 2	Remainder	Remainder (Digit)	Bit #
(7392)/2	3696	0 **(LSB)**	0
(3696)/2	1848	0	1
(1848)/2	924	0	2
(924)/2	462	0	3
(462)/2	231	0	4
(231)/2	115	(0.5 * 2) 1	5
(115)/2	57	(0.5 * 2) 1	6

Convert <u>from</u> decimal to binary

- Divide by the base to get the digits from the remainders
    - Base 2 (Binary): $7392_{10} = 1110011100000_2$

Division by 2	**Remainder**	**Remainder (Digit)**	**Bit #**
(7392)/2	3696	0 **(LSB)**	0
(3696)/2	1848	0	1
(1848)/2	924	0	2
(924)/2	462	0	3
(462)/2	231	0	4
(231)/2	115	(0.5 * 2) 1	5
(115)/2	57	(0.5 * 2) 1	6
(57)/2	28	(0.5 * 2) 1	7

Convert <u>from</u> decimal to binary

- Divide by the base to get the digits from the remainders
  - Base 2 (Binary): $7392_{10} = 1110011100000_2$

Division by 2	Remainder	Remainder (Digit)	Bit #
(7392)/2	3696	0 **(LSB)**	0
(3696)/2	1848	0	1
(1848)/2	924	0	2
(924)/2	462	0	3
(462)/2	231	0	4
(231)/2	115	(0.5 * 2) 1	5
(115)/2	57	(0.5 * 2) 1	6
(57)/2	28	(0.5 * 2) 1	7
(28)/2	14	0	8

Convert <u>from</u> decimal to binary

- Divide by the base to get the digits from the remainders
  - Base 2 (Binary): $7392_{10} = 1110011100000_2$

Division by 2	Remainder	Remainder (Digit)	Bit #
(7392)/2	3696	0 **(LSB)**	0
(3696)/2	1848	0	1
(1848)/2	924	0	2
(924)/2	462	0	3
(462)/2	231	0	4
(231)/2	115	(0.5 * 2) 1	5
(115)/2	57	(0.5 * 2) 1	6
(57)/2	28	(0.5 * 2) 1	7
(28)/2	14	0	8
(14)/2	7	0	9

Convert <u>from</u> decimal to binary

- Divide by the base to get the digits from the remainders
    - Base 2 (Binary): $7392_{10} = 1110011100000_2$

Division by 2	Remainder	Remainder (Digit)	Bit #
(7392)/2	3696	0 **(LSB)**	0
(3696)/2	1848	0	1
(1848)/2	924	0	2
(924)/2	462	0	3
(462)/2	231	0	4
(231)/2	115	(0.5 * 2) 1	5
(115)/2	57	(0.5 * 2) 1	6
(57)/2	28	(0.5 * 2) 1	7
(28)/2	14	0	8
(14)/2	7	0	9
(7)/2	3	(0.5 * 2) 1	10

Convert <u>from</u> decimal to binary

- Divide by the base to get the digits from the remainders
    - Base 2 (Binary): $7392_{10} = 1110011100000_2$

Division by 2	Remainder	Remainder (Digit)	Bit #
(7392)/2	3696	0 **(LSB)**	0
(3696)/2	1848	0	1
(1848)/2	924	0	2
(924)/2	462	0	3
(462)/2	231	0	4
(231)/2	115	(0.5 * 2) 1	5
(115)/2	57	(0.5 * 2) 1	6
(57)/2	28	(0.5 * 2) 1	7
(28)/2	14	0	8
(14)/2	7	0	9
(7)/2	3	(0.5 * 2) 1	10
(3)/2	1	(0.5 * 2) 1	11

Convert from decimal to binary

- Divide by the base to get the digits from the remainders
  - Base 2 (Binary): 7392_{10} = 1110011100000_2

Division by 2	Remainder	Remainder (Digit)	Bit #
(7392)/2	3696	0 **(LSB)**	0
(3696)/2	1848	0	1
(1848)/2	924	0	2
(924)/2	462	0	3
(462)/2	231	0	4
(231)/2	115	(0.5 * 2) 1	5
(115)/2	57	(0.5 * 2) 1	6
(57)/2	28	(0.5 * 2) 1	7
(28)/2	14	0	8
(14)/2	7	0	9
(7)/2	3	(0.5 * 2) 1	10
(3)/2	1	(0.5 * 2) 1	11
(1)/2	0	(0.5 * 2) 1 **(MSB)**	12

Convert from decimal to hexadecimal

- Divide by the base to get the digits from the remainders
  - Base 16 (Hexadecimal): $7392_{10} = ?_{16}$

Division by 16	Remainder	Remainder (Digit)	Digit #
(7392)/16	462	0 (“LSB”)	0

Decimal	Binary	Octal	Hexa
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Convert from decimal to hexadecimal

- Divide by the base to get the digits from the remainders
  - Base 16 (Hexadecimal): $7392_{10} = ?_{16}$

Division by 16	Remainder	Remainder (Digit)	Digit #
(7392)/16	462	0 (“LSB”)	0
(462)/16	28	(0.875 * 16) 14 = E	1

Decimal	Binary	Octal	Hexa
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Convert from decimal to hexadecimal

- Divide by the base to get the digits from the remainders
  - Base 16 (Hexadecimal): $7392_{10} = ?_{16}$

Division by 16	Remainder	Remainder (Digit)	Digit #
(7392)/16	462	0 (“LSB”)	0
(462)/16	28	(0.875 * 16) 14 = E	1
(28)/16	1	(0.75 * 16) 12 = C	2

Decimal	Binary	Octal	Hexa
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Convert <u>from</u> decimal to hexadecimal

- Divide by the base to get the digits from the remainders
  - Base 16 (Hexadecimal): $7392_{10} = ?_{16}$

Division by 16	Remainder	Remainder (Digit)	Digit #
(7392)/16	462	**0** (“LSB”)	0
(462)/16	28	(0.875 * 16) 14 = **E**	1
(28)/16	1	(0.75 * 16) 12 = **C**	2
(1)/16	0	(0.0625 * 16) **1** (“MSB”)	3
7392_{10} = $\mathbf{1CE0}_{16}$			

Decimal	Binary	Octal	Hexa
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Numbering systems (floating-point)

- Continue until the fractional part become zero, and continue with negative integers

Point

2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
64	32	16	8	4	2	1	0.5	0.25	0.125

  - Example:

$$\mathbf{101000.101_2} = 1*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = 40.625_{\mathbf{10}}$$

$\mathbf{40.625_{10}}$ =

40 / 2 = 20 with remainder 0 20 / 2 = 10 with remainder 0 10 / 2 = 5 with remainder 0 5 / 2 = 2 with remainder 1 2 / 2 = 1 with remainder 0 1 / 2 = 0 with remainder 1 10100	**0.625 × 2 = 1 + 0.25**

Numbering systems (floating-point)

- Continue until the fractional part become zero, and continue with negative integers

Point

2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
64	32	16	8	4	2	1	0.5	0.25	0.125

  - Example:

$\mathbf{101000.101_2} = 1*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = 40.625_{\mathbf{10}}$

$\mathbf{40.625_{10}} =$

40 / 2 = 20 with remainder 0 20 / 2 = 10 with remainder 0 10 / 2 = 5 with remainder 0 5 / 2 = 2 with remainder 1 2 / 2 = 1 with remainder 0 1 / 2 = 0 with remainder 1 10100	0.625 × 2 = 1 + 0.25 **0.25 × 2 = 0 + 0.5**

Numbering systems (floating-point)

- Continue until the fractional part become zero, and continue with negative integers

Point

2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
64	32	16	8	4	2	1	0.5	0.25	0.125

  - Example:

$$\mathbf{101000.101_2} = 1*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = 40.625_{\mathbf{10}}$$

40.625_{10} =

40 / 2 = 20 with remainder 0 20 / 2 = 10 with remainder 0 10 / 2 = 5 with remainder 0 5 / 2 = 2 with remainder 1 2 / 2 = 1 with remainder 0 1 / 2 = 0 with remainder 1 10100	0.625 × 2 = 1 + 0.25 0.25 × 2 = 0 + 0.5 **0.5 × 2 = 1 + 0** (stop)

Numbering systems (floating-point)

- Continue until the fractional part become zero, and continue with negative integers

Point

2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
64	32	16	8	4	2	1	0.5	0.25	0.125

  - Example:

$$\mathbf{101000.101_2} = 1*2^5+ 0*2^4+ 1*2^3+ 0*2^2+ 0*2^1+ 0*2^0+ 1*2^{-1}+ 0*2^{-2}+ 1*2^{-3} = 40.625_{\mathbf{10}}$$

$\mathbf{40.625_{10}}$ =

40 / 2 = 20 with remainder 0 20 / 2 = 10 with remainder 0 10 / 2 = 5 with remainder 0 5 / 2 = 2 with remainder 1 2 / 2 = 1 with remainder 0 1 / 2 = 0 with remainder 1 10100	0.625 × 2 = **1** + 0.25 0.25 × 2 = **0** + 0.5 0.5 × 2 = **1** + 0 (stop) (You can calculate upto more digits) **.101**

Task: Convert between numbering systems (20 min)

Convert between decimal, binary, and hexadecimal numbering systems. Write down the intermediate steps leading to the result.

Decimal and Binary	**Decimal and Hexadecimal**	**Binary and Hexadecimal**
1. Convert the decimal number 1. 25 to binary. 2. 255 to binary. 3. 9.5625 to binary. 4. 19.875 to binary. 2. Convert the binary number 1. 1000 0011 to decimal. 2. 100101 to decimal. 3. 100011.0011 to decimal. 4. 1.001011 to decimal.	1. Convert the decimal number 1. 123 to hexadecimal. 2. 500 to hexadecimal. 2. Convert the hexadecimal number 1. FF to decimal. 2. C8 to decimal.	1. Convert the binary number 1. 1001 1100 to hexadecimal. 2. 1111 0000 to hexadecimal. 2. Convert the hexadecimal number 1. 4C to binary. 2. 1E to binary.

Validate your answers online, eg. here:
https://www.rapidtables.com/convert/number/decimal-to-binary.html?x=0

Microcontroller

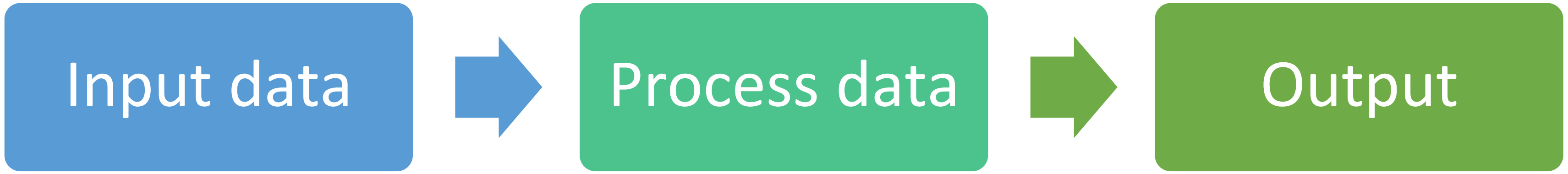
In-Class Q/A: *“What is a **Microcontroller**?*

What is a Microcontroller?

- A small Computer on a single integrated circuit (**Computer-on-a-chip** / “mini-computer”)

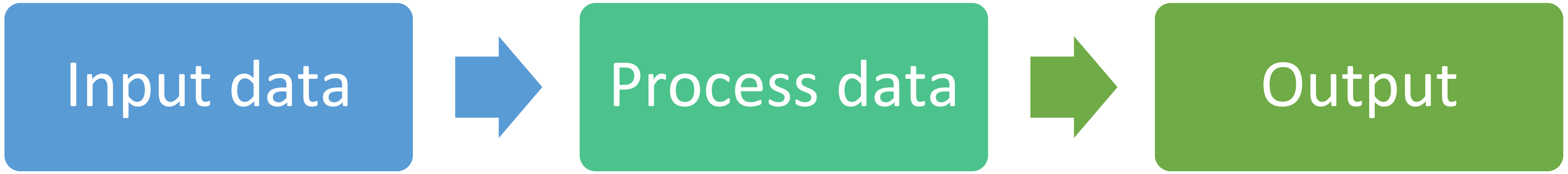
What is a Microcontroller?

- A small Computer on a single integrated circuit (**Computer-on-a-chip** / “mini-computer”)
- An integrated circuit that is **programmed to do a specific** (simple) task, typically:
  - Collect input from various sensors (Temperature, humidity, image, …)
  - Process collected data to make some decisions
  - Use the output mechanism on the Microcontroller to do something useful (Turning on/off actuators, LEDs, …)



What is a Microcontroller?

- A small Computer on a single integrated circuit (**Computer-on-a-chip** / “mini-computer”)
- An integrated circuit that is **programmed to do a specific** (simple) task, typically:
    - Collect input from various sensors (Temperature, humidity, image, …)
    - Process collected data to make some decisions
    - Use the output mechanism on the Microcontroller to do something useful (Turning on/off actuators, LEDs, …)



Input data → Process data → Output

- **Different Platforms, Different Goals**

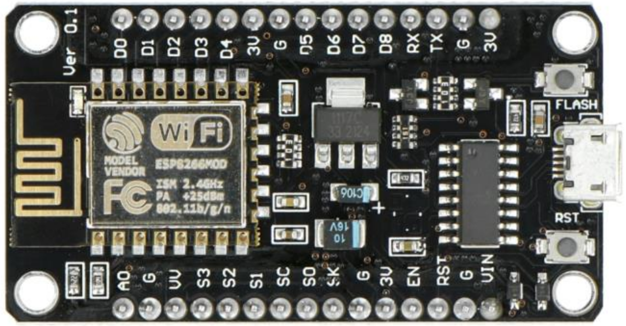
Types of Microcontroller?

- **RP2040** (Raspberry Pi Pico)
  - Dual-core ARM Cortex-M0+, up to 133 MHz, 264 KB SRAM, external flash.
- **ATmega328P** (Arduino Uno)
  - 8-bit AVR, up to 20 MHz, 2 KB SRAM, 32 KB flash.
- **ESP32**
  - Dual-core Xtensa LX6, up to 240 MHz, 520 KB SRAM, integrated Wi-Fi and Bluetooth.
- **STM32F103:**
  - ARM Cortex-M3, up to 72 MHz, 20 KB SRAM, 64-512 KB flash.
- **ATSAMD21** (Arduino Zero)
  - ARM Cortex-M0+, up to 48 MHz, 32 KB SRAM, 256 KB flash.
- **ESP8266** (NodeMCU)
  - 32-bit Tensilica L106, 80 MHz, 160 KB SRAM, integrated Wi-Fi, popular in IoT.

Specs sounds a lot like a laptop with a (micro)processor?

Raspberry Pi Pico

Arduino Uno



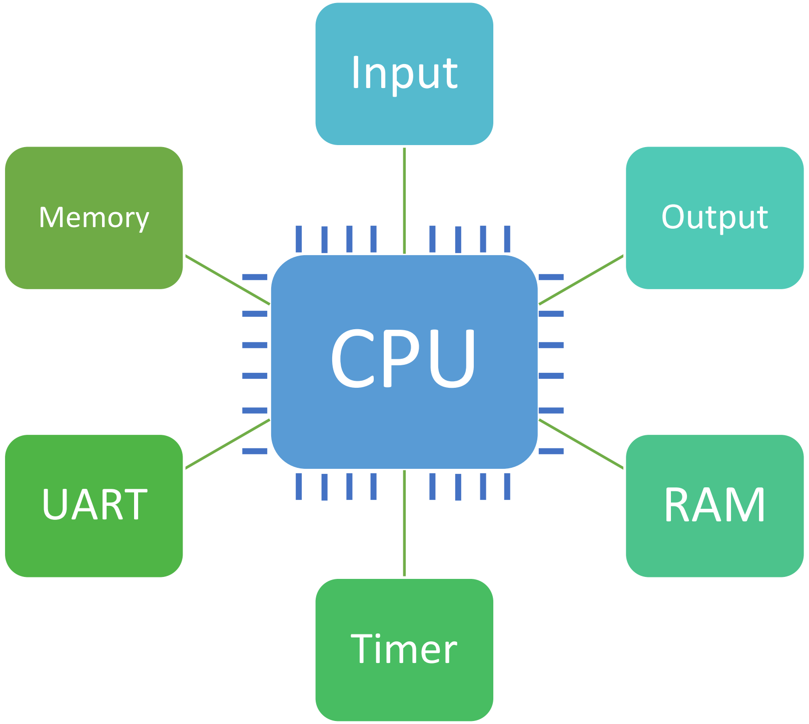
NodeMCU

Microcontroller versus Microprocessor?

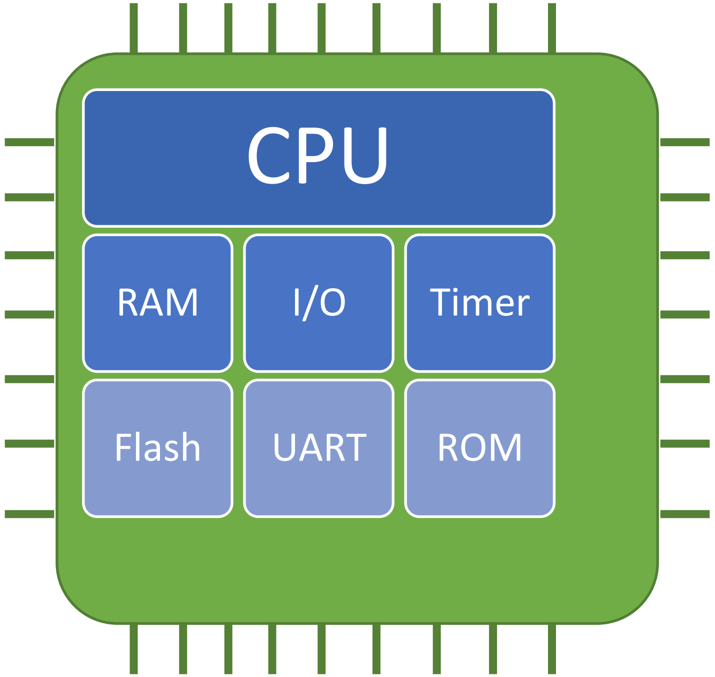
In-Class Q/A: *“What is the difference between a* ***Microcontroller*** *and a* ***Microprocessor****?"*

Microcontroller versus Microprocessor?

- Components / Peripherals?



Microprocessor
(General computing)



Microcontroller
(Specialized devices)

Illustration: Datateknik course, W1 (T540050101, by Ali Sahafi)

Microcontroller versus Microprocessor?

	Microprocessor **(General computing)**	**Microcontroller** **(Specialized devices, application depended)**
Memory	Requires external memory and data storage. (only processes data, not storing)	On-chip memory modules (ROM, RAM). (Stand-alone device)
Peripherals	Needs additional parts. Connect with the external bus.	On-chip peripherals (timers, I/O ports, signal converter).
Computing	Capable of complex computing tasks.	Limited to specific application logic.
Clock speed	Very fast. GHz range.	Fast but slower than microprocessors. kHz to MHz range.
Power	High power consumption. No power saving mode.	Consumes minimal power. Built-in power saving modes.
Communication	Handles high-speed data transfer. Supports USB 3.0 and Gigabit Ethernet.	Supports low to moderate speed communication. Serial Peripheral Interface (SPI) and I²C. Universal asynchronous receiver-transmitter (UART).
Cost	Expensive because of the additional components.	Cheaper because a single integrated circuit provides multiple functionalities.

Source: https://aws.amazon.com/compare/the-difference-between-microprocessors-microcontrollers/

Micro-architecture

In-Class Q/A: “*What is it?”*

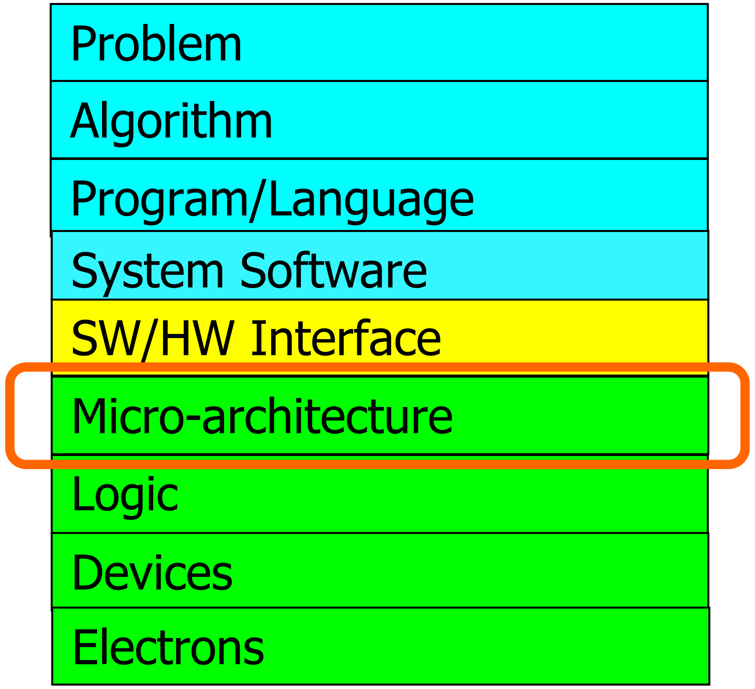
Microcontroller architecture (micro-architecture)

“The way to structure and design of a computer's Central Processing Unit (CPU) or other hardware components”

- **Key components** (structure and design)
  - Central Processing Unit (CPU)
  - Memory
  - Digital Converters (ADC/DAC)
  - Input/Output (I/O) Ports
  - Serial Bus Interface (data transfer each component)

- **Types:** RISC, CISC, ARM, AVR, PIC, Intel 8051, ……….
  - Computer architectures are a study by itself….

- **Two basic/traditional designs:**
  - Von Neumann architecture
  - Harvard architecture



Microcontroller architecture (microarchitecture)

- Harvard vs. Von Neumann Architecture
  - **Von Neumann architecture:** uses the same memory space for both.
    - Slower, CPU can not access instructions and read/write at the same time.
    - Cheaper and easier to program
  - **Harvard architecture:** separates memory for instructions and data.
    - Faster, CPU can access instructions and read/write at the same time.
    - More costly and complex, compared to Von Neumann

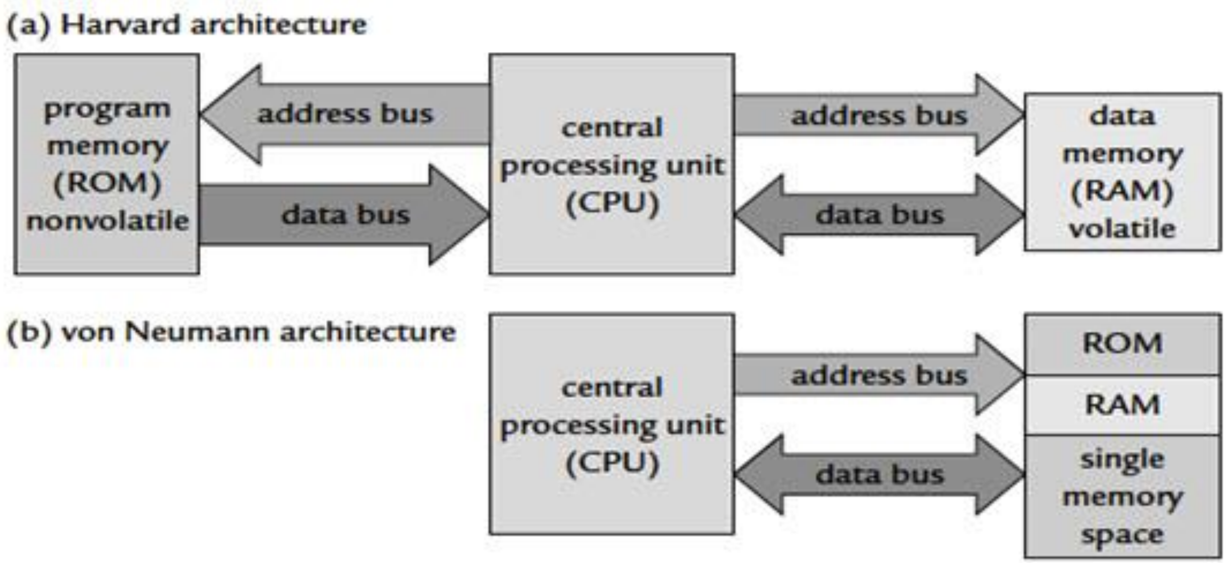

(a) Harvard architecture
program memory (ROM) nonvolatile
address bus
data bus
central processing unit (CPU)
address bus
data bus
data memory (RAM) volatile
(b) von Neumann architecture
central processing unit (CPU)
address bus
data bus
ROM
RAM
single memory space

Illustration: http://fabacademy.org/2018/labs/fablabcept/week-09---embedded-programming.html

Microcontroller architecture (Common components)

- **CPU (Central Processing Unit)**
  The brain of the microcontroller, responsible for executing instructions.
  - **Control Unit**
  - **Arithmetic Logic Unit (ALU)**
  - **Memory / Registers**

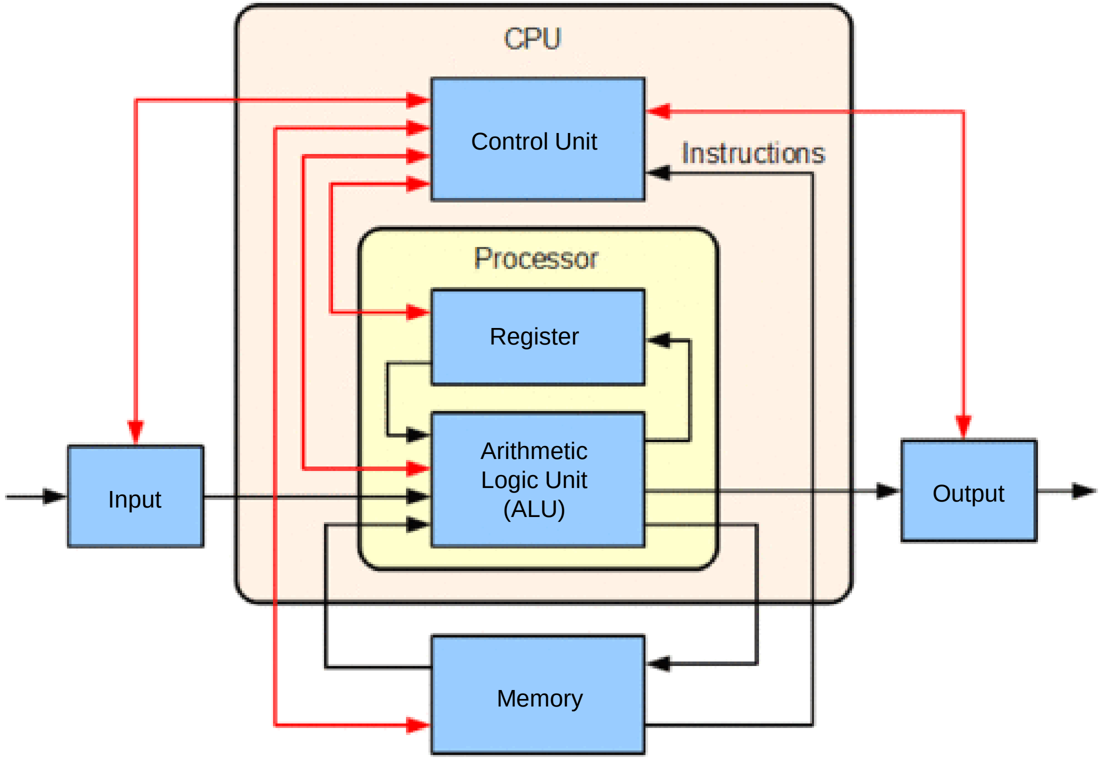


Illustration: https://www.partitionwizard.com/partitionmanager/cpu-architecture.html

Microcontroller architecture (Common components)

- **CPU (Central Processing Unit)**
  The brain of the microcontroller, responsible for executing instructions.
  - **Control Unit:** Handles the instruction cycle
    1. **Fetches** instructions from main memory,
    2. **Decodes** them, and then
    3. **Executes** them by coordinating the actions of the ALU, registers, and other components.
  - **Arithmetic Logic Unit (ALU)**
  - **Memory / Registers**

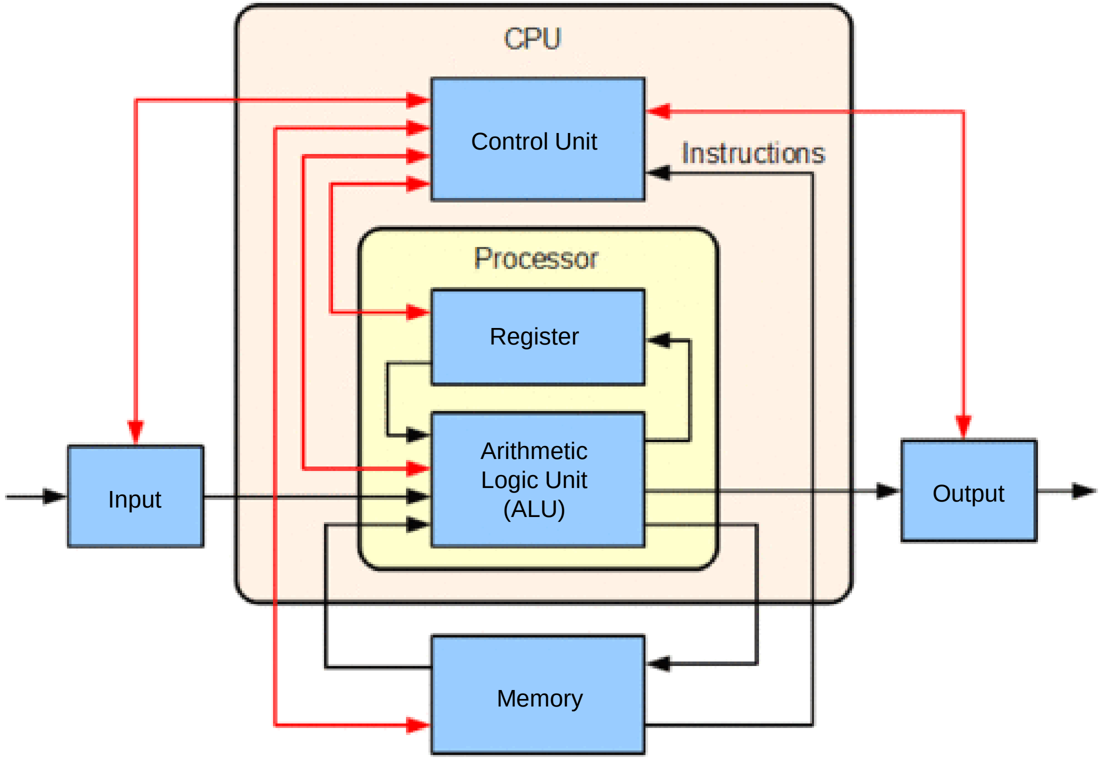


Illustration: https://www.partitionwizard.com/partitionmanager/cpu-architecture.html

Microcontroller architecture (Common components)

- **CPU (Central Processing Unit)**
  The brain of the microcontroller, responsible for executing instructions.
  - **Control Unit**
  - **Arithmetic Logic Unit (ALU):** Part of the CPU that performs arithmetic and logical operations
    - addition,
    - subtraction,
    - multiplication,
    - division,
    - logical operations (and, or, etc.),
    - increment and decrement,
    - ect.
  - **Memory / Registers**

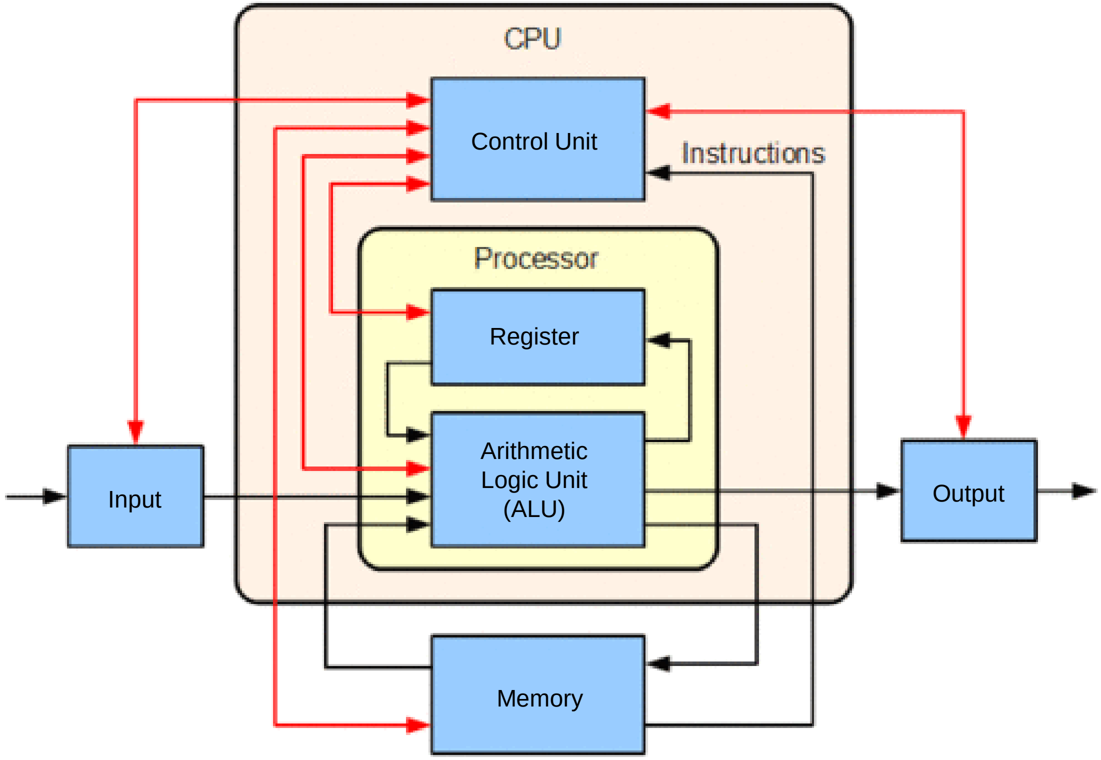
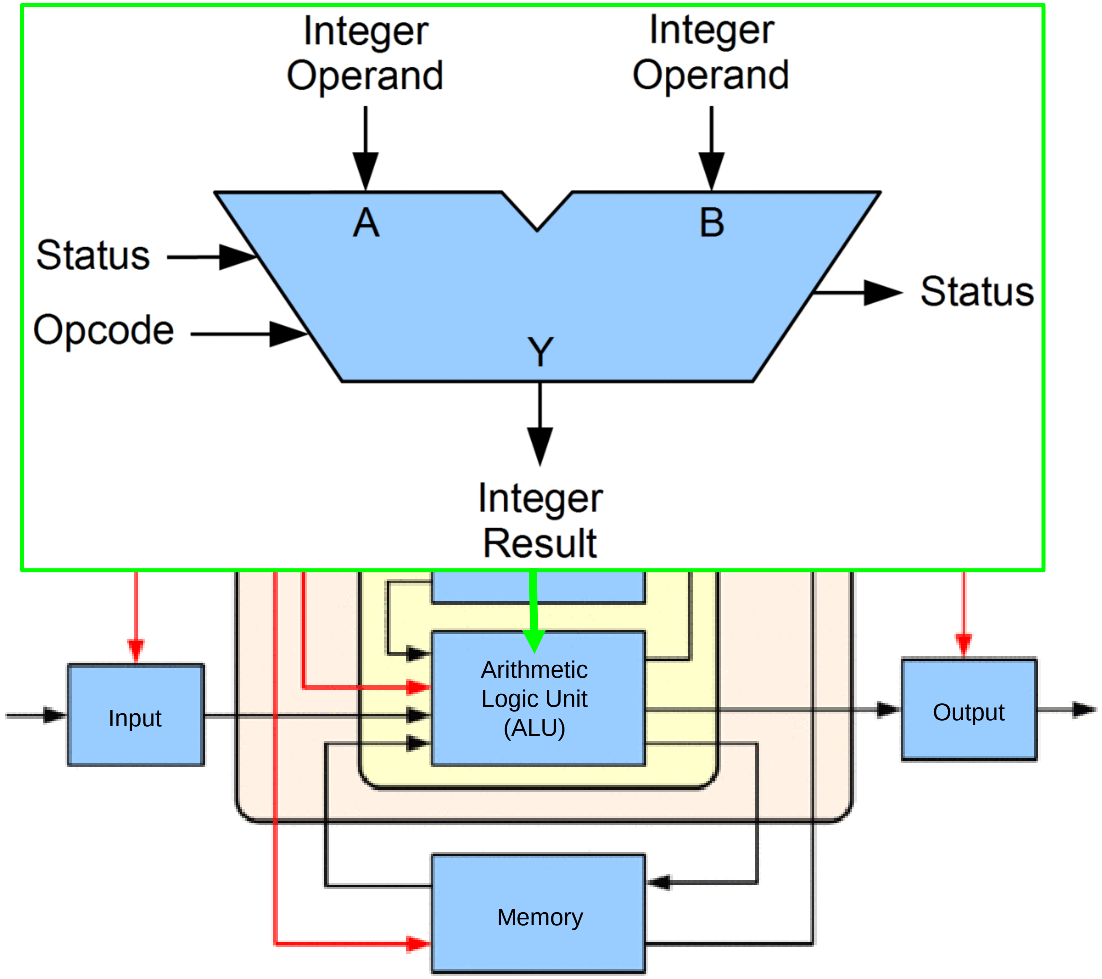


Illustration: https://www.partitionwizard.com/partitionmanager/cpu-architecture.html

Microcontroller architecture (Common components)

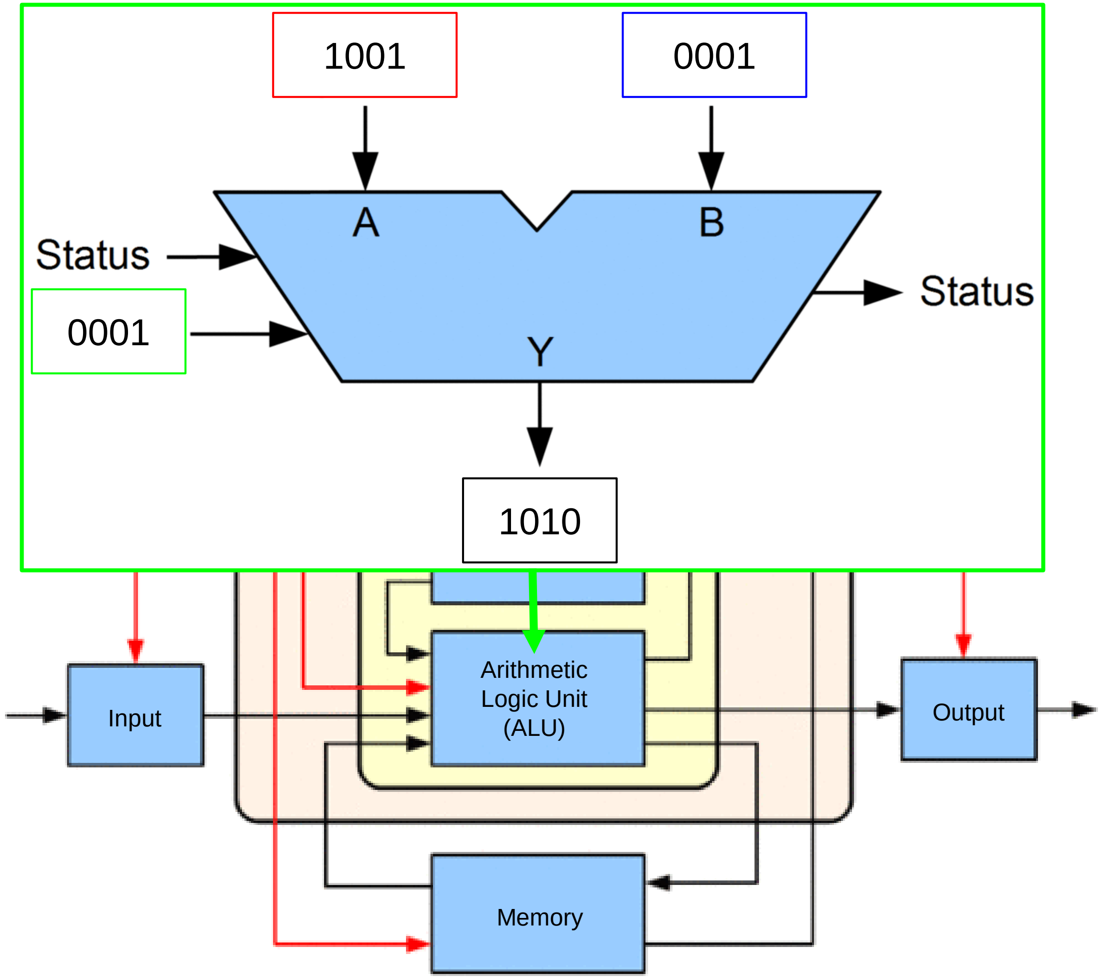
- **CPU (Central Processing Unit)**
  The brain of the microcontroller, responsible for executing instructions.
  - **Control Unit**
  - **Arithmetic Logic Unit (ALU):** Part of the CPU that performs arithmetic and logical operations
    - **Opcode:** This part of the instruction specifies the operation (addition,subtraction, Multiplication, division, ect.)
    - **Operands:** These are the values or addresses on which the operation specified by the opcode will be performed.
    - (**Status:** Additional information, eg. about a previous operation)
  - **Memory / Registers**



Illustrations: https://www.partitionwizard.com/partitionmanager/cpu-architecture.html
https://en.wikipedia.org/wiki/Arithmetic_logic_unit

Microcontroller architecture (Common components)

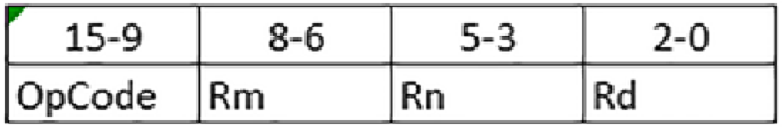
- **CPU (Central Processing Unit)**
  The brain of the microcontroller, responsible for executing instructions.
  - **Control Unit**
  - **Arithmetic Logic Unit (ALU):** Part of the CPU that performs arithmetic and logical operations
    - **Example:** ADD Instruction
      - Suppose the instruction 0001 1001 0001 is sent to the CPU.
    - The <u>first</u> 4 bits (0001) could be the opcode, representing the ADD operation.
    - The <u>next bits</u> (1001 0001) might represent the addresses of the operands or the operands themselves.
  - **Memory / Registers**



Illustrations: https://www.partitionwizard.com/partitionmanager/cpu-architecture.html
https://en.wikipedia.org/wiki/Arithmetic_logic_unit

Microcontroller architecture (Common components)

- **CPU (Central Processing Unit)**
  The brain of the microcontroller, responsible for executing instructions.
  - **ARM Cortex-M0+** binary instructions are 16 bits long.
    - (The processor on the Raspberry Pi Pico)



15-9	8-6	5-3	2-0
OpCode	Rm	Rn	Rd

    - **Opcode:** Which instruction are we performing (ADD, SUB, ect.)
    - **Rm and Rn:** The two registers to add
    - **Rd:** The destination register, where to put the result of the addition

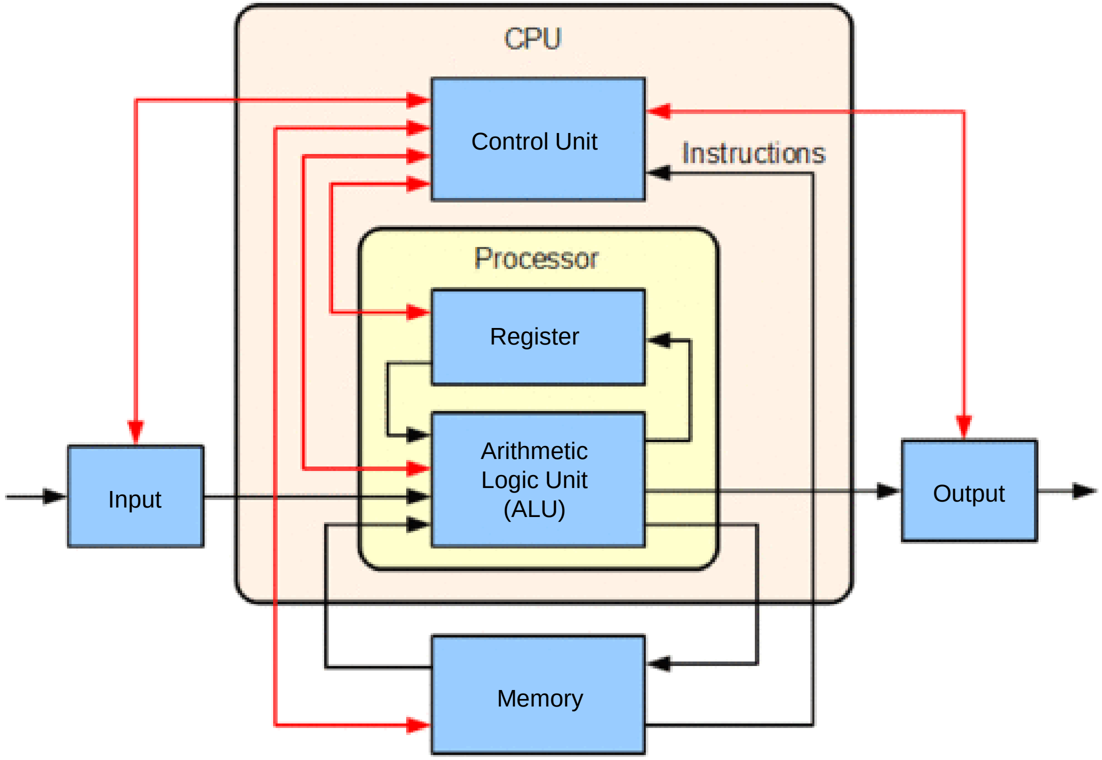
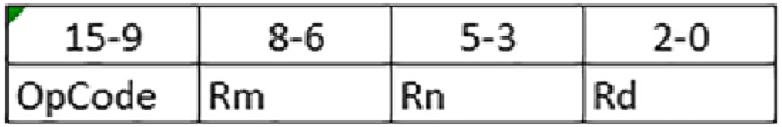


Illustration: https://www.partitionwizard.com/partitionmanager/cpu-architecture.html
https://en.wikipedia.org/wiki/Arithmetic_logic_unit

Microcontroller architecture (Common components)

- **CPU (Central Processing Unit)**
  The brain of the microcontroller, responsible for executing instructions.
  - **ARM Cortex-M0+** binary instructions are 16 bits long.
    - (The processor on the Raspberry Pi Pico)



15-9	8-6	5-3	2-0
OpCode	Rm	Rn	Rd

  - **More human-readable?** Assembly Instruction!

ADD R5, R3, R2

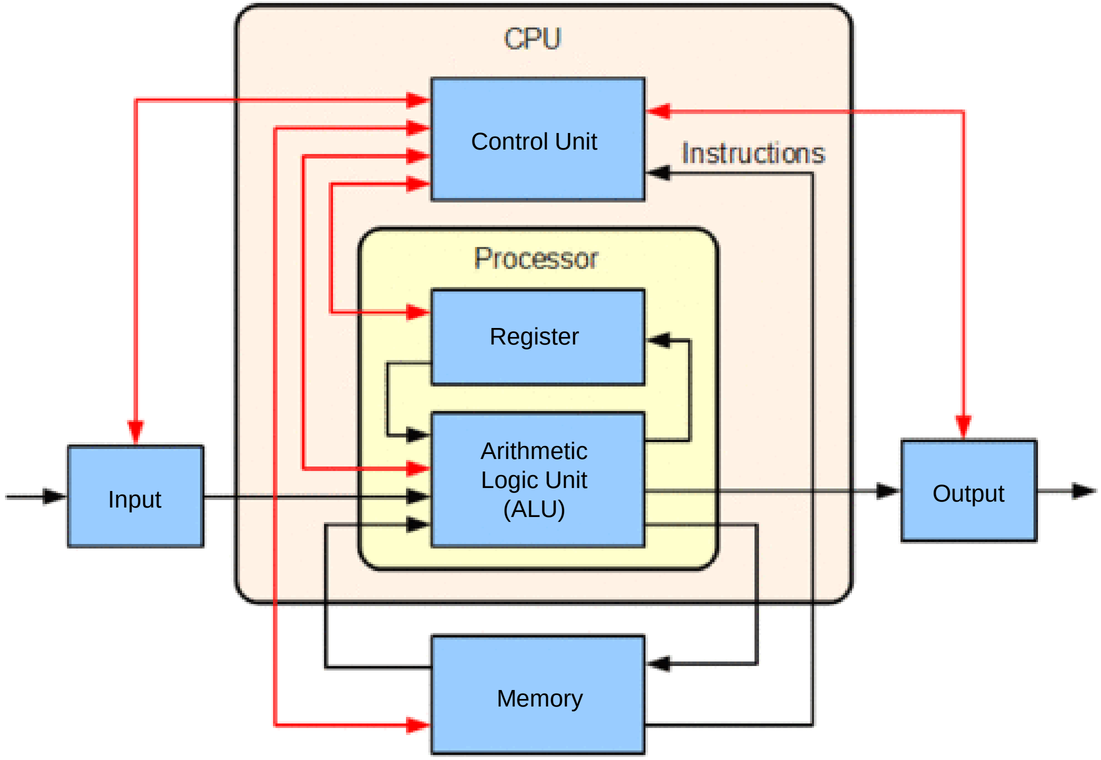
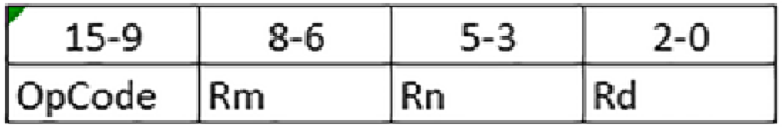


Illustration: https://www.partitionwizard.com/partitionmanager/cpu-architecture.html
https://en.wikipedia.org/wiki/Arithmetic_logic_unit

Microcontroller architecture (Common components)

- **CPU (Central Processing Unit)**
  The brain of the microcontroller, responsible for executing instructions.
  - **ARM Cortex-M0+** binary instructions are 16 bits long.
    - (The processor on the Raspberry Pi Pico)



15-9	8-6	5-3	2-0
OpCode	Rm	Rn	Rd

  - **“More” human-readable?** Assembly Instruction!

ADD R5, R3, R2

  - **Instruction to computer** R5 = R3 + R2
    - OpCode = 0001100 meaning ADD
    - Rm = 010 = 2 (i.e., R2)
    - Rn = 011 = 3 (i.e., R3)
    - Rd = 101 = 5 (i.e., R5)
    - **In binary** →0001 1000 1001 1101

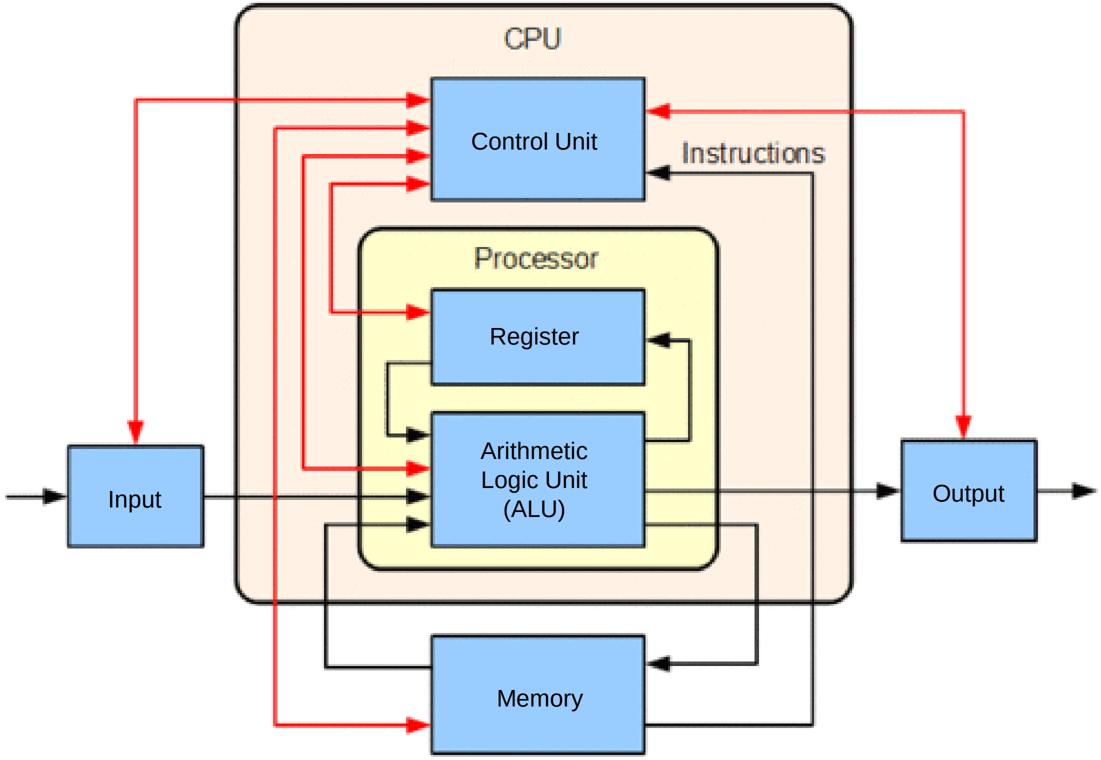


Illustration: https://www.partitionwizard.com/partitionmanager/cpu-architecture.html

Microcontroller architecture (Common components)

- **CPU (Central Processing Unit)**
  The brain of the microcontroller, responsible for executing instructions.
  - **Control Unit**
  - **Arithmetic Logic Unit (ALU)**
  - **Memory / Registers**
    - **Registers:** Small, fast storage locations within the CPU used for quick data manipulation and storage during processing.
      - **Accumulator:** Often used for arithmetic and logic operations.
      - **Instruction Register:** Holds the current instruction being executed.
      - **Address Register:** Holds the address of the location to be accessed from memory.
      - **Program Counter:** Holds the address of the next instruction to be executed.

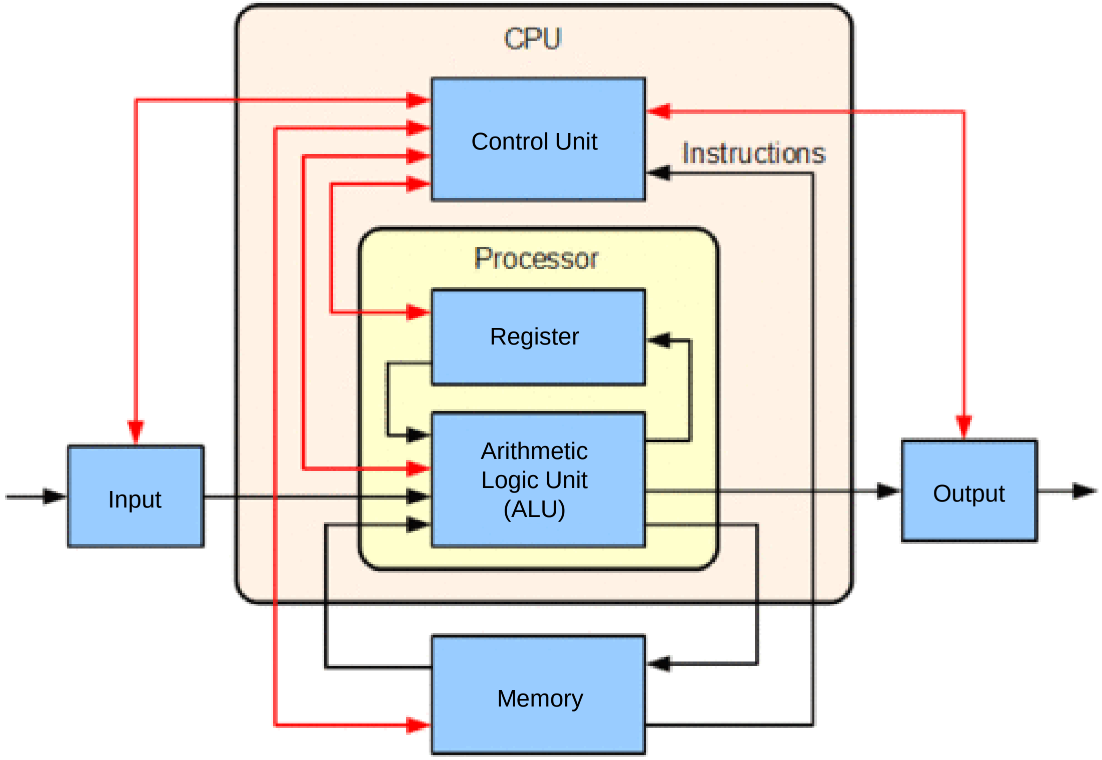
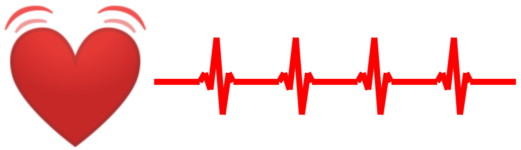
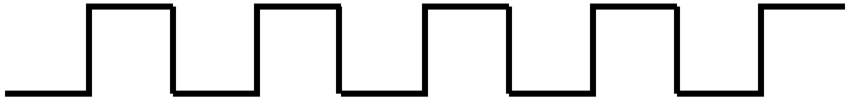


Illustration: https://www.partitionwizard.com/partitionmanager/cpu-architecture.html

Pipeline code execution



CPU Clock



(The “heartbeat” of the microcontroller)

Program/Code Memory

PC = 0

PC = 1

PC = 2

PC = 3

PC = 4

Address	Memory	Fetch / Decode	Execution
0	Instruction 0	Instruction 0	
1	Instruction 1	Instruction 1	Instruction 0
2	Instruction 2	Instruction 2	Instruction 1
3	Instruction 3	Instruction 3	Instruction 2
4	Instruction 4	Instruction 4	Instruction 3
5	Instruction 5		
6	Instruction 6		

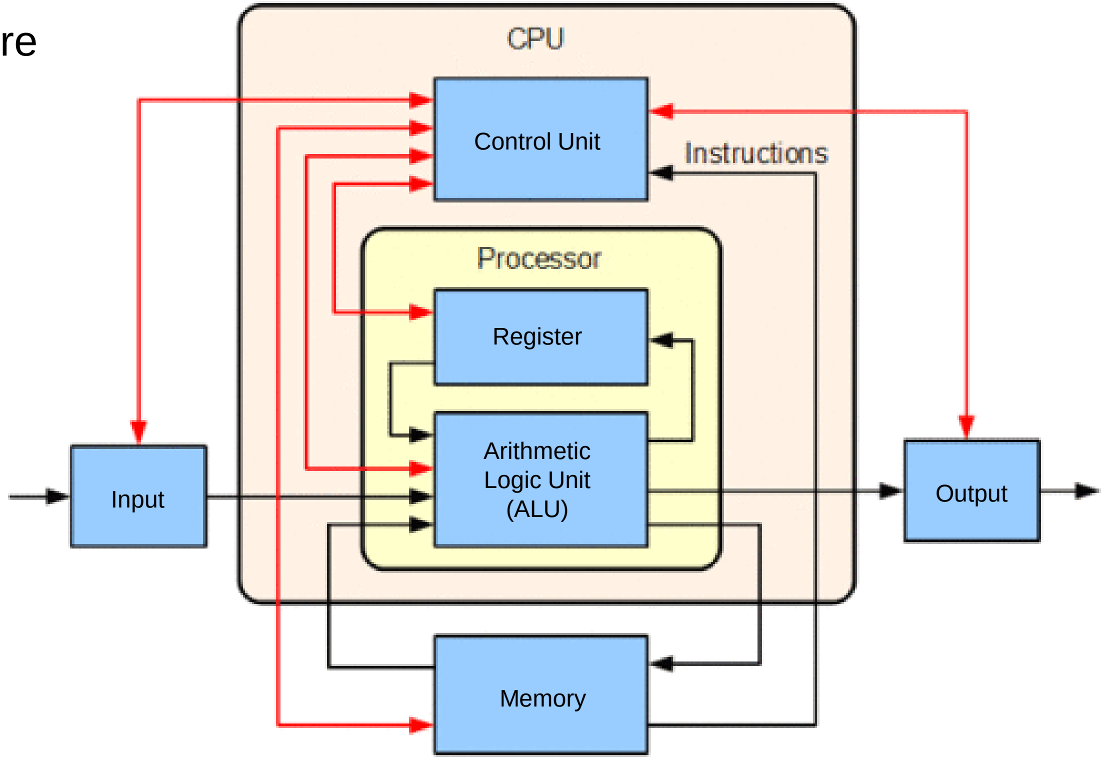
Illustration: Datateknik course, W1 (T540050101, by Ali Sahafi)

Microcontroller architecture (Common components)

- **Memory**
  The main memory of the computer where data and instructions are stored temporarily during execution.

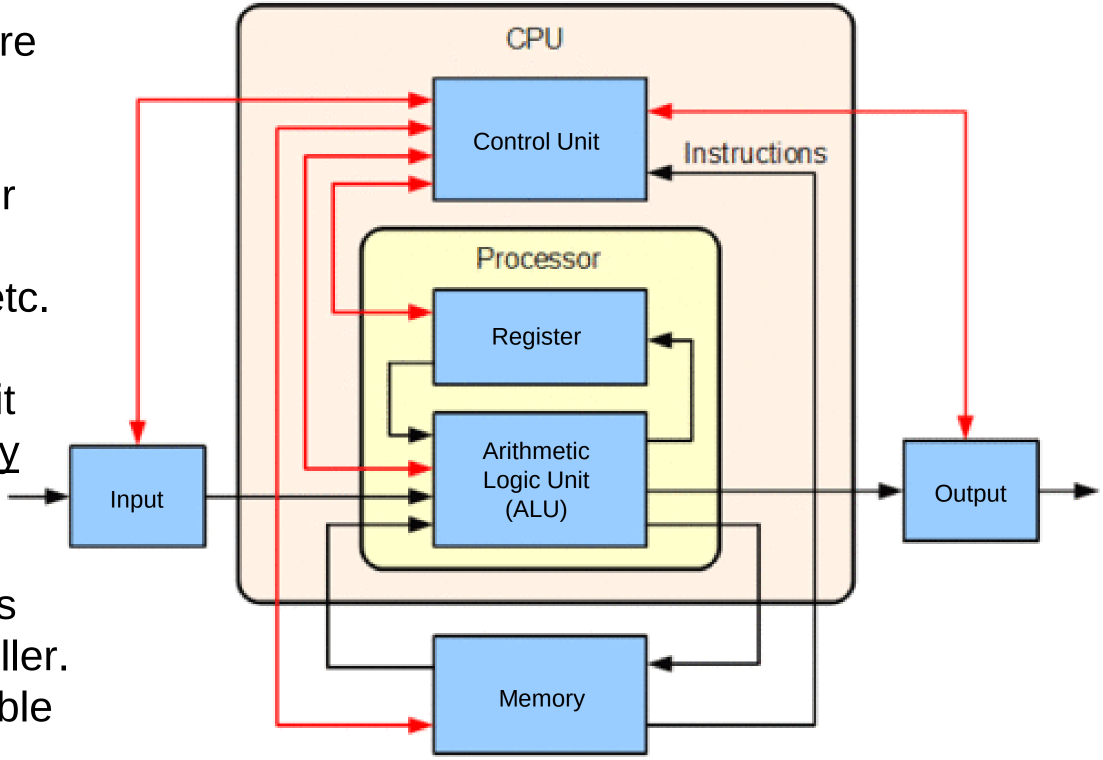
  - **Two types** of memory:

Volatile	Non-Volatile
• Temporary memory • Once the system is turned off the data is deleted automatically • Fast access to data • It is more costly per unit size • Has less storage capacity	• Data remains stored even if it is powered-off • It is slower than volatile memory. • Data can not be easily transferred in comparison to volatile memory • Non-volatile memory is less costly per unit size • Processor has no direct access to data.



Microcontroller architecture (Common components)

- **Memory**
  The main memory of the computer where data and instructions are stored temporarily during execution.
  - **RAM (Random Access Memory):** Volatile memory used for temporary storage while the microcontroller is running.
    - **Types:** SRAM (Static RAM), DRAM (Dynamic RAM), etc.
    - **Example:** When a program runs on a microcontroller, it stores variables, function call data, and other temporary data in RAM.
  - **ROM (Read-Only Memory):** Non-volatile memory that holds the firmware, or the program code that runs the microcontroller.
    - **Types:** PROM (Programmable ROM), EPROM (Erasable Programmable ROM), EEPROM (Electrically Erasable Programmable ROM), etc…
    - **Example:** The firmware that controls a microcontroller's operations and the program code is stored in ROM, ensuring that it is available every time the device is powered on.



Assignments

Assignments

- **Little Man Computer:** Simulate a Von Neumann architecture to get an understanding of:
  - Basic CPU operations: fetch, decode, execute cycle.
  - The role of memory and how instructions are stored and executed.
  - Simple low-level programming

- **Demo:** https://www.peterhigginson.co.uk/LMC