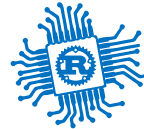


# Introduction into PM

Lecture 1



# Welcome

to the *Proiectarea cu Microprocesoare* engineering class

## You will learn

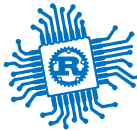
- how hardware works
- how to actually build your own hardware device
- the Rust programming Language
- a little bit of low level C

## We expect

- to come to class
- ask a lot of questions
- maybe some work at home

2025 is an experiment - we will keep it chill

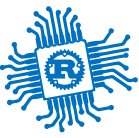




# DISCLAIMER

- These slides represent a summary.
- The slides do not cover all the explanations, simulations, or demonstrations provided during the course.
- The slides do not limit, in any way, the material required for the exam.
- For the complete version, you are welcome to attend the course.

(copyright info) These slides may contain materials shared with my colleagues Alexandru Radovici, Dan Tudose, Alexandru Vaduva, Razvan Tataroiu

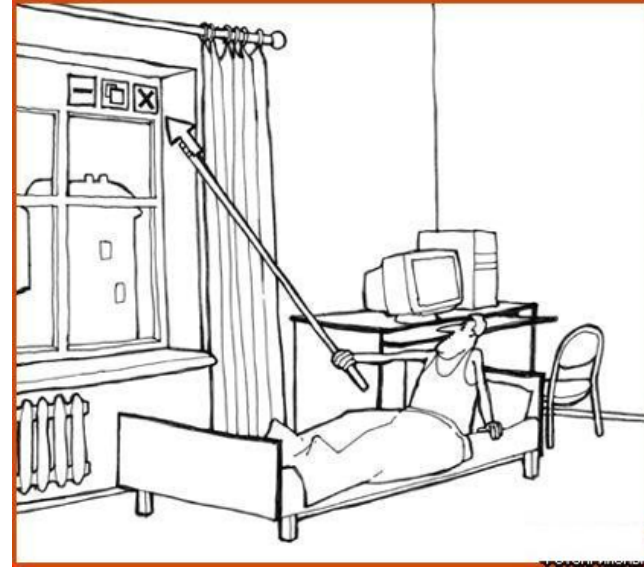


# ENG

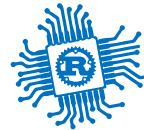
Scientific understanding of the natural world

Used to invent, design, and build things

Used to solve problems and achieve practical goal







# Why PM

Computing systems with microprocessors > everywhere

## Questions for an engineer:

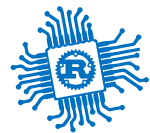
- What is inside a computing system?
- How do the components interact?
- How do I design a system that interacts with the physical environment?
- How do I choose the best hardware option for an embedded system?

"Data-based decisions" – based on IoT infrastructure require:

- Actual physical sensors
- Lots of IoT custom hardware



# Team



# Our team

**Daniel Rosner**



Course Professor

**Irina Niță**



Lab Professor  
Software

**Irina Bradu**



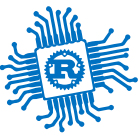
Lab Professor

**Teodor Dicu**



Lab Professor  
Hardware

---



# Despre Daniel Rosner

Cursuri

DEEA

PM

How To Build Your Cyber Security  
Startup

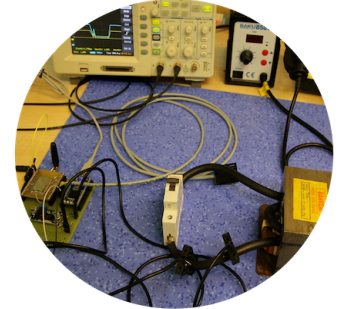
VZ & PoliFest

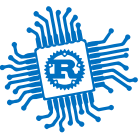
Innovation Labs &  
Concursuri (tech)

Tech area

Automotive

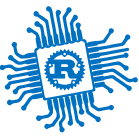
MedTech





# Outline





# Outline

## Lectures

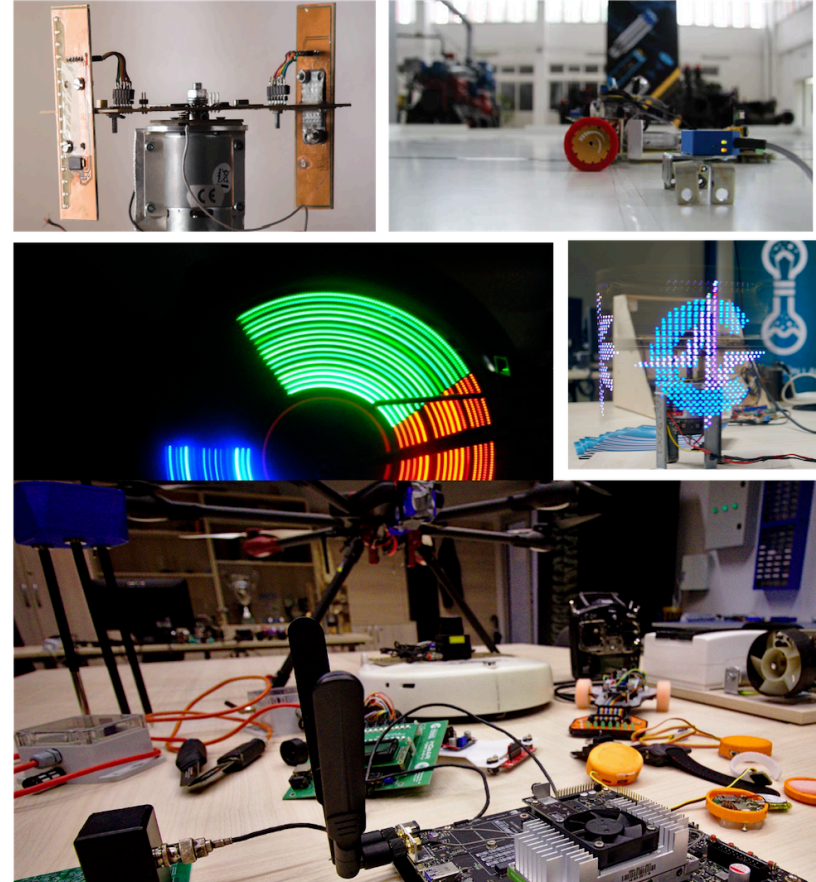
- 12 lectures
- 1 Q&A lecture for the project

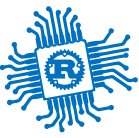
## Labs

- 12 labs

## Project

- Build a hardware device running software written in Rust or C on a microcontroller-based board
- The cost for the hardware is around 150 RON
- Presented at PM Fair during the last week of the semester





# Scoring Structure

1 point for lab activity

1 point for lab assignment (final lab exam)

3 points PROJECT

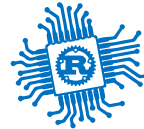
2 points lectures activity (announced tests)

3 points @ Final Exam

## *Bonus*

*+0.75 bonus for top 30 projects of the year (top 7%)*

*+0.75 bonus for top 10 projects of the year*



# Project

## Structure

Documentation / Hard / Soft

PM-fair

## Project scope

Needs to be approved by your laboratory teacher

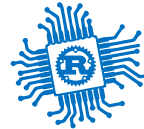
## It can not be super-simple!

*(digital clock, digital thermometer)*

## A few reference points:

It can not be simpler than one laboratory

It can not be based on a 30 min youtube tutorial



## Extra

### Bonus for competition & activity results

Up to 1 point for results in the top at technical profile competitions

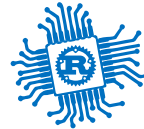
Up to 0.5 bonus points for involvement in student volunteer activities

Email in pre-session with Subject: Bonus\_PM FirstName\_LastName\_32xCC

### Equivalencies

Up to 3 points for results at technical competitions:

- ACM (top 50%);
- Innovation Labs (SemiFinals);
- Suceava Hard and Soft (top 50%);



# (Example) Innovation Labs

Why join:



CV



Team-Work



Profesional Networking



Presentations skills



Build your own start-up with a super support structure



500.000 EURO Investment Prize



Summer Internship @ your own start-up



8 - 9 March - the largest, coolest, most fun Hackathon in Romania

PS: 🕒 Is it a good time considering how the IT market looks?



Yes! > It's the best time:



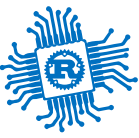
gain practical experience & boost your CV;



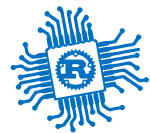
build a public profile & establish 👤 relationships with IL partner IT companies (e.g., Adobe, Keysight, NXP, UiPath, Stripe);



improve your skills beyond coding (lowers the risk of being replaced by ChatGPT :))



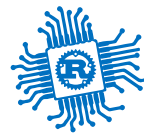
# Apollo Guidance Computer



# *We choose to go to the moon*

John F. Kennedy, Rice University, 1961

*in this decade and do the other things, **not because they are easy, but because they are hard**, because **that goal will serve to organize and measure the best of our energies and skills**, because that challenge is one that we are willing to accept, one we are unwilling to postpone, and one which we intend to win, and the others, too.*

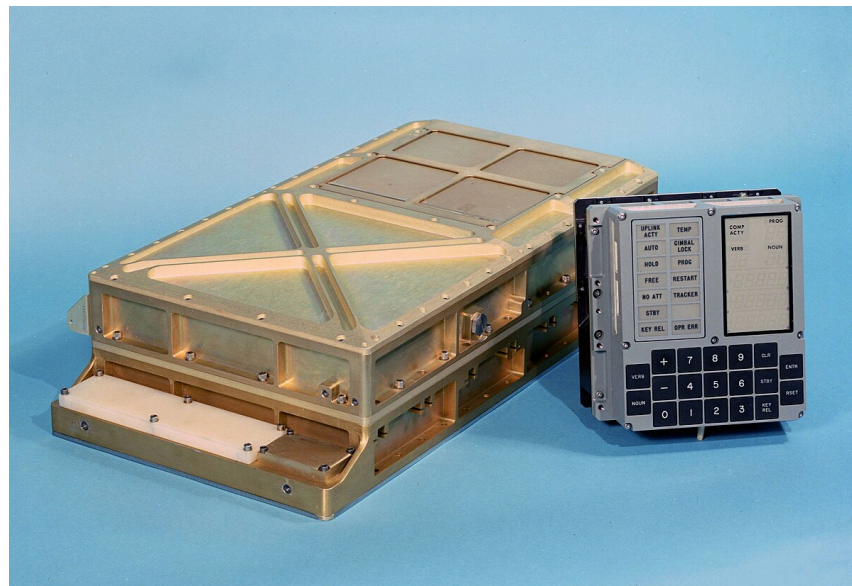


# AGC

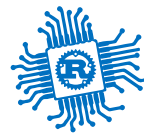
August 1966

|              |                       |
|--------------|-----------------------|
| Frequency    | 2.048 MHz             |
| World Length | 15 + 1 bit            |
| RAM          | 4096 B                |
| Storage      | 72 KB                 |
| Software API | AGC Assembly Language |

This landed the *moon eagle*.

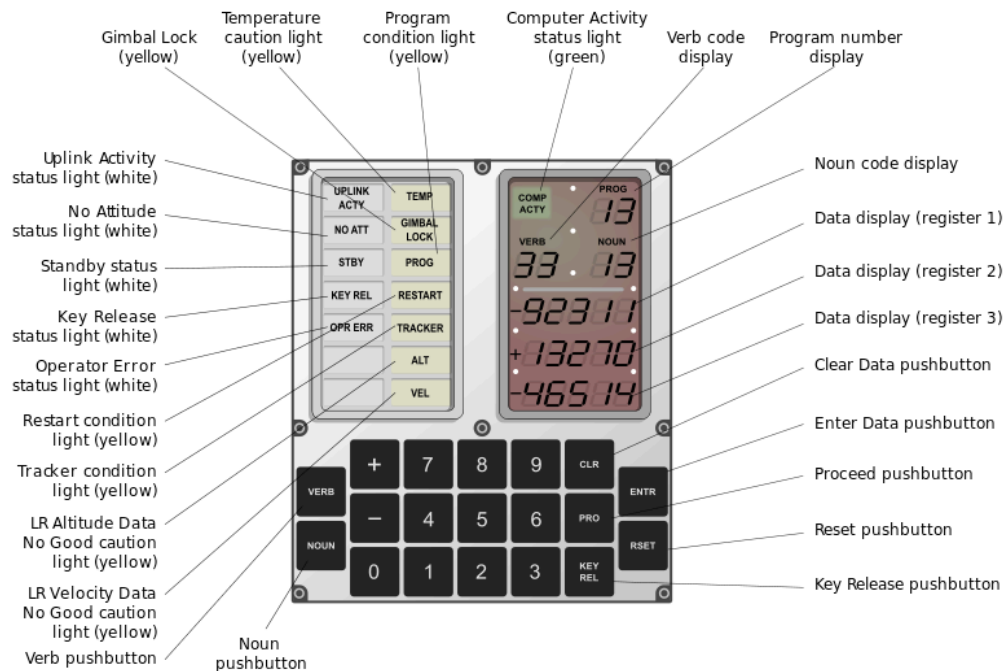




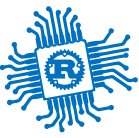


# DSKY

## Display and keyboards



## Simulator



# Where we are now





# Embedded Systems

In general, they have a dedicated function.

Common constraints:

Real-time requirements

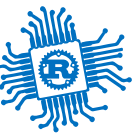
Fixed response time:

- Control (e.g., constant-time sampling)
- Safety (response within a limited time upon detection)

Limited resources (processing power/memory)

Robustness requirements (aka high uptime)

# Example

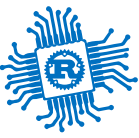




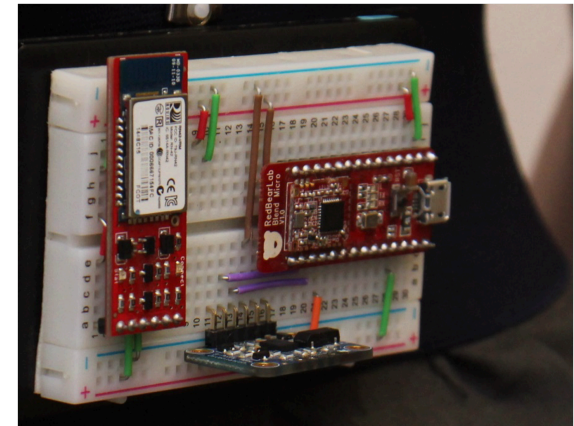
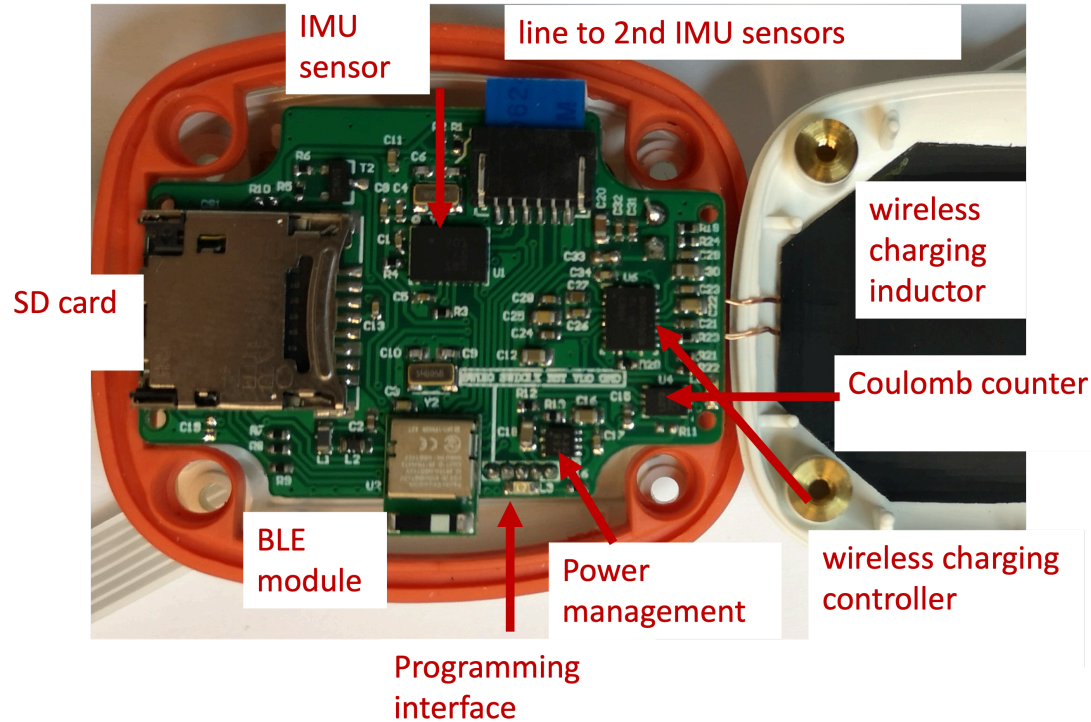
# Example controller

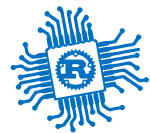
NXP S32ZE

STM32H



# Example ENTy





# Example Companies

NXP

Infineon

Microchip

EPG

Renault

Continental

Viavi

Siemens

Emerson

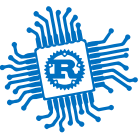
GE

Honeywell

Thales

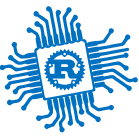
Hella

Bosch



# What is a microprocessor?

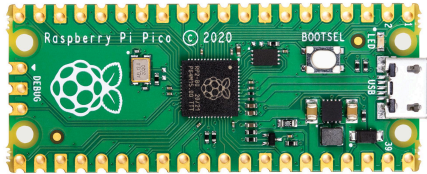




# Microcontroller (MCU)

Integrated in embedded systems for certain tasks

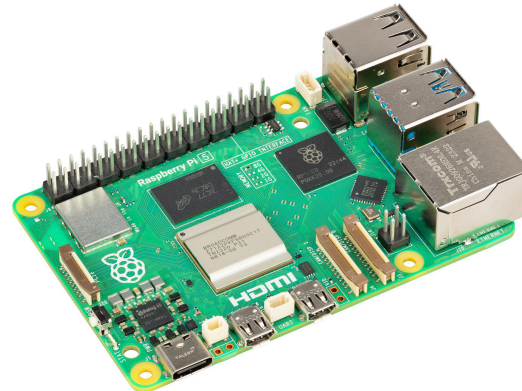
- low operating frequency (MHz)
- a lot of I/O ports
- controls hardware
- does not require an Operating System
- costs \$0.1 - \$25
- annual demand is billions



# Microprocessor (CPU)

General purpose, for PC & workstations

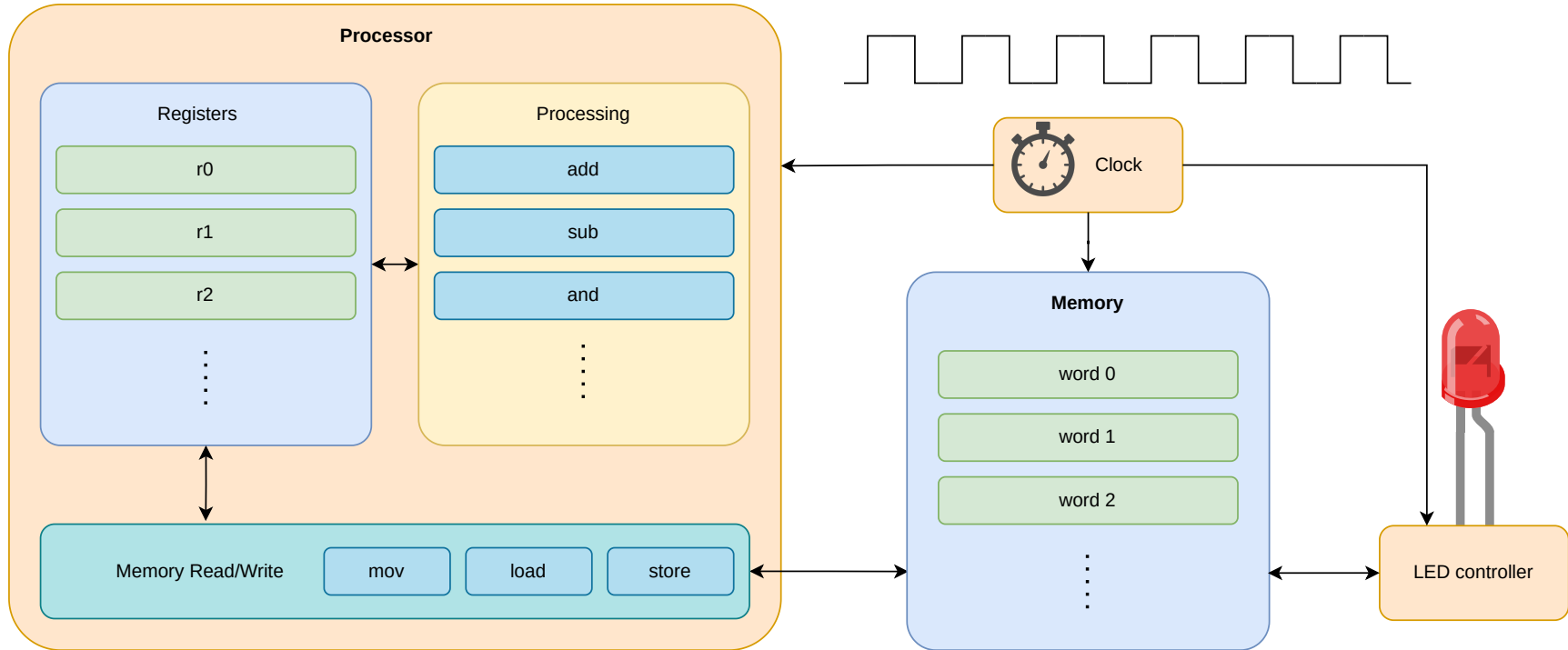
- high operating frequency (GHz)
- limited number of I/O ports
- usually requires an Operating System
- costs \$75 - \$500
- annual demand is tens of millions





# How a microprocessor works

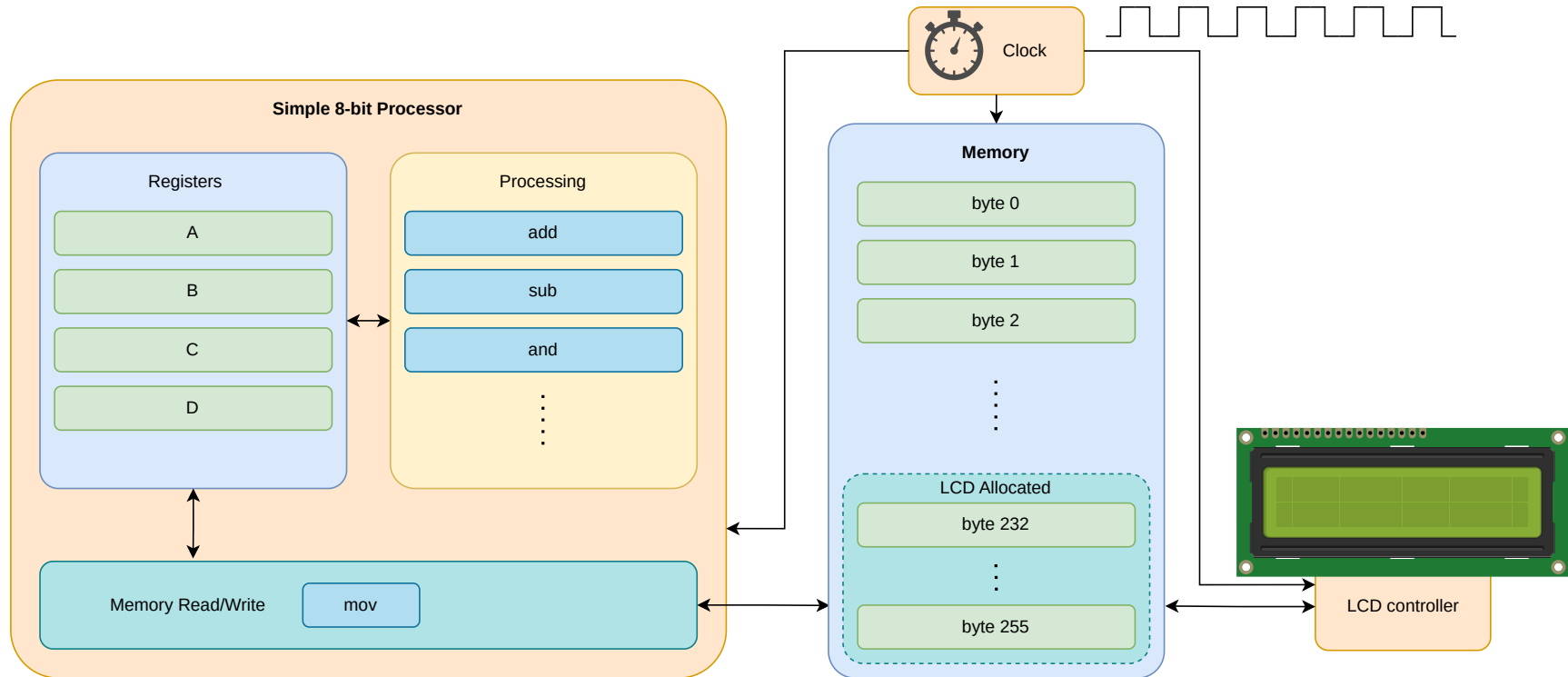
This is a simple processor

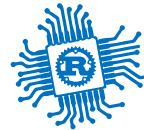




# 8 bit processor

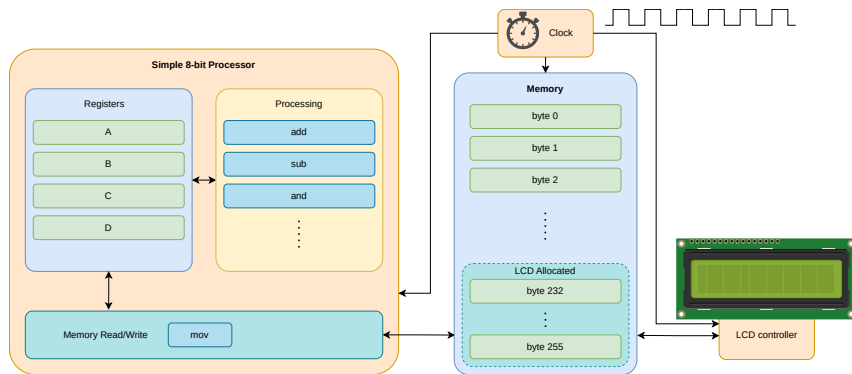
a simple 8 bit processor with a text display





# Programming

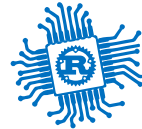
in Rust



```
1 use eight_bit_processor::print;
2
3 static hello: &str = "Hello World!";
4
5 #[start]
6 fn start() {
7     print(hello);
8 }
```

# Assembly

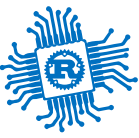
```
1     JMP start
2     hello: DB "Hello World!" ; Variable
3           DB 0 ; String terminator
4     start:
5         MOV C, hello ; Point to var
6         MOV D, 232 ; Point to output
7         CALL print
8         HLT ; Stop execution
9     print: ; print(C:*from, D:*to)
10        PUSH A
11        PUSH B
12        MOV B, 0
13    .loop:
14        MOV A, [C] ; Get char from var
15        MOV [D], A ; Write to output
16        INC C
17        INC D
18        CMP B, [C] ; Check if end
19        JNZ .loop ; jump if not
20
21        POP B
22        POP A
23        RET
```



# Demo

a working example for the previous code

Start



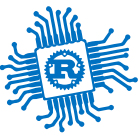
# Microprocessors VS Microcontrollers

## Microcontroller

A microcontroller is a small computer on a single integrated circuit (IC).

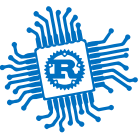
## Microprocessor

A microprocessor is a computer central processing unit (CPU) on a single integrated circuit (IC).

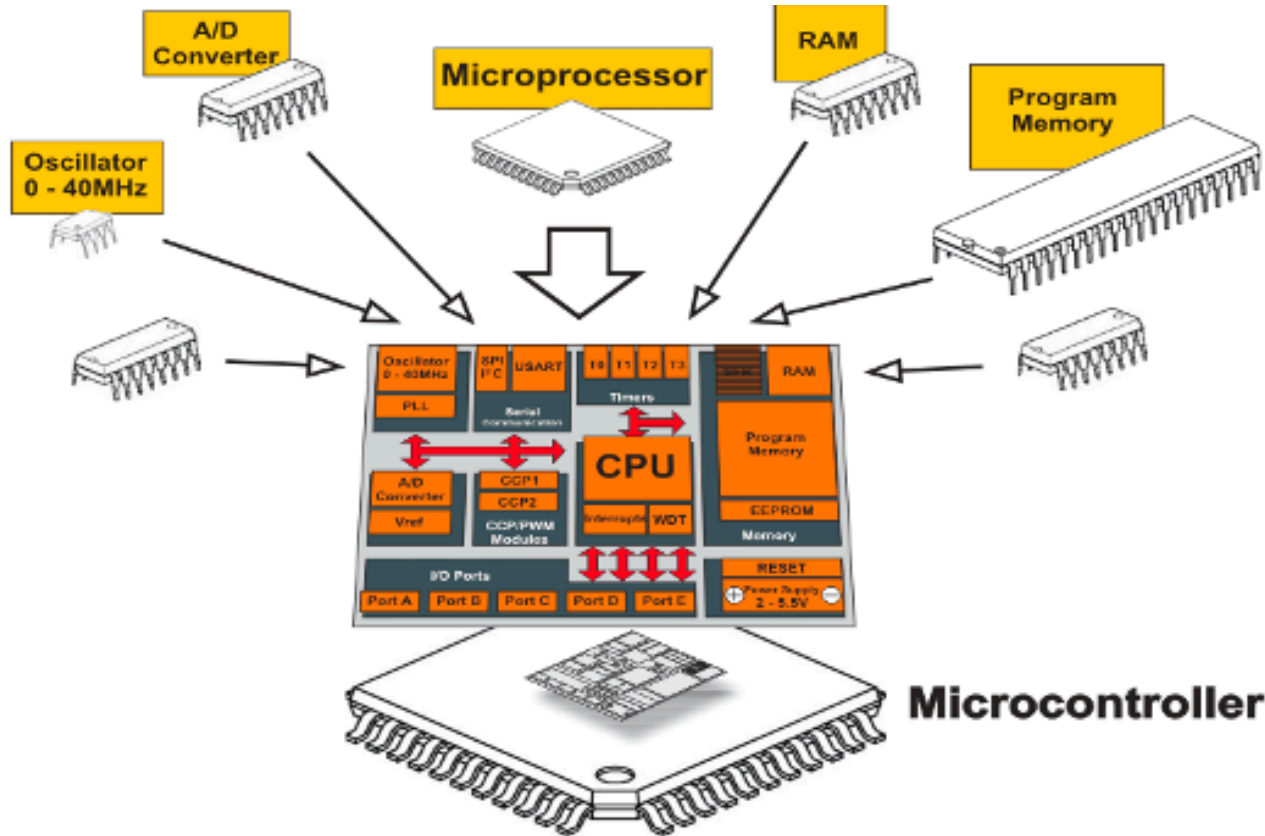


# Comparison

| Characteristic | Microcontroller             | Microprocessor               |
|----------------|-----------------------------|------------------------------|
| Function       | Includes CPU, mem & I/O     | Includes only the CPU        |
| Cost           | >> <i>cheaper</i>           | >> <i>expensive</i>          |
| Complexity     | >> <i>simple</i>            | >> <i>complex</i>            |
| Use case       | <i>Incorporated devices</i> | <i>PCs, Servers, Laptops</i> |



# Graphic representation









# Note: von Neumann VS Harvard

From the point of view of memory access, there are 2 architectures:

von Neumann, where memory contains both instructions and data.

Today's PCs are all von Neumann

Harvard, where memory access is done on separate buses, one for data, one for instructions.

AVR, PIC, DSPs and many microcontrollers are Harvard

Note: ARM is von Neumann with some \* Note: GPUs (NVIDIA) are mixed architecture



## Note: microcontrollers - general observations

*Microcontroller (MCU) – a mini computer on a single silicon chip that integrates:*

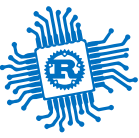
Processor

Data memory

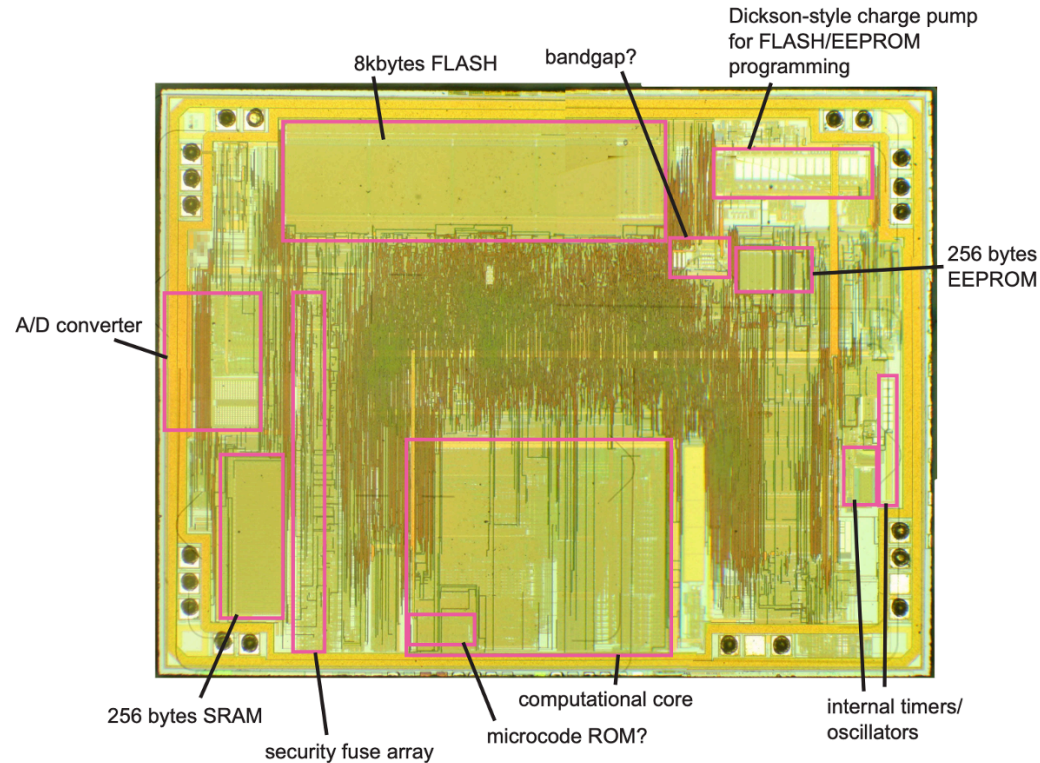
Program memory

Peripherals

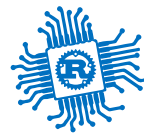
In contrast to a microprocessor that needs other external chips for memory, control, peripherals



# Under the microscope



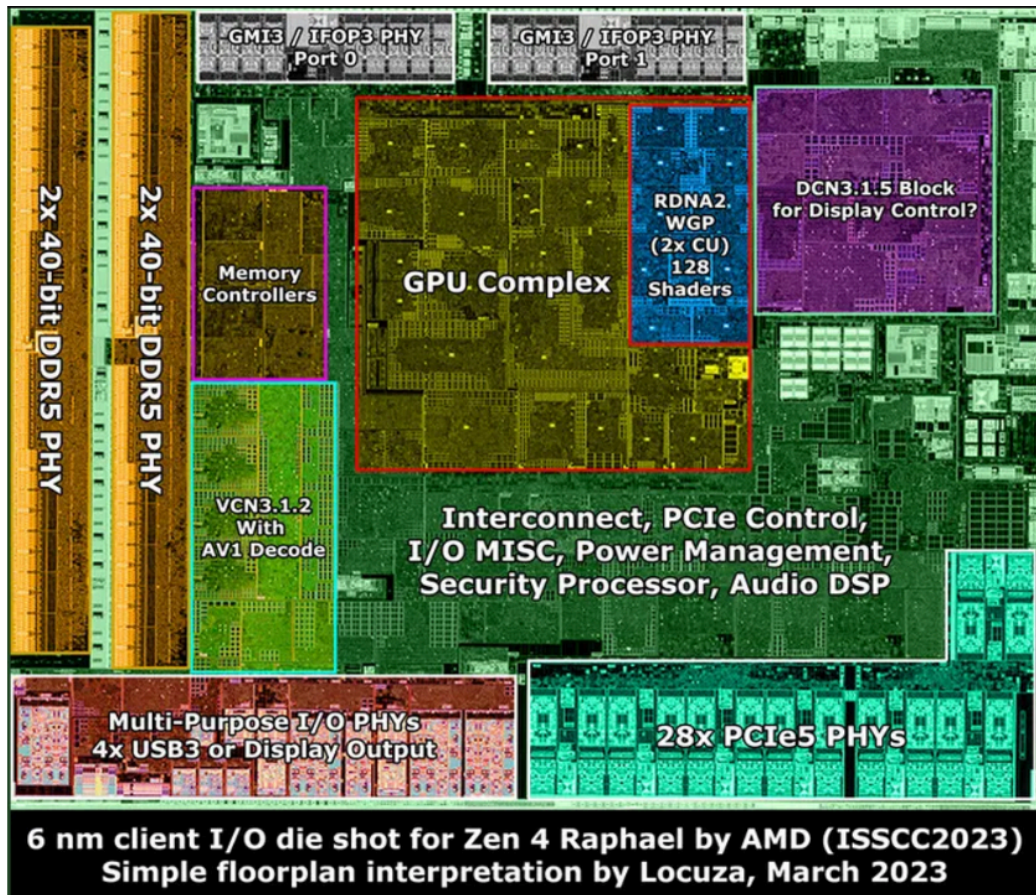


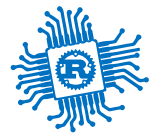


(extra)

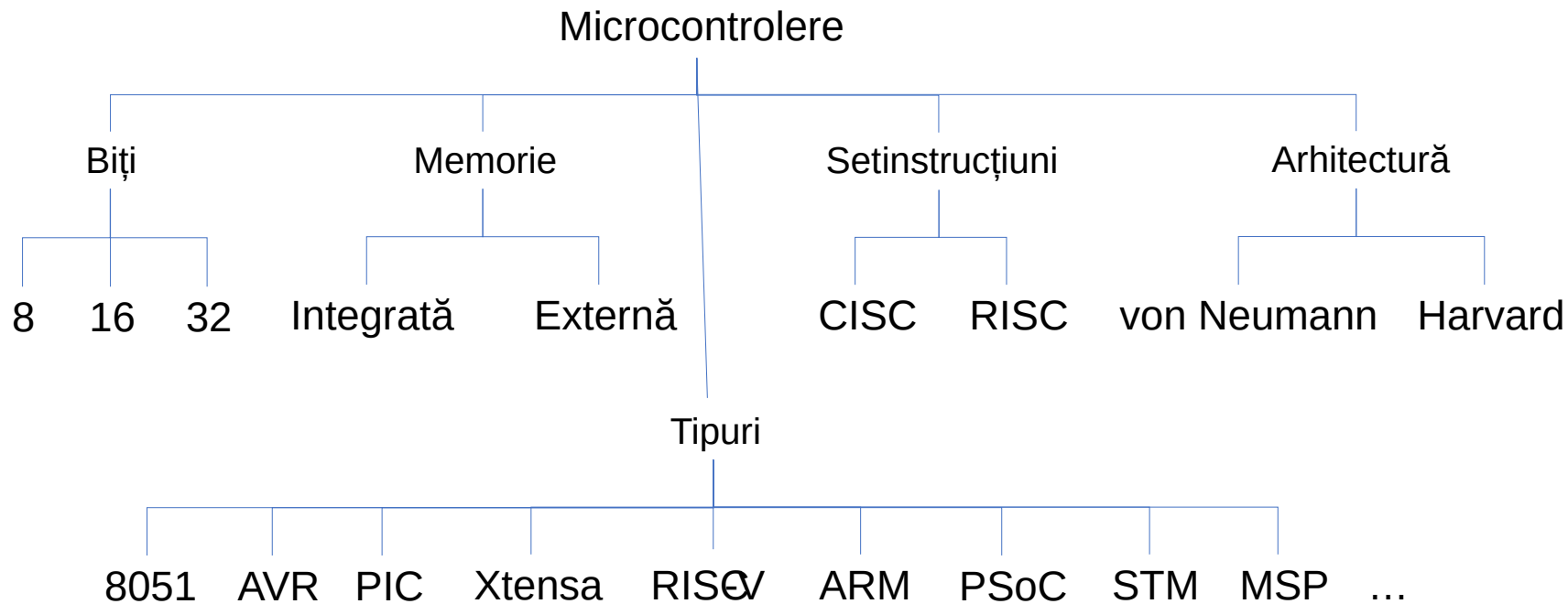
©

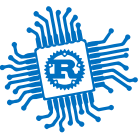
<https://www.tomshardware.com/news/amd-shares-new-second-gen-3d-v-cache-chiplet-details-up-to-25-tbs>





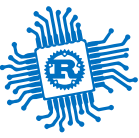
# Types





# How to choose the right one ?

- ? Energy consumption
  - ? Operating frequency
  - ? IO Pins & Supported Peripheral / Interface Types
- (discussion)
- ? Memory
  - ? Internal functions
  - ? Software availability & support!

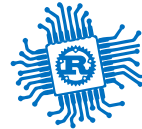


# Hello World on AVR in C

```
1  #include <avr/io.h>
2  #include <util/delay.h>
3
4  #define F_CPU 12000000UL //MCU clock frequency
5
6  int main()
7  {
8      DDRC = (1 << PC0); //Set pin 0 of PORT C as output
9      //DDRC = Data Direction Register for PORT C
10     while(1)
11     {
12         PORTC ^= (1 << PC0); //Toggle pin 0 of PORT C (XOR)
13         _delay_ms(500);
14     }
15 }
```

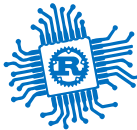
Note: the above code can toggle an LED on/ off every 500ms





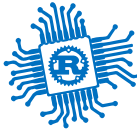
# Let's go lower level

```
1  //00000000 <__vectors>:
2  //__vectors():
3  0: 0c 94 3e 00 jmp 0x7c ; 0x7c <__ctors_end> //reset
4  4: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
5  8: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
6  c: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
7  10: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
8  14: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
9  18: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
10 1c: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
11 20: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
12 24: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
13 28: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
14
15 .....
16
17 60: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
18 64: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
19 68: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
20 6c: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
21 70: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
22 74: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
23 78: 0c 94 48 00 jmp 0x90 ; 0x90 <__bad_interrupt>
24 0000007c <__ctors_end>:
```



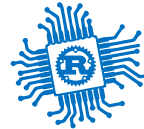
# Next code

```
1  //__trampolines_start():
2      7c: 11 24 eor r1, r1 ; r1 = 0          //program jumps here at reset
3      7e: 1f be out 0x3f, r1 ; SREG = r1
4      80: cf ef ldi r28, 0xFF ; 255
5      82: d8 e0 ldi r29, 0x08 ; 8
6      84: de bf out 0x3e, r29 ; SPH = 0x8      //stack pointer on the last RAM address - 0x08FF for 328P
7      86: cd bf out 0x3d, r28 ; SPL = 0xFF.    //stack Pointer High and Low - to get a 16b address on a 8bit MCU
8      88: 0e 94 4a 00 call 0x94 ; 0x94 <main>
9      8c: 0c 94 59 00 jmp 0xb2 ; 0xb2 <_exit> 00000090
10
11  //<__bad_interrupt>: __vector_22():
12      90: 0c 94 00 00 jmp 0 ; 0x0 <__vectors>.    //any interrupt triggers a reset
```



# We get to the code

```
1      94: 38 9a sbi 0x07, 0 ; DDRC = 0x01          //DDRC |= (1 << PC0);
2
3      96: 91 e0 ldi r25, 0x01 ; r25 = 1
4      98: 88 b1 in r24, 0x08 ; r24 = PORTC          //from here PORTC ^= (1 << PC0);
5      9a: 89 27 eor r24, r25 ; r24 = r24 ^ 1
6      9c: 88 b9 out 0x08, r24 ; PORTC = r24
7
8      9e: 2f e9 ldi r18, 0x9F ; 159          //from here _delay_ms():
9      a0: 36 e8 ldi r19, 0x86 ; 134
10     a2: 81 e0 ldi r24, 0x01 ; 1
11     a4: 21 50 subi r18, 0x01 ; 1
12     a6: 30 40 sbci r19, 0x00 ; 0
13     a8: 80 40 sbci r24, 0x00 ; 0
14     aa: e1 f7 brne .-8 ; 0xa4 <main+0x10>
15     ac: 00 c0 rjmp .+0 ; 0xae <main+0x1a>
16     ae: 00 00 nop b0: f3 cf rjmp .-26 ; 0x98 <main+0x4> //jumps back to the loop (98)
```



# Real World Microcontrollers

Intel / AVR / PIC / TriCore / ARM Cortex-M / RISC-V rv32i(a)mc



# Bibliography

for this section

**Joseph Yiu**, *The Definitive Guide to ARM® Cortex®-M0 and Cortex-M0+ Processors, 2nd Edition*

- Chapter 1 - *Introduction*
- Chapter 2 - *Technical Overview*



|           |                   |
|-----------|-------------------|
| Vendor    | Intel             |
| ISA       | 8051, 8051        |
| Word      | 8 bit             |
| Frequency | a few MHz         |
| Storage   | ?                 |
| Variants  | <i>8048, 8051</i> |

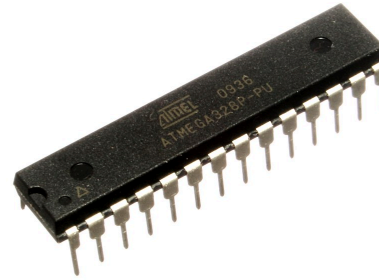




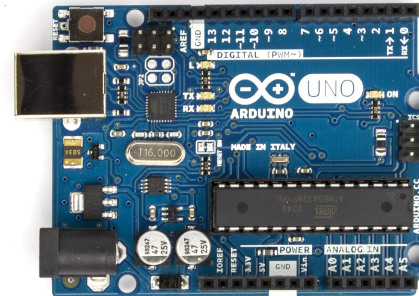
# AVR

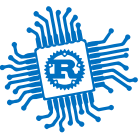
probably *Alf and Vegard's RISC processor*

|           |                                  |
|-----------|----------------------------------|
| Authors   | Alf-Egil Bogen and Vegard Wollan |
| Vendor    | Microchip ( <i>Atmel</i> )       |
| ISA       | AVR                              |
| Word      | 8 bit                            |
| Frequency | 1 - 20 MHz                       |
| Storage   | 4 - 256 KB                       |
| Variants  | <i>ATmega</i> , <i>ATtiny</i>    |



Board

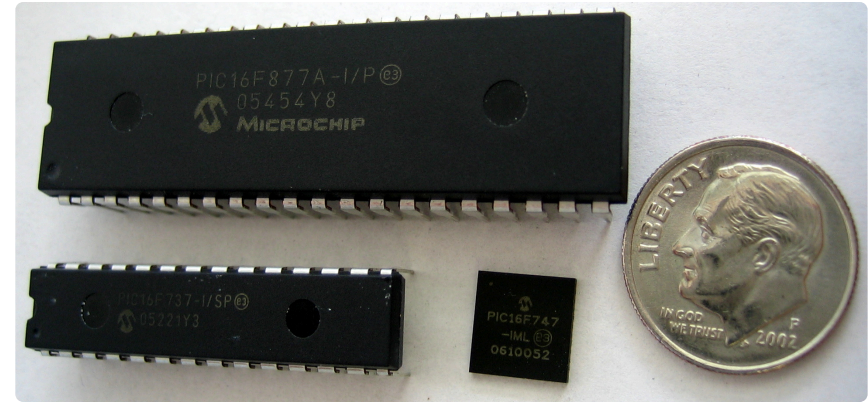




# PIC

Peripheral Interface Controller / Programmable Intelligent Computer

|           |                                                 |
|-----------|-------------------------------------------------|
| Vendor    | Microchip                                       |
| ISA       | PIC                                             |
| Word      | 8 - 32                                          |
| Frequency | 1 - 20 MHz                                      |
| Storage   | 256 B - 64 KB                                   |
| Variants  | <i>PIC10, PIC12, PIC16, PIC18, PIC24, PIC32</i> |



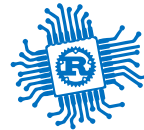




# TriCore



|           |                            |
|-----------|----------------------------|
| Vendor    | Infineon                   |
| ISA       | AURIX32                    |
| Word      | 32 bit                     |
| Frequency | hundreds of MHz            |
| Storage   | a few MB                   |
| Variants  | <i>TC2xx, TC3xx, TC4xx</i> |

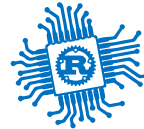


# ARM Cortex-M

Advanced RISC Machine

arm

|           |                                                                                                |
|-----------|------------------------------------------------------------------------------------------------|
| Vendor    | Qualcomm, NXP, Nordic Semiconductor, Broadcom, Raspberry Pi                                    |
| ISA       | ARMv6-M (Thumb and some Thumb-2)<br>ARMv7-M (Thumb and Thumb-2)<br>ARMv8-M (Thumb and Thumb-2) |
| Word      | 32                                                                                             |
| Frequency | 1 - 900 MHz                                                                                    |
| Storage   | up to a few MB                                                                                 |
| Variants  | <i>M0, M0+, M3, M4, M7, M23, M33</i>                                                           |

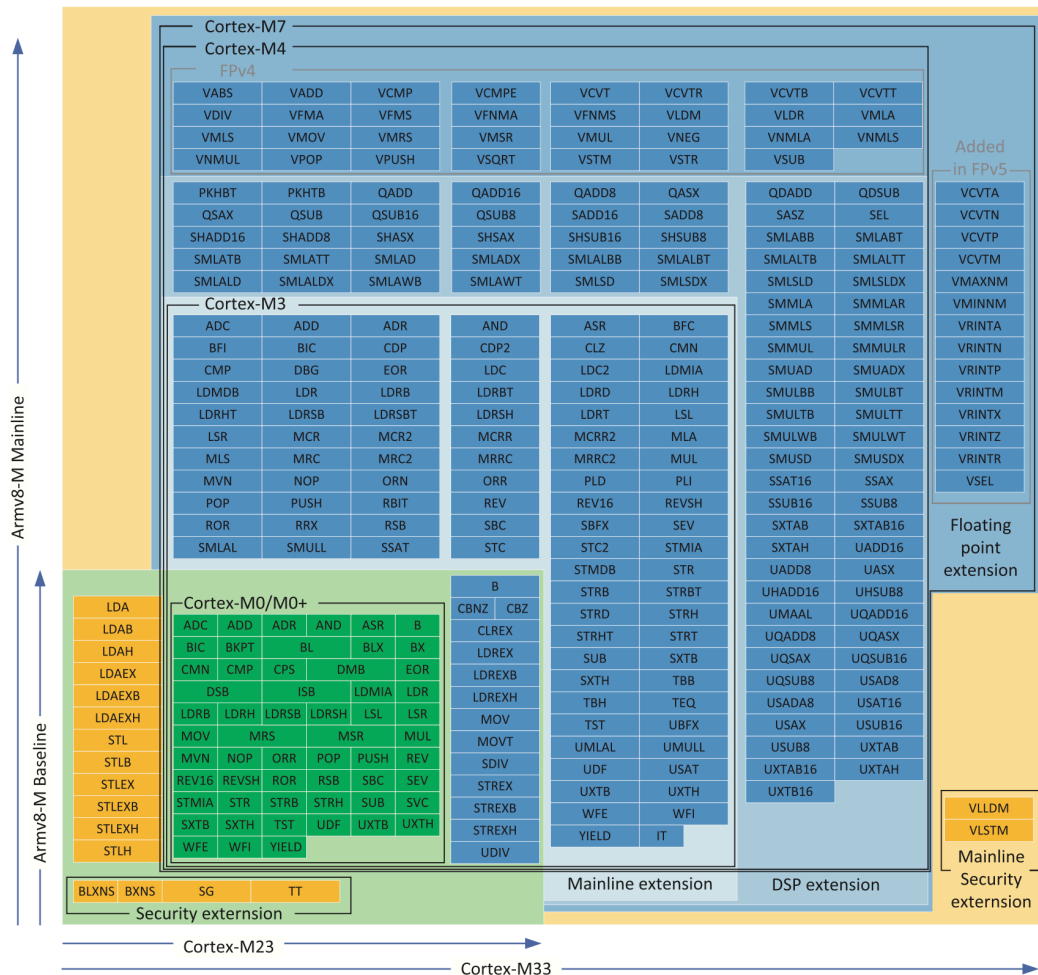


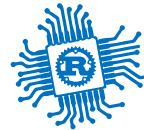
# ARM Cortex-M Instruction Set

what the MCU can do

## Fun Facts

- M0/M0+ has no `div`
- M0 - M3 have no floating point
- M23 and M33 have security extensions



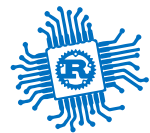


# RISC-V rv32i(a)mc

Fifth generation of RISC ISA

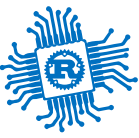


|           |                                    |
|-----------|------------------------------------|
| Authors   | University of California, Berkeley |
| Vendor    | Espressif System                   |
| ISA       | rv32i(a)mc                         |
| Word      | 32 bit                             |
| Frequency | 1 - 200 MHz                        |
| Storage   | 4 - 256 KB                         |
| Variants  | <i>rv32imc, rv32iamc</i>           |



# RP2350

ARM Cortex-M33, built by Raspberry Pi



# Bibliography

for this section

**Raspberry Pi Ltd**, *RP2350 Datasheet*

- Chapter 1 - *Introduction*
- Chapter 2 - *System Description*
  - Section 2.1 - *Bus Fabric*



# RP2350

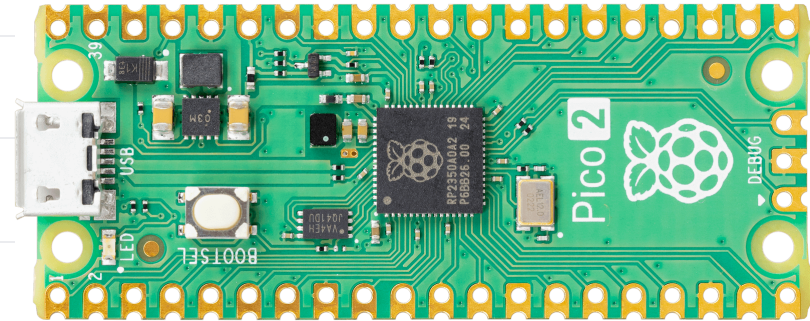
the MCU

# Boards

that use RP2350

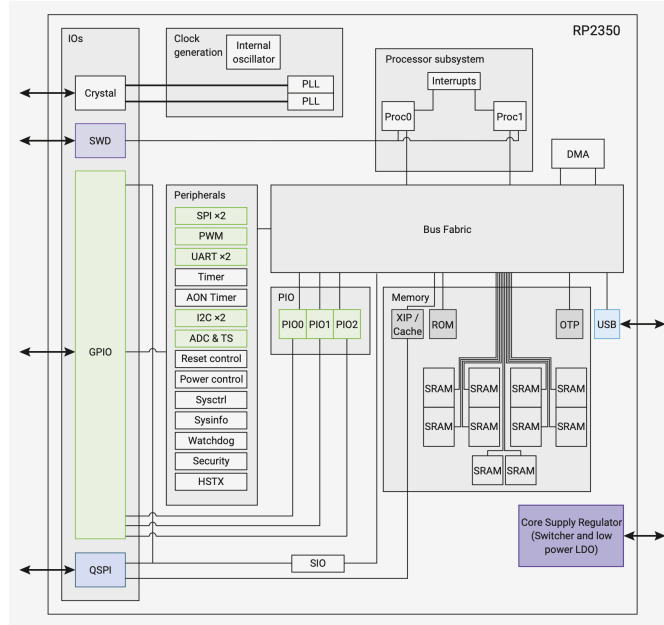
## Raspberry Pi Pico 2 (W)

|           |                                 |
|-----------|---------------------------------|
| Vendor    | Raspberry Pi                    |
| Variant   | ARM Cortex-M33 / Hazard3 RISC-V |
| ISA       | ARMv8-M / rv32iamc              |
| Cores     | 2                               |
| Word      | 32 bit                          |
| Frequency | up to 150 MHz                   |
| RAM       | 520 KB                          |





# The Chip



*GPIO*: General Purpose Input/Output

*SWD*: Debug Protocol

*DMA*: Direct Memory Access

[Datasheet RP2350](#)

## Peripherals

SIO      Single Cycle I/O (implements GPIO)

PWM      Pulse Width Modulation

ADC      Analog to Digital Converter

(Q)SPI      (Quad) Serial Peripheral Interface

UART      Universal Async. Receiver/Transmitter

RTC      Real Time Clock

I2C      Inter-Integrated Circuit

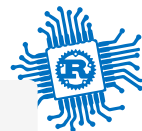
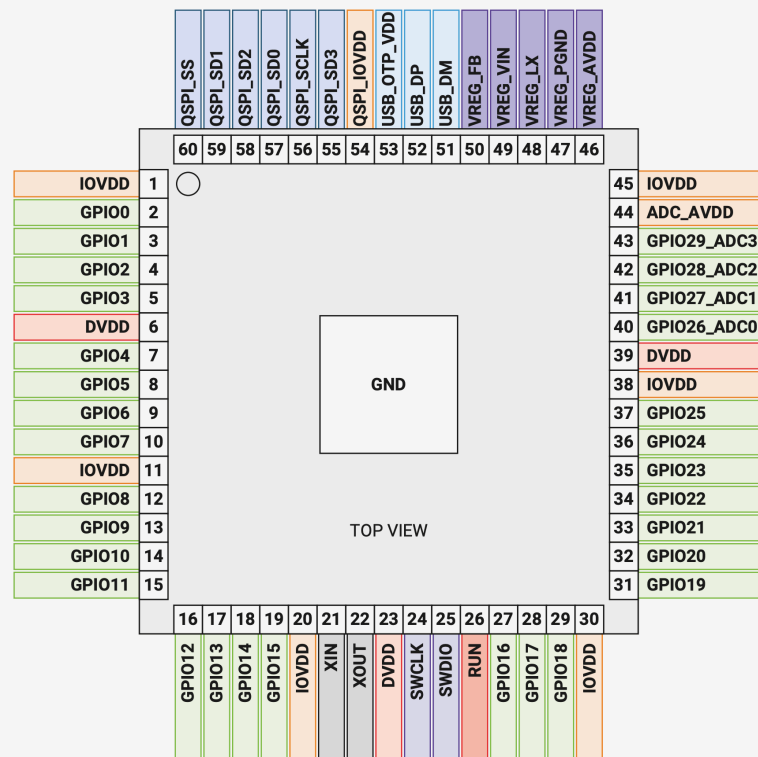
PIO      Programmable Input/Output

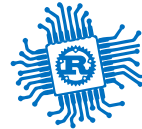


# Pins

have multiple functions

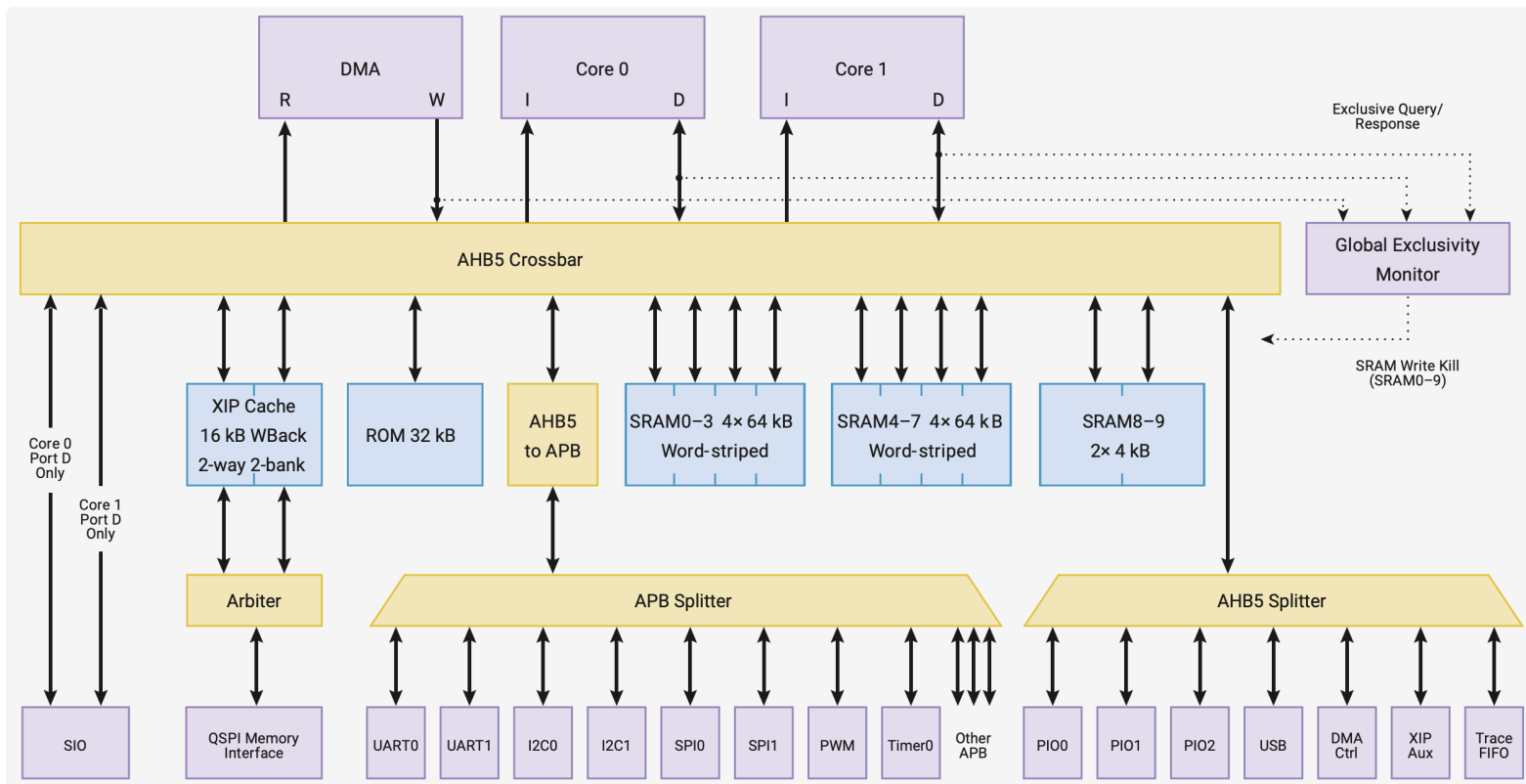
| GPIO | F0   | F1       | F2        | F3       | F4     | F5  | F6   | F7   | F8   | F9           | F10           | F11      |
|------|------|----------|-----------|----------|--------|-----|------|------|------|--------------|---------------|----------|
| 0    |      | SPI0 RX  | UART0 TX  | I2C0 SDA | PWM0 A | SIO | PI00 | PI01 | PI02 | QMI CS1n     | USB OVCUR DET |          |
| 1    |      | SPI0 CSn | UART0 RX  | I2C0 SCL | PWM0 B | SIO | PI00 | PI01 | PI02 | TRACECLK     | USB VBUS DET  |          |
| 2    |      | SPI0 SCK | UART0 CTS | I2C1 SDA | PWM1 A | SIO | PI00 | PI01 | PI02 | TRACEDATA0   | USB VBUS EN   | UART0 TX |
| 3    |      | SPI0 TX  | UART0 RTS | I2C1 SCL | PWM1 B | SIO | PI00 | PI01 | PI02 | TRACEDATA1   | USB OVCUR DET | UART0 RX |
| 4    |      | SPI0 RX  | UART1 TX  | I2C0 SDA | PWM2 A | SIO | PI00 | PI01 | PI02 | TRACEDATA2   | USB VBUS DET  |          |
| 5    |      | SPI0 CSn | UART1 RX  | I2C0 SCL | PWM2 B | SIO | PI00 | PI01 | PI02 | TRACEDATA3   | USB VBUS EN   |          |
| 6    |      | SPI0 SCK | UART1 CTS | I2C1 SDA | PWM3 A | SIO | PI00 | PI01 | PI02 |              | USB OVCUR DET | UART1 TX |
| 7    |      | SPI0 TX  | UART1 RTS | I2C1 SCL | PWM3 B | SIO | PI00 | PI01 | PI02 |              | USB VBUS DET  | UART1 RX |
| 8    |      | SPI1 RX  | UART1 TX  | I2C0 SDA | PWM4 A | SIO | PI00 | PI01 | PI02 | QMI CS1n     | USB VBUS EN   |          |
| 9    |      | SPI1 CSn | UART1 RX  | I2C0 SCL | PWM4 B | SIO | PI00 | PI01 | PI02 |              | USB OVCUR DET |          |
| 10   |      | SPI1 SCK | UART1 CTS | I2C1 SDA | PWM5 A | SIO | PI00 | PI01 | PI02 |              | USB VBUS DET  | UART1 TX |
| 11   |      | SPI1 TX  | UART1 RTS | I2C1 SCL | PWM5 B | SIO | PI00 | PI01 | PI02 |              | USB VBUS EN   | UART1 RX |
| 12   | HSTX | SPI1 RX  | UART0 TX  | I2C0 SDA | PWM6 A | SIO | PI00 | PI01 | PI02 | CLOCK GPIN0  | USB OVCUR DET |          |
| 13   | HSTX | SPI1 CSn | UART0 RX  | I2C0 SCL | PWM6 B | SIO | PI00 | PI01 | PI02 | CLOCK GPOUT0 | USB VBUS DET  |          |
| 14   | HSTX | SPI1 SCK | UART0 CTS | I2C1 SDA | PWM7 A | SIO | PI00 | PI01 | PI02 | CLOCK GPIN1  | USB VBUS EN   | UART0 TX |
| 15   | HSTX | SPI1 TX  | UART0 RTS | I2C1 SCL | PWM7 B | SIO | PI00 | PI01 | PI02 | CLOCK GPOUT1 | USB OVCUR DET | UART0 RX |
| 16   | HSTX | SPI0 RX  | UART0 TX  | I2C0 SDA | PWM0 A | SIO | PI00 | PI01 | PI02 |              | USB VBUS DET  |          |
| 17   | HSTX | SPI0 CSn | UART0 RX  | I2C0 SCL | PWM0 B | SIO | PI00 | PI01 | PI02 |              | USB VBUS EN   |          |
| 18   | HSTX | SPI0 SCK | UART0 CTS | I2C1 SDA | PWM1 A | SIO | PI00 | PI01 | PI02 |              | USB OVCUR DET | UART0 TX |
| 19   | HSTX | SPI0 TX  | UART0 RTS | I2C1 SCL | PWM1 B | SIO | PI00 | PI01 | PI02 | QMI CS1n     | USB VBUS DET  | UART0 RX |
| 20   |      | SPI0 RX  | UART1 TX  | I2C0 SDA | PWM2 A | SIO | PI00 | PI01 | PI02 | CLOCK GPIN0  | USB VBUS EN   |          |
| 21   |      | SPI0 CSn | UART1 RX  | I2C0 SCL | PWM2 B | SIO | PI00 | PI01 | PI02 | CLOCK GPOUT0 | USB OVCUR DET |          |
| 22   |      | SPI0 SCK | UART1 CTS | I2C1 SDA | PWM3 A | SIO | PI00 | PI01 | PI02 | CLOCK GPIN1  | USB VBUS DET  | UART1 TX |

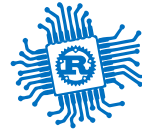




# The Bus

that interconnects the cores with the peripherals

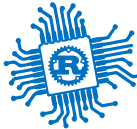




# Conclusion

we talked about

- How a processor functions
- Microcontrollers (MCU) / Microprocessors (CPU)
- Microcontroller architectures
- ARM Cortex-M
- RP2040



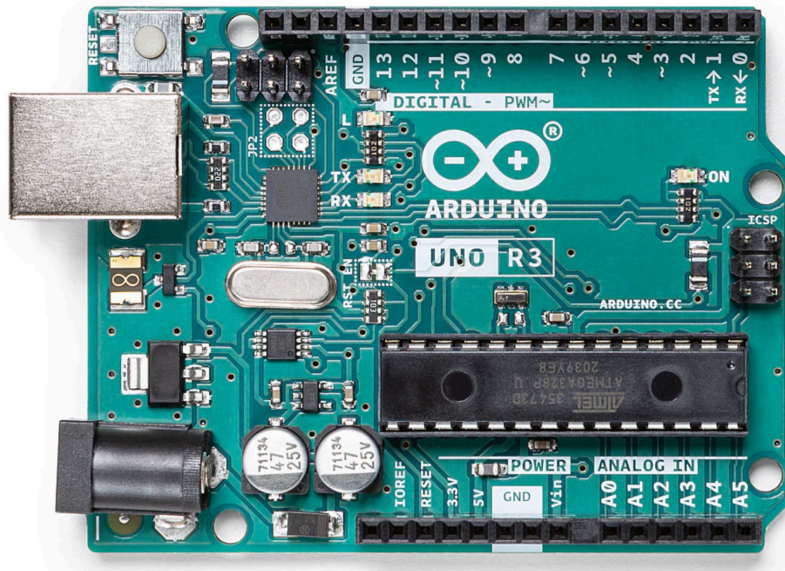
# Atmega328P

the MCU

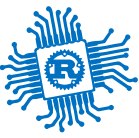
## Boards

that use 328P - many :)

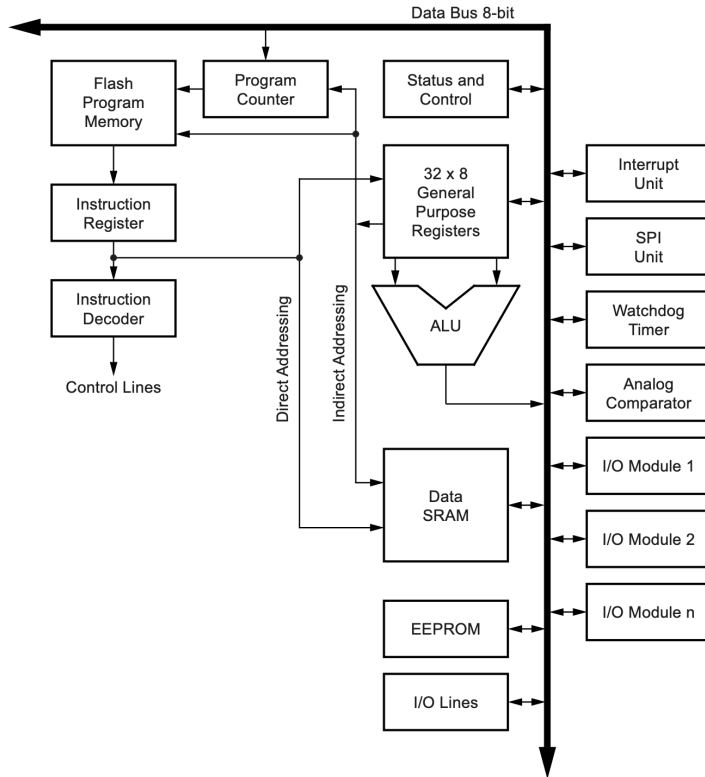
### Example: Arduino Uno



|           |                          |
|-----------|--------------------------|
| Vendor    | Arduino & others         |
| Variant   | 328p/ 328P               |
| Cores     | 1                        |
| Word      | 8 bit                    |
| Frequency | up to 16 MHz             |
| RAM       | 2 KB                     |
| Storage   | 32KB Flash & 1 KB EEPROM |



# The Chip



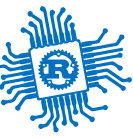
## Peripherals

|      |                                         |
|------|-----------------------------------------|
| PWM  | Pulse Width Modulation                  |
| ADC  | Analog to Digital Converter             |
| SPI  | Serial Peripheral Interface             |
| UART | Universal Async. Receiver/Transmitter   |
| RTC  | Real Time Clock                         |
| I2C  | Inter-Integrated Circuit <sup>[1]</sup> |
| PIO  | Programmable Input/Output               |

1. Actually 2-wire serial interface ↔

# Pins

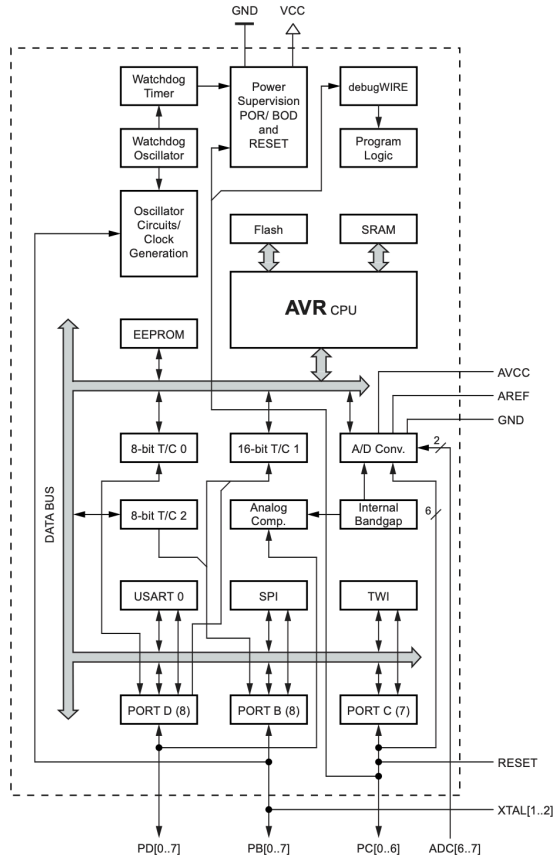
have multiple functions

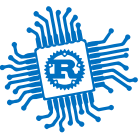




# The Bus

For more details - check-out the 328P DataSheet





# Embedded Software





# Why Embedded Software is Different

## It tends to be very application-specific

- It comes in the form of a blob, which contains data, configuration, application and drivers
- While some operating systems exist for embedded devices, they are very rare

## It uses specialized hardware to achieve its goal

- DSPs for audio/video processing
- On-chip/off-chip peripherals (ADCs/DACs for data acquisition, audio playback, capacitive touch)
- Displays, buttons for user interfaces

## It is much more tightly coupled to hardware than PC/server software

- This allows for smaller binaries but the trade-off is less portable code
- It must be designed in parallel with the hardware



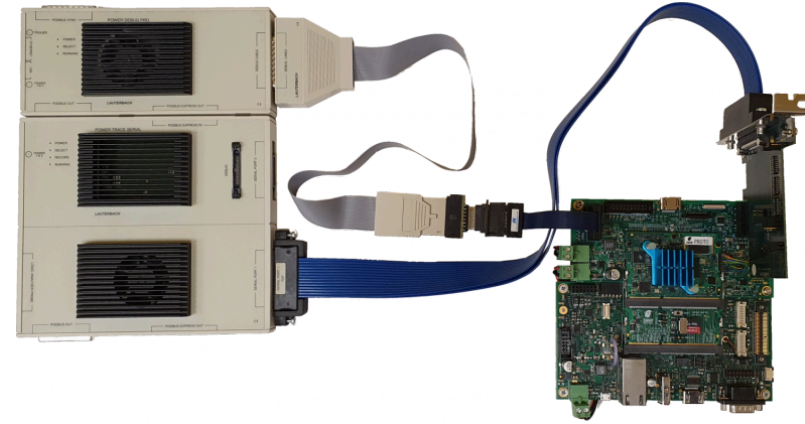
# Hardware Programming & Debugging Devices

Software tools + hardware tools:

- IDE
- compiler
- programming device/ debugger
- hardware device

Extras:

- oscilloscope
- waveform analyzer
- power analyzer



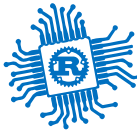
[1]

1. [https://wiki.dave.eu/index.php/MITO8M-AN-001: Advanced multicore debugging, tracing, and energy profiling with Lauterbach TRACE32](https://wiki.dave.eu/index.php/MITO8M-AN-001:_Advanced_multicore_debugging,_tracing,_and_energy_profiling_with_Lauterbach_TRACE32) ↔



# Program Flow - ARM vs AVR

| What                | ARM                                                                         | AVR                                               |
|---------------------|-----------------------------------------------------------------------------|---------------------------------------------------|
| Program Load        | Using an external programmer or bootloader                                  | (same)                                            |
| Execution launch    | When the microcontroller is reset, execution starts from a preset address   | (same)                                            |
| Execution threads   | Supports multiple threads, multiple values for the Program Counter PC (R15) | Single thread, controlled by PC (Program Counter) |
| In/ Out interaction | Memory mapped I/O                                                           | Port-mapped I/O                                   |



# The code

## How do we program a microcontroller?

1. The code is compiled and a binary file containing the machine code instructions is produced.

- .UF2 / .BIN / .HEX on ARM
- .HEX on AVR

2. The binary must end up in the microcontroller's program memory (Flash) <sup>[1]</sup>

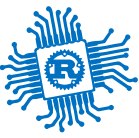
- Using an external programmer (In-System Programmer or JTAG)
- using a bootloader

The bootloader takes up space in the program memory for AVR (for RPI it resides in ROM).

3. After programming, a RESET is automatically applied to the processor, and it starts execution from the start address.

Depending on the configuration (eg where the bootloader is written), it may not be 0.

1. ARM microcontrollers are able to execute code from RAM ←



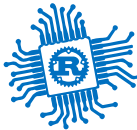
# In / Out

## No

- screen :)
- console :)

## Yes

- LEDs
- LCD
- Serial interface
- Hardware Debugger



# Variables

## Allocation

- Local variables > stack

Be careful when using recursive functions

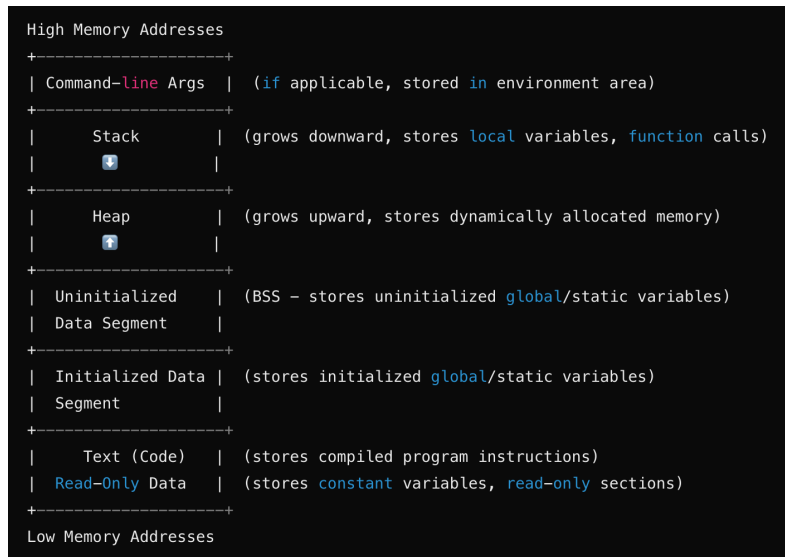
- Global variables > data

- Dynamic variables > heap

Dynamic variables require an allocator - might not be ideal on an AVR / when you are low on memory

- Const > flash memory (program memory - written at compile time)

Const on AVR can also be stored on EEPROM (slow)





# Memory on AVR - 328P example

## ATmega328P Memory Details

| Memory Type               | Size      | Purpose                                          |
|---------------------------|-----------|--------------------------------------------------|
| Flash (ROM)               | 32 KB     | Stores program instructions (non-volatile).      |
| SRAM (RAM)                | 2 KB      | Stores variables, stack, heap, and registers.    |
| EEPROM                    | 1 KB      | Stores persistent data (non-volatile, writable). |
| General Purpose Registers | 32 Bytes  | Fast-access CPU registers.                       |
| I/O Registers             | 64 Bytes  | Port-mapped peripheral control registers.        |
| Extended I/O Registers    | 160 Bytes | Memory mapped peripheral control registers.      |



# Memory on ARM - RP2350 example - M33 based

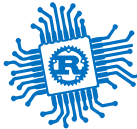
## RP2350 Memory Breakdown

| Memory Type              | Size                   | Purpose                                                 |
|--------------------------|------------------------|---------------------------------------------------------|
| XIP <sup>[1]</sup> Flash | Up to 16 MB            | Stores program code (external QSPI Flash).              |
| SRAM (On-chip)           | 520 KB                 | Stores stack, heap, variables, and data.                |
| Boot ROM                 | 32 KB                  | Stores bootloader, factory firmware.                    |
| OTP                      | 8 KB                   | One-time-programmable (Product id, cryptographic keys). |
| Peripheral Space         | Varies                 | Memory-mapped I/O for GPIO, UART, SPI, DMA.             |
| Registers                | 16 + control registers | General purpose + program flow + special purpose        |

---

1. XIP = Execute in Place (without this, the code would need to be copied in RAM first) ↩





# Let's see some code

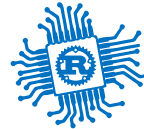
```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  void printBinary(uint32_t num) {
5      for (int i = 31; i >= 0; i--) {
6          printf("%d", (num >> i) & 1);
7          if (i % 8 == 0) printf(" ");
8      }
9      printf("\n");
10 }
11
12 int main()
13 {
14     uint8_t a;
15     uint32_t b;
16
17     a = 0x01;
18     b = a << 24;
19
20     printBinary(a);
21     printBinary(b);
22
23     return 0;
24 }
```

What is the resulting value?

it depends on the compiler and on the architecture

Solution

```
1  b = (uint32_t) a << 24;
2  //b will be 00000001 00000000 00000000 00000000
3  //same result on any architecture and compiler;
```

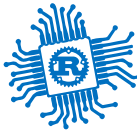


# Variables in C

```
1  #include <stdio.h>
2
3  int8_t, uint8_t
4  int16_t, uint16_t
5  int32_t, uint32_t
```

# Variables in Rust

```
1  u8, u16, u32, u64, u128
2  i8, i16, i32, i64, i128
3  usize //word size (eg - 32b for 32b processor)
4  isize //word size (eg - 32b for 32b processor)
5
6  //NOTES:
7  char // 4 bytes != u8 //UTF-8 not ASCII like in C
8  b"str" //ASCII string
9  "str" UTF-8 string
10
11 's' // char
12 b's' // u8
```



# Why Rust-lang

The tagline of Rust is No Undefined Behavior.

- no null reference; the Rust compiler explicitly asks developers to check this;
- no implicit cast, even adding a u32 to a u8 must be casted;
- safe access to shared data across threads verified at compile time;
- uses type states to move runtime checks to compile time and force developers to check;
- clearly defined data types, unlike i8 or u128;
- safe unions, that provide a discriminant to prevent wrong interpretation of data;
- clear code organization into crates and modules;
- backward compatibility at crate level.