

Introduction into PM

Lecture 1

Welcome

to the *Proiectarea cu Microprocesoare* engineering class

You will learn

- how hardware works
- how to engineer a hardware system - while accounting for practical constraints
- how to actually build your own hardware device
- the Rust programming Language
- a little bit of low-level C

We expect you

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

2026 is an experiment - iteration V2 - 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.

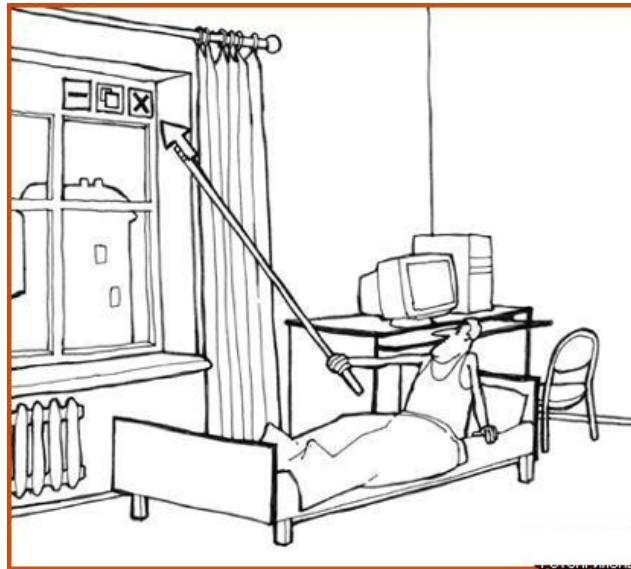
(copyright info) These slides may contain materials shared with my colleagues Alexandru Radovici, Dan Tudose, Alexandru Vaduva, Razvan Tataroiu

Engineer

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

Extra questions

What else should we account for?

- **How do we go from requirements to production?** (*process / gates*)
 - **How do we design a maintainable product?** (*modularity / versioning*)
 - **What documentation must we deliver for a hardware product?** (*HW+FW+test*)
 - **What are the classic constraints & trade-offs?** (*power / cost / perf / time / risk*)
 - **How do we make it robust?** (*watchdog / brownout / safe-state*)
 - **How do we approach security?** (*threat model / secure boot / keys*)
 - **How do we do remote updates?** (*OTA / signing / rollback*)
 - **How do we do observability & real-world debugging?** (*logs / tracing / SWD*)
 - **How do we design for test & manufacturing?** (*DFT/DFM / fixtures / calibration*)
-

2026 VS 2025

Modification	How it helps
We are dropping the Raspberry Pi Pico and moving to an ST MCU board with an integrated debugger.	Fewer hardware-related issues (the current Pico generation has had multiple hw problems). Also removes the need for 2 separate boards for the project.
New lab boards (as a shield) designed for the selected ST variant.	Cleaner, more reliable lab setup and easier reproduction of results across students.
Added protection features on the shield (general I/O protections).	Fewer board failures and fewer “mystery” faults caused by wiring mistakes or accidental shorts.
Dedicated power-supply protections (lab PCs have unstable/problematic USB power).	More stable power, fewer USB-related resets/brownouts, and reduced risk of damaging boards.

2026 VS 2025

Modification	How it helps
Each student will receive one board for the project.	Lower overall costs for students.
Lab grading becomes attendance-based: 10 points for being present and successfully flashing/uploading code to the board.	Less “grade pressure” during lab sessions and less focus on chasing points; more focus on learning and experimenting.
Publish lab solutions during the semester (for each lab, in the following week).	Students get more code examples earlier, and can reuse patterns for the project while the semester is ongoing.

>> **Solutions for all the lab works:** <https://github.com/UPB-PMRust/lab-solutions/tree/main>

Team

Our team //WIP

Andrei Zamfir

Andrei Batasev

Cristiana Precup

Daniel Rosner

Danut Aldea

Eva Cosma

George Popa

Irina Niță

Irina Bradu

Despre Daniel Rosner

Cursuri

DEEA

PM

Innovation in Tehnology

VZ & PoliFest

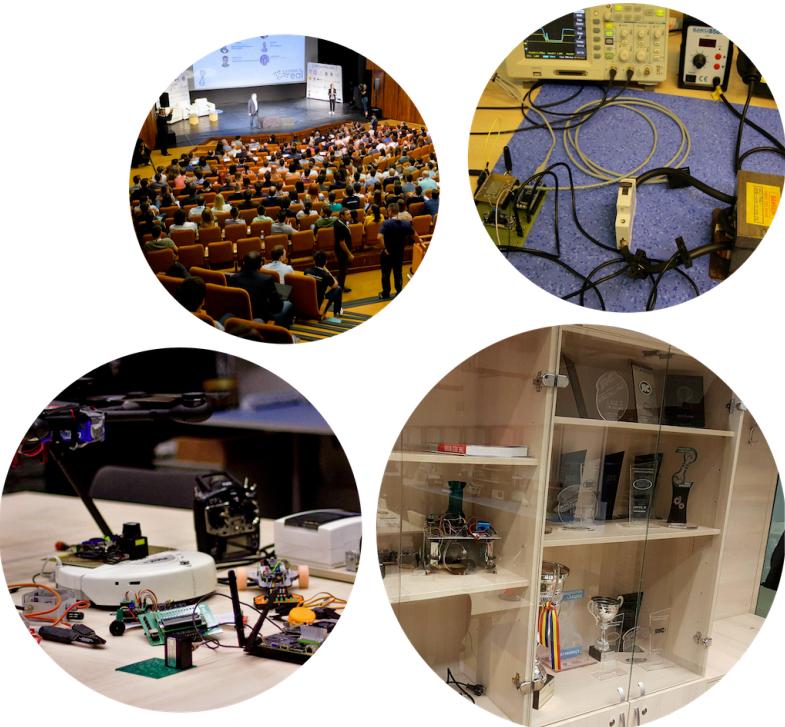
Innovation Labs &
Concursuri (tech)

Tech area

Automotive

MedTech

Energy & IoT



Outline & Rules

Outline

Lectures

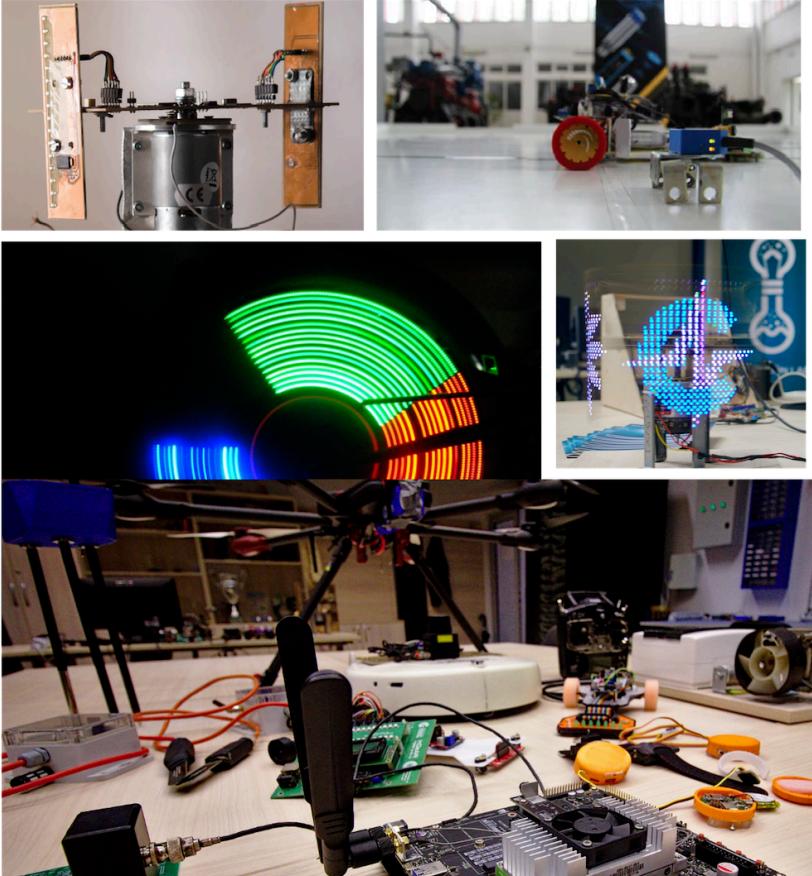
- 15 lectures + 2 Q&A lecture for the project + 2 lectures with invitees from the industry

Labs

- 7 labs + 6 project work slots + 1 test

Project

- Build a hardware device running software written in Rust on a microcontroller-based board
- The cost for the hardware is around 100 RON
- You can opt to receive a free dev board
- Presented at PM Fair during the last week of the semester



Scoring Structure

1 point for lab activity

1 point for "colocviu laborator"

3 points PROJECT

1 point - 2 quizzes @ course

4 points @ Final Exam

Note: the lab activity is for attendance; it is scaled in the end with the mark from the colocviu laborator

Bonus

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

+0.75 bonus for top 10 projects of the year

Special

10 out of 10 for complex projects "POLIfest ready" by the 26th of March

Exam

4 points out of 10

On paper

Paper-based open book - you can print whatever you want - no phones/laptop/ tablet / smart watches allowed

50% - multiple choice answers

50% - an engineering problem (design a system that does X in Y scenario with Z constraints) with subpoints that might include:

- hardware block diagram
- argument design choices
- software block diagram
- calculate certain parameters

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

Project details

Available dev-boards (that you can borrow from us for the project):

- Nucleo-U545RE-Q
- Raspberry Pi Pico-series microcontrollers documentation //not recommended

Examples - good projects 2025

PoliCube

Money counting machine with coin and bill storage

Bluetooth-Controlled Car

RFID-Controlled Corporate Elevator System

Special projects (for 10/10)

Requirements for projects for 10/ 10 without exam)

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



Professional Networking



Presentations skills



Build your own start-up with a super support structure



500.000 EURO Investment Prize



Summer Internship @ your own start-up



21 - 22 March - the largest, coolest, most fun Hackathon in Romania (>110 teams)

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 :))

Why Embedded &
Where it gets us

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

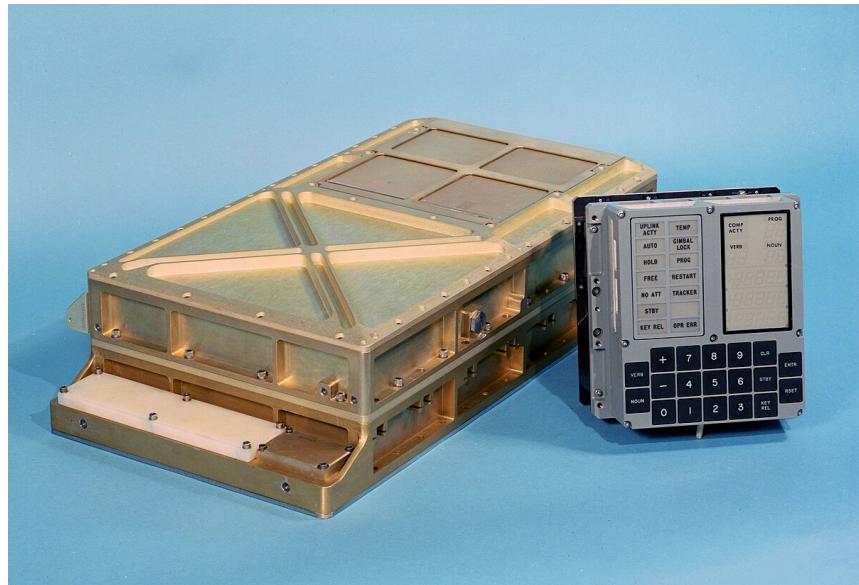
Frequency 2.048 MHz

Word Length 15 + 1 bit

RAM 4096 B

Storage 72 KB

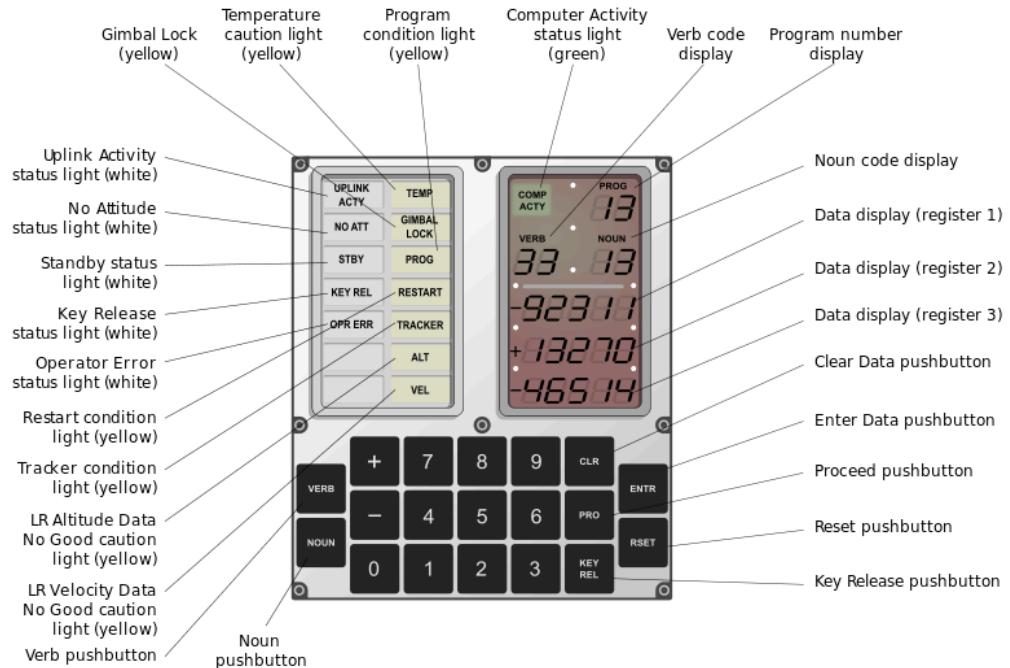
Software API AGC Assembly Language



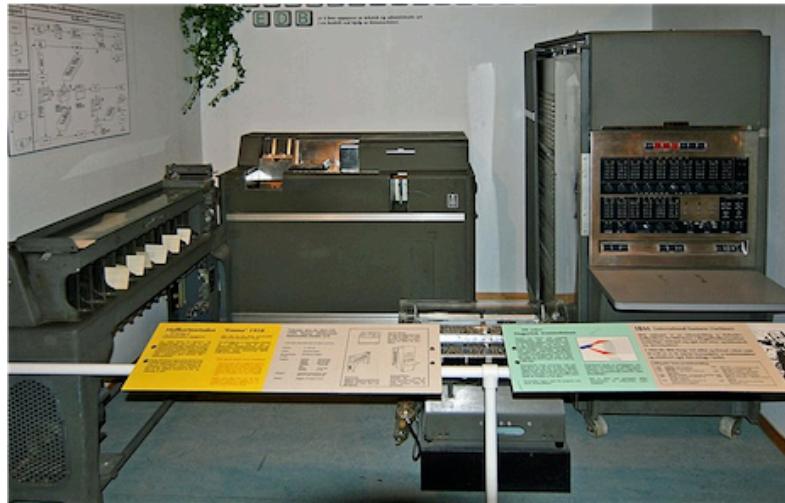
This landed the *moon eagle*.

DSKY

Display and keyboards



Where we are now



Embedded Systems

In general, they have a dedicated function.

Common constraints:

Real-time requirements

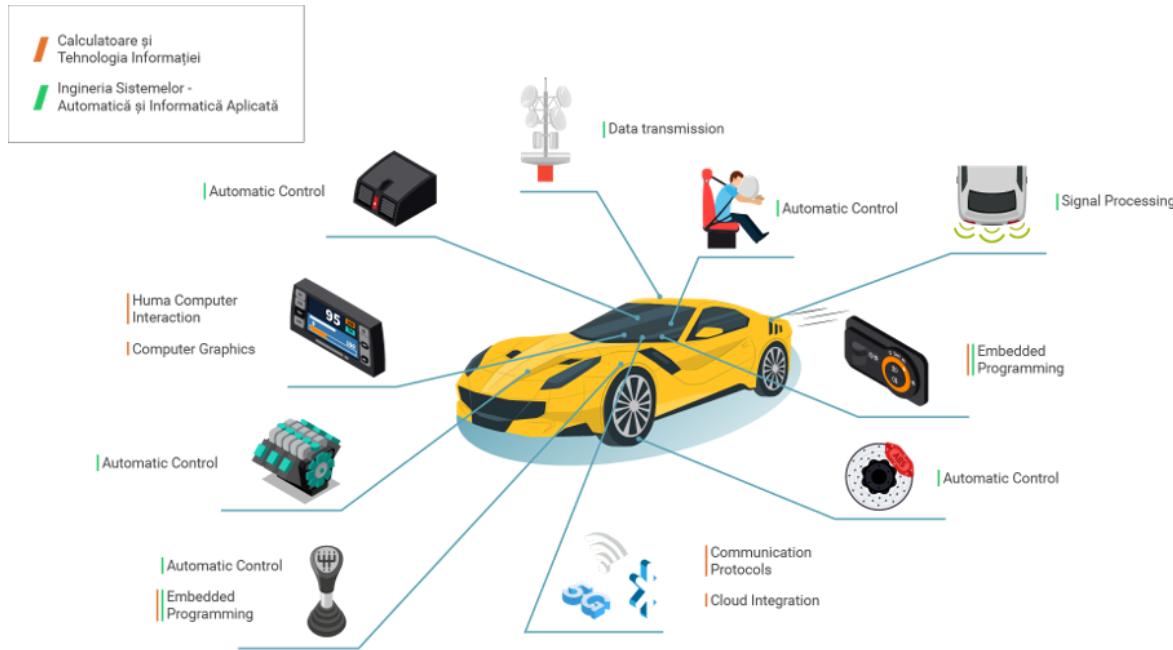
Fixed response time:

- 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

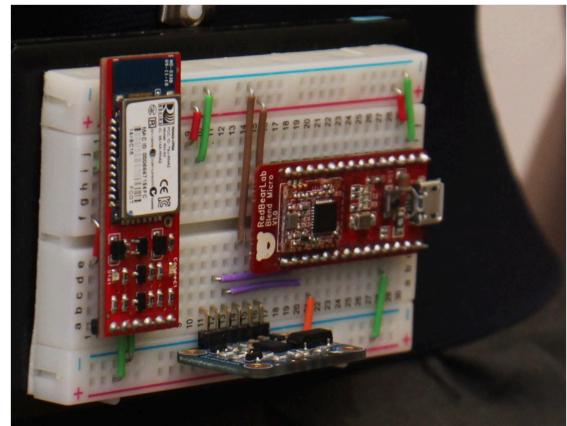
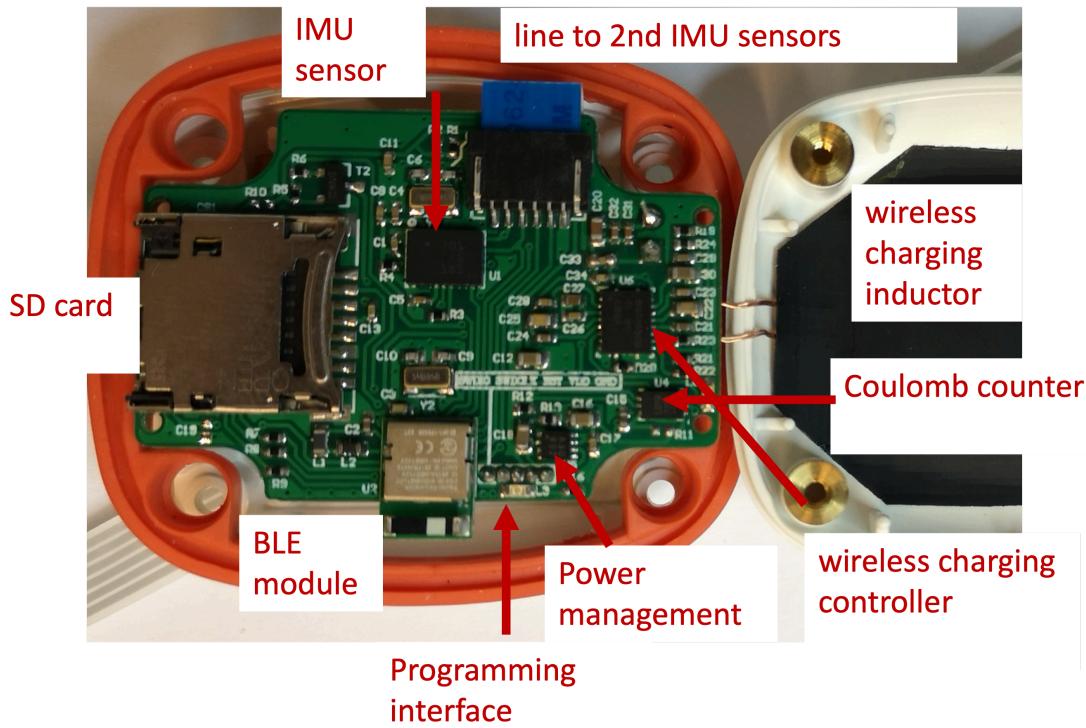


Example controller

NXP S32ZE

STM32H

Example ENTy



Example Companies (Romania) - HW + Embedded

Semiconductors

NXP

Infineon

Microchip

Synopsys

Analog Devices

Automotive Tier-1

Continental

Bosch

FORVIA HELLA

HARMAN

Industrial / Energy /
Automation

Emerson

Siemens

Honeywell

Networking / Test &
Measurement

Nokia

Ericsson

Keysight

Aerospace

Thales

Legend: Bucharest • Cluj, Timisoara, Sibiu

Btw...

#	Company	Market cap	What it does	#	Company	Market cap	What it does
1	NVIDIA	\$4.450 T	AI GPUs and accelerator systems	6	TSMC	\$1.900 T	Foundry manufacturing for advanced chips
2	Apple	\$3.759 T	Consumer devices with custom silicon	7	Saudi Aramco	\$1.643 T	Oil production and energy refining
3	Alphabet (Google)	\$3.701 T	Search, ads, YouTube, cloud	8	Meta Platforms	\$1.618 T	Social networks and digital advertising
4	Microsoft	\$2.982 T	Software, cloud services, enterprise AI	9	Tesla	\$1.566 T	EVs, batteries, onboard compute
5	Amazon	\$2.133 T	E-commerce and AWS cloud platform	10	Broadcom	\$1.541 T	Networking chips for data centers

What is a microprocessor?

Microcontroller (MCU)

Integrated in embedded systems for certain tasks

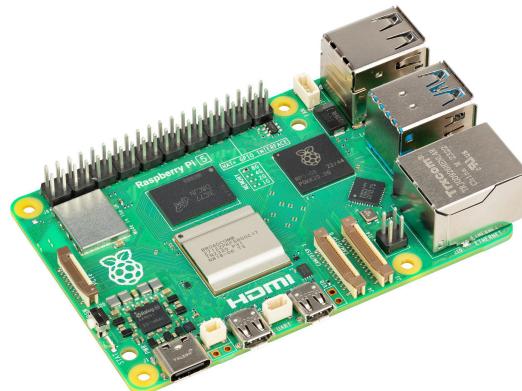
- low operating frequency (MHz)
- a lot of I/O ports
- controls hardware
- does not require an Operating System
- costs \$0.1 - \$25
- annual demand is billions



Microprocessor (CPU)

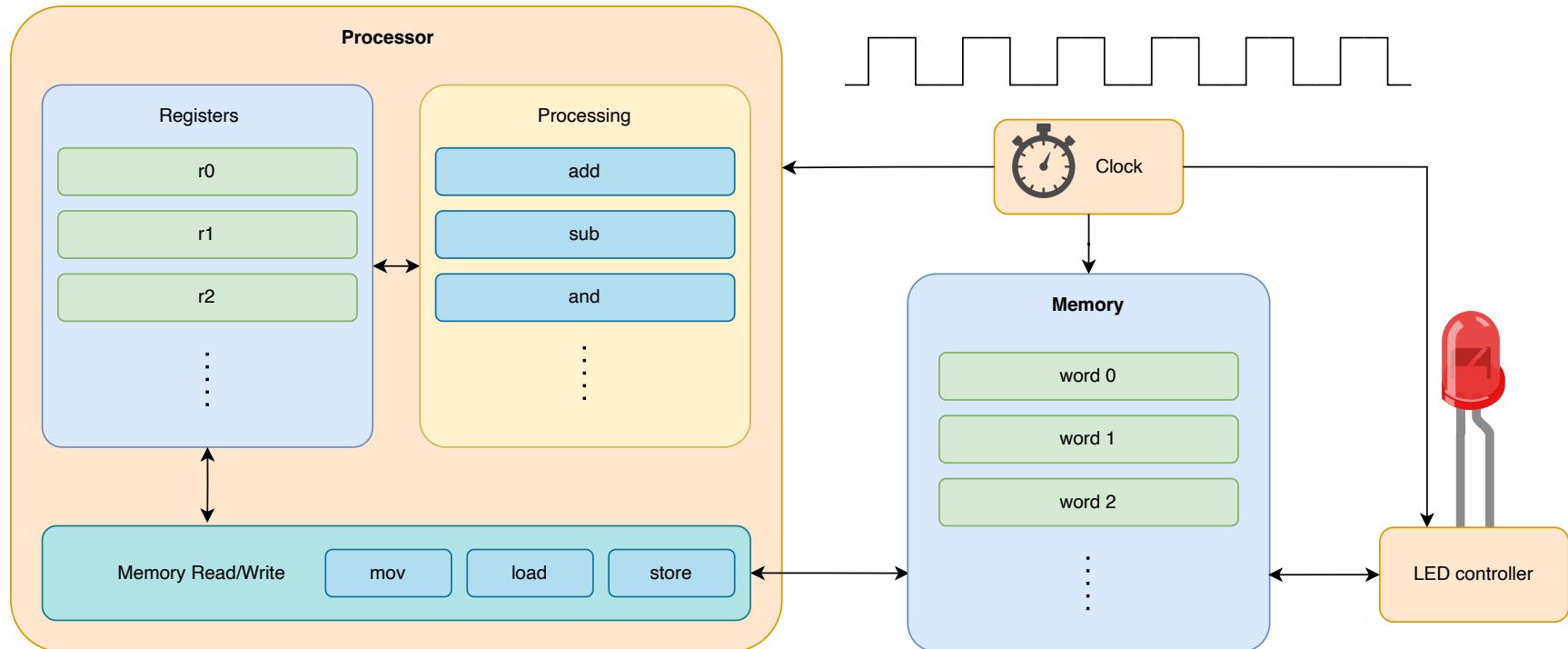
General purpose, for PC & workstations

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



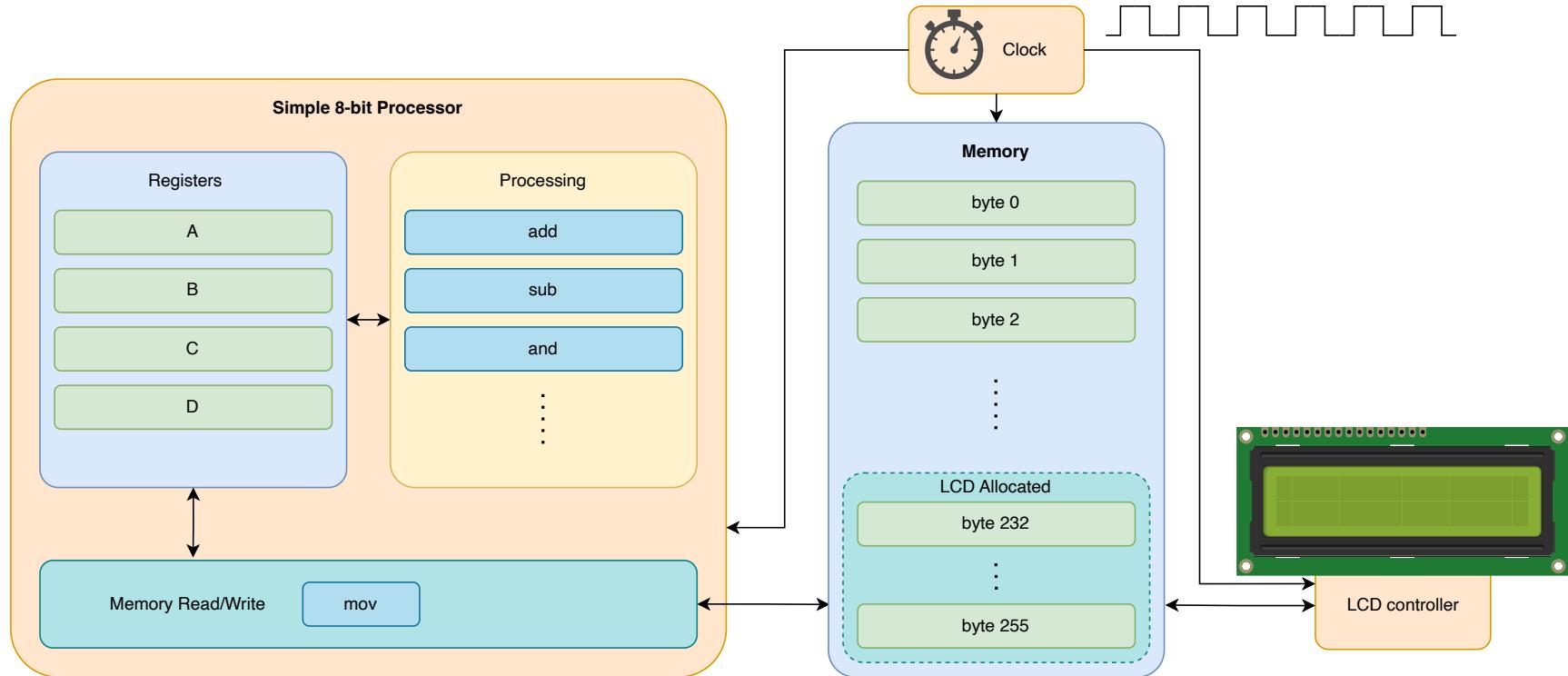
How a microprocessor works

This is a simple processor



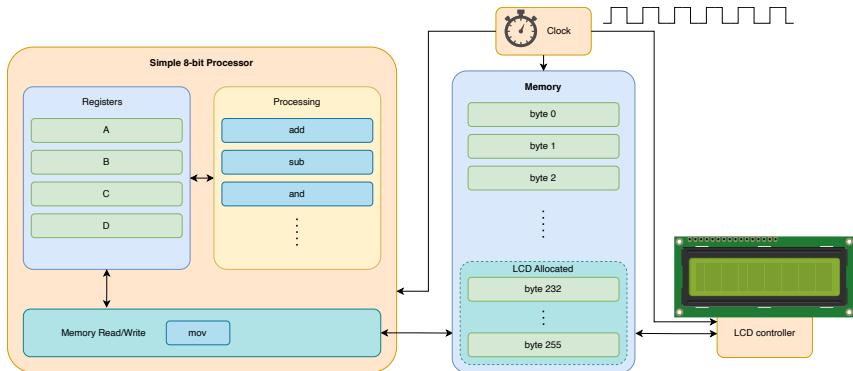
8 bit processor

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

Microprocesors VS Microcontrollers

Microcontroller

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

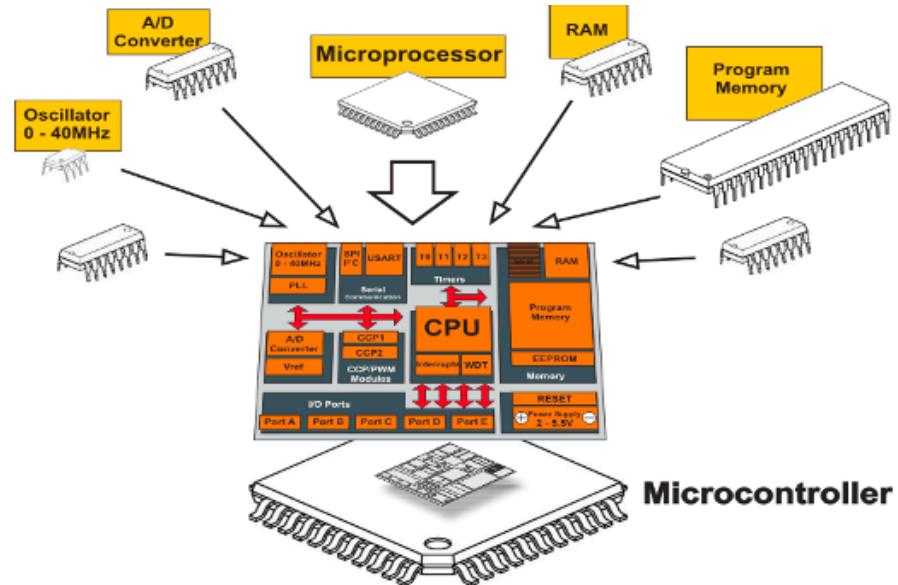
Microprocessor

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

Comparation

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



© <https://www.electronicsforu.com/resources/difference-between-microprocessor-and-microcontroller>

Note: why a motherboard



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

Note1: ARM is von Neumann with some *

Note2: GPUs (NVIDIA) are mixed architecture

Note: microcontrollers - general observations

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

Processor

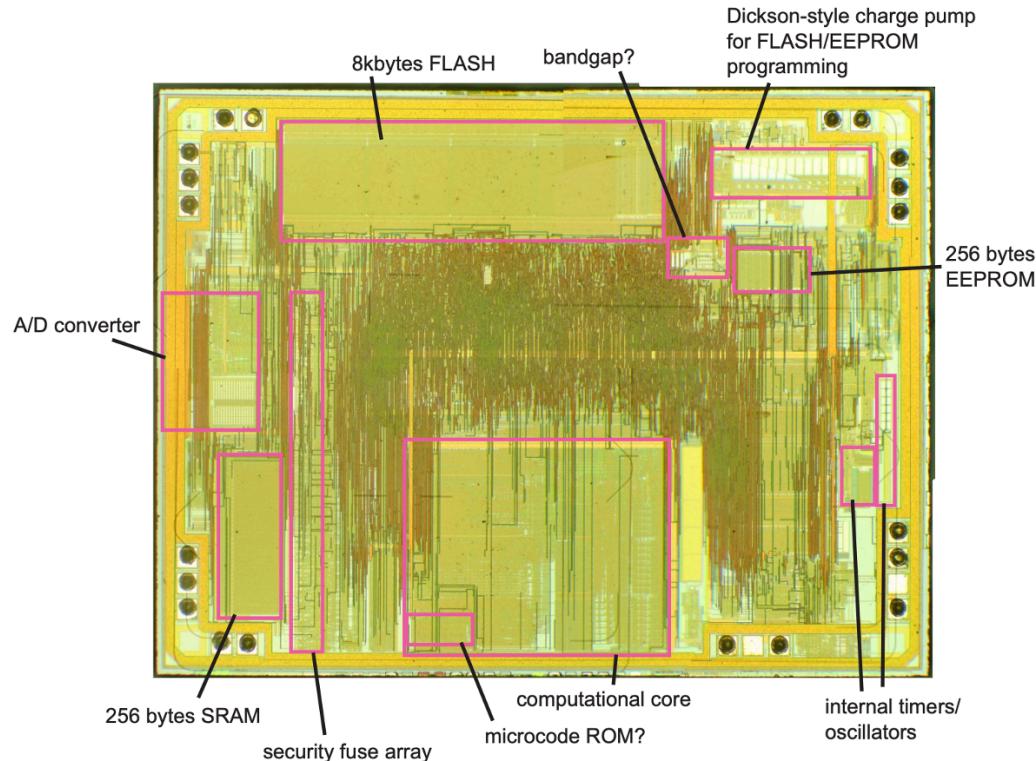
Data memory

Program memory

Peripherals

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

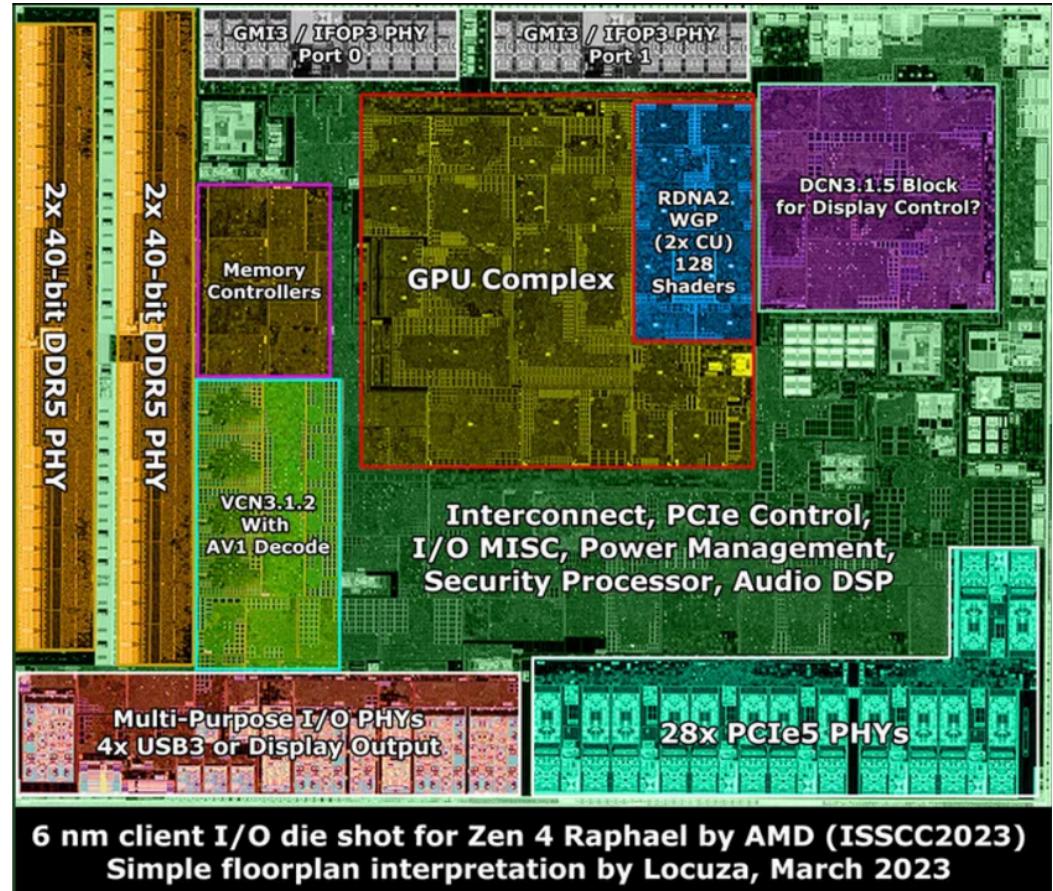
Under the microscope



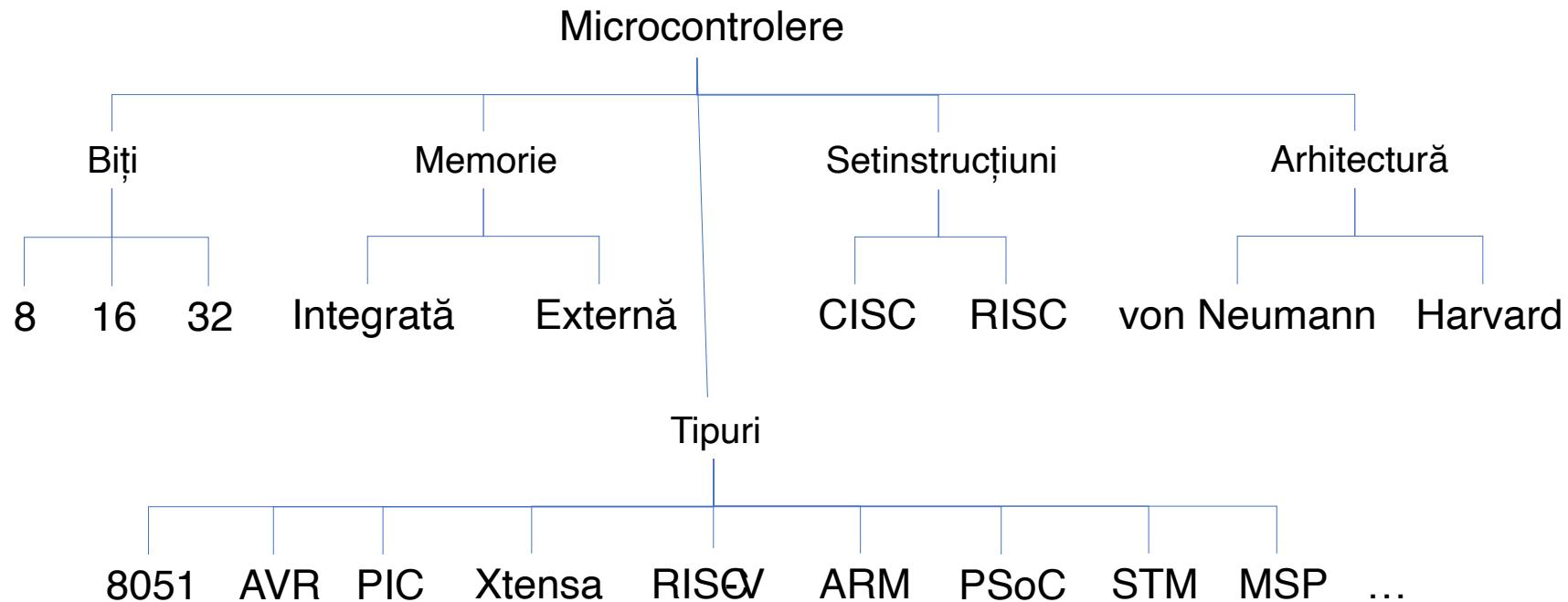
(extra)

©

<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*

Intel

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 (*Atmel*)

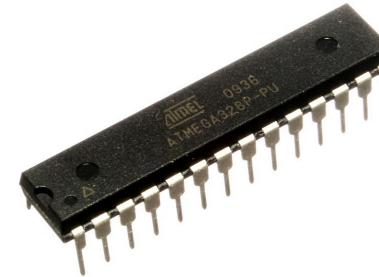
ISA AVR

Word 8 bit

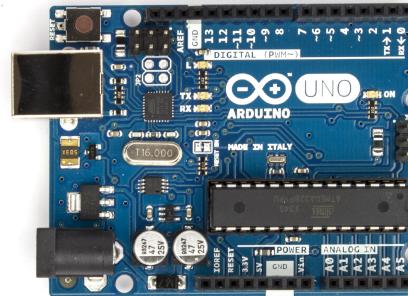
Frequency 1 - 20 MHz

Storage 4 - 256 KB

Variants *ATmega, ATTiny*



Board



PIC

Peripheral Interface Controller / Programmable Intelligent Computer

Vendor Microchip

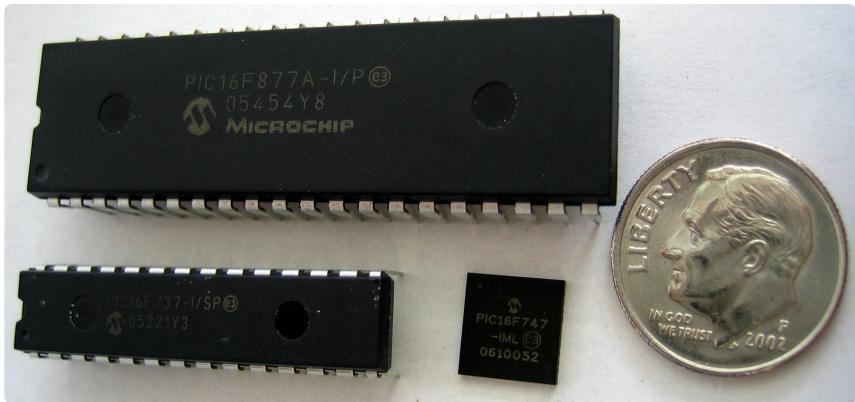
ISA PIC

Word 8 - 32

Frequency 1 - 20 MHz

Storage 256 B - 64 KB

Variants *PIC10, PIC12, PIC16, PIC18, PIC24,
PIC32*



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



Vendor Qualcomm, NXP, Nordic
 Semiconductor, Broadcom, Raspberry Pi

ISA ARMv6-M (Thumb and some Thumb-2)
 ARMv7-M (Thumb and Thumb-2)
 ARMv8-M (Thumb and Thumb-2)

Word 32

Frequency 1 - 900 MHz

Storage up to a few MB

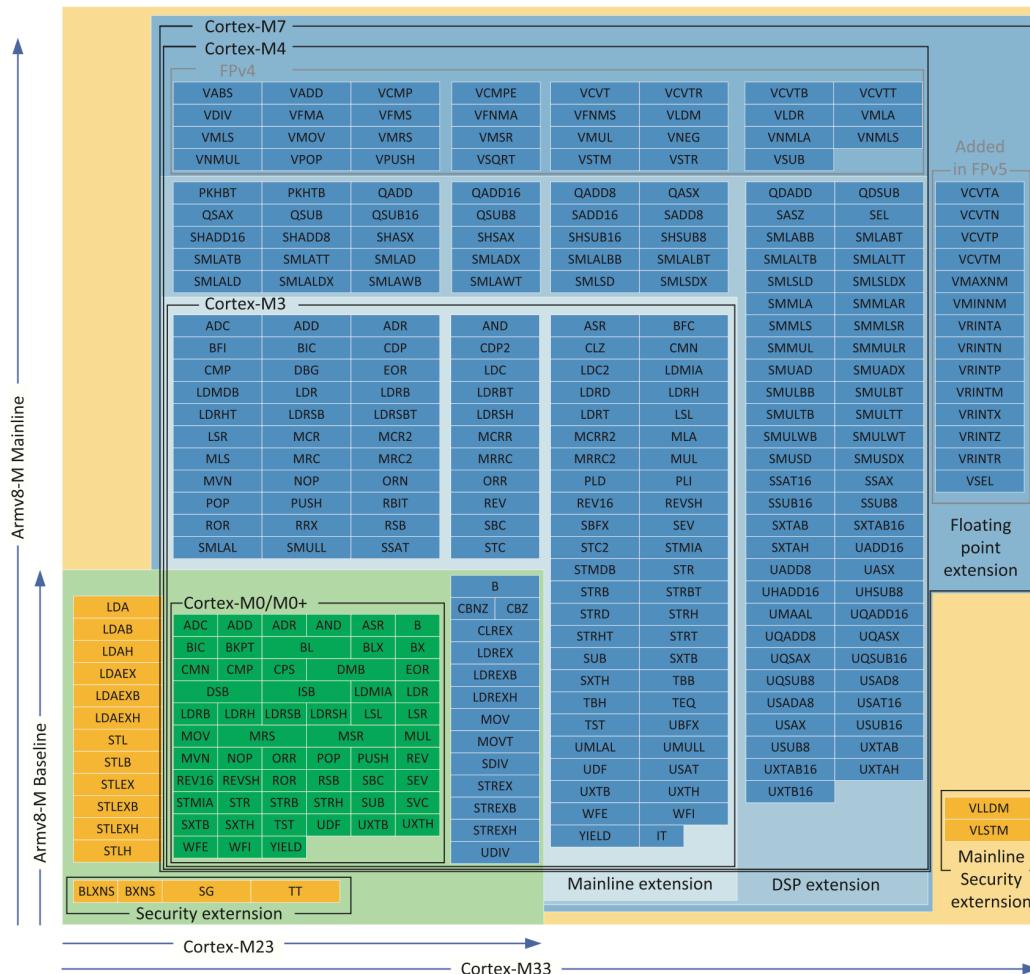
Variants *M0, M0+, M3, M4, M7, M23, M33*

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 *rv32imc, rv32iamc*



STM32U545RE

ARM Cortex-M33, built by STMicroelectronics

Bibliography

for this section

STMicroelectronics, STM32U5 Reference Manual

- Chapter 2 - *Memory and bus architecture*
 - Section 2.1 - *System architecture*

STMicroelectronics, STM32U5 Datasheet

- Chapter 2 - *"Description"*
- Chapter 4 - *"Pinout, pin description, and alternate function"*

STM32U545RE

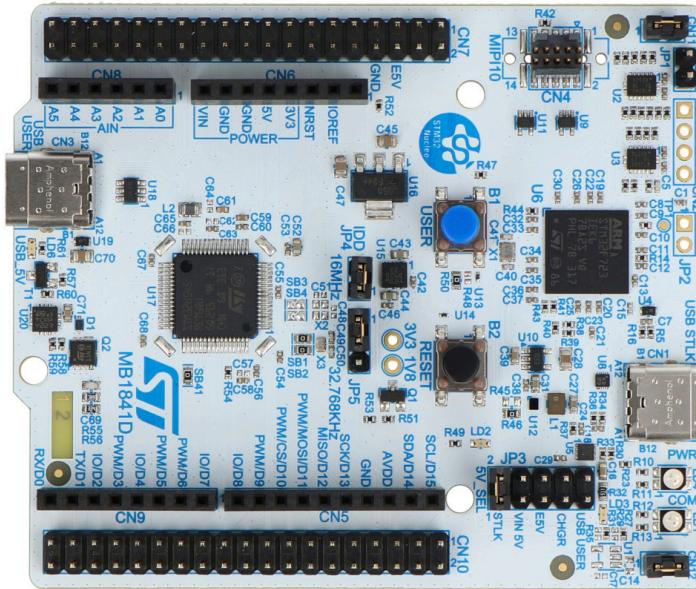
the MCU

Board

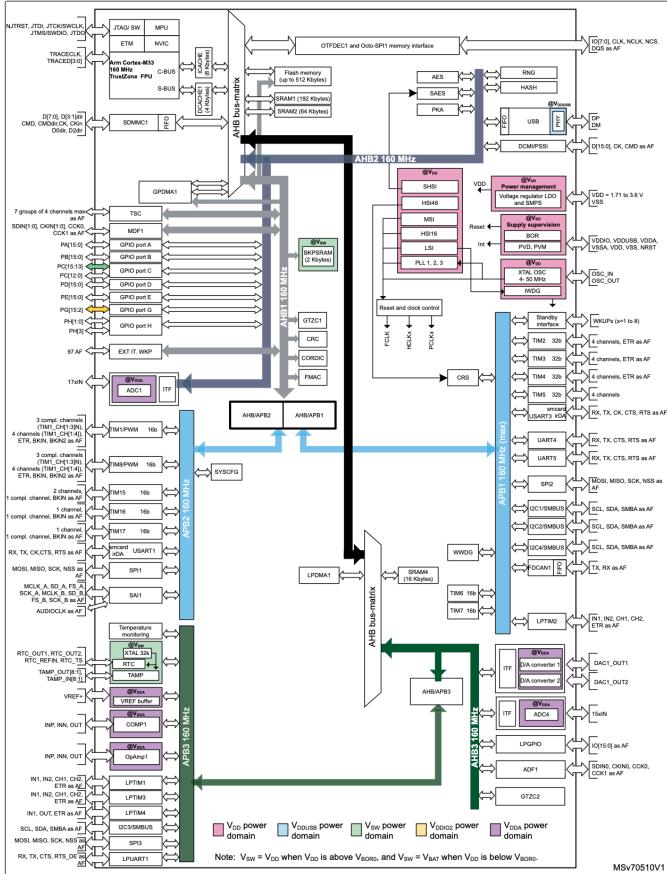
that use STM32U545RE

Nucleo U545RE-Q

Vendor	STMicroelectronics
Variant	ARM Cortex-M33
ISA	ARMv8-M
Cores	1
Word	32 bit
Frequency	up to 160 MHz
RAM	272 KB



The Chip



Peripherals

Parallel interface

**FSMC 8-/16-bit
(TFT-LCD, SRAM,
NOR, NAND)**

Timers

**19 timers including:
2 x 16-bit advanced
motor control timers
4 x ULP timers
5 x 16-bit timers
4 x 32-bit timers**

I/Os

Touch-sensing controller Camera Interface

Arm® Cortex®-M33 CPU

160 MHz
TrustZone®
FPU
MPU
ETM

LPDMA

ART Accelerator™

CORDIC
FMAC

Connectivity

USB Host/Device
1 x SD/SDIO/MMC, 3 x SPI,
4 x I²C, CAN FD,
1 x Octo-SPI,
4 x USART + 1 x LPUART

Digital

AES (256-bit),
SHA-1, SHA-256,
TRNG, PKA, 1 x SAI,
1 x MDF, 2 x ADF

Analog

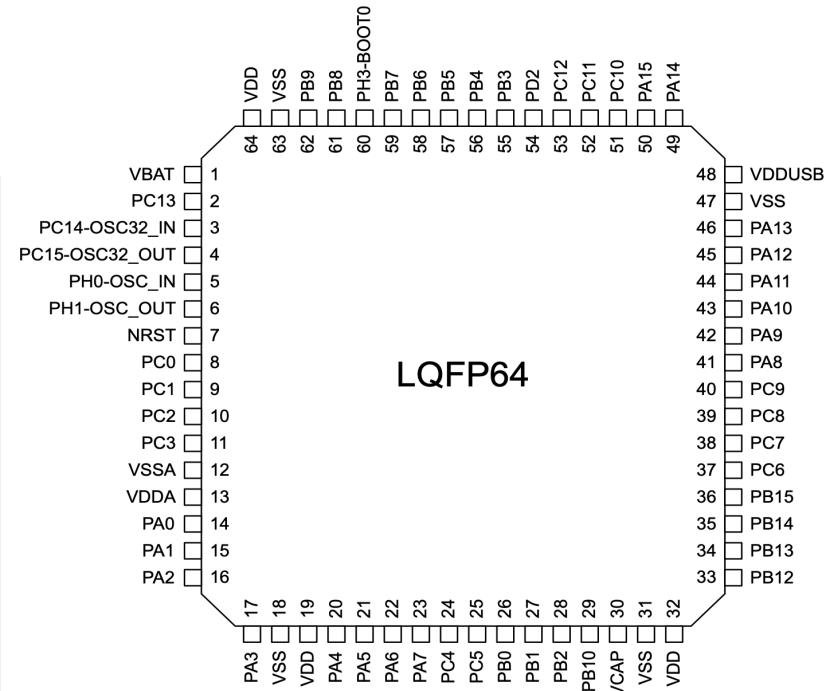
**1x 14-bit ADC 2 MSPS,
1x 12-bit ADC 2 MSPS,
2 x DAC, 2 x comparators,
1 x op amps
1 x temperature sensor**

Datasheet STM32U545RE

Pins

have multiple functions

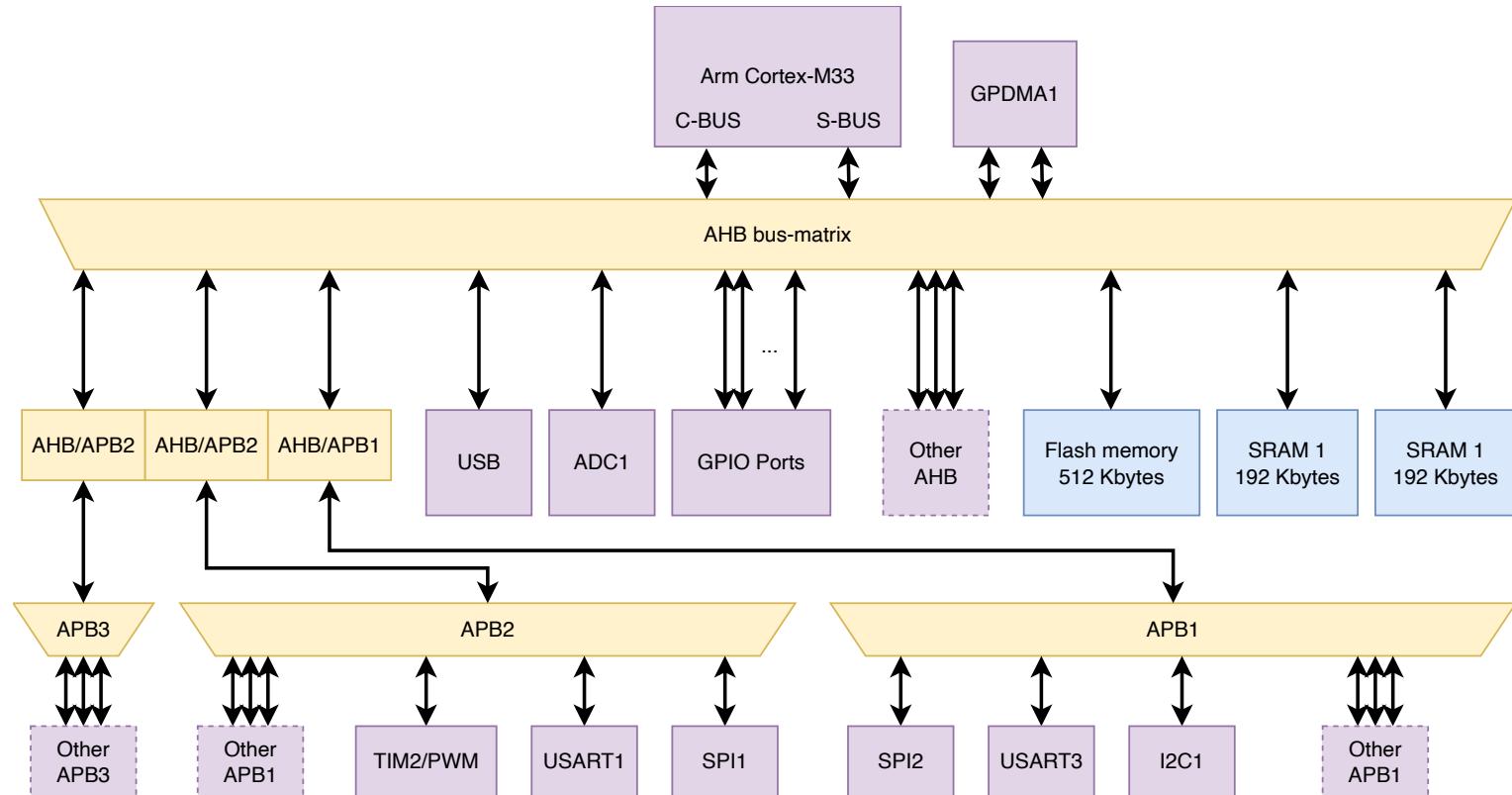
GPIO	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
0		SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1	PIO2	QMI CS1n	USB OVCUR DET	
1		SPI0 CSn	UART0 RX	I2C0 SCL	PWM0 B	SIO	PIO0	PIO1	PIO2	TRACECLK	USB VBUS DET	
2		SPI0 SCK	UART0 CTS	I2C1 SDA	PWM1 A	SIO	PIO0	PIO1	PIO2	TRACEDATA0	USB VBUS EN	UART0 TX
3		SPI0 TX	UART0 RTS	I2C1 SCL	PWM1 B	SIO	PIO0	PIO1	PIO2	TRACEDATA1	USB OVCUR DET	UART0 RX
4		SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1	PIO2	TRACEDATA2	USB VBUS DET	
5		SPI0 CSn	UART1 RX	I2C0 SCL	PWM2 B	SIO	PIO0	PIO1	PIO2	TRACEDATA3	USB VBUS EN	
6		SPI0 SCK	UART1 CTS	I2C1 SDA	PWM3 A	SIO	PIO0	PIO1	PIO2	USB OVCUR DET	UART1 TX	
7		SPI0 TX	UART1 RTS	I2C1 SCL	PWM3 B	SIO	PIO0	PIO1	PIO2		USB VBUS DET	UART1 RX
8		SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1	PIO2	QMI CS1n	USB VBUS EN	
9		SPI1 CSn	UART1 RX	I2C0 SCL	PWM4 B	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	
10		SPI1 SCK	UART1 CTS	I2C1 SDA	PWM5 A	SIO	PIO0	PIO1	PIO2		USB VBUS DET	UART1 TX
11		SPI1 TX	UART1 RTS	I2C1 SCL	PWM5 B	SIO	PIO0	PIO1	PIO2		USB VBUS EN	UART1 RX
12	HSTX	SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIN0	USB OVCUR DET	
13	HSTX	SPI1 CSn	UART0 RX	I2C0 SCL	PWM6 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT0	USB VBUS DET	
14	HSTX	SPI1 SCK	UART0 CTS	I2C1 SDA	PWM7 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIN1	USB VBUS EN	UART0 TX
15	HSTX	SPI1 TX	UART0 RTS	I2C1 SCL	PWM7 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT1	USB OVCUR DET	UART0 RX
16	HSTX	SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1	PIO2		USB VBUS DET	
17	HSTX	SPI0 CSn	UART0 RX	I2C0 SCL	PWM0 B	SIO	PIO0	PIO1	PIO2		USB VBUS EN	
18	HSTX	SPI0 SCK	UART0 CTS	I2C1 SDA	PWM1 A	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	UART0 TX
19	HSTX	SPI0 TX	UART0 RTS	I2C1 SCL	PWM1 B	SIO	PIO0	PIO1	PIO2	QMI CS1n	USB VBUS DET	UART0 RX
20		SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIN0	USB VBUS EN	
21		SPI0 CSn	UART1 RX	I2C0 SCL	PWM2 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT0	USB OVCUR DET	
22		SPI0 SCK	UART1 CTS	I2C1 SDA	PWM3 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIN1	USB VBUS DET	UART1 TX



• •

The Bus

that interconnects the core with the peripherals



Conclusion

we talked about

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

Atmega328P

the MCU

Vendor Arduino & others

Variant 328p/ 328P

Cores 1

Word 8 bit

Frequency up to 16 MHz

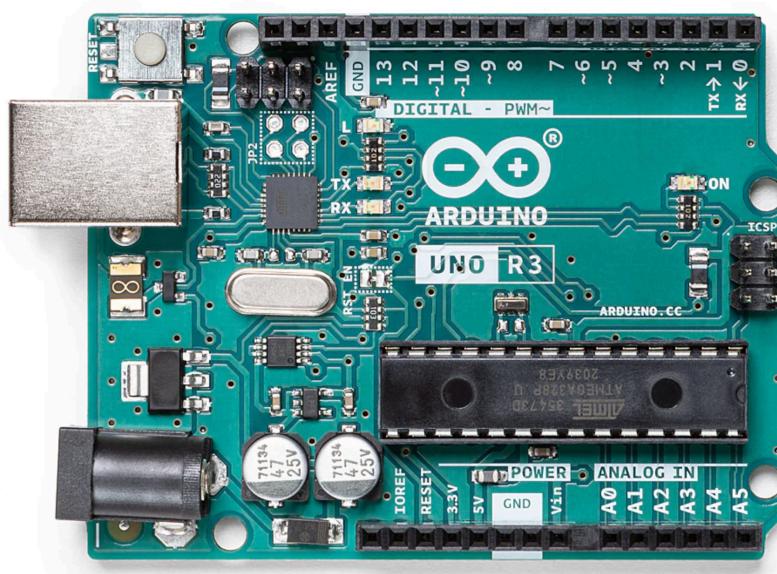
RAM 2 KB

Storage 32KB Flash & 1 KB EEPROM

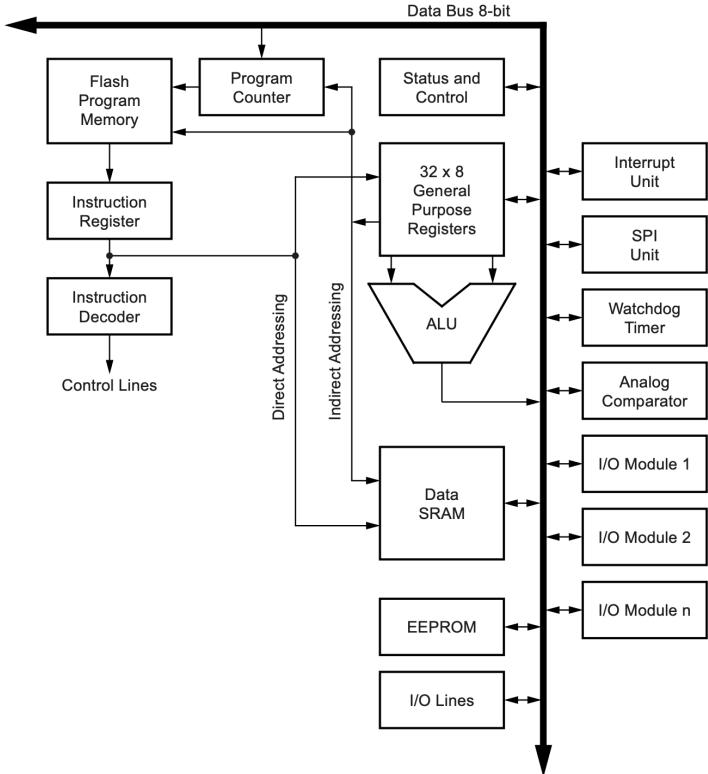
Boards

that use 328P - many :)

Example: Arduino Uno



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

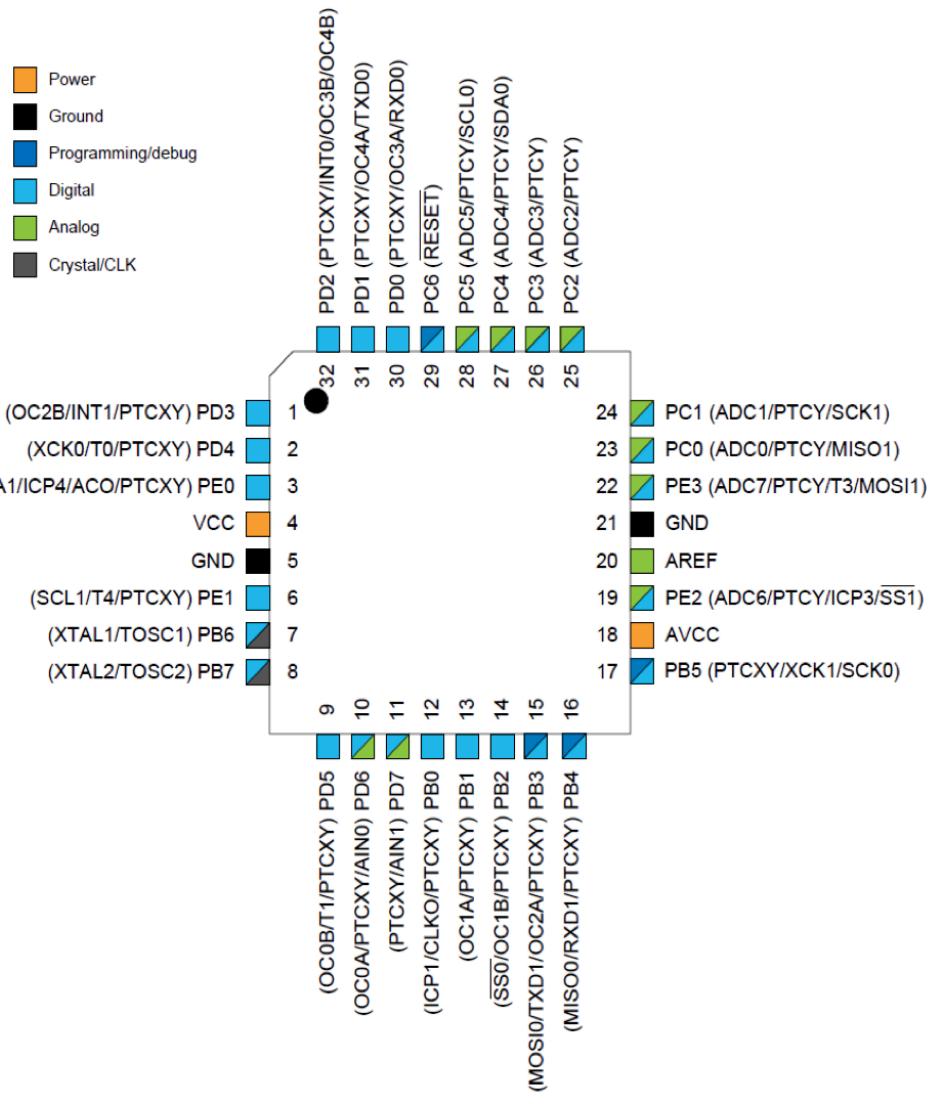
I2C Inter-Integrated Circuit [1]

PIO Programmable Input/Output

1. Actually 2-wire serial interface ↵

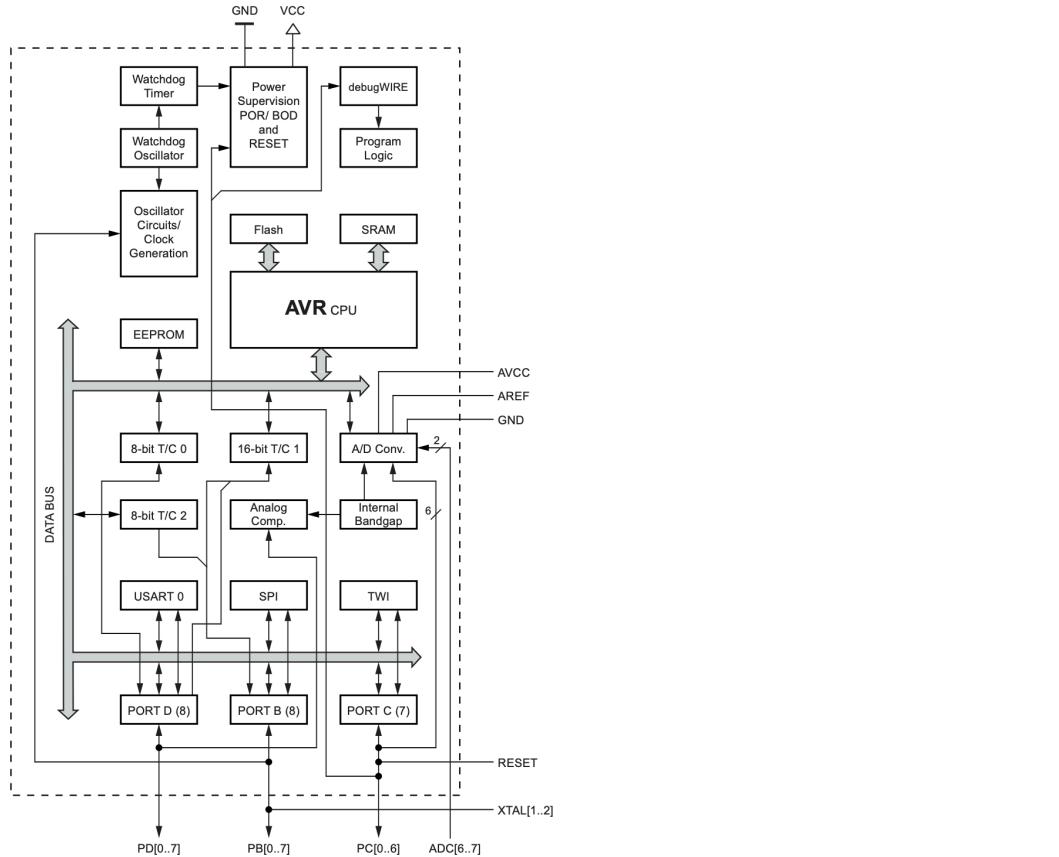
Pins

have multiple functions



The Bus

For more details - check-out the [328P DataSheet](#)



Embedded Software

From bare metal to Linux

Controller	RAM	Flash / storage	Runs	Example OS / framework
ATmega324	2 KB	32 KB Flash	Bare metal	Superloop + ISR
ATmega128	4 KB	128 KB Flash	Bare metal	Superloop + timer tick
Microchip SAMD21	32 KB	256 KB Flash	Bare metal, RTOS	FreeRTOS
STM32F103C8	20 KB	64 KB Flash	Bare metal, RTOS	FreeRTOS
Nordic nRF52832	64 KB	512 KB Flash	Bare metal, RTOS	FreeRTOS + BLE stack
ESP32-C3	400 KB	4 MB	RTOS	FreeRTOS (ESP-IDF)
RP2350	520 KB	4 MB	BM, RTOS, async	Embassy, FreeRTOS
STM32U545RE	274k	2 MB Flash	Bare metal, RTOS, async	Embassy, FreeRTOS, Zephyr
STM32MP157	512 MB	eMMC / SD / NAND	Full OS	Linux (minimal/rootfs)
Raspberry Pi	1–8 GB	SD / eMMC / SSD	Full OS	Linux (Raspberry Pi OS, Ubuntu)

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

Why Bare Metal / Async

Bare metal / async is often the better default

- You control the whole exec model (superloop/interrupts or async tasks) => predictable latency and low jitter
- Lower RAM, flash, & power, because there's no kernel, scheduler overhead, or background services

An OS adds overhead and complexity you must “pay for”

- Context switches, tick interrupts, stacks per task, drivers, IPC => CPU/RAM cost and extra failure modes
- More moving parts (scheduler, priorities, locks) => deadlocks, priority inversion, starvation, timing surprises

Verification, robustness, and debugging are harder with an OS

- Worst-case timing and system state become harder to reason about (more interleavings, shared state, nondeterminism)
- When it fails, it can fail “system-wide” (scheduler stuck, heap fragmentation, task crash) and root cause is harder to reproduce

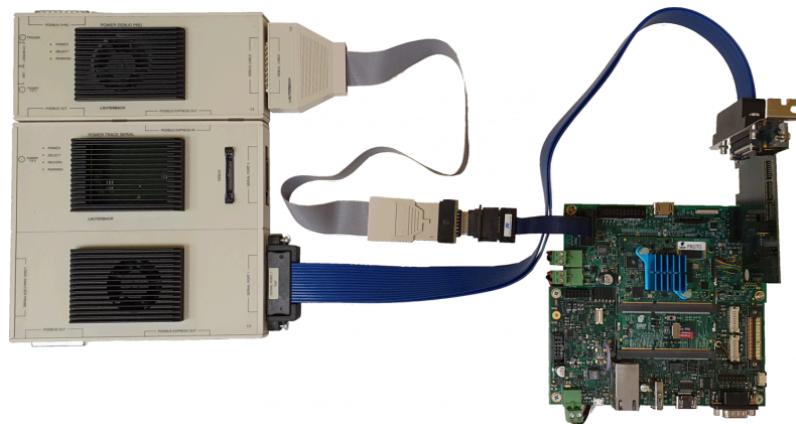
Hardware Programming & Debugging Devices

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/MIT08M-AN-001:_Advanced_multicore_debugging,_tracing,_and_energy_profiling_with_Lauterbach_TRACE32 ↵](https://wiki.dave.eu/index.php/MIT08M-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) ^[1]
 - 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

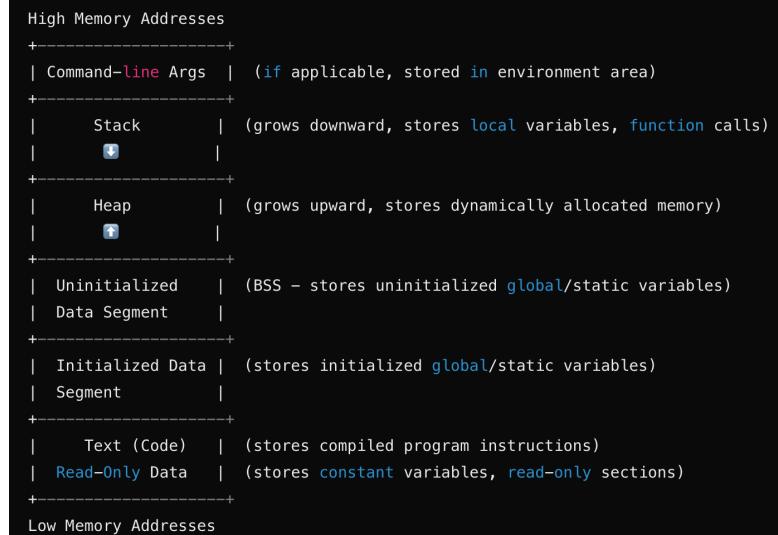
- Global variables > data

- 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 [1] 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) ↵

Memory on ARM - STM example - M33 based

STM32U545RE-Q Memory Breakdown

Memory Type	Size	Purpose
Embedded Flash	512 KB	Stores program code and non-volatile data in internal Flash.
SRAM (on-chip)	274 KB	Stores stack, heap, variables, and runtime data.
System Memory	64 KB	Factory-programmed ST bootloader (“system memory”).
OTP	512 bytes	One-time-programmable user data (e.g., IDs/keys).
Peripheral Space	Varies	Memory-mapped I/O regions for GPIO, UART, SPI, DMA, etc.
Registers	16 + control registers	General purpose + program flow + special purpose

Why Rust-lang

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 }
```

Variabiles in C

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

Variabiles 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
```

Lets see how C++ is doing

Link	Memory-safety issue	Why Rust prevents it
ZDI-24-854	Unchecked AES-key length copied into fixed stack buffer => overflow enables network-adjacent remote code execution.	Safe Rust enforces bounds checks; unchecked stack copies aren't possible.
Toyota: single bit flip	Stack overflow/memory corruption can kill RTOS task, bypass fail-safes => loss of throttle control (unintended acceleration).	Rust prevents overflows/races; hardware bit-flips and logic bugs remain possible.
CrowdStrike RCA (Channel File 291)	Template field-count mismatch caused out-of-bounds input-array read in sensor, triggering Windows system crash/BSOD.	Rust bounds-checked indexing prevents OOB reads; you get an error instead.

How do we keep C++ code functional in safety-critical applications?

Lots of tooling and processes

Coding standards / restricted subsets: MISRA C/C++, AUTOSAR C++14, SEI CERT C/C++, JSF AV C++ (plus documented deviations/waivers).

Static analysis & compliance checking: rule checkers + dataflow analyzers (e.g., Polyspace, Coverity/CodeSonar/Parasoft, clang-tidy) integrated in CI.

Compiler hardening & build gates: warnings-as-errors, stack protection, fortified libc checks / hardening bundles (e.g., `_FORTIFY_SOURCE`, `-fstack-protector-strong`, `-fharden`ed).

Dynamic bug finding (test builds): sanitizers for memory/UB/races (ASan/UBSan/TSan), plus coverage-guided fuzzing (libFuzzer).

Safety evidence overhead: mandated reviews + traceability + V&V activities (ISO 26262 / DO-178C / IEC 61508-style workflows).

Why Rust-lang - some tech insights

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 safeguard to prevent wrong interpretation of data;
- clear code organization into crates and modules;
- backward compatibility at crate level.

Does Rust remove the need for tooling?

No, but it sure makes code safer and dev faster

Rust's advantage is biggest in safe Rust.

As unsafe/FFI grows, assurance shifts back to: unsafe policy & reviews, FFI boundary correctness, sanitizers/fuzzing on risky edges, dependency/toolchain governance; plus the same ISO/DO-178 traceability & V&V.

Rust in Android: move fast and fix things

- A 1000x reduction in memory safety vulnerability density compared to Android's C and C++ code
- With Rust changes having a 4x lower rollback rate and spending 25% less time in code review, the safer path is now also the faster one

Who supports Rust-lang

Some links to read

the NSA: U.S. and International Partners Issue Recommendations to Secure Software Products Through Memory Safety

The White House: Back to the building blocks: A path toward secure and measurable software

How Rust went from a side project to the world's most-loved programming language

On Rust-lang adoption based on git-hub adoption

Rust developers at Google are twice as productive as C++ teams

Some companies that are building up Rust teams in embedded:

- Airbus, Ampere, Bae Systems, Boeing, Ford, General Dynamics, Hyundai, Northrop Grumman, NXP, Thales, Toyota, Volvo