Introduction

Lecture 1

Welcome

to the *Microprocessor Architecture* engineering class

You will learn

- how hardware works
- how to actually build your own hardware device
- the Rust programming Language

We expect

- to come to class
- ask a lot of questions

Team

Our team

Lectures

- Alexandru Radovici

Labs

- Alexandru Radovici
- Teodor Dicu (Hardware)
- Genan Omer (Software)

Outline

Lectures

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

Labs

- 7 labs

Project

- Build a hardware device running software written in Rust
- Presented at PM Fair during the last week of the semester

Grading

Part	Description	Points
Lecture tests	You will have a test at every class with subjects from the previous class.	1p
Final Lecture test	You will have a test during one of the lectures in January.	4p
Lab	Your work at every lab will be graded.	1p
Project	You will have to design and implement a hardware device. Grading will be done for the documentation, hardware design and software development.	3p
Final Test	You will have to take an exam during the last week of the semester.	2p
Total	*You will need at least 4.5 points to pass the subject.*	**11p**

Subjects

Theory

- How a microprocessor works
- How the ARM Cortex-M processor works
- Using digital signals to control devices
- Using analog signals to read data from sensors
- How interrupts work
- How asynchronous programming works (async/await)
- How embedded operating systems work

Practical

- How to use the STM32 Nucleo-U545RE-Q
  - Affordable
  - Powerful processor
  - Good documentation
- How to program in Rust
  - Memory Safe
  - *Java-like features, without Java's penalties*
  - Defines an embedded standard interface *embedded-hal*

Apollo Guidance Computer

We choose to go to the moon

John F. Kennedy, Rice University, 1961

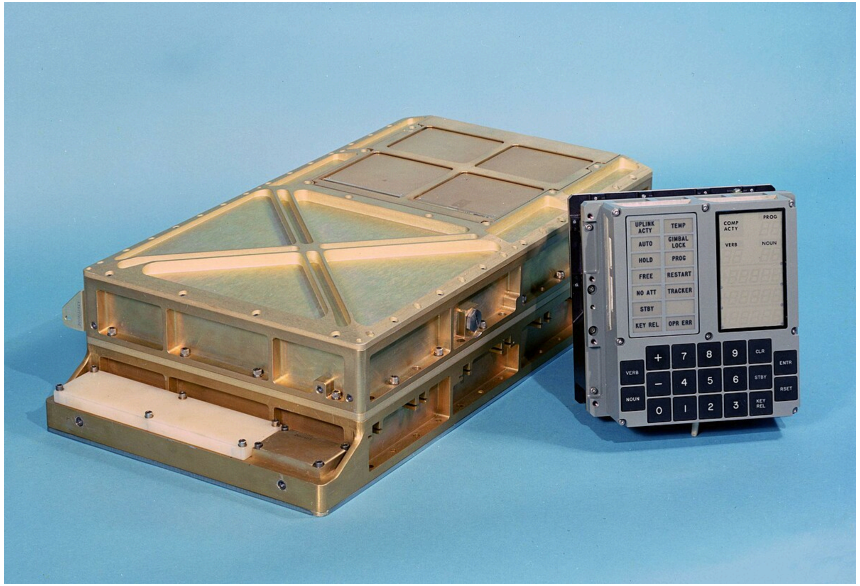
in this decade and do the other things, ***not because they are easy, but because they are hard****, because* ***that goal will serve to organize and measure the best of our energies and skills****, because that challenge is one that we are willing to accept, one we are unwilling to postpone, and one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

AGC

August 1966

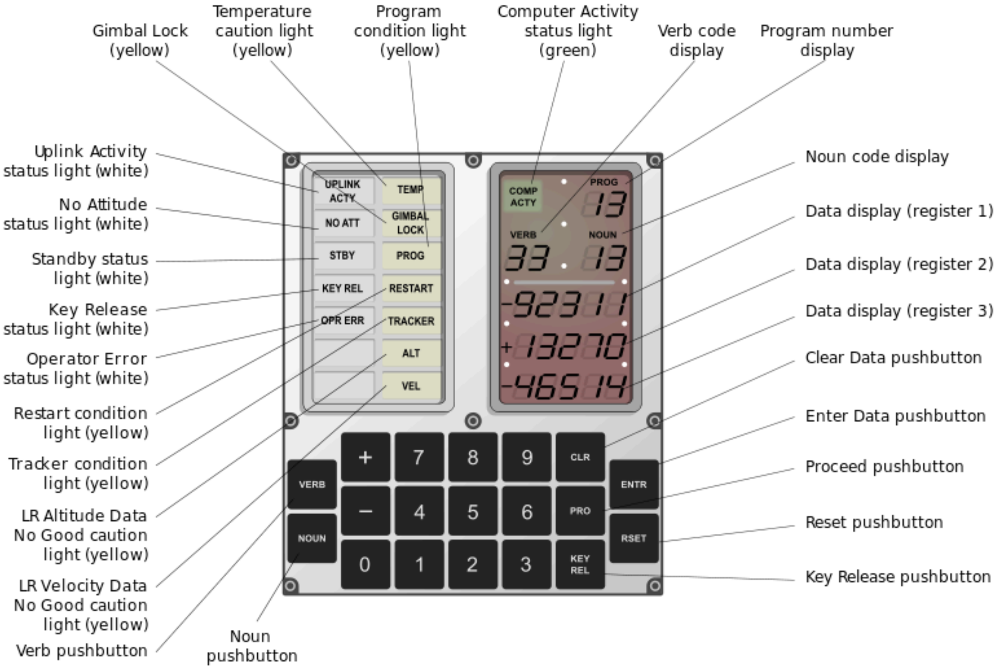
Frequency	2.048 MHz
World Length	15 + 1 bit
RAM	4096 B
Storage	72 KB
Software API	AGC Assembly Language

This landed the *moon eagle*.



DSKY

Display and keyboards



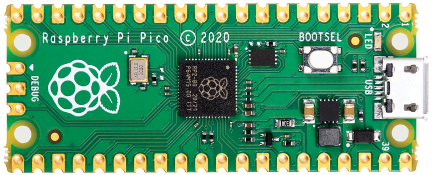
Simulator

What is a microprocessor?

Microcontroller (MCU)

Integrated in embedded systems for certain tasks

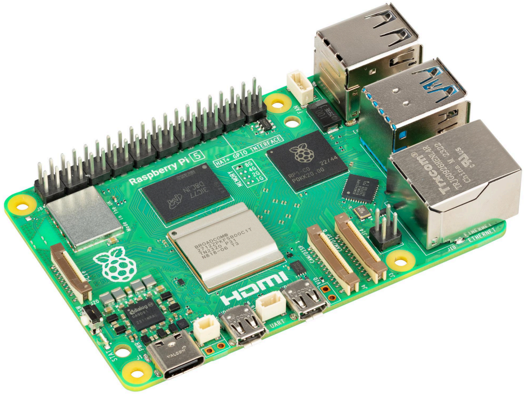
- low operating frequency (MHz)
- a lot of I/O ports
- controls hardware
- does not require an Operating System
- costs $0.1 - $25
- annual demand is billions



Microprocessor (CPU)

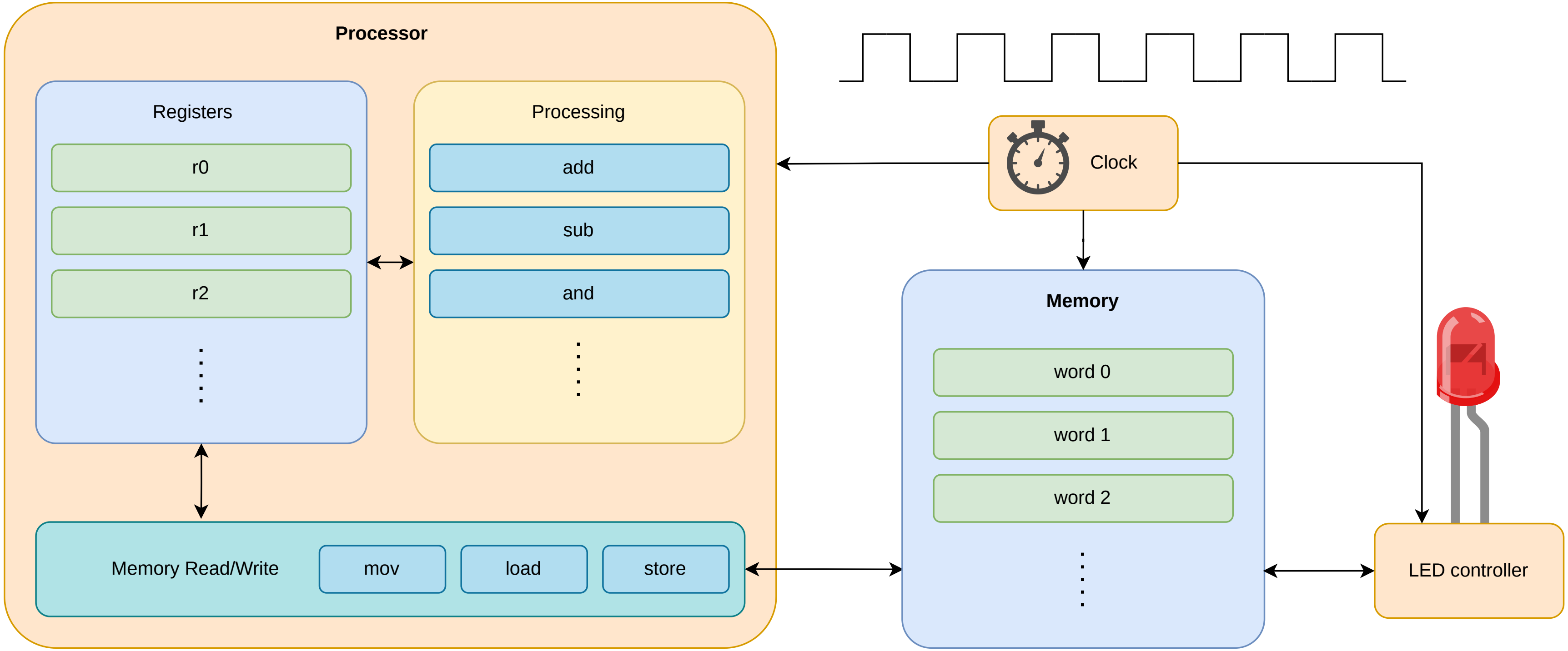
General purpose, for PC & workstations

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



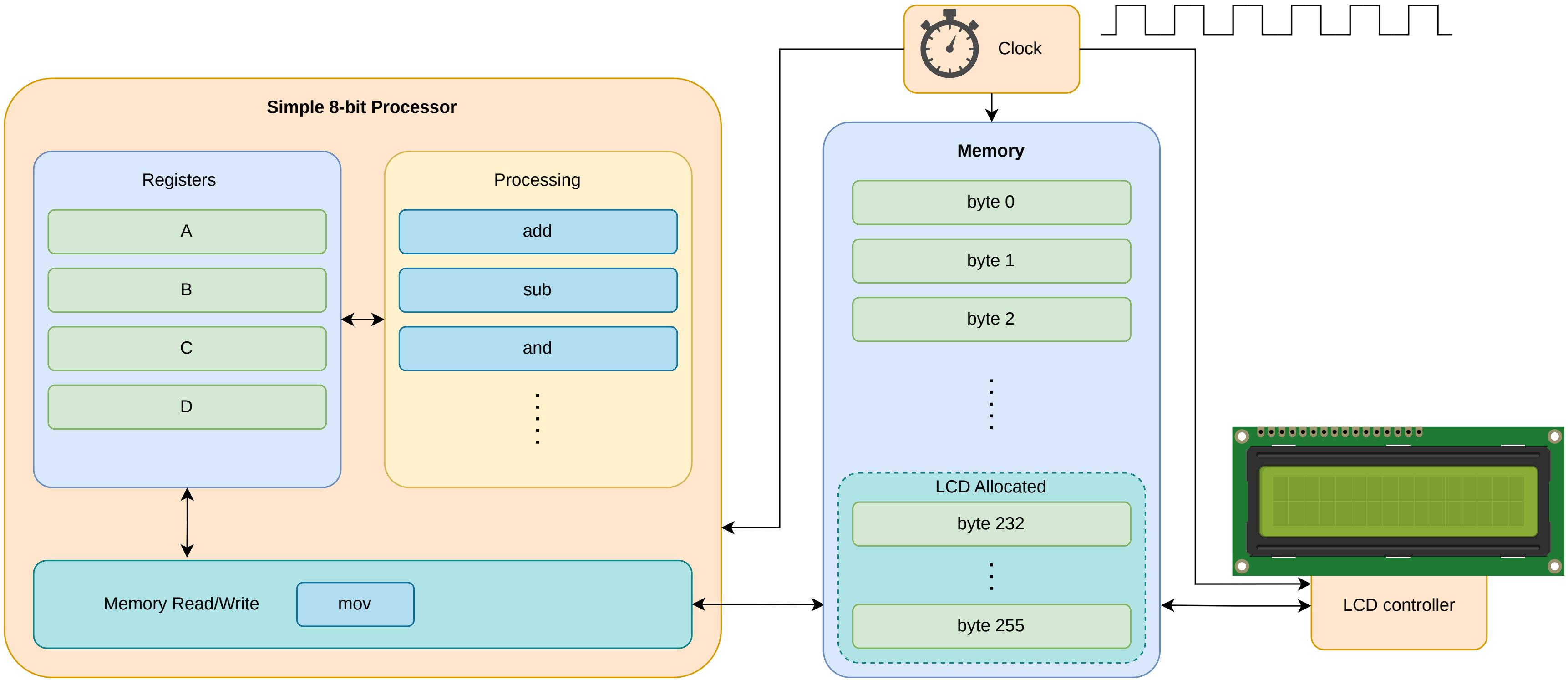
How a microprocessor works

This is a simple processor



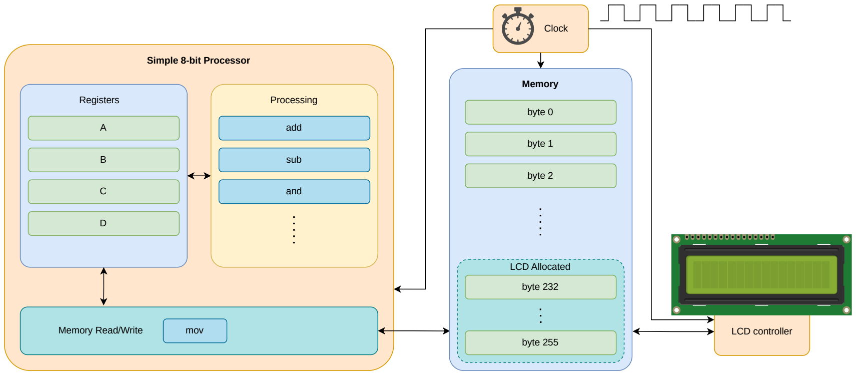
8 bit processor

a simple 8 bit processor with a text display



Programming

in Rust




```rust
use eight_bit_processor::print;

static hello: &str = "Hello World!";

#[start]
fn start() {
    print(hello);
}
```

Assembly

```asm
  JMP start
hello: DB "Hello World!" ; Variable
       DB 0  ; String terminator
start:
  MOV C, hello    ; Point to var
  MOV D, 232  ; Point to output
  CALL print
       HLT               ; Stop execution
print:       ; print(C:*from, D:*to)
  PUSH A
  PUSH B
  MOV B, 0
.loop:
  MOV A, [C]  ; Get char from var
  MOV [D], A  ; Write to output
  INC C
  INC D
  CMP B, [C]  ; Check if end
  JNZ .loop ; jump if not

  POP B
  POP A
  RET
```

Demo

a working example for the previous code

Start

Real World Microcontrollers

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

Bibliography

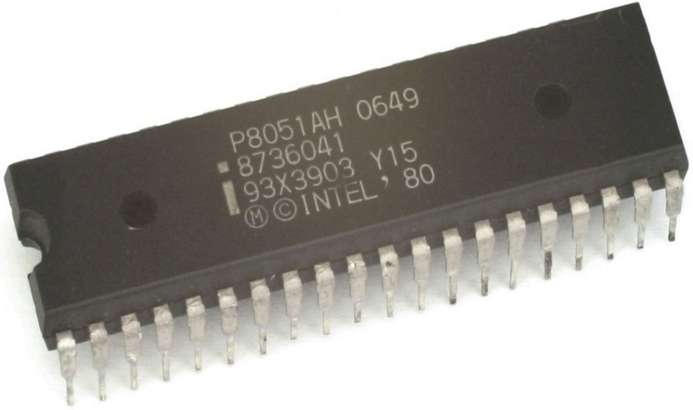
for this section

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

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

Intel

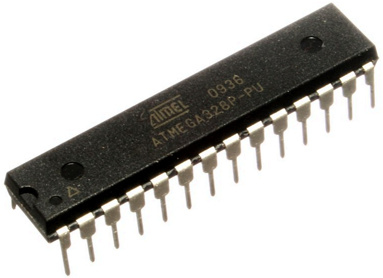
Vendor	Intel
ISA	8051, 8051
Word	8 bit
Frequency	a few MHz
Storage	?
Variants	*8048, 8051*



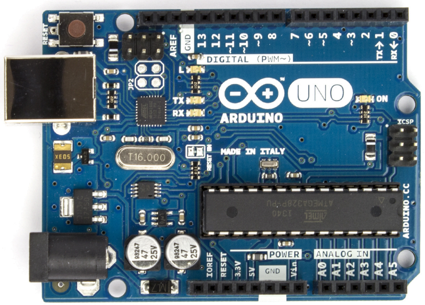
AVR

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*



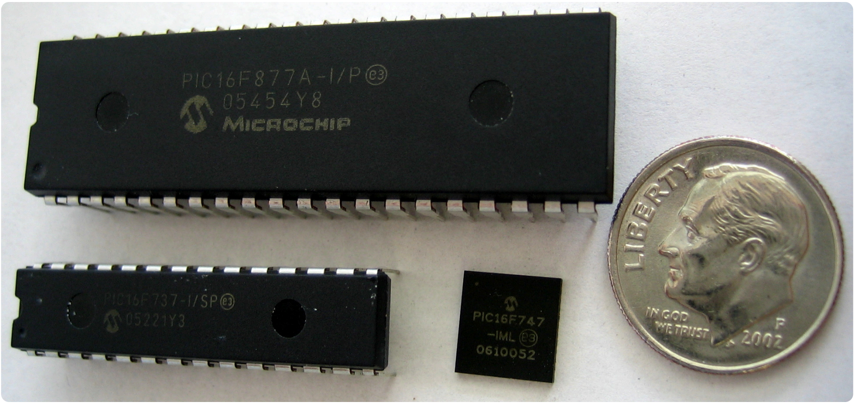
Board



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	*TC2xx*, *TC3xx*, *TC4xx*

ARM Cortex-M

Advanced RISC Machine

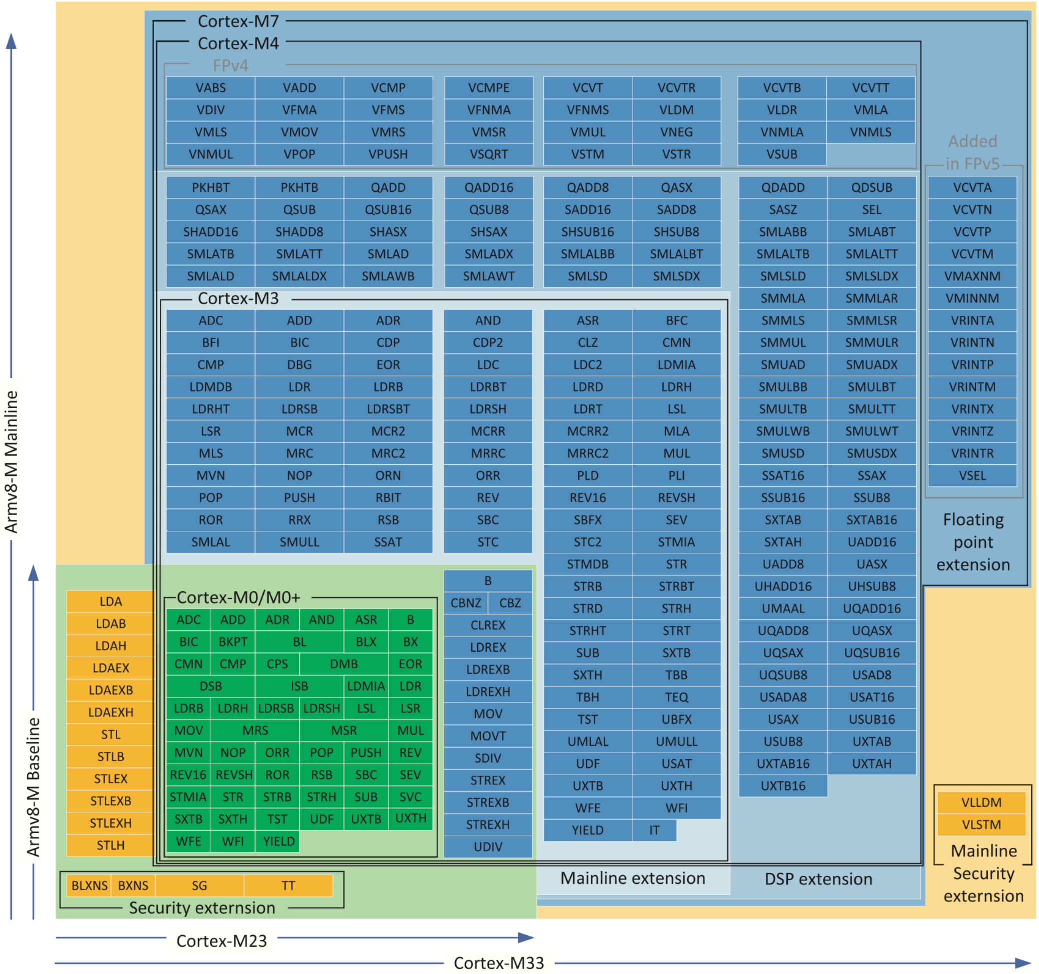
Vendor	Qualcomm, NXP, Nordic Semiconductor, Broadcom, Raspberry Pi
ISA	ARMv6-M (Thumb and some Thumb-2) ARMv7-M (Thumb and Thumb-2) ARMv8-M (Thumb and Thumb-2)
Word	32
Frequency	1 - 900 MHz
Storage	up to a few MB
Variants	*M0, M0+, M3, M4, M7, M23, M33*

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*

RP2040

ARM Cortex-M0+, built by Raspberry Pi

Bibliography

for this section

Raspberry Pi Ltd, *RP2040 Datasheet*

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

RP2040

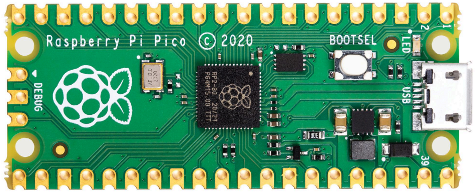
the MCU

Vendor	Raspberry Pi
Variant	ARM Cortex-M0+
ISA	ARMv6-M (Thumb and some Thumb-2)
Cores	2
Word	32 bit
Frequency	up to 133 MHz
RAM	264 KB

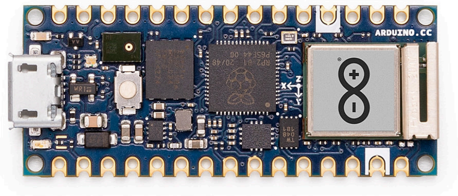
Boards

that use RP2040

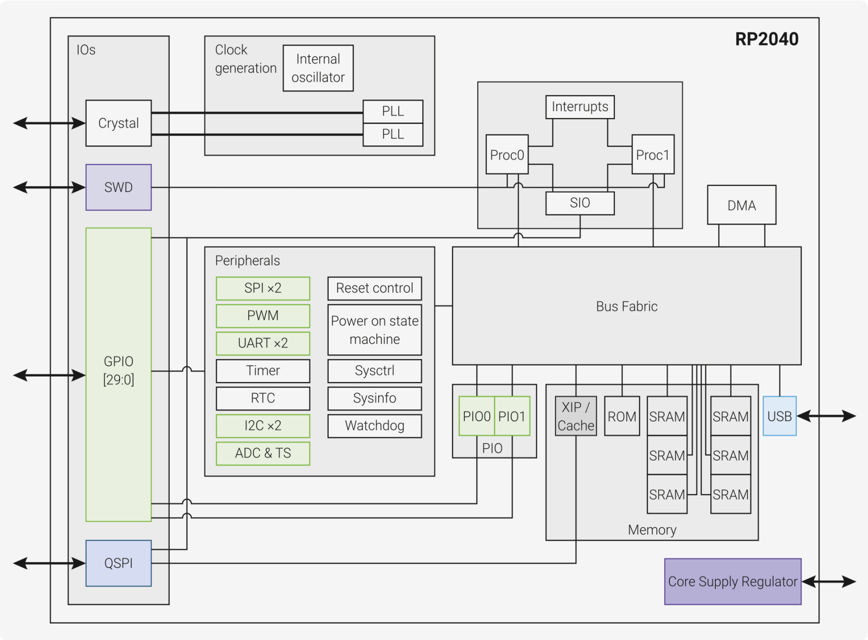
Raspberry Pi Pico (W)



Arduino Nano RP2040 Connect



The Chip



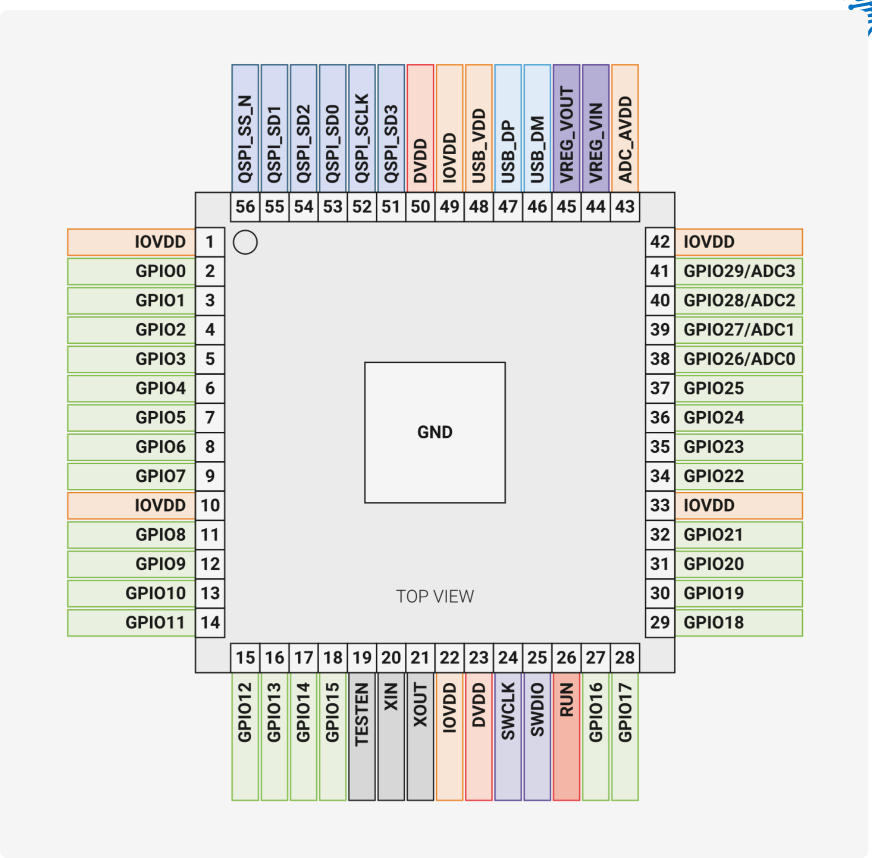
GPIO: General Purpose Input/Output
SWD: Debug Protocol
DMA: Direct Memory Access

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

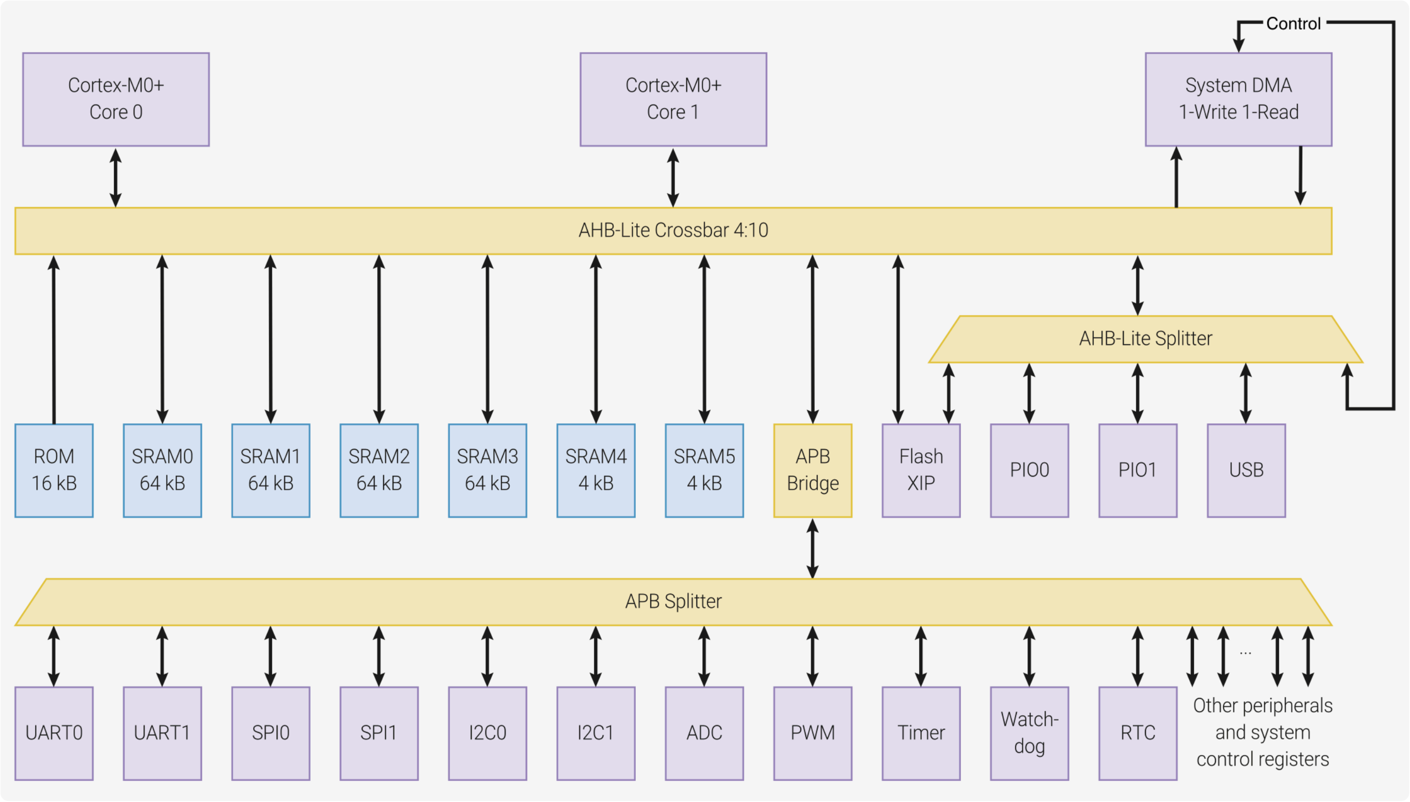
Pins

have multiple functions



The Bus

that interconnects the cores with the peripherals



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*

STM32U545RE

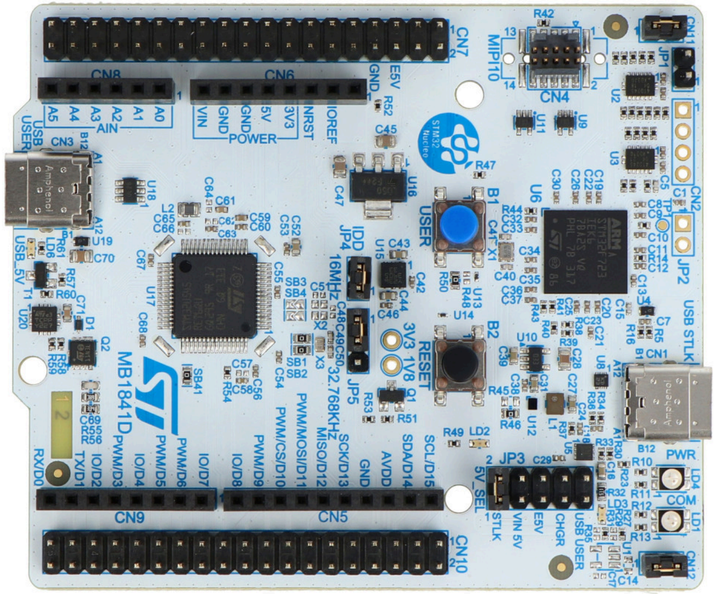
the MCU

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

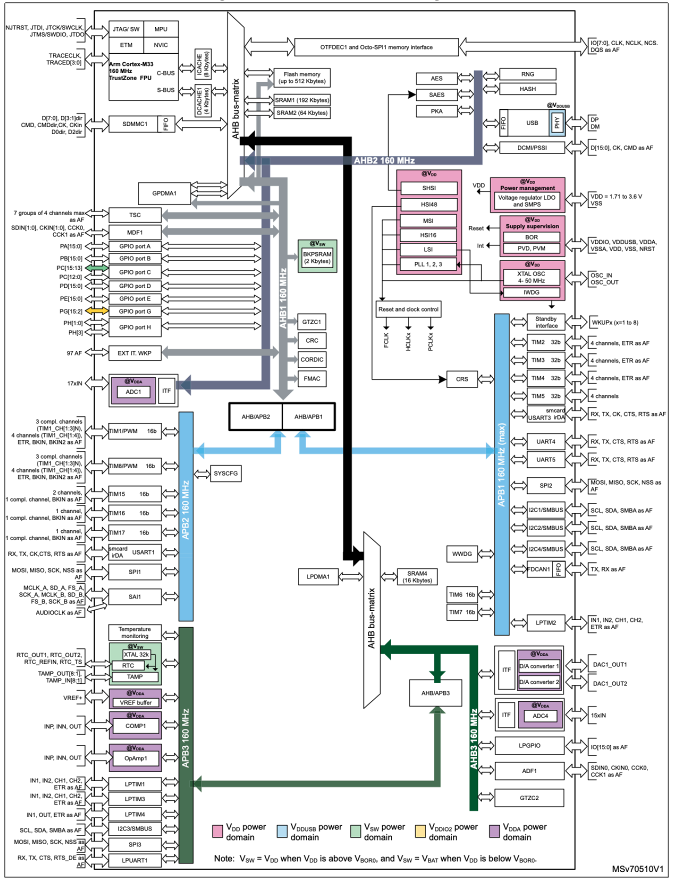
Board

that use STM32U545RE

Nucleo U545RE-Q



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

Up to 512-Kbyte Flash memory Dual Bank

274-Kbyte RAM

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

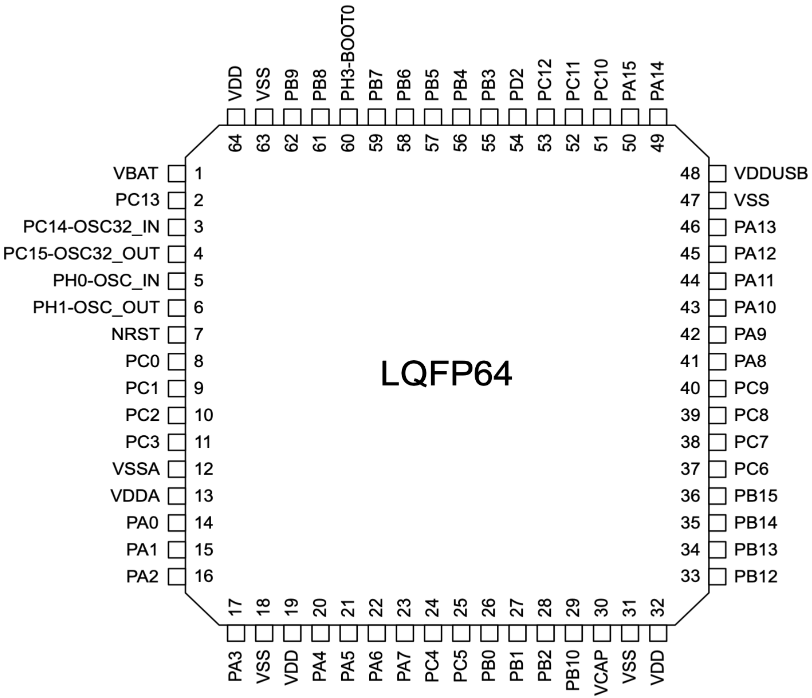
1x 14-bit ADC 2 MSPS,
1x 12-bit ADC 2 MSPS,
2 x DAC, 2 x comparators,
1 x op amps
1 x temperature sensor

Datasheet STM32U545RE

Pins

have multiple functions

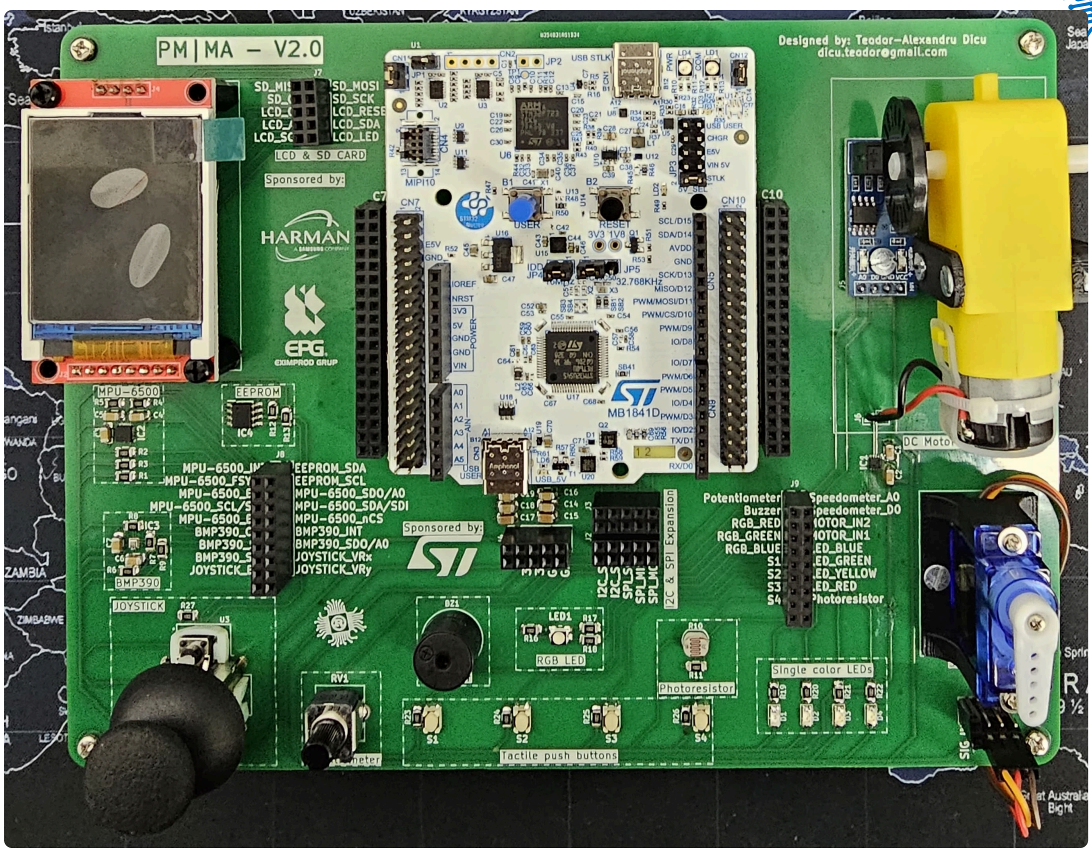
Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
		CRS/LPTIM1/ SYS_AF	**LPTIM1/ TIM1/2/8**	**LPTIM1/2/3/ TIM1/2/3/4/5**	**ADF1/I2C4/ OCTOSPI/ SAI1/SPI2/ TIM1/8/ USB**	**DCMI/ I2C1/2/3/4/ LPTIM3**	**DCMI/I2C4/ MDF1/ OCTOSPI/ SPI1/2/3**	**I2C3/MDF1/ OCTOSPI/ SPI3**	**USART1/3**
Port A	PA0	-	TIM2_CH1	TIM5_CH1	TIM8_ETR	-	-	SPI3_RDY	-
	PA1	LPTIM1_CH2	TIM2_CH2	TIM5_CH2	-	I2C1_SMBA	SPI1_SCK	-	-
	PA2	-	TIM2_CH3	TIM5_CH3	-	-	SPI1_RDY	-	-
	PA3	-	TIM2_CH4	TIM5_CH4	SAI1_CK1	-	-	-	-
	PA4	-	-	-	OCTOSPI1_NCS	-	SPI1_NSS	SPI3_NSS	-
	PA5	CSLEEP	TIM2_CH1	TIM2_ETR	TIM8_CH1N	PSSI_D14	SPI1_SCK	-	USART3_RX
	PA6	CDSTOP	TIM1_BKIN	TIM3_CH1	TIM8_BKIN	DCMI_PIXCLK/ PSSI_PDCK	SPI1_MISO	-	USART3_CTS
	PA7	SRDSTOP	TIM1_CH1N	TIM3_CH2	TIM8_CH1N	I2C3_SCL	SPI1_MOSI	-	USART3_TX
	PA8	MCO	TIM1_CH1	-	SAI1_CK2	-	SPI1_RDY	-	USART1_CK
	PA9	-	TIM1_CH2	-	SPI2_SCK	-	DCMI_D0/ PSSI_D0	-	USART1_TX
	PA10	CRS_SYNC	TIM1_CH3	LPTIM2_IN2	SAI1_D1	-	DCMI_D1/ PSSI_D1	-	USART1_RX
	PA11	-	TIM1_CH4	TIM1_BKIN2	-	-	SPI1_MISO	-	USART1_CTS
	PA12	-	TIM1_ETR	-	-	-	SPI1_MOSI	OCTOSPI1_ NCS	USART1_ RTS_DE
	PA13	JTMS/SWDIO	IR_OUT	-	-	-	-	-	-
	PA14	JTCK/SWCLK	LPTIM1_CH1	-	-	I2C1_SMBA	I2C4_SMBA	-	-
	PA15	JTDI	TIM2_CH1	TIM2_ETR	-	-	SPI1_NSS	SPI3_NSS	USART3_ RTS_DE



...

Lab Board

- Nucleo U545RE-Q Slot / Board
- 4 buttons
- 5 LEDs
- potentiometer
- buzzer
- photoresistor
- I2C EEPROM
- MPU-6500 accelerometer & Gyro
- BMP 390 Pressure sensor
- SPI LCD Display
- SD Card Reader
- servo connectors
- stepper motor



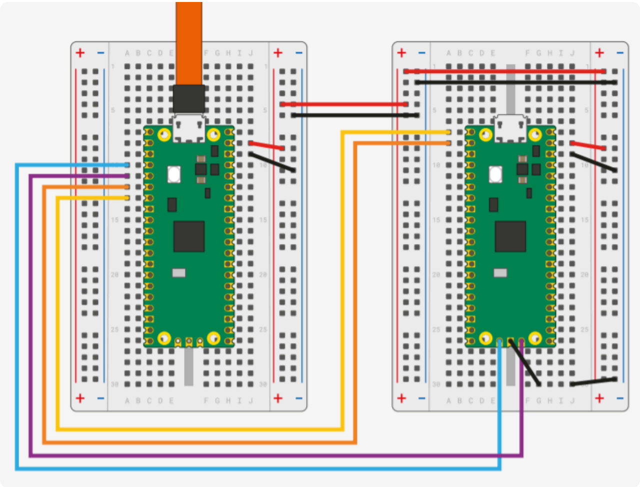
Project

suggested hardware

- the hardware should not cost more than 150 RON
- STM32 Nucleo F446RE or Nucleo U545RE-Q board (include debuggers)
- Raspberry Pi Pico with a debugger

Raspberry Pi Pico 2W + Debug Probe

Raspberry Pi Pico 2W + Raspberry Pi Pico 1



Bitwise Ops

How to set and clear bits

Set bit

set the `1` on position `bit` of `register`

```rust
fn set_bit(register: usize, bit: u8) -> usize {
    // assume register is 0b1000, bit is 2
    //   1 << 2 is 0b0100
    //   0b1000 | 0b0100 is 0b1100
    register | 1 << bit
}
```

Set multiple bits

```rust
fn set_bits(register: usize, bits: usize) -> usize {
    // assume register is 0b1000, bits is 0b0111
    //   0b1000 | 0b0111 is 0b1111
    register | bits
}
```

Clear bit

Set the `0` on position `bit` of `register`

```rust
fn clear_bit(register: usize, bit: u8) -> usize {
    // assume register is 0b1100, bit is 2
    //   1 << 2 is 0b0100
    //   !(1 << 2) is 0b1011
    //   0b1100 & 0b1011 is 0b1000
    register & !(1 << bit)
}
```

Clear multiple bits

```rust
fn clear_bits(register: usize, bits: usize) -> usize {
    // assume register is 0b1111, bits is 0b0111
    //   !bits = 0b1000
    //   0b1111 & 0b1000 is 0b1000
    register & !bits
}
```

Flip bit

Flip the bit on position `bit` of `register`

```rust
fn flip_bit(register: usize, bit: u8) -> usize {
    // assume register is 0b1100, bit is 2
    //   1 << 2 is 0b0100
    //   0b1100 ^ 0b0100 is 0b1000
    register ^ 1 << bit
}
```

Flip multiple bits

```rust
fn flip_bits(register: usize, bits: usize) -> usize {
    // assume register is 0b1000, bits is 0b0111
    //   0b1000 ^ 0b0111 is 0b1111
    register ^ bits
}
```

Let’s see a combined operation for value extraction

- We presume an 32 bits ID = `0b1100_1010_1111_1100_0000_1111_0110_1101`
- And want to extract a portion `0b`**`1100_1010_1111`**`_1100_0000_1111_0110_1101`

```
const MASK: u32 = 0b0000_0000_0000_0000_0000_1111_1111_1111;

fn print_binary(label: &str, num: u32) {
    println!("{}: {:032b}", label, num);
}

fn main() {
    let large_id: u32 = 0b1100_1010_1111_1100_0000_1111_0110_1101;
    let extracted_bits = (large_id >> 20) & MASK;

    // Print values in binary
    print_binary("Original_", large_id);
    print_binary("Mask_____", MASK);
    print_binary("Extracted", extracted_bits);
}
/* RESULT
Original_: 11001010111111000000111101101101
Mask_____: 00000000000000000000111111111111
Extracted: 00000000000000000000110010101111 */
```

With nice formating

```rust
const MASK: u32 = 0b0000_0000_0000_0000_0000_1111_1111_1111;
fn format_binary(num: u32) -> String {
    (0..32).rev()
        .map(|i| {
            if i != 0 && i % 4 == 0 {
                format!("{}_", (num >> i) & 1)
            } else {
                format!("{}", (num >> i) & 1)
            }
        })
        .collect::<Vec<_>>()
        .join("")
}
fn print_binary(label: &str, num: u32) { println!("{}: {}", label, format_binary(num));}
fn main() {
    let large_id: u32 = 0b1100_1010_1111_1100_0000_1111_0110_1101;
    let extracted_bits = (large_id >> 20) & MASK;
    print_binary("Original_", large_id);
    print_binary("Extracted", extracted_bits);
}
/* RESULTS:
Original_: 1100_1010_1111_1100_0000_1111_0110_1101
Extracted: 0000_0000_0000_0000_0000_1100_1010_1111 */
```

Conclusion

we talked about

- How a processor functions
- Microcontrollers (MCU) / Microprocessors (CPU)
- Microcontroller architectures
- ARM Cortex-M
- RP2040 and STM32U545RE