CS301
Embedded System and Microcomputer Principle

Lecture 2: STM32 MCU & GPIO

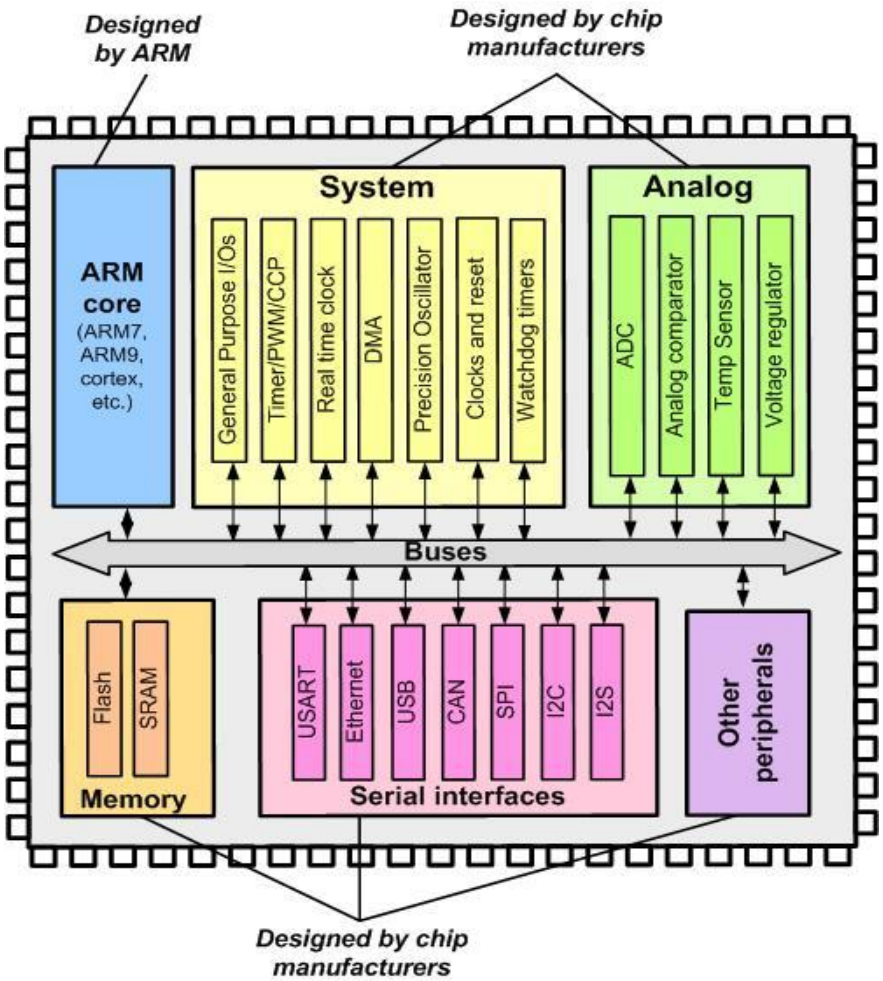
2023 Fall

Outline

- **CPU Overview**
- CPU Registers & Memory Map
- GPIO

ARM MCU inside view

- The ARM microprocessor + Different peripherals manufactured by chip designers

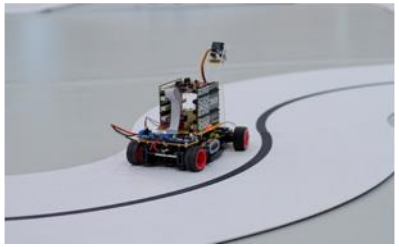


ARM Microprocessor

- ARM Cortex-A family:
  - **A**pplications processors
  - Support OS and high-performance applications
  - Such as Smartphones, Smart TV

- ARM Cortex-R family:
  - **R**eal-time processors with high performance and high reliability
  - Support real-time processing and mission-critical control

- ARM Cortex-M family:
  - **M**icrocontroller
  - Cost-sensitive, support SoC



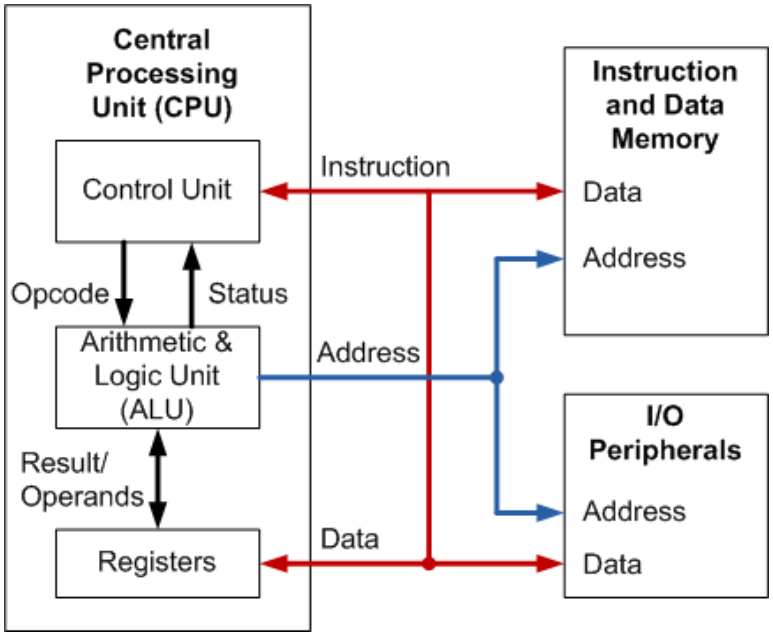
Arm families and Architectures

CORE	Application Cortex Processors			Cortex-A15		
	Embedded Cortex Processors			Cortex-A9		
	Classic ARM Processors			Cortex-A8		
			ARM11MP	Cortex-A5	Cortex-M7	
		ARM926	ARM176JZ	Cortex-R7	SC300	SC00
	ASC100	ARM968	ARM1136J	Cortex-R5	Cortex-M4	Cortex-M1
	ARM7TDMI	ARM946	ARM1156T2	Cortex-R4	Cortex-M3	Cortex-M0
Family	ARM7TDMI	ARM9E	ARM11	Cortex-A/R	Cortex-M	Cortex-M
Architecture Version	ARMv4T	ARMv5TJ	ARMv6	ARMv7A/R	ARMv7M/ME	ARMv8M

Architecture

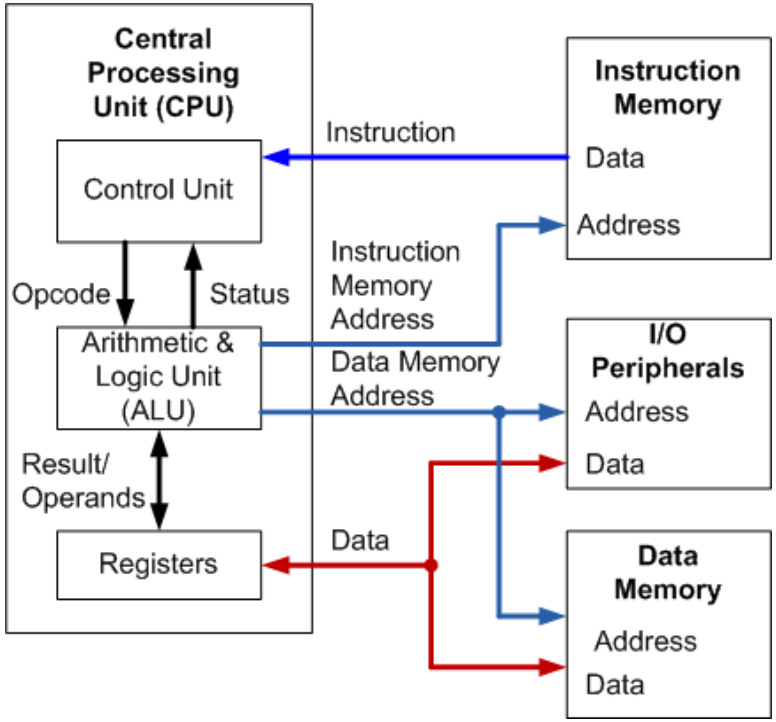
- ARM Cortex M3
  - Harvard Architecture: CPU, Memories, Input/Output

Von-Neumann



Instructions and data are stored in the **same** memory.

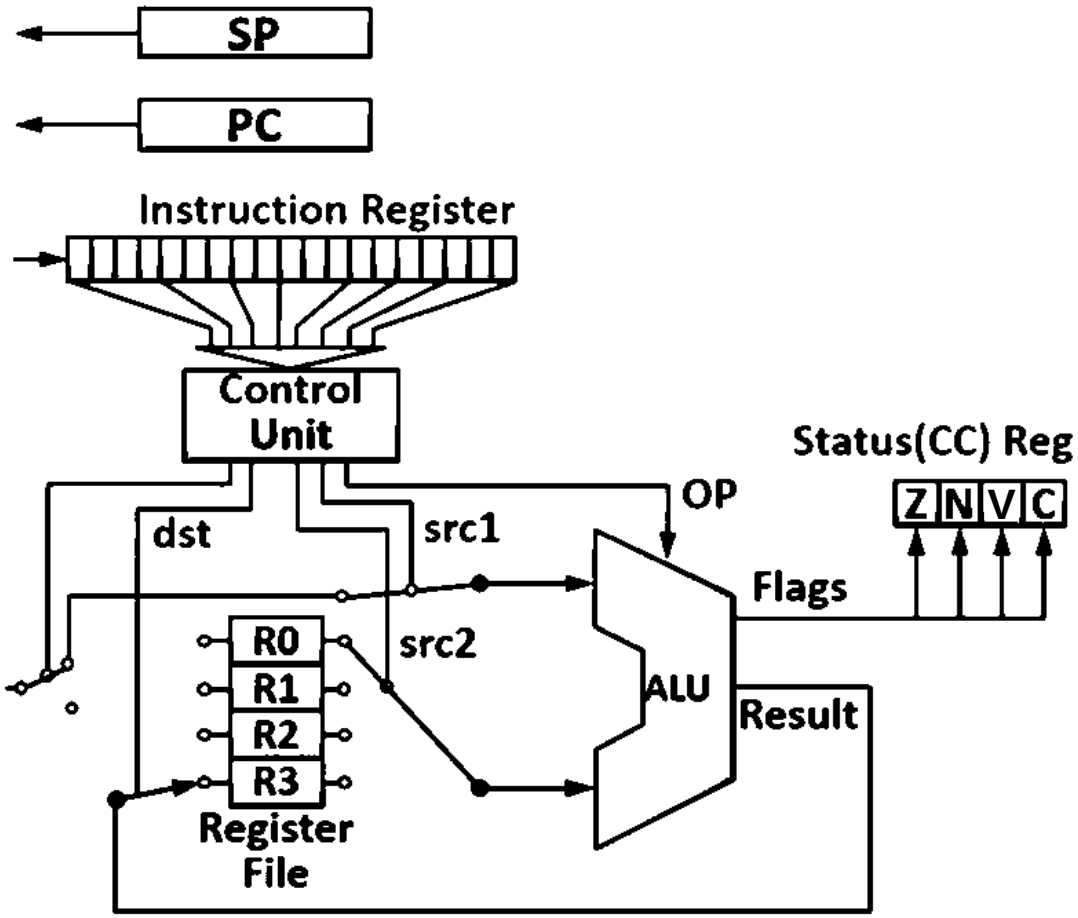
Harvard



Data and instructions are stored into **separate** memories.

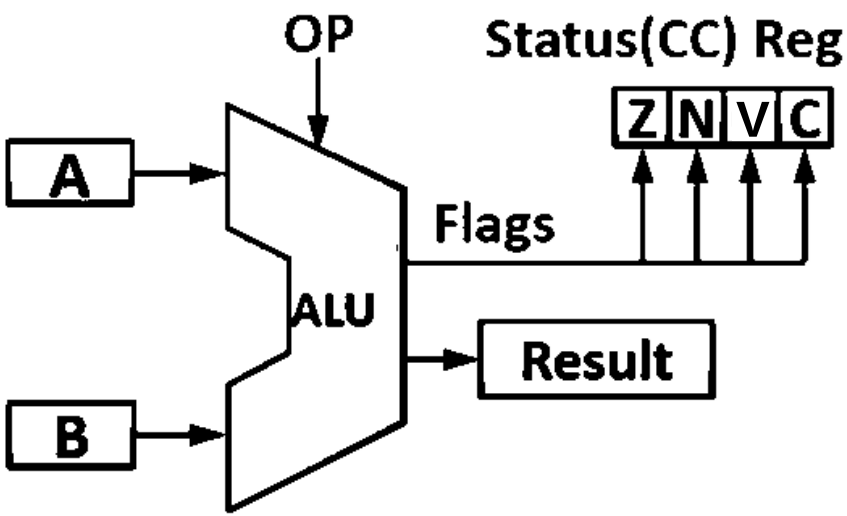
A Complete CPU

- Arithmetic Logic Unit (ALU)
- Register file
- Control Unit



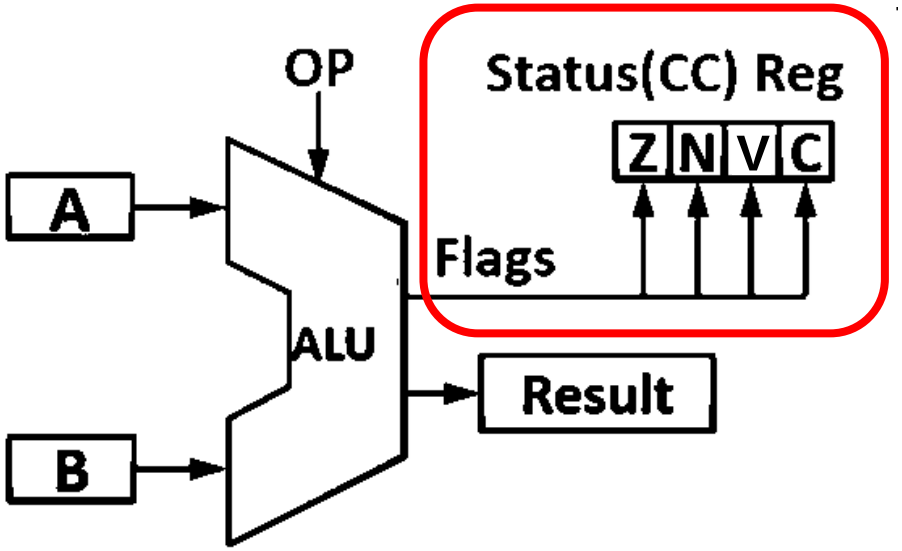
ALU

- Performs
  - arithmetic functions such as add, subtract, multiply, divide
  - logic functions such as AND, OR, NOT, XOR,
  - bit functions such shift, rotation
- Let's find out where are the following key elements
  - Operands
  - Operation
  - Flags
  - result



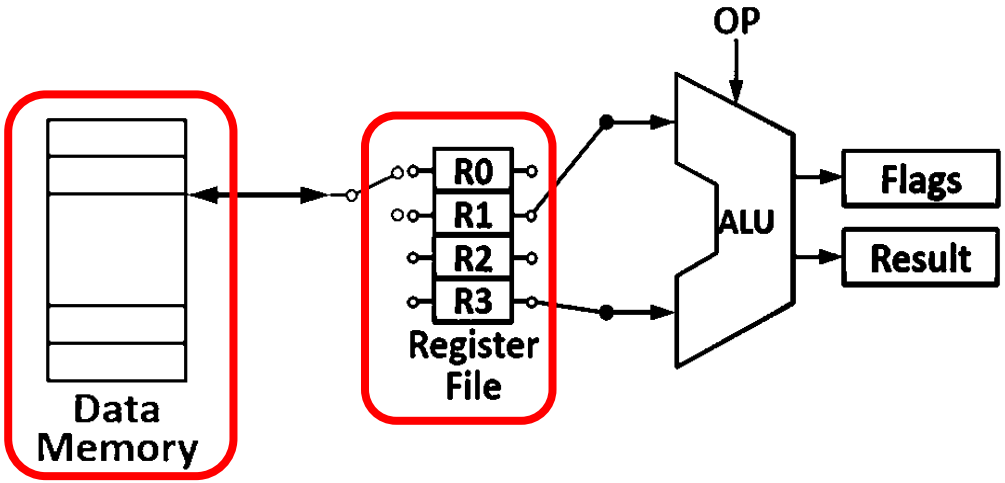
ALU Flags

- Flags: special bits that provide information about the results of arithmetic and logical operations.
  - Z: Zero
  - N: Negative
  - V: oVerflow
  - C: Carry
- Where are the Flags?
  - stored in PSR
    - Program Status Register



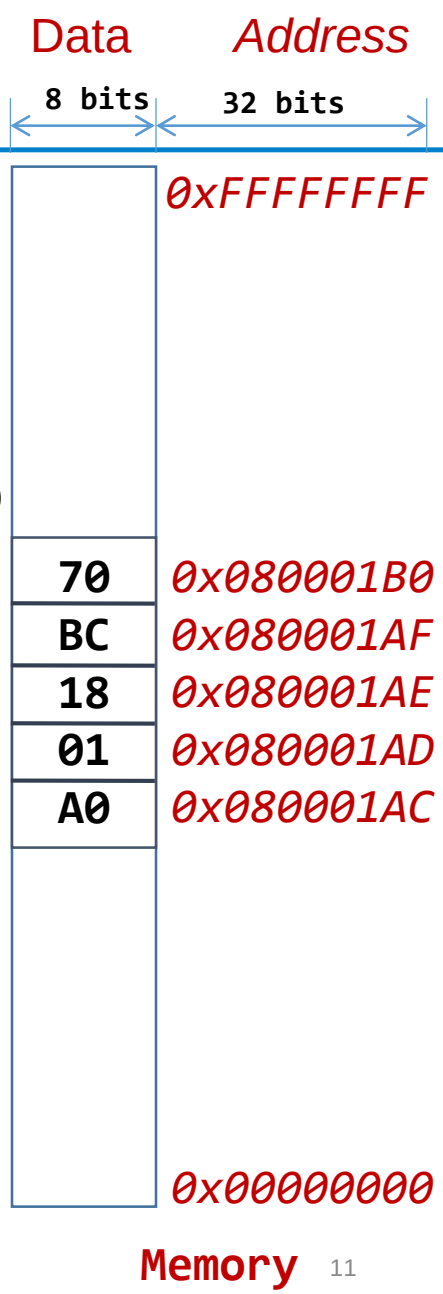
ALU Operands

- Operands: inputs to an arithmetic or logic operation
- Where are the operands?
  - Registers
    - Registers are used to temporarily store/retrieve operands
    - Every CPU includes several general/special purpose registers.
    - The number and width of registers are important metrics for measuring a CPU's performance.
  - Data Memory
    - Accessing memory is significantly slower than accessing registers



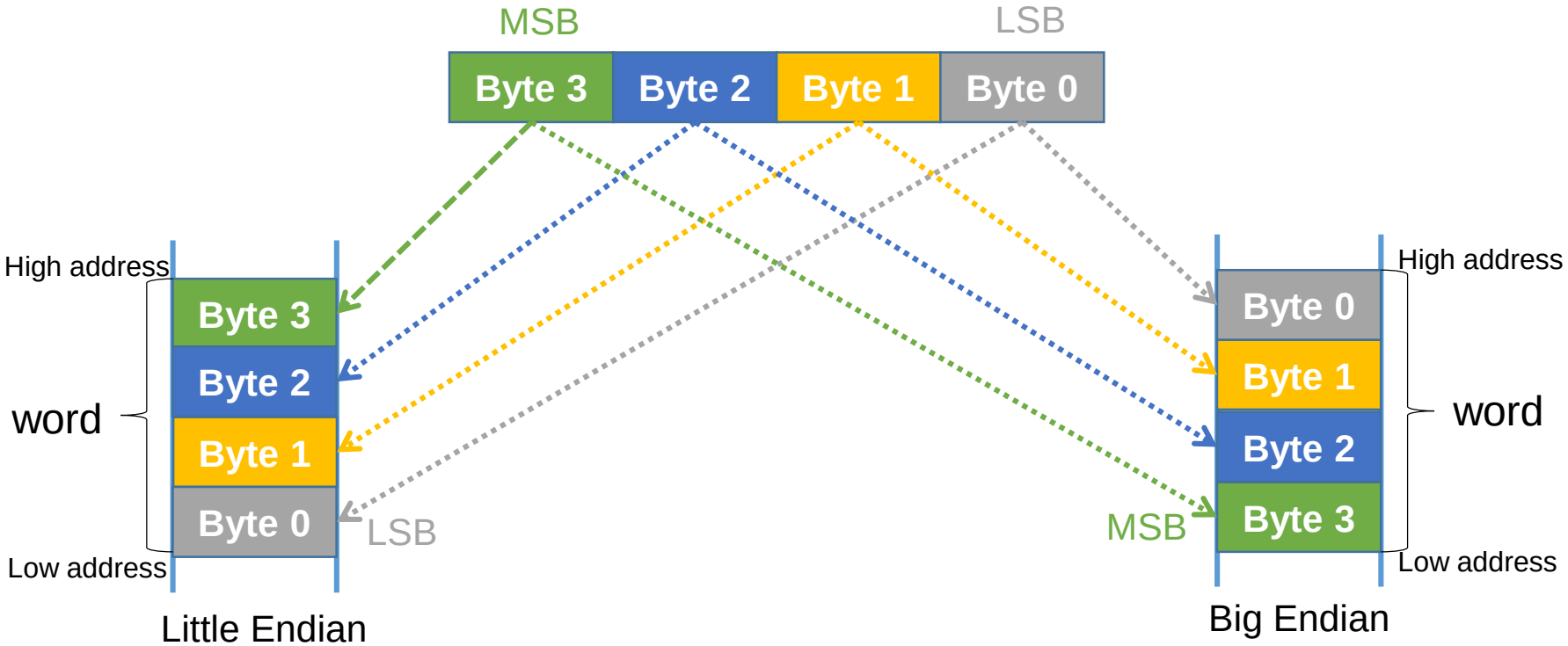
Memory Organization

- Memory is arranged as a series of "locations"
  - Each location has a unique "address"
  - Each location holds a byte (***byte-addressable***)
  - e.g. the memory location at address `0x080001B0` contains the byte value `0x70`, i.e., `112`
- 32-bit Address line
  - Max size: 2^{32} = 4G (bytes)
  - Address range: 0x00000000 ~ 0XFFFFFFFF
- Values stored at each location can represent
  - either program data
  - or program instructions



Little Endian vs Big Endian

- Little-endian
  - LSB of a word is at least memory address
- Big-endian
  - MSB of a word is at least memory address

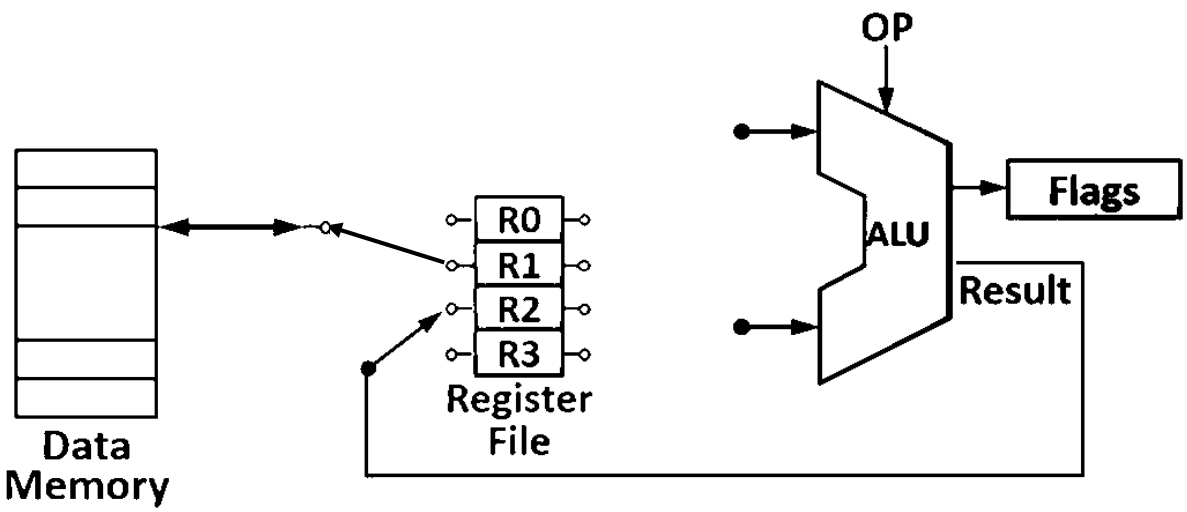


LSB is at least address!

MSB is at least address!

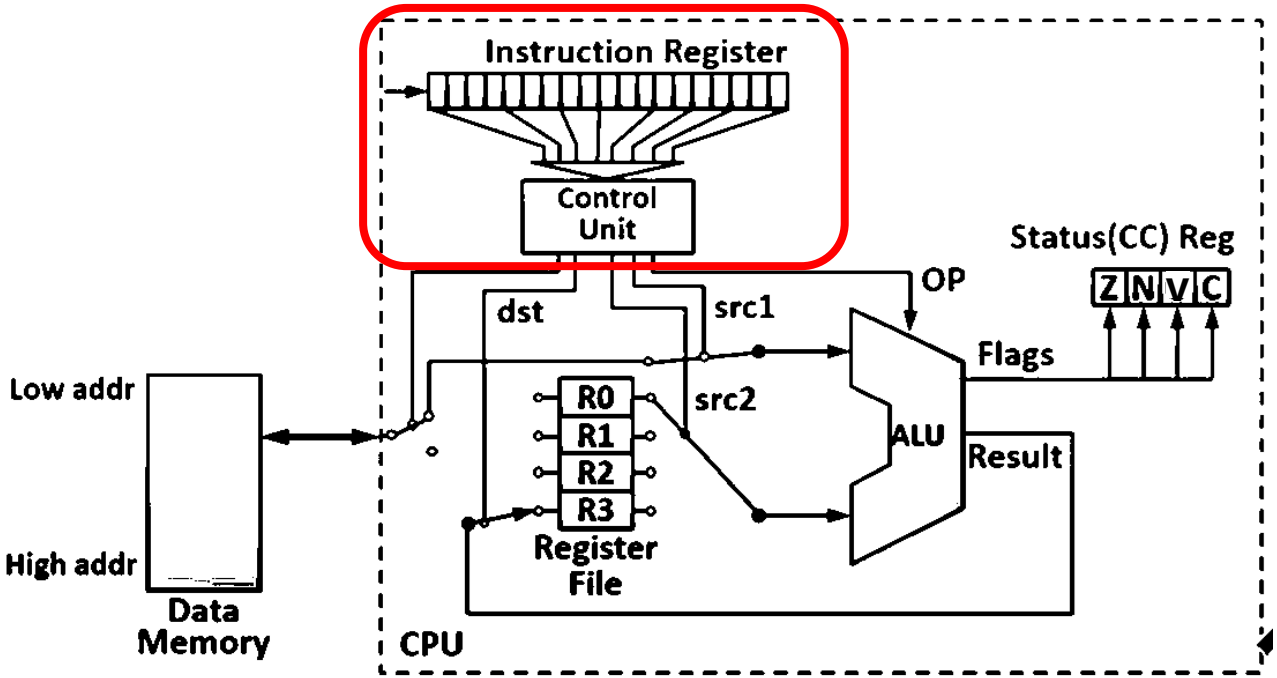
ALU Result

- Where to store the results?
  - Generally, the same as the operands
  - Registers
  - Data Memory



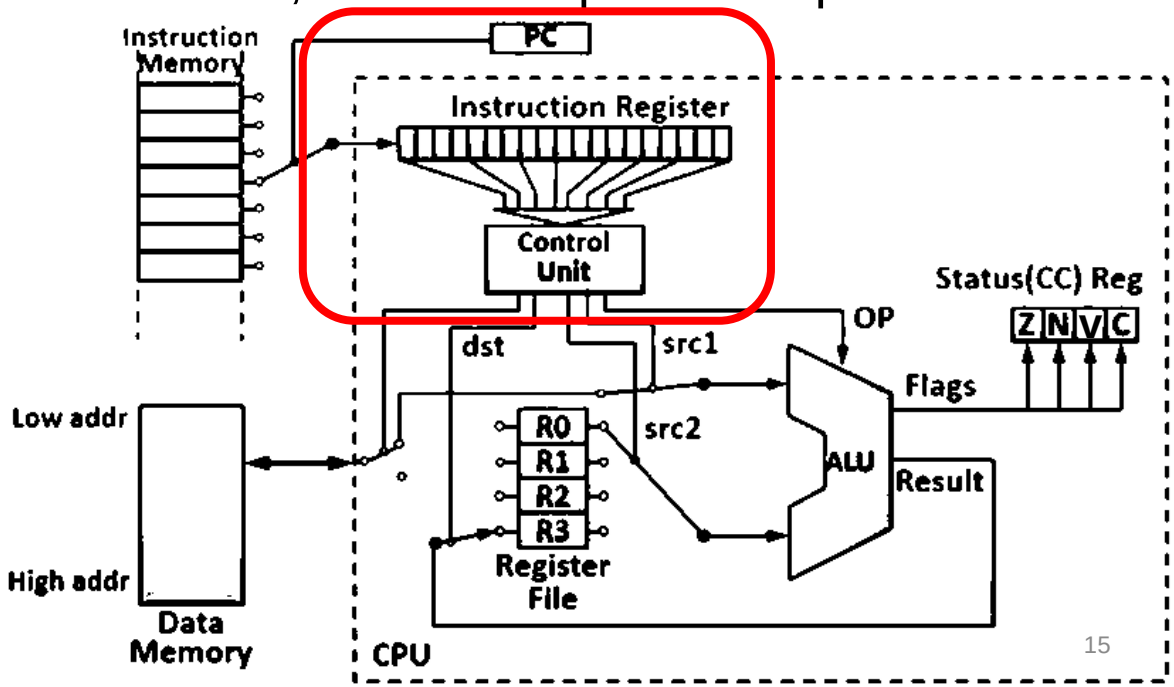
Control Unit

- Instruction Decoding
  - Analyz the Operation to Be Executed by the Instruction
- Dataflow
  - Identify the Sources of Operands and the Destination of the Result for the Instruction



Program Counter

- Instruction fetch
  - A program consists of an instruction sequence stored in the program memory.
  - Instructions are processed sequentially, one after the other, the address of the next instruction to be executed is stored in the PC register (Program Counter)
  - After an instruction is fetched, the PC is updated to point to the next instruction.



Outline

- CPU Overview
- **CPU Registers & Memory Map**
- GPIO

CPU Registers

- Fastest way to read and write
- Registers are within the processor chip
- Each register has 32 bits

ARM Cortex-M3 has

Register Bank: **R0 – R15**

R0-R12: 13 general-purpose registers

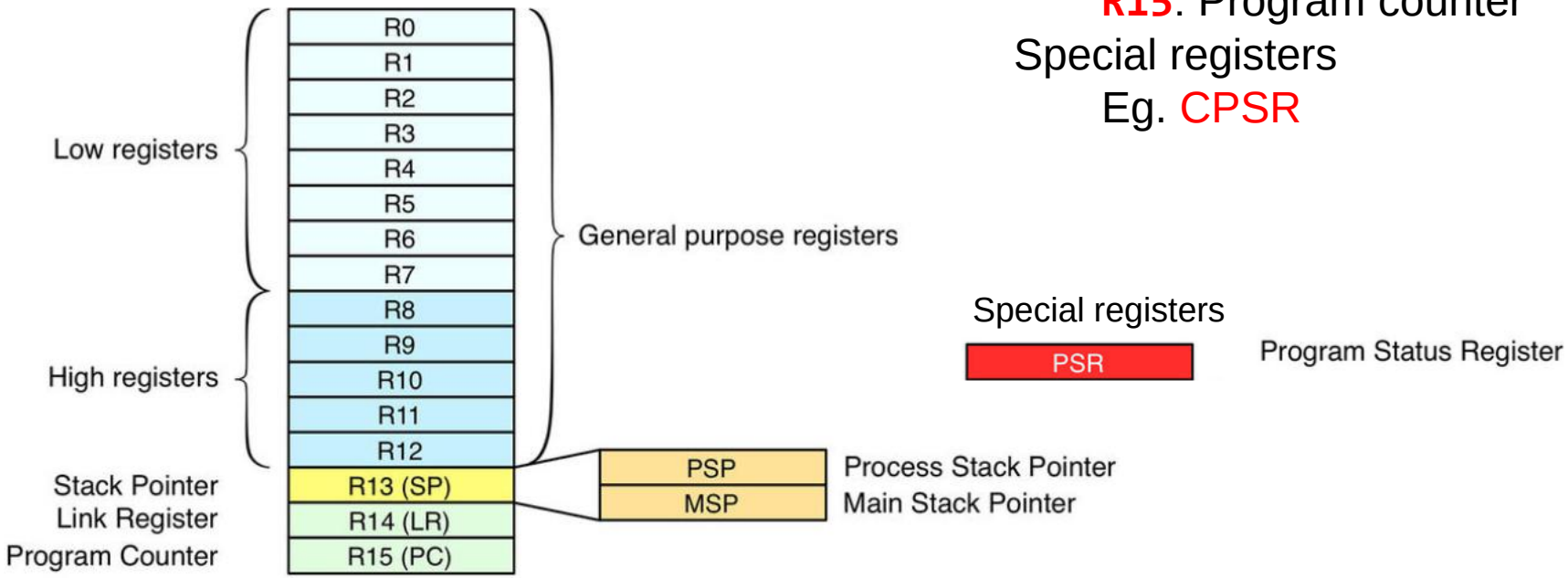
R13: Stack pointer

R14: Link register

R15: Program counter

Special registers

Eg. CPSR



CPU Registers

- Low Registers (R0 – R7)
  - Can be accessed by any instruction
- High Register (R8 – R12)
  - Can only be accessed by some instructions
- Stack Pointer (R13)
  - Cortex-M3 supports two stacks
  - Main SP (MSP) for privileged access (e.g. exception handler)
  - Process SP (PSP) for application access
- Link Register
  - Stores the return address for function calls
- Program Counter (R15)
  - Memory address of the to be executed instruction

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13 (SP)
R14 (LR)
R15 (PC)

- Program Status Register
  - CPSR

D31	D30	D29	D28	.....	D7	D6	D5	D4	D3	D2	D1	D0
N	Z	C	V	Reserved	I	F	T	M4	M3	M2	M1	M0
Condition code					Operating mode							

Memory Map

- STM32 Memory is mapped into 8 blocks, each having 512 MB
- We specially take care about the following blocks.
  - Code
  - SRAM
  - Peripheral

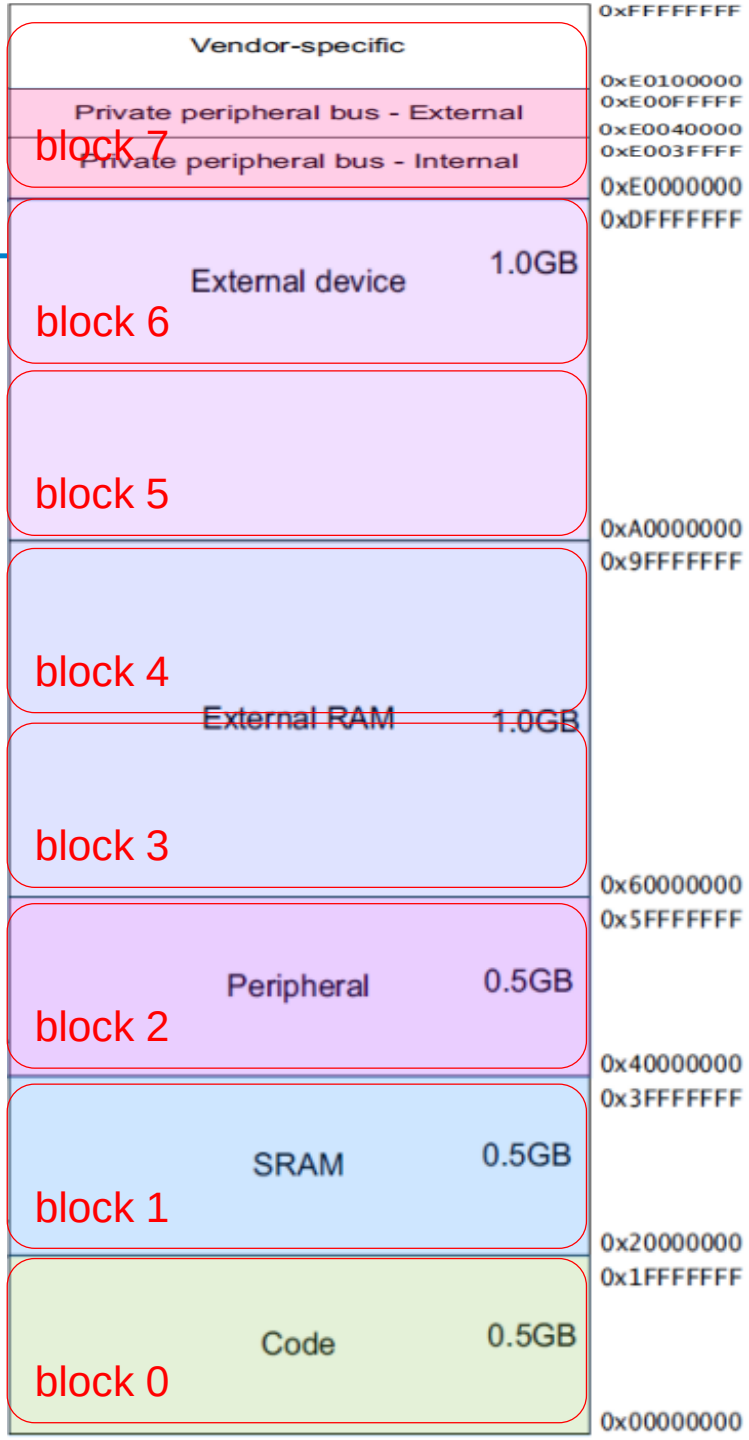
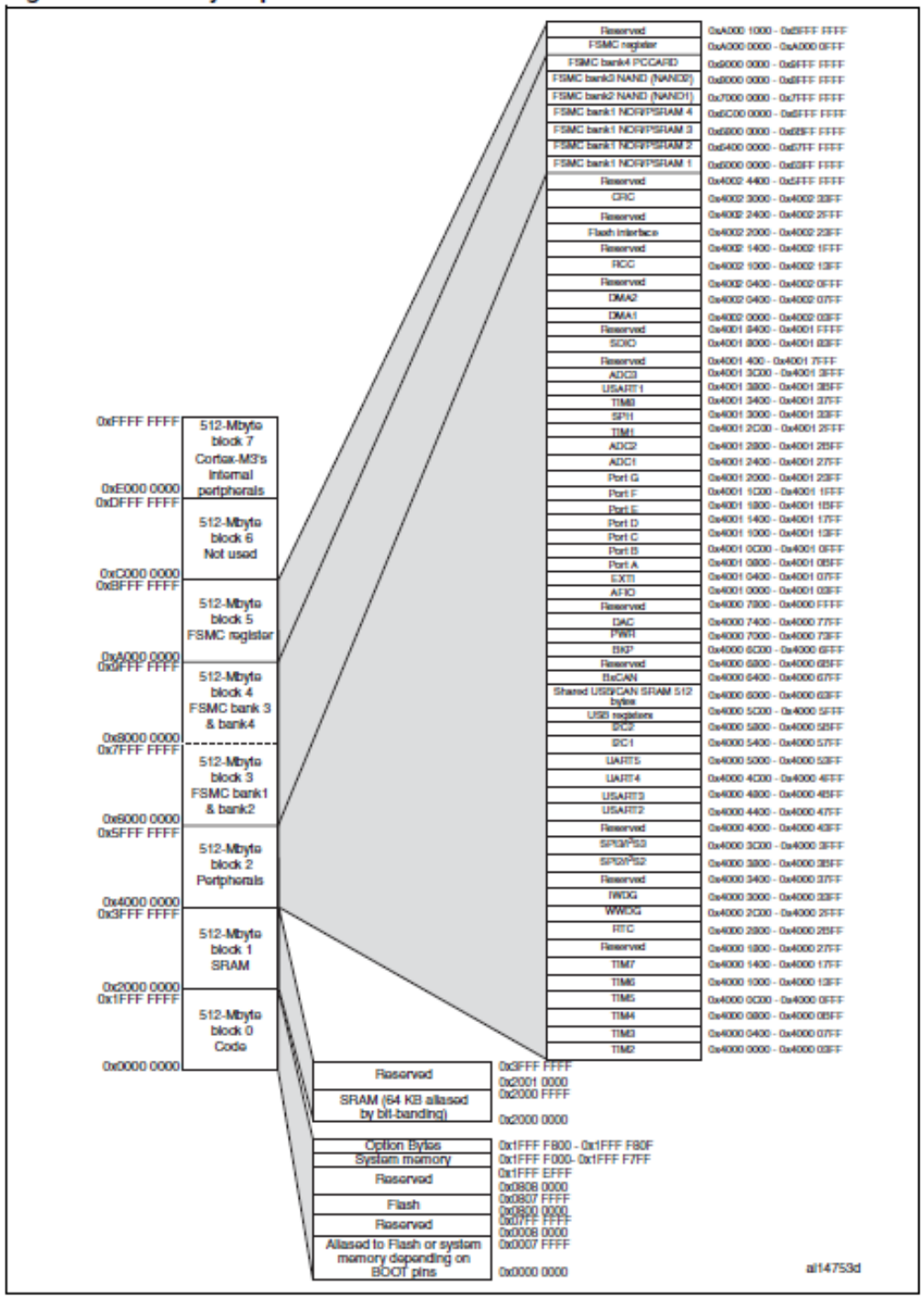


Figure 9. Memory map



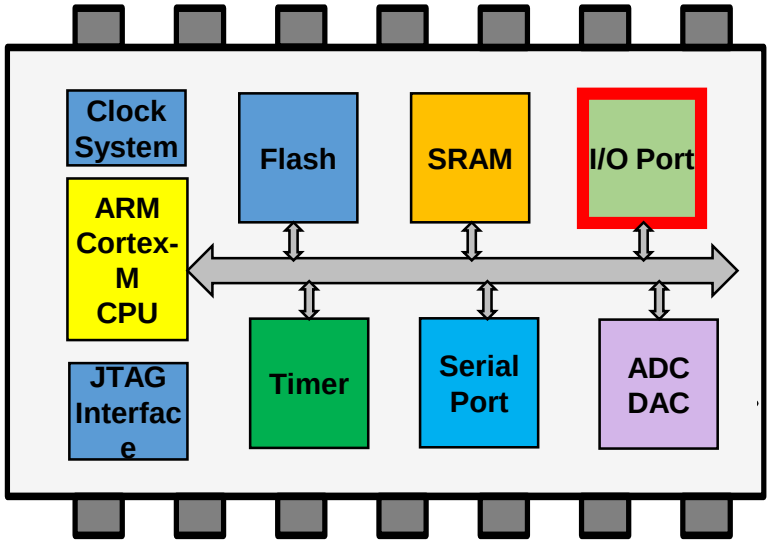
STM32F103 Datasheet, Figure 9

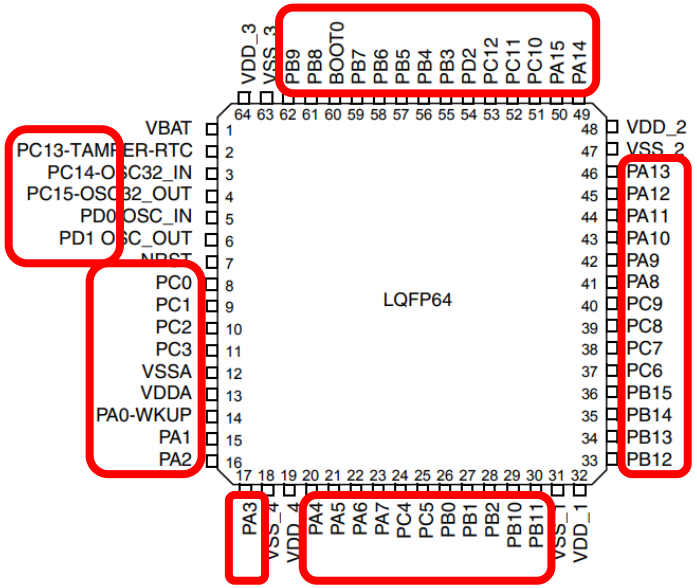
Outline

- CPU Overview
- CPU Registers & Memory Map
- **GPIO**

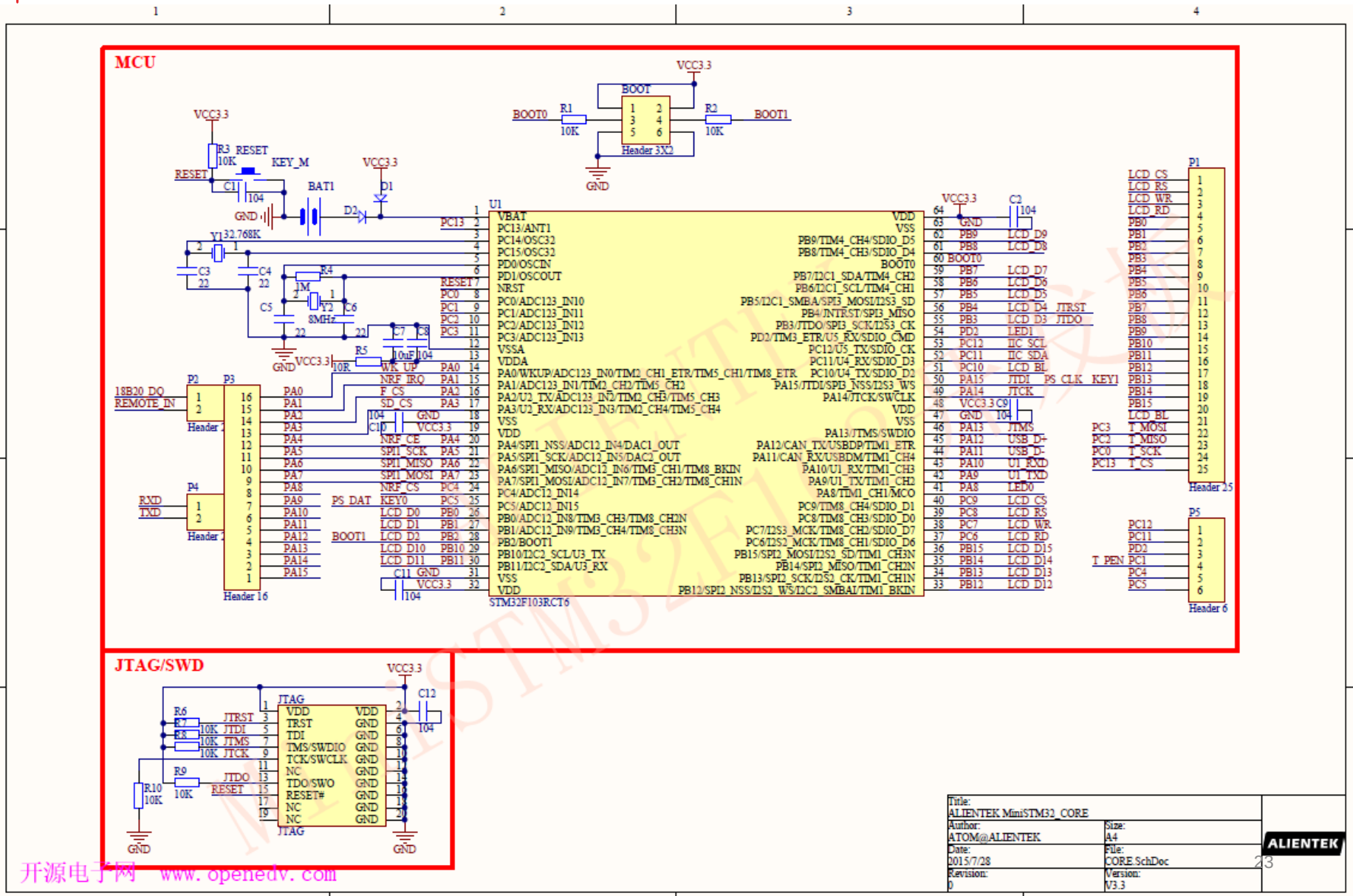
GPIO

- GPIO (General Purpose Input Output) pins are commonly used pins that can control the voltage levels (high or low) and can be read from or written to.
  - GPIO pins are named in groups, as ports PA, PB, PC, etc
  - Each port contains 16 pins numbered from 0 to 15
  - The ports appear to the CPU as registers (memory-mapped I/O), each bit corresponds to a pin and a port may be associated to many registers for different purposes (next page)



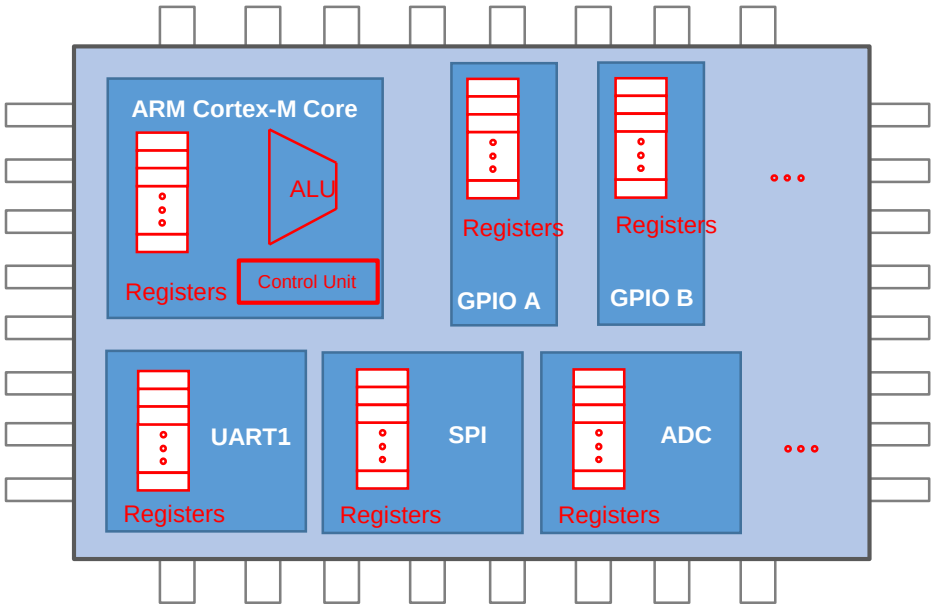


STM32F103 Schematic

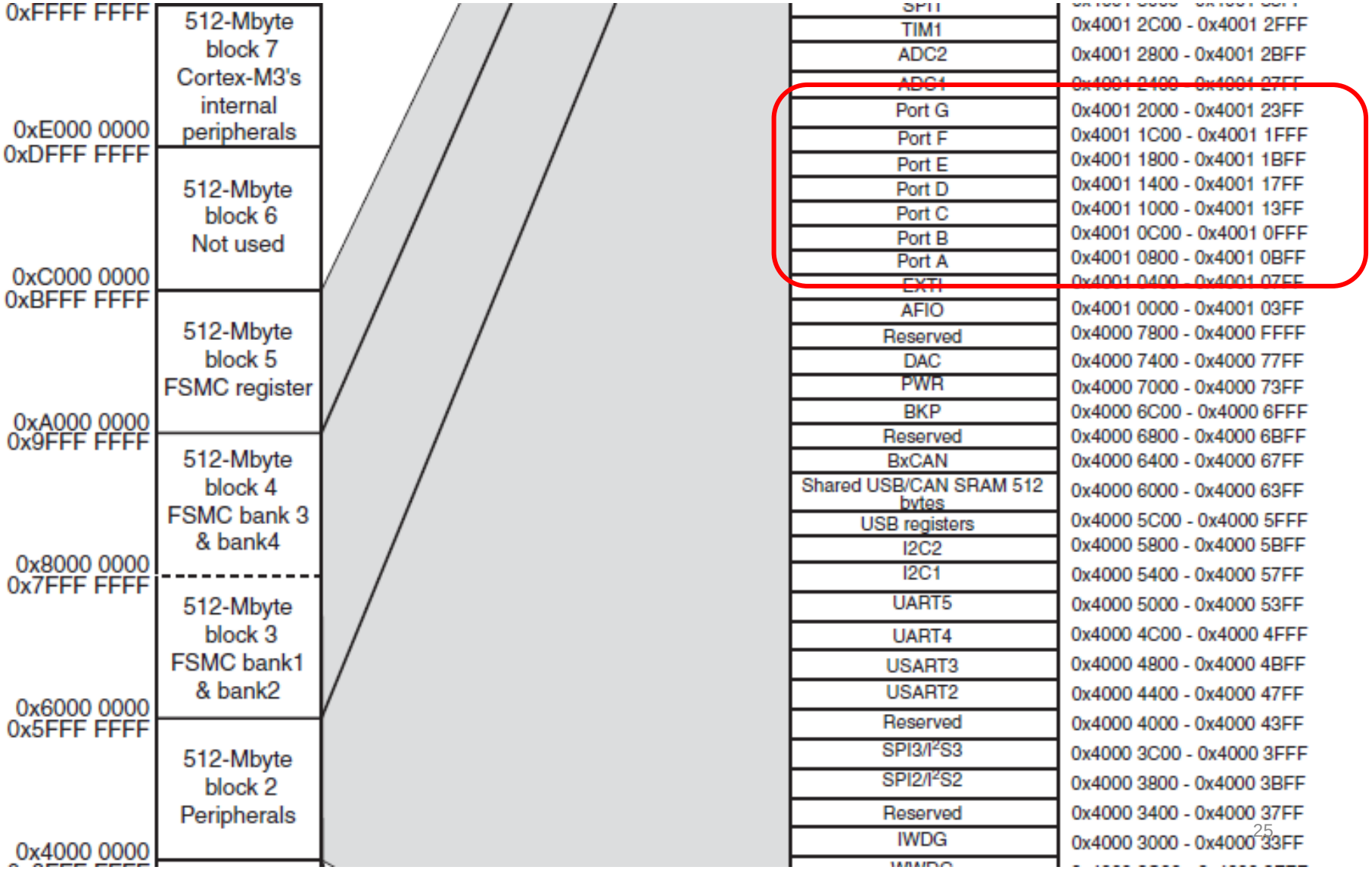


CPU Registers vs GPIO Registers

- Processor can directly access processor registers
  - `ADD r3,r1,r0  ; r3 = r1 + r0`
- Processor accesses peripheral registers via **memory mapped I/O**
  - Each peripheral register is assigned a fixed memory address at the chip design stage
  - Processor treats peripherals registers the same as data memory
  - Processor uses load/store instructions to read from/write to memory (to be covered in future lectures)

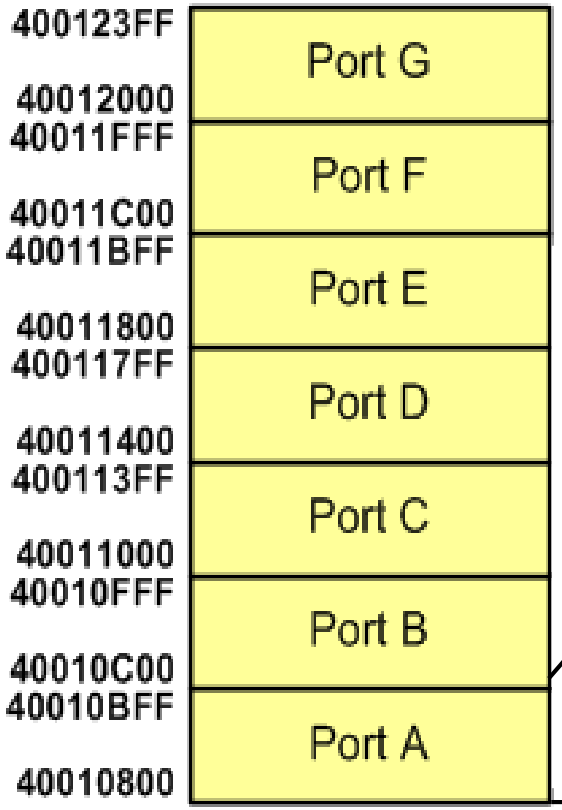


Memory Mapped IO



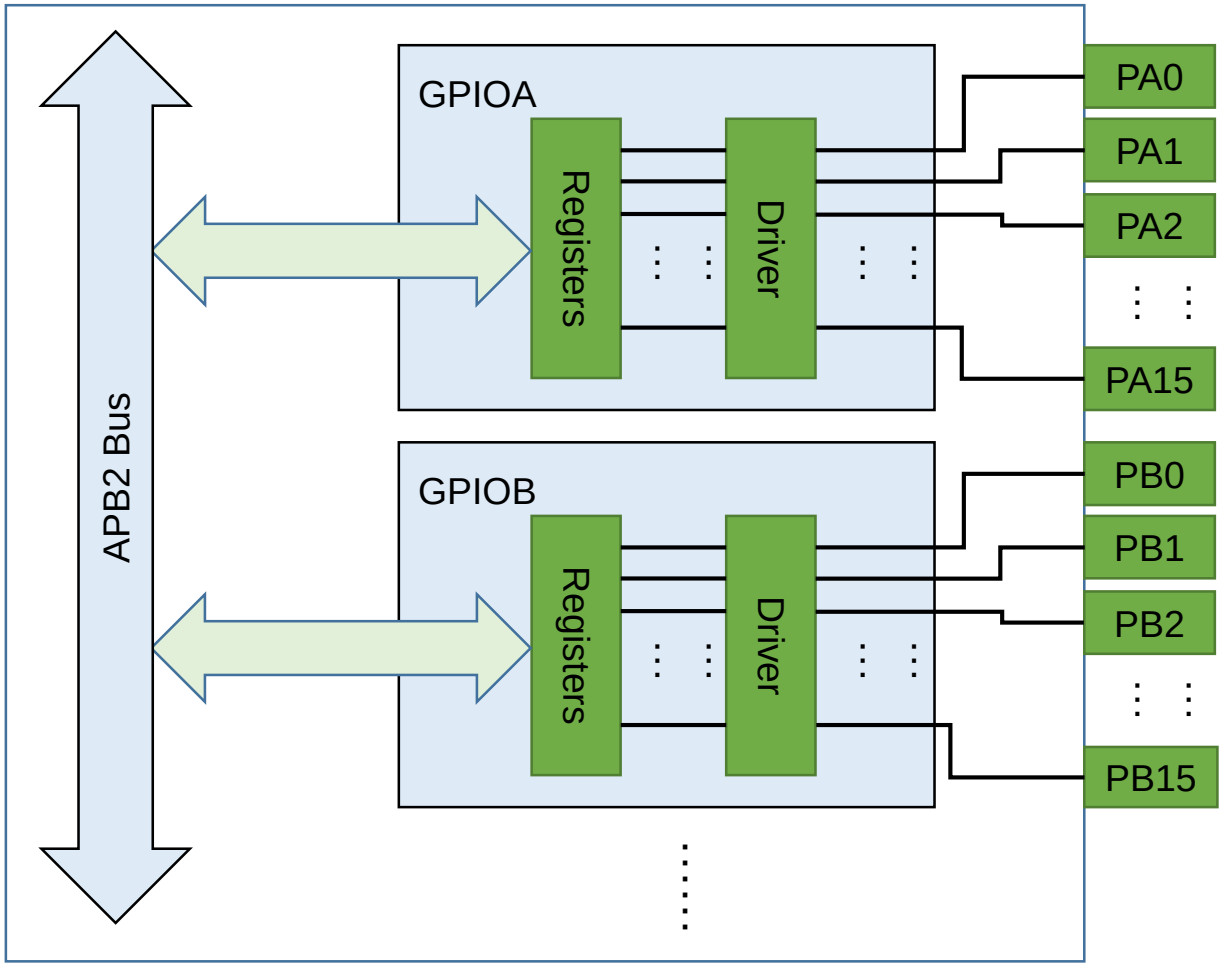
GPIO Register Mapping

- Each port has seven I/O registers associated with it
- Each register has a specific memory address, Register Mapping assigned a name to each register address.



I/O Register	Address
GPIOA_LCKR	0x40010818
GPIOA_BRR	0x40010814
GPIOA_BSRR	0x40010810
GPIOA_ODR	0x4001080C
GPIOA_IDR	0x40010808
GPIOA_CRH	0x40010804
GPIOA_CRL	0x40010800

GPIO Inside


APB2 Bus
GPIOA
Registers
Driver
GPIOB
Registers
Driver
PA0
PA1
PA2
PA15
PB0
PB1
PB2
PB15

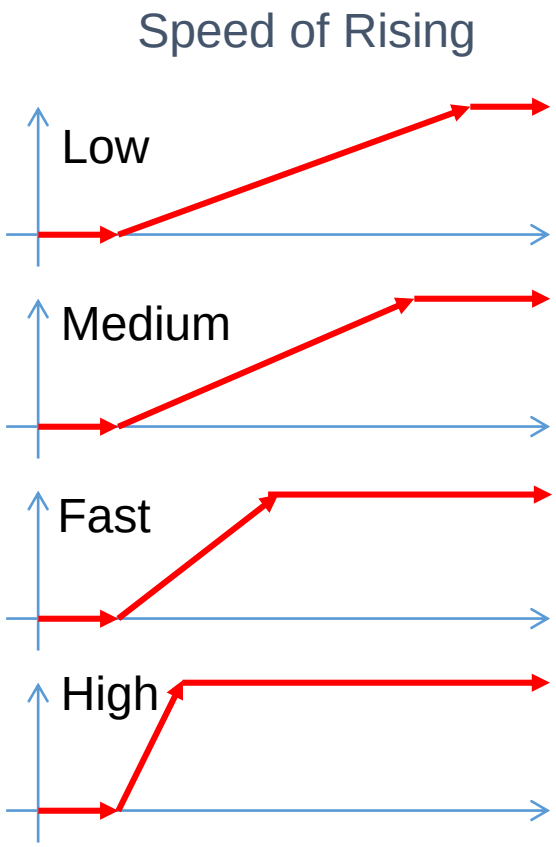
GPIO Mode

- Default: floating input
- Often used: Input with pull-up/down (上拉/下拉输入), output push-pull (推挽输出)

GPIO Mode	Usage
Floating input (reset state)	Completely floating, and the state is undefined
Input with pull-up	With internal pull-up, defaults to high level (button)
Input with pull-down	With internal pull-down, defaults to low level
Analog mode	ADC, DAC
General purpose output Open-drain	Software I2C, SDA, SCL, etc
General purpose output push-pull	Strong driving capability, general-purpose output (LED)
Alternate function output Open-drain	On-chip peripheral functions (hardware I2C, SDA, SCL pins, etc)
Alternate function output Push-pull	On-chip peripheral functions (SPI, SCK, MISO, MOSI pins, etc)

GPIO Output Speed

- Output Speed:
  - Speed of rising and falling
  - Four speeds: Low, Medium, Fast, High
- Tradeoff
  - Higher GPIO speed increases EMI noise and power consumption
  - Configure based on peripheral speed
    - Low speed for toggling LEDs
    - High speed for SPI



Programming GPIO

- Basic Steps of GPIO programming
  - Enable the corresponding GPIO Clock
    - RCC->APB2ENR (GPIO is on APB2 bus)
  - Configure the GPIO Mode
    - Setting **CRL/CRH** to configure input/output mode
  - Set the output status if you are using GPIO as output
    - Setting **ODR** to configure output status
  - Read the input status if you are using GPIO as input
    - Setting **ODR** (to configure input with Pull-up/Pull-Down)
    - Reading from **IDR** (to get the input status)

CRL and CRH (Configuration Registers)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CRH:	CNF15		MODE15		CNF14		MODE14		CNF13		MODE13		CNF12		MODE12	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CNF11		MODE11		CNF10		MODE10		CNF9		MODE9		CNF8		MODE8	

pin15

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CRL:	CNF7		MODE7		CNF6		MODE6		CNF5		MODE5		CNF4		MODE4	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CNF3		MODE3		CNF2		MODE2		CNF1		MODE1		CNF0		MODE0	

pin1 pin0

Default value: 0x4444 4444

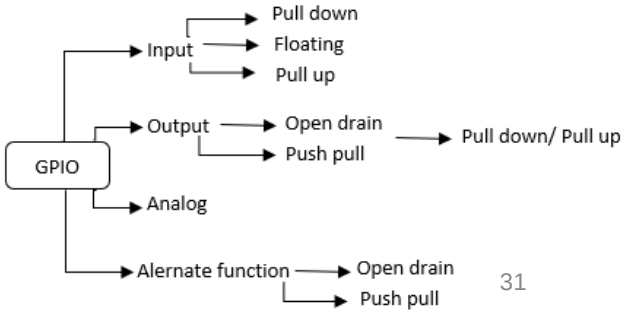
Output (MODE>00)

CNFx bits	
00	General purpose output push-pull
01	General purpose output Open-drain
10	Alternate function output Push-pull
11	Alternate function output Open-drain

MODEx bits	Direction	Max speed
00	Input	
01	Output	10 MHz
10		2 MHz
11		50 MHz

Input (MODE=00)

CNFx bits	Configuration	Description
00	Analog mode	Select this mode when you use a pin as an ADC input.
01	Floating input	In this mode, the pin is high-impedance.
10	Input with pull-up/pull-down	The value of ODR chooses if the pull-up or pull-down resistor is enabled. (1: pull-up, 0:pull-down)
11	reserved	



CRL and CRH

- Common settings:
  - 0x3
    - MODEx 11 → output
    - CNFx 00 → pushpull
  - 0x4 (default)
    - MODEx 00 → input
    - CNFx 01 → Hiz
  - 0x8
    - MODEx 00 → input
    - CNFx 10 → pull-up/pull-down

Output (MODE>00)

CNFx bits	
00	General purpose output push-pull
01	General purpose output Open-drain
10	Alternate function output Push-pull
11	Alternate function output Open-drain

Input (MODE=00)

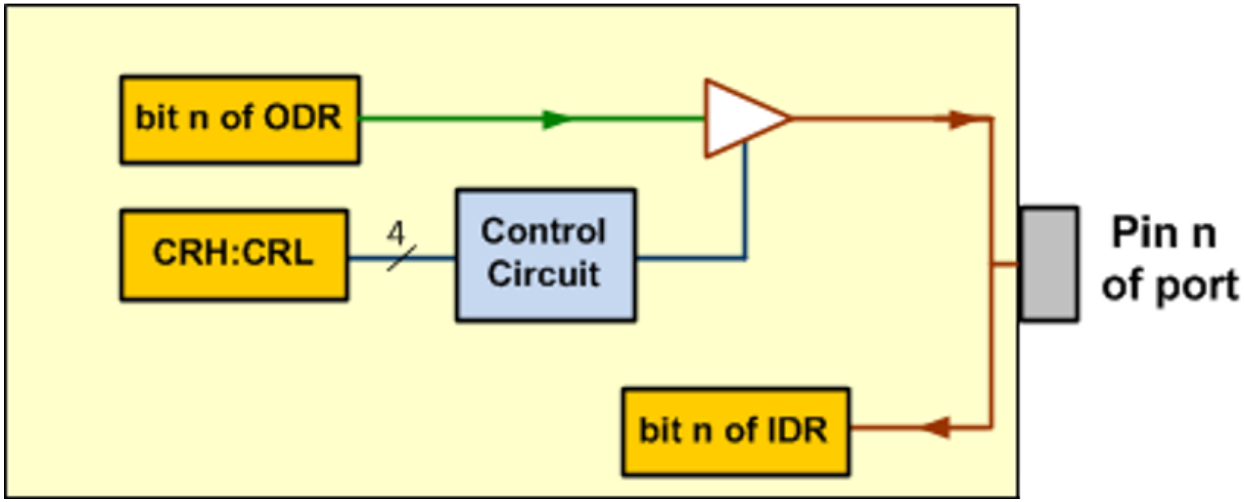
CNFx bits	Configuration	Description
00	Analog mode	Select this mode when you use a pin as an ADC input.
01	Floating input	In this mode, the pin is high-impedance.
10	Input with pull-up/pull-down	The value of ODR chooses if the pull-up or pull-down resistor is enabled. (1: pull-up, 0:pull-down)
11	reserved	

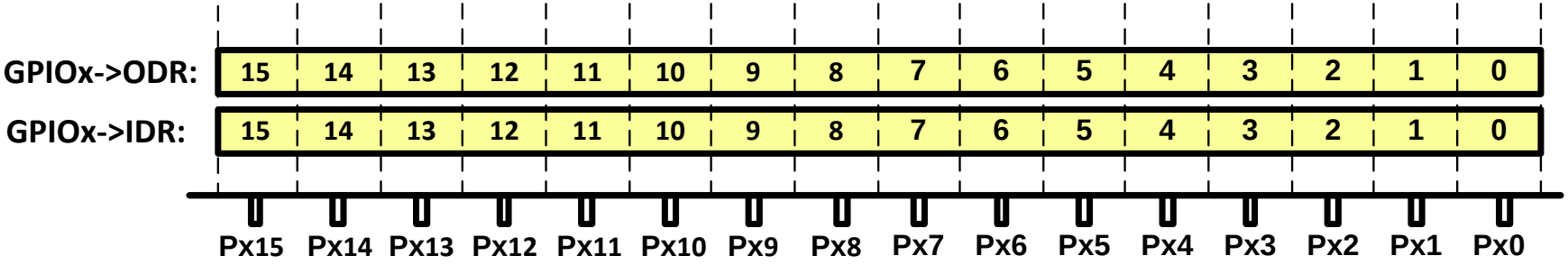
- Example

```
GPIOC->CRH &= 0xFFF00FFF; //clear settings of PC11 and PC12
GPIOC->CRH |= 0x00038000; //PC11 as input, PC12 as output
GPIOC->ODR |= 1<<11;      //set PC11 as input with pull-up
```

&= is often used for clearing bits
|= is often used for setting bits

IDR (Input Data Reg.) and ODR (Output Data Reg.)

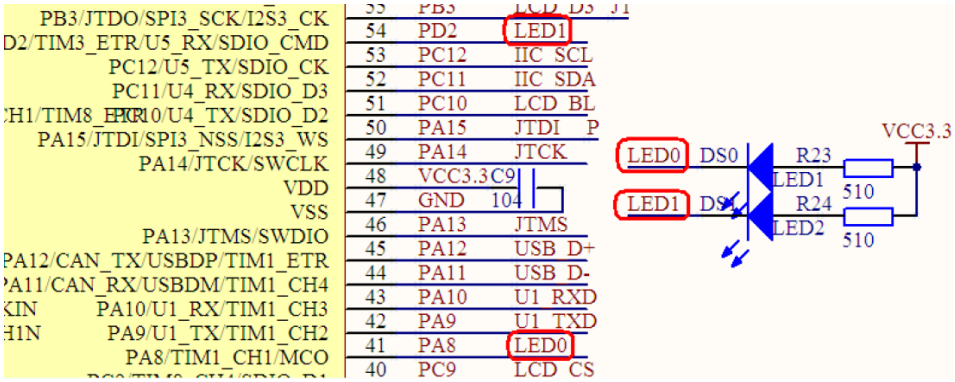




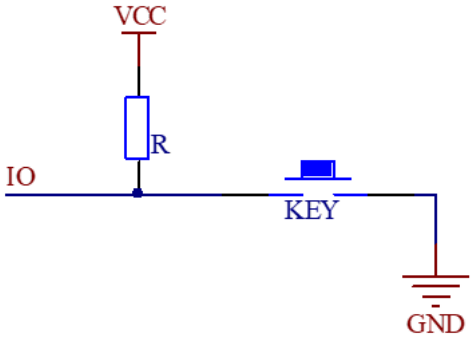

```
GPIOC->CRH &= 0x0FFFFFFF; //clear settings pf PC15
GPIOC->CRH |= 0x30000000; //set PC15 as output
GPIOC->ODR |= 1<<15;      //Set output pin PC15 as high
```

IDR and ODR

- For LED0 and LED1 in STM32, should we set ODR bit to low or high in order to turn on the LED?

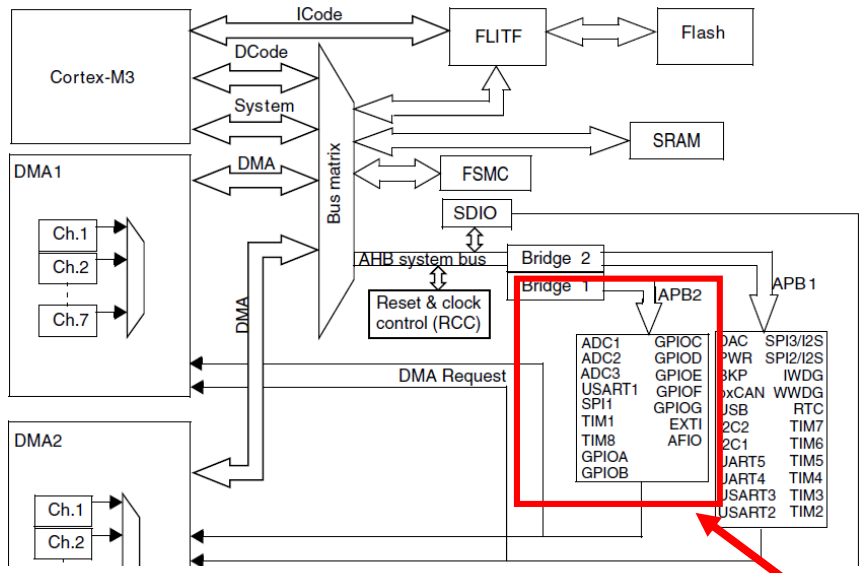


- For KEY in STM32, should we set pull-up or pull-down input mode?



Enabling Clocks

- Example:
  - RCC->APB2ENR |= 1<<4; /* Enable clocks for GPIO C ports */
  - RCC->APB2ENR |= 0xFC; /* Enable clocks for all GPIO ports */



RCC_APB2ENR:
(RCC->APB2ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved										TIM11 EN	TIM10 EN	TIM9 EN	Res.		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USART 1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN

IOPx: I/O port x clock enable

Sample Code

- Toggling PA2

```
void delay_ms(uint32_t t);

int main()
{
    /* System clock must be initialed before */
    RCC->APB2ENR |= 0xFC; /* Enable clocks for GPIO
ports */
    GPIOA->CRL = 0x44444344; /* PA2 as output */
    while(1)
    {
        GPIOA->ODR ^= (1<<2); /* toggle PA2 */
        delay_ms(1000);
    }
}
```

^= is often used for toggling bits