

CS301

Embedded System and Microcomputer Principle

Lecture 2: STM32 MCU & GPIO

2024 Fall

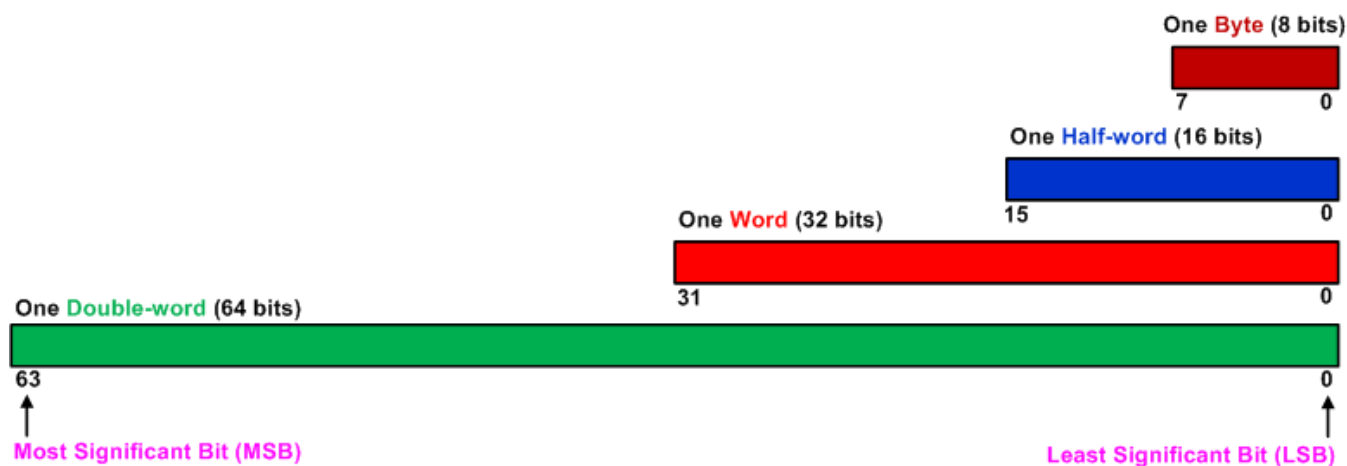
This PowerPoint is for internal use only at Southern University of Science and Technology.
Please do not repost it on other platforms without permission from the instructor.

Outline

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

Value notions

- For a 32 bit processor like the ARM
 - a word is equal to 32 bits or 4 bytes
 - a 'half word' is 16 bits or 2 bytes



Power	Meaning	Prefix	Symbol
2^{10}	1024	Kilo	K
2^{20}	1024^2	Mega	M
2^{30}	1024^3	Giga	G
2^{40}	1024^4	Tera	T
2^{50}	1024^5	Peta	P
2^{60}	1024^6	Exa	E
2^{70}	1024^7	Zetta	Z

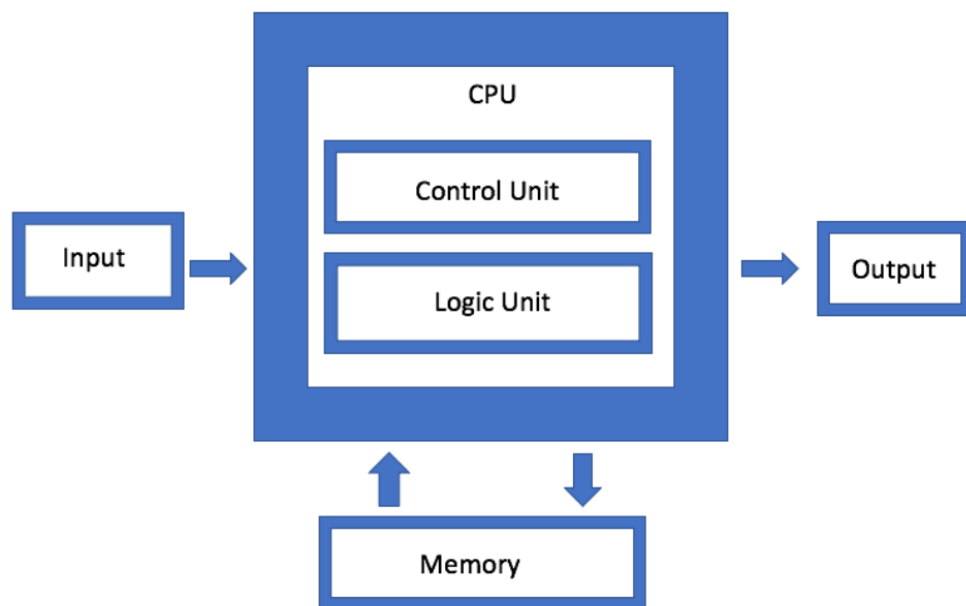
Number System

- Radix-r to Decimal Conversion:
 - In positional number systems: Let r be the radix (or base)
 - then the $(n+m)$ -digit number: $D = \sum_{i=-m}^{n-1} d_i r^i$
- Decimal to Radix-r Conversion:
 - Integer part: Successive divisions by r and observe the remainders
 - Fraction: Successive multiplications by r and observe the integer part
- Three ways to represent signed binary integers:
 - Signed magnitude
 - One's complement
 - Two's complement (used in the modern computers)
 - Example: to represent ± 7 in 8-bit binary
 - $+7_{10} = 00000111_2$
 - $-7_{10} = 11111001_2$ (One's complement plus one)

Computer Architecture

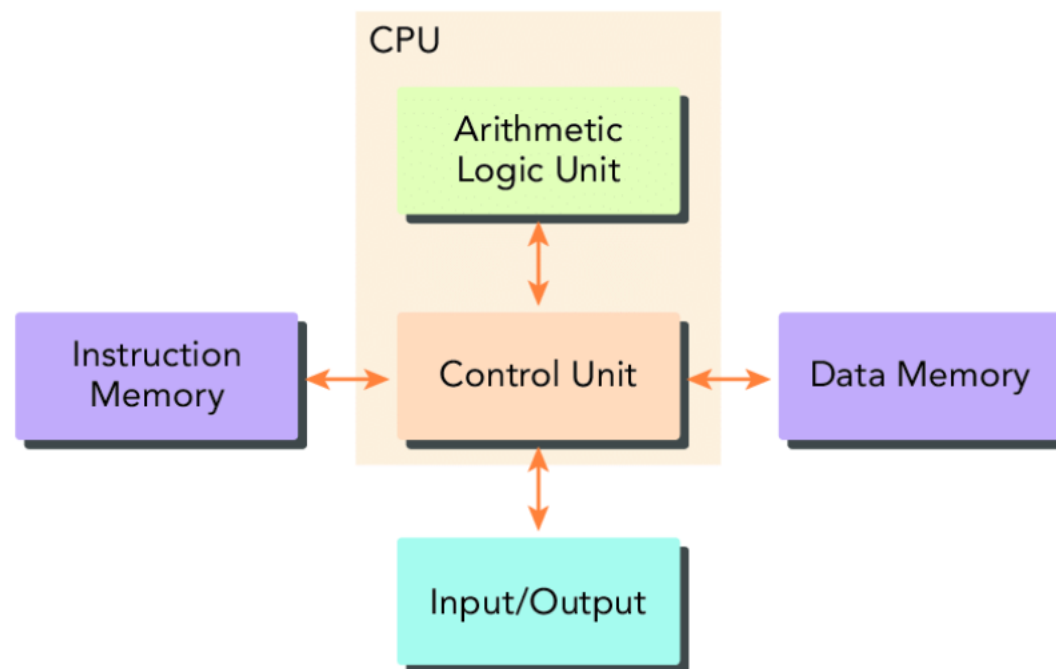
Von-Neumann

Instructions and data are stored in the same memory.



Harvard

Data and instructions are stored into separate memories.



Levels of Program Code

C Program

```
int main(void){  
    int i;  
    int total = 0;  
    for (i = 0; i < 10; i++) {  
        total += i;  
    }  
    while(1); // Dead loop  
}
```

Compile

Assembly Program

```
        MOVS r1, #0  
        MOVS r0, #0  
        B     check  
loop    ADD  r1, r1, r0  
        ADDS r0, r0, #1  
check   CMP  r0, #10  
        BLT  loop  
self    B     self
```

Assemble

Machine Program

```
0010000100000000  
0010000000000000  
1110000000000001  
0100010000000001  
0001110001000000  
0010100000001010  
1101110011111011  
1011111100000000  
1110011111111110
```

▶ High-level language

- ▶ Level of abstraction closer to problem domain
- ▶ Provides for productivity and portability

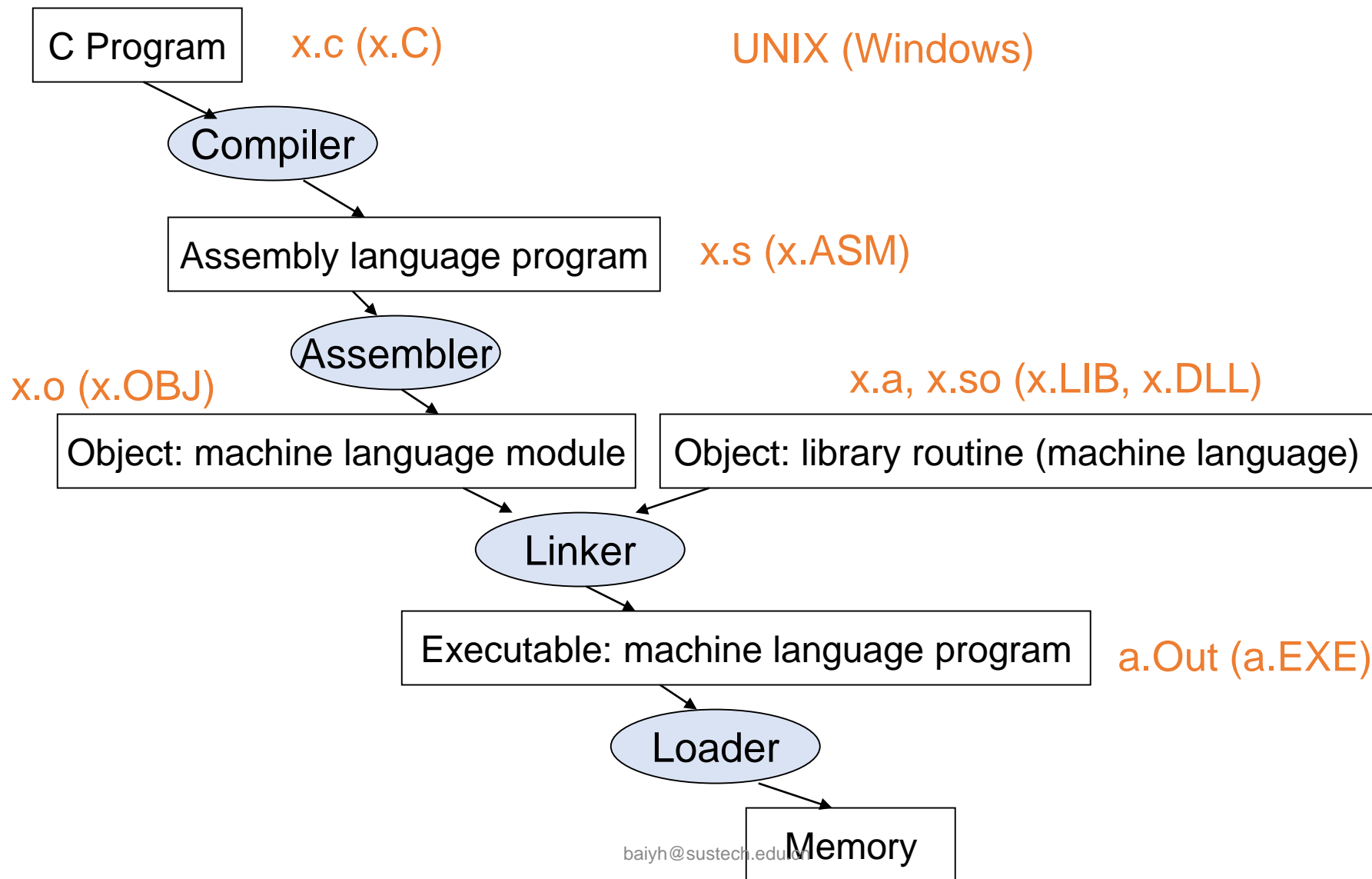
▶ Assembly language

- ▶ Textual representation of instructions
- ▶ Human-readable format instructions

▶ Hardware representation

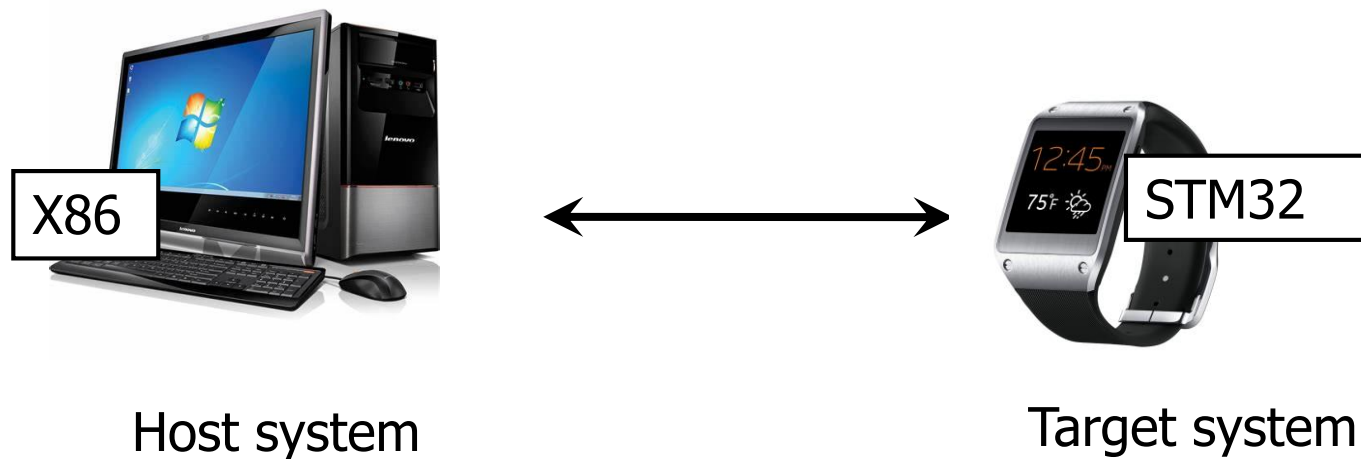
- ▶ Binary digits (bits)
- ▶ Encoded instructions and data
- ▶ Computer-readable format instructions

Running a Program in General Computer



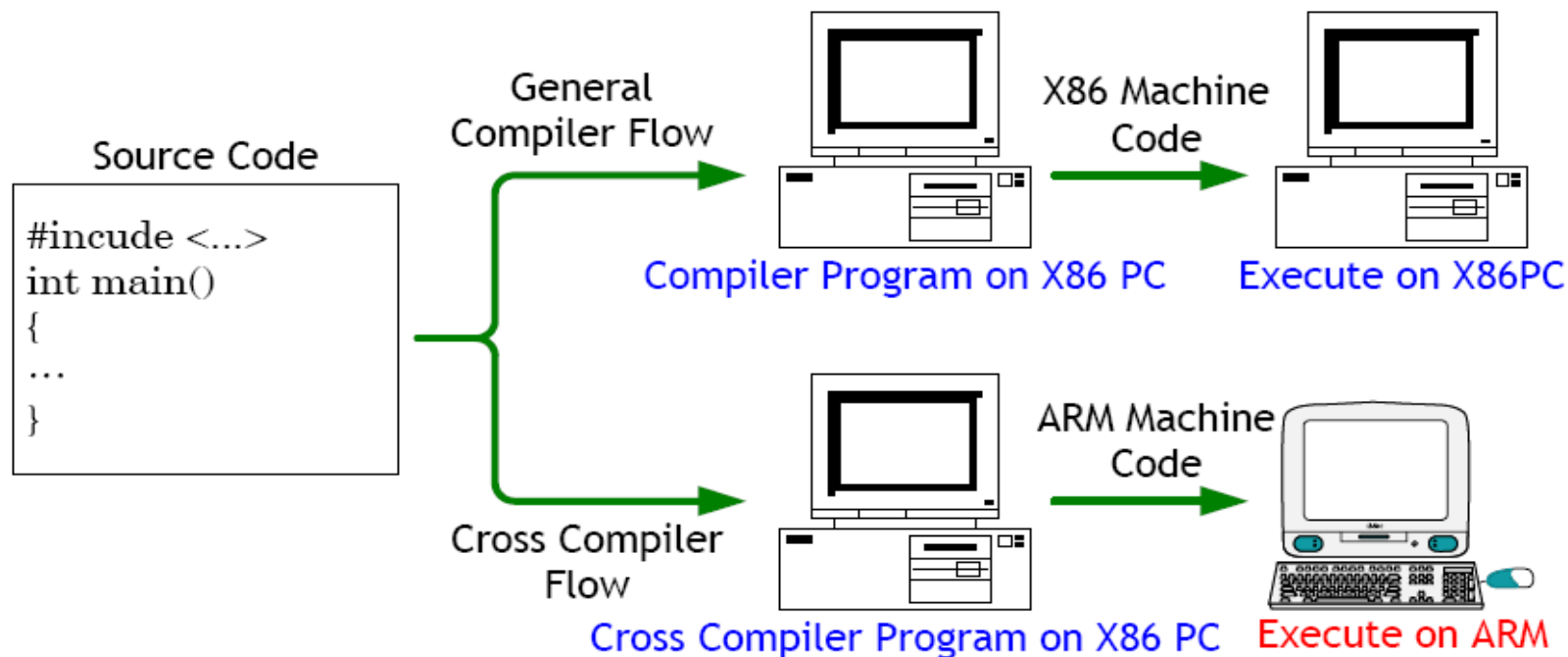
MCU Development Environment

- Host: a computer running programming tools for development of the programs
- Target: the HW on which code will run
- After program is written, compiled, assembled and linked, it is transferred to the target



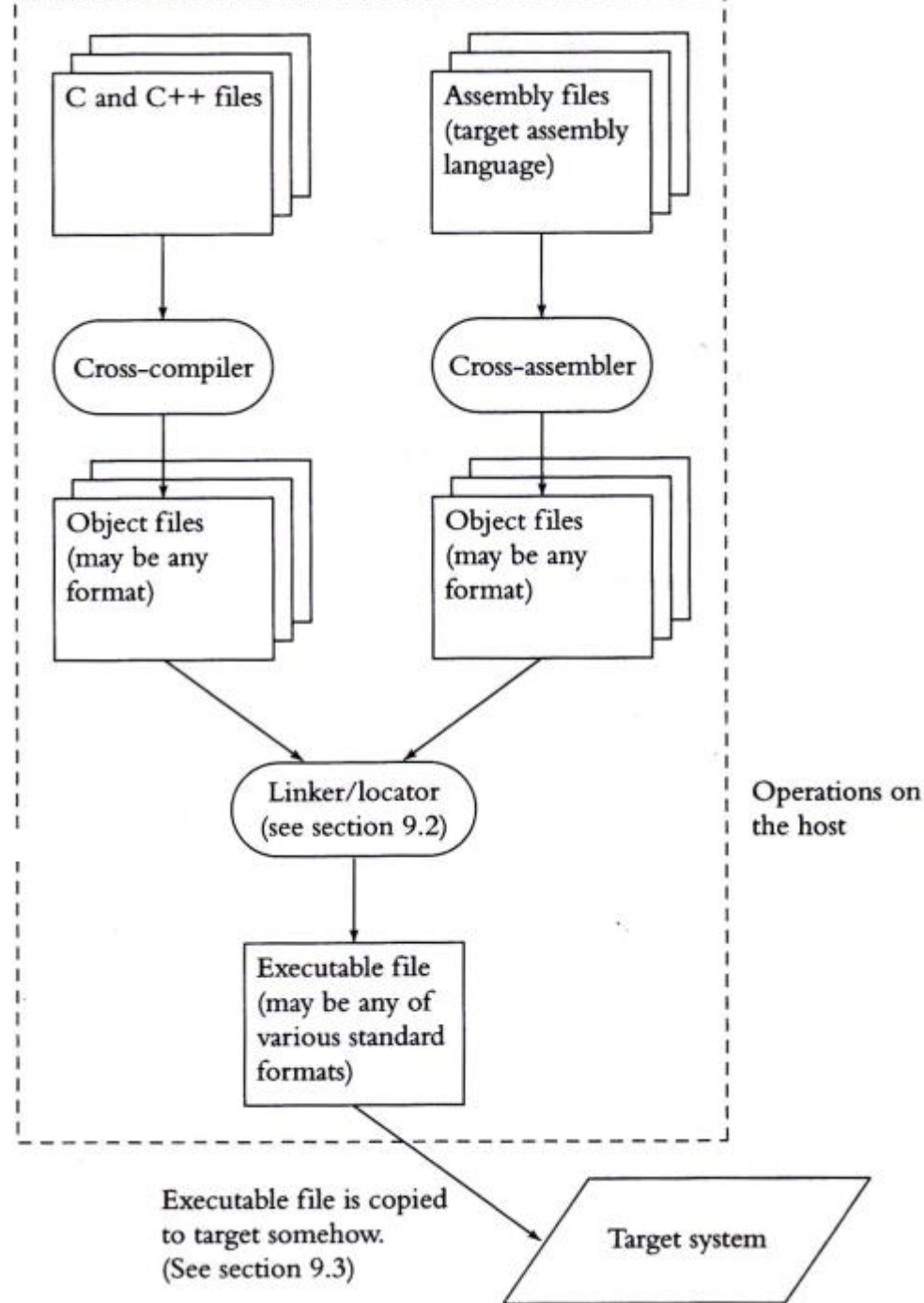
Cross Compiler

- Runs on host but generates code for target
 - Target usually have different architecture from host. Hence compiler on host has to produce binary instructions that will be understood by target



Cross Compilation Process

- Cross Compiler
- Cross Assembler
- Linker
- Locator
 - For embedded systems: creates a file, containing binary image or other format, that will be copied onto target, which run on its own (not through loader)

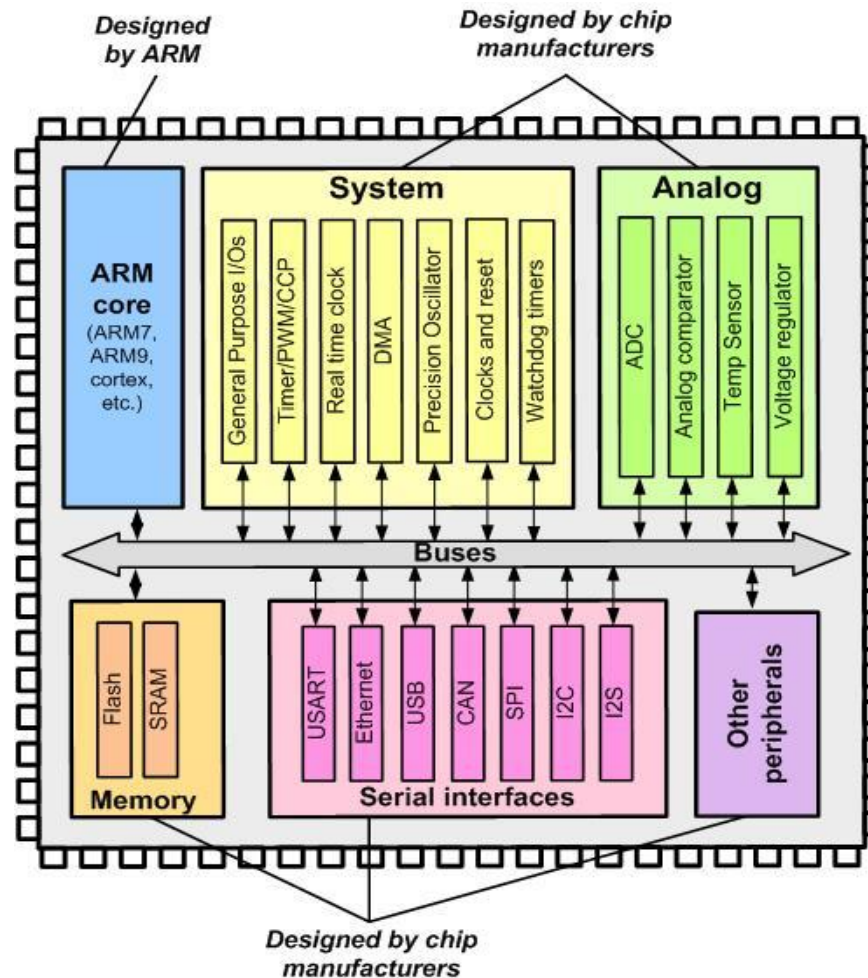


Outline

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

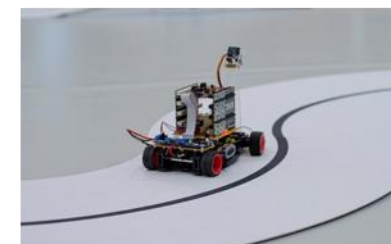
ARM MCU inside view

- The ARM microprocessor + Different peripherals manufactured by chip designers

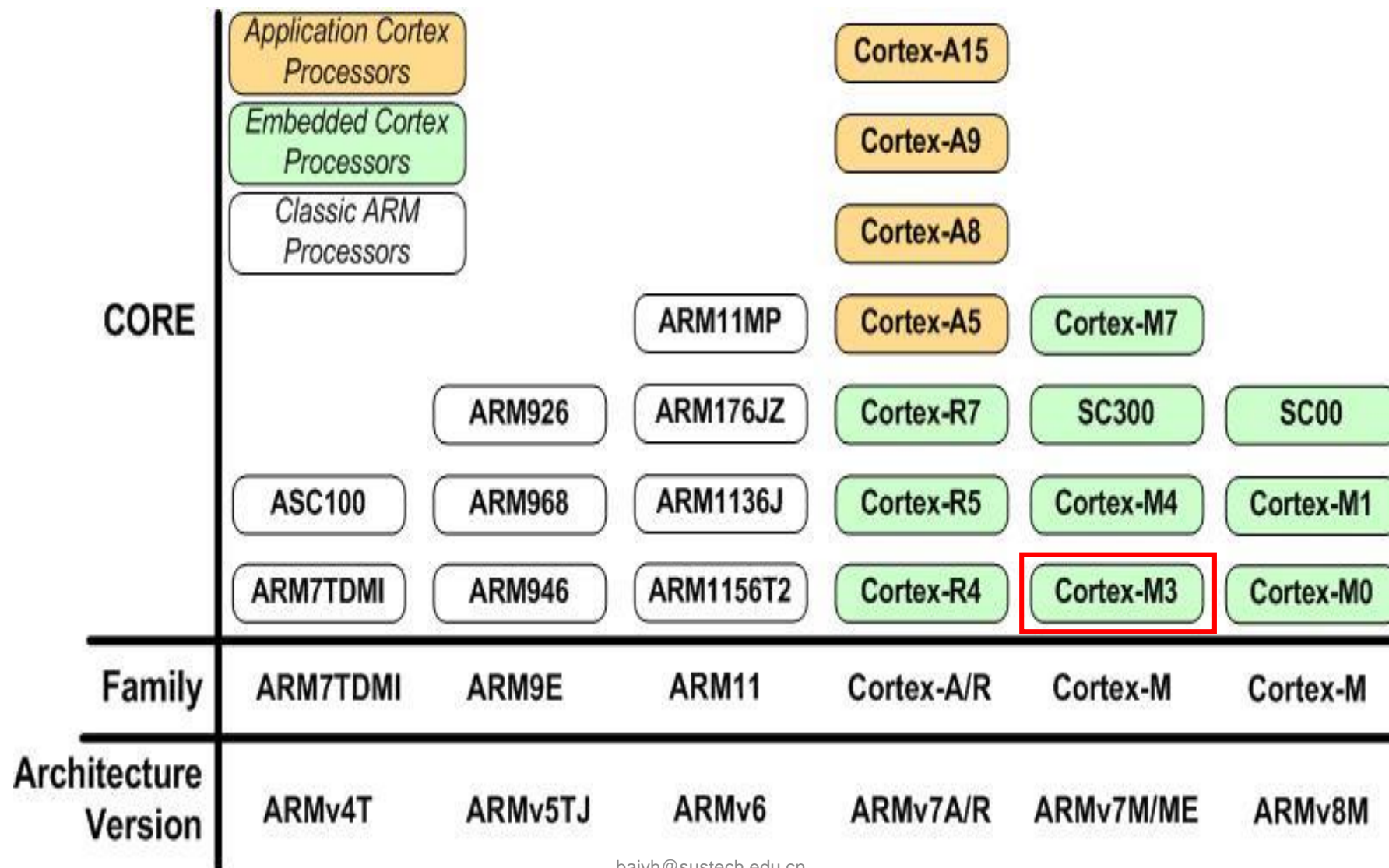


ARM Microprocessor

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



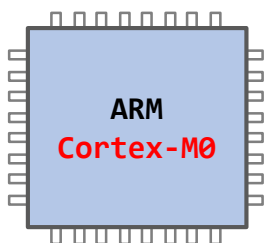
Arm families and Architectures



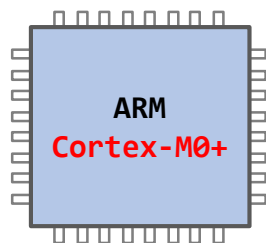
ARM Cortex-M Series Family

Von-Neumann

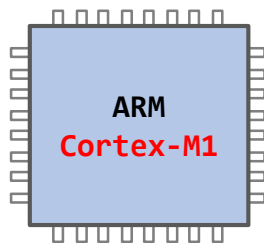
Instructions and data are stored in the same memory.



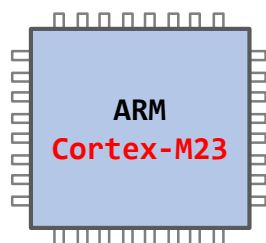
ARMv6-M



ARMv6-M



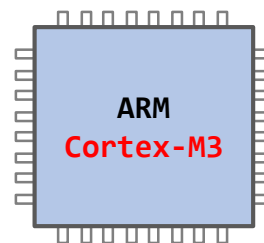
ARMv6-M



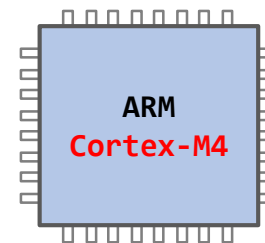
ARMv8-M

Harvard

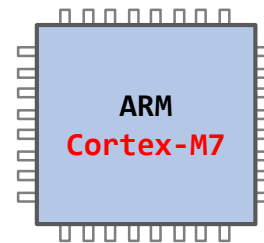
Data and instructions are stored into separate memories.



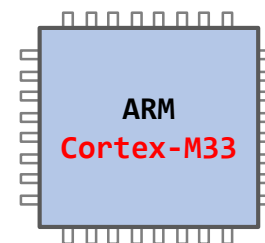
ARMv7-M



ARMv7E-M



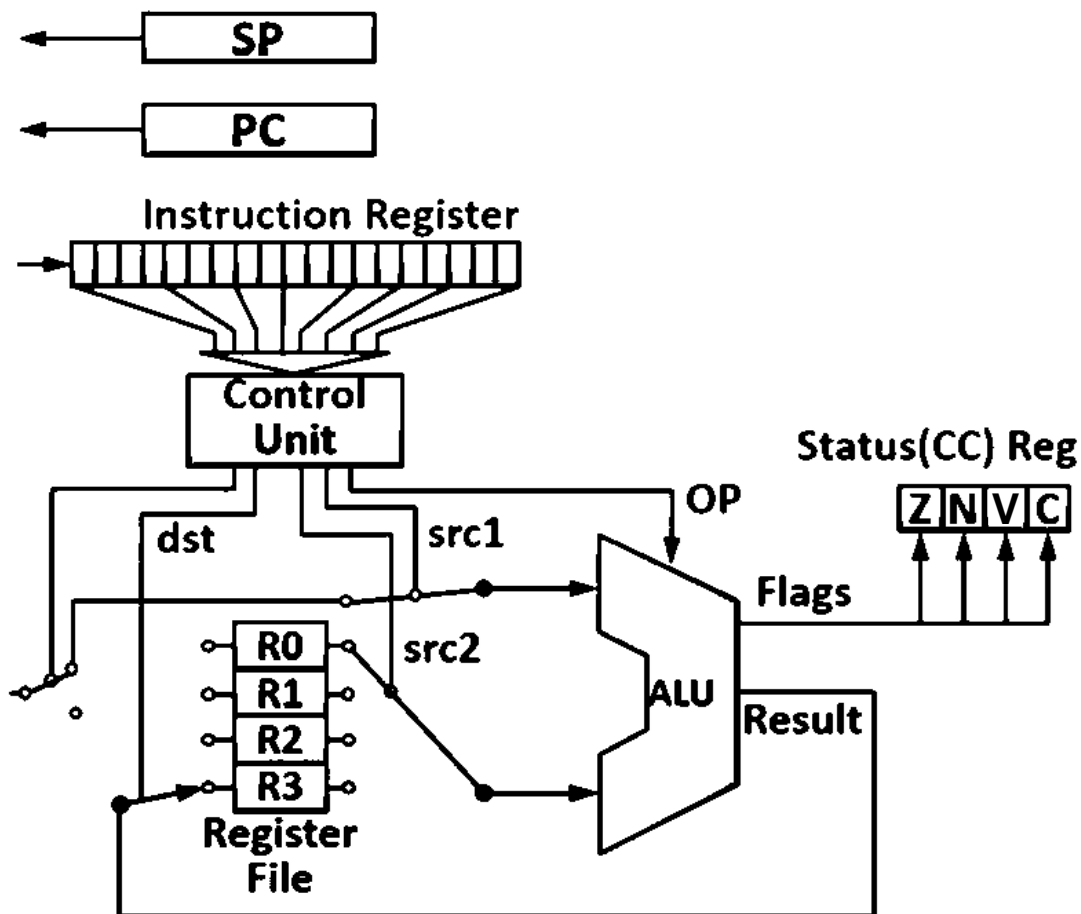
ARMv7E-M



ARMv8-M

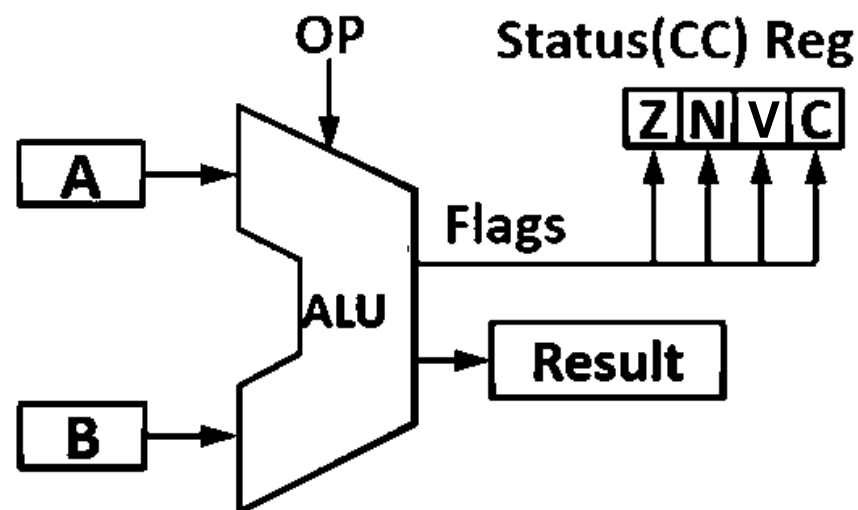
A Complete CPU

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



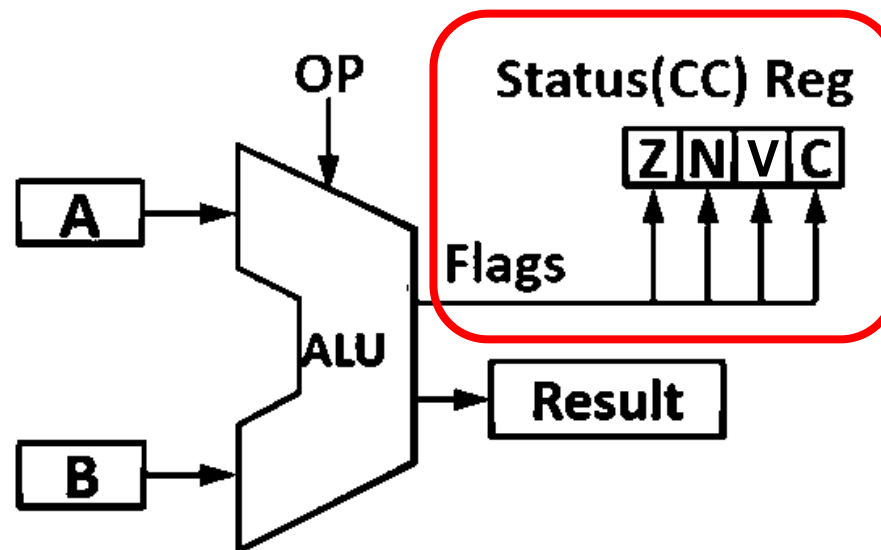
ALU

- 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



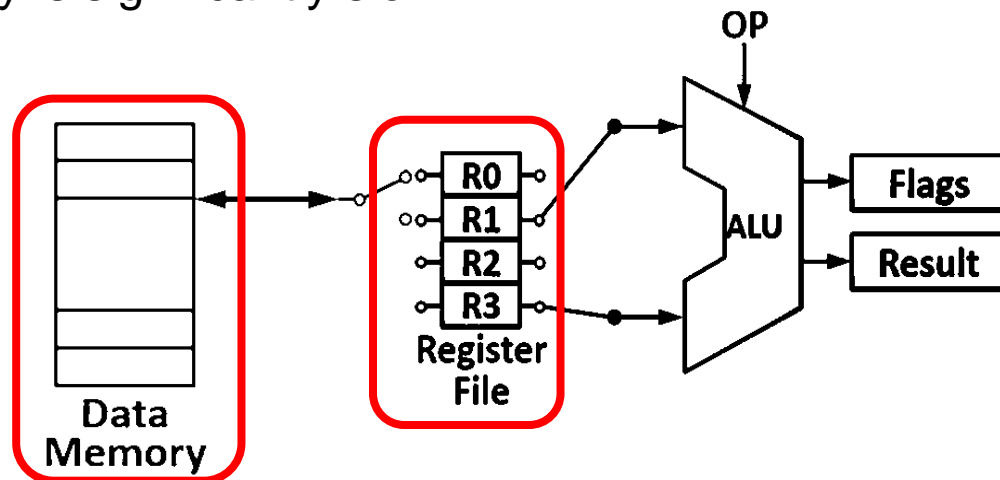
ALU Flags

- 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



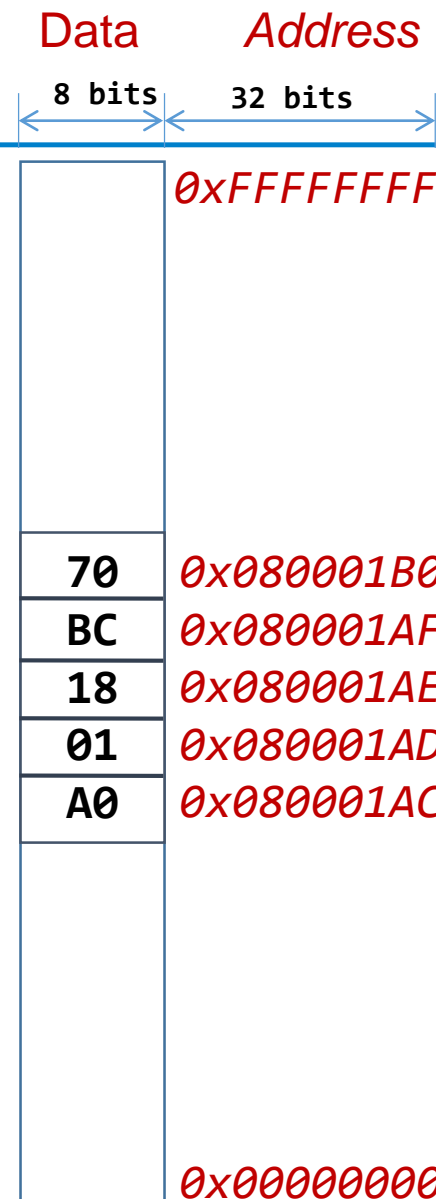
ALU Operands

- 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

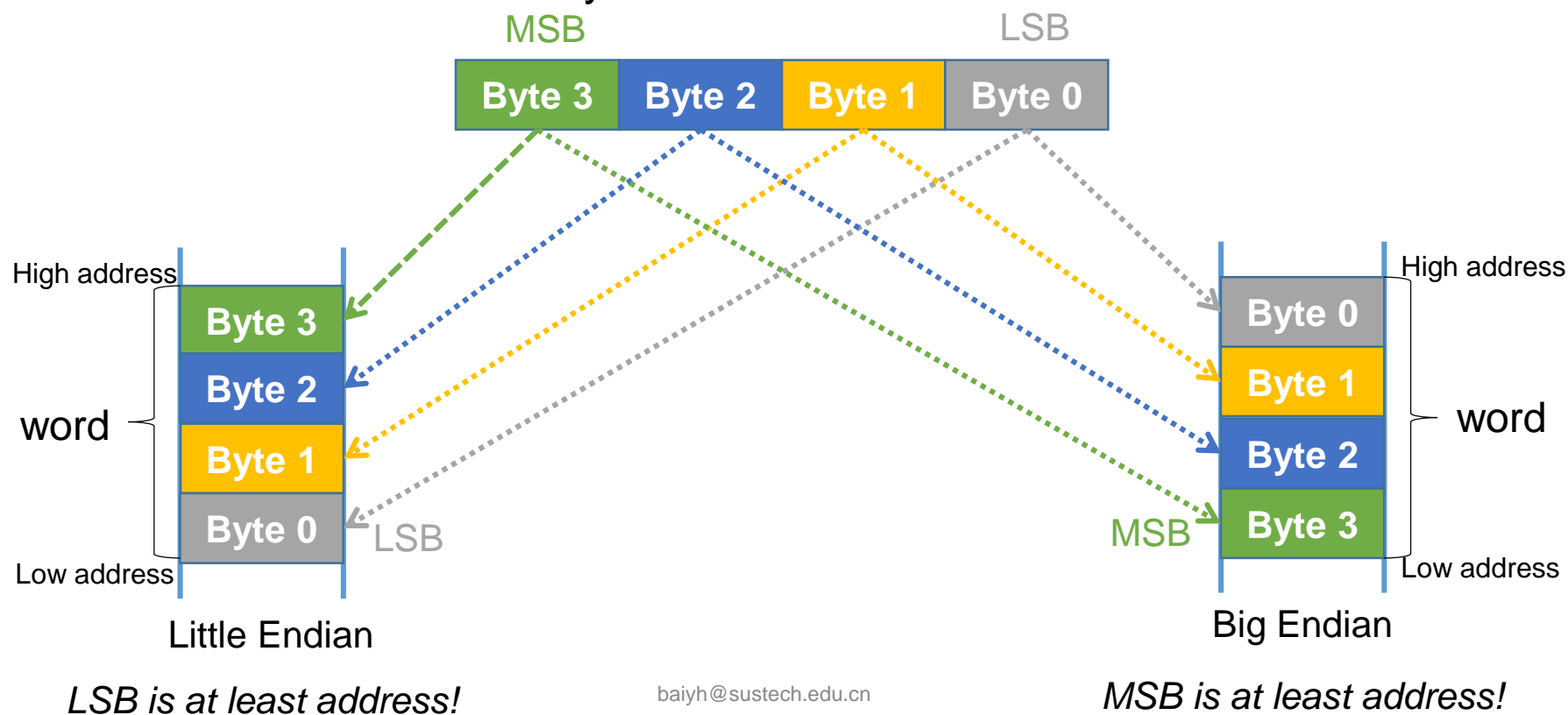
- Memory is arranged as a series of “locations”
 - Each location has a unique “address”
 - Each location holds a byte (**byte-addressable**)
 - e.g. the memory location at address 0x080001B0 contains the byte value 0x70, i.e., 112
- 32-bit Address line
 - Max size: $2^{32} = 4\text{G}$ (bytes)
 - Address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



Memory

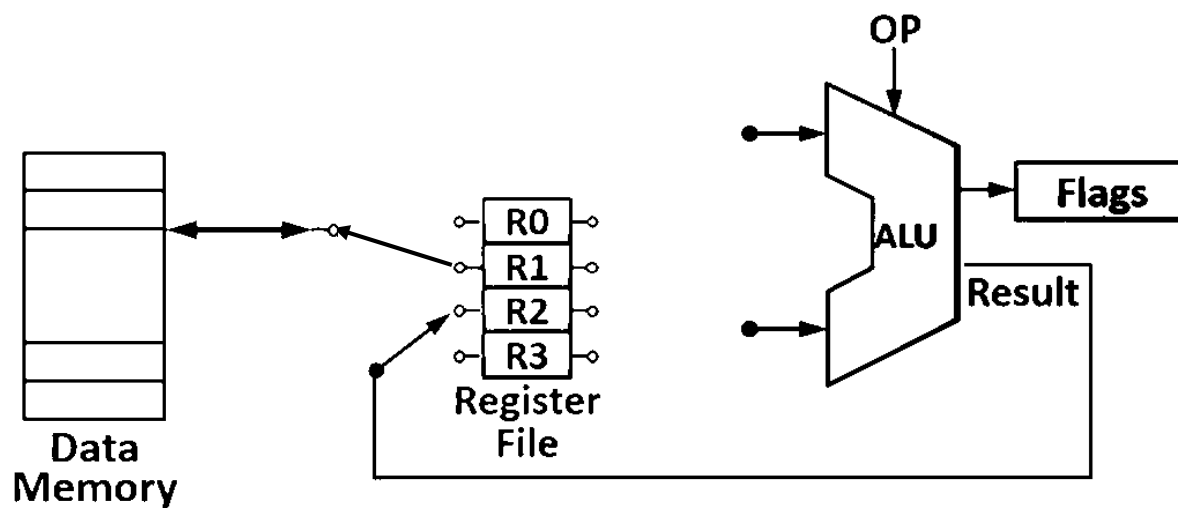
Little Endian vs Big Endian

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



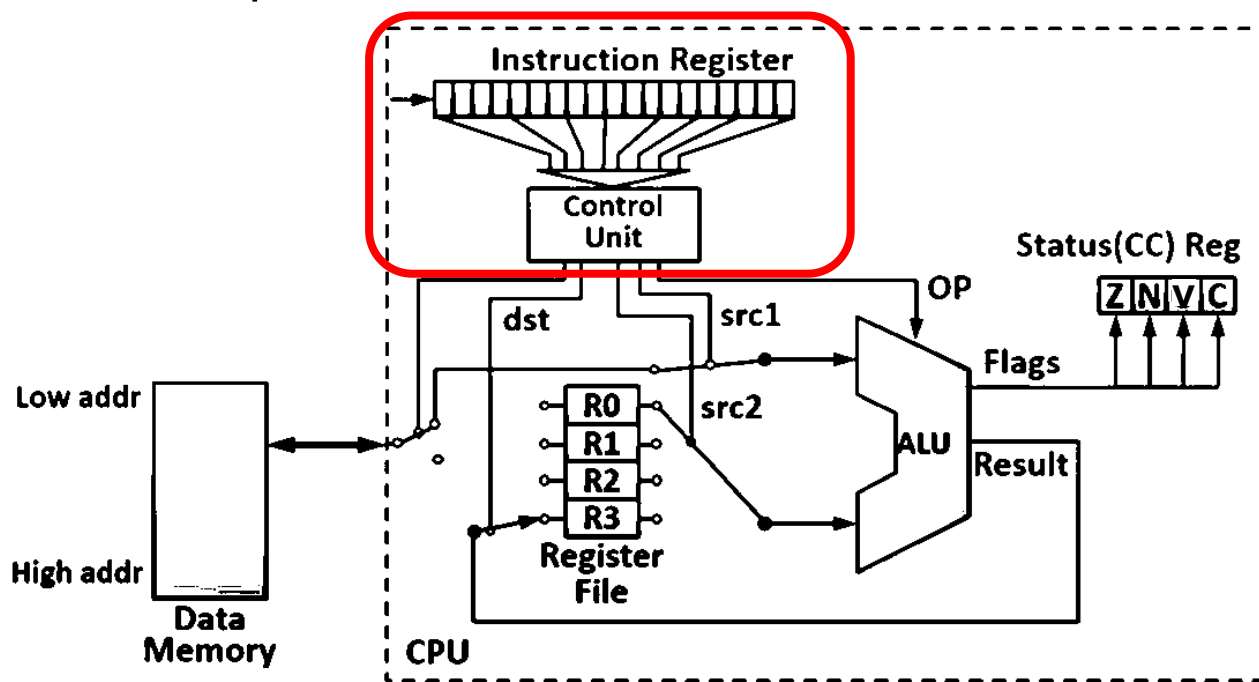
ALU Result

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



Control Unit

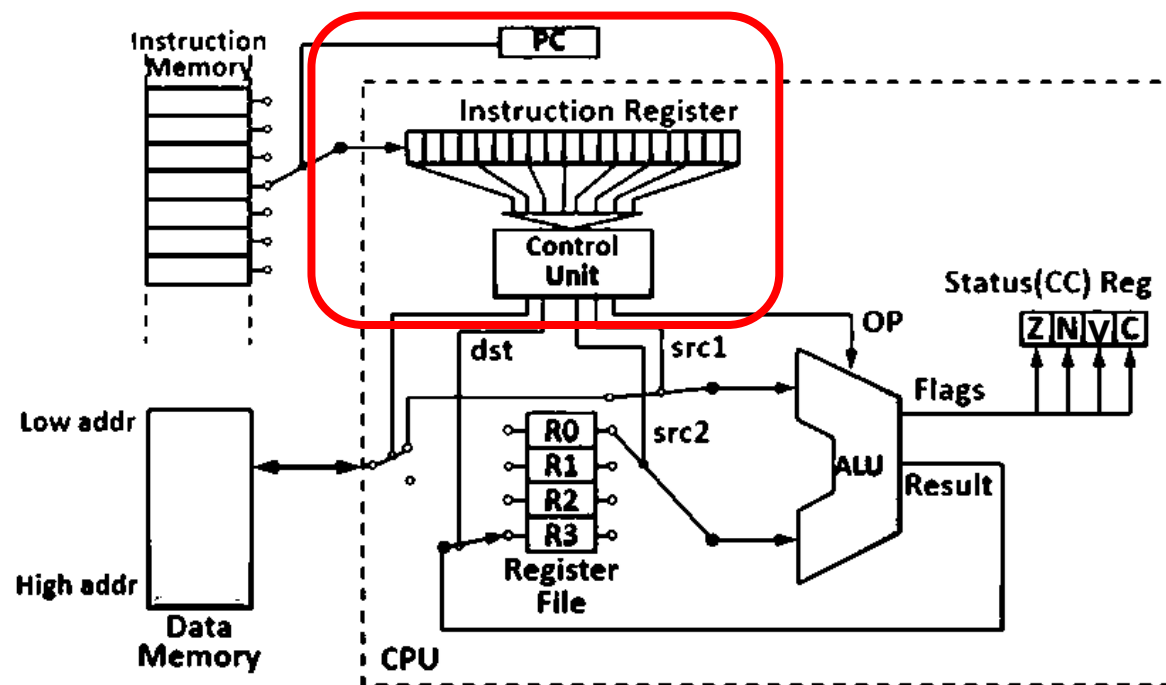
- Instruction Decoding
 - Analyz the Operation to Be Executed by the Instruction
- Dataflow
 - Identify the Sources of Operands and the Destination of the Result for the Instruction



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.

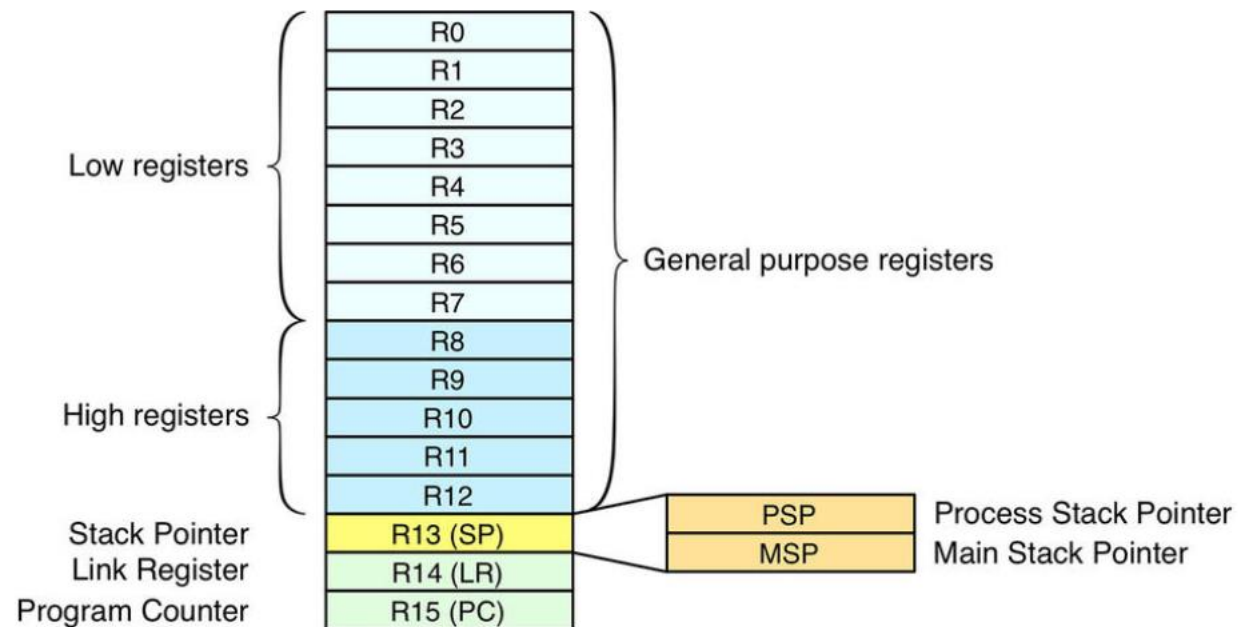


Outline

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

ARM 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**

Special registers

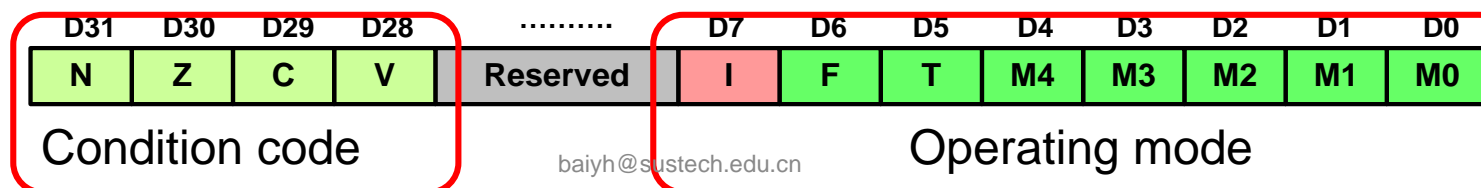
PSR

Program Status Register

CPU Registers

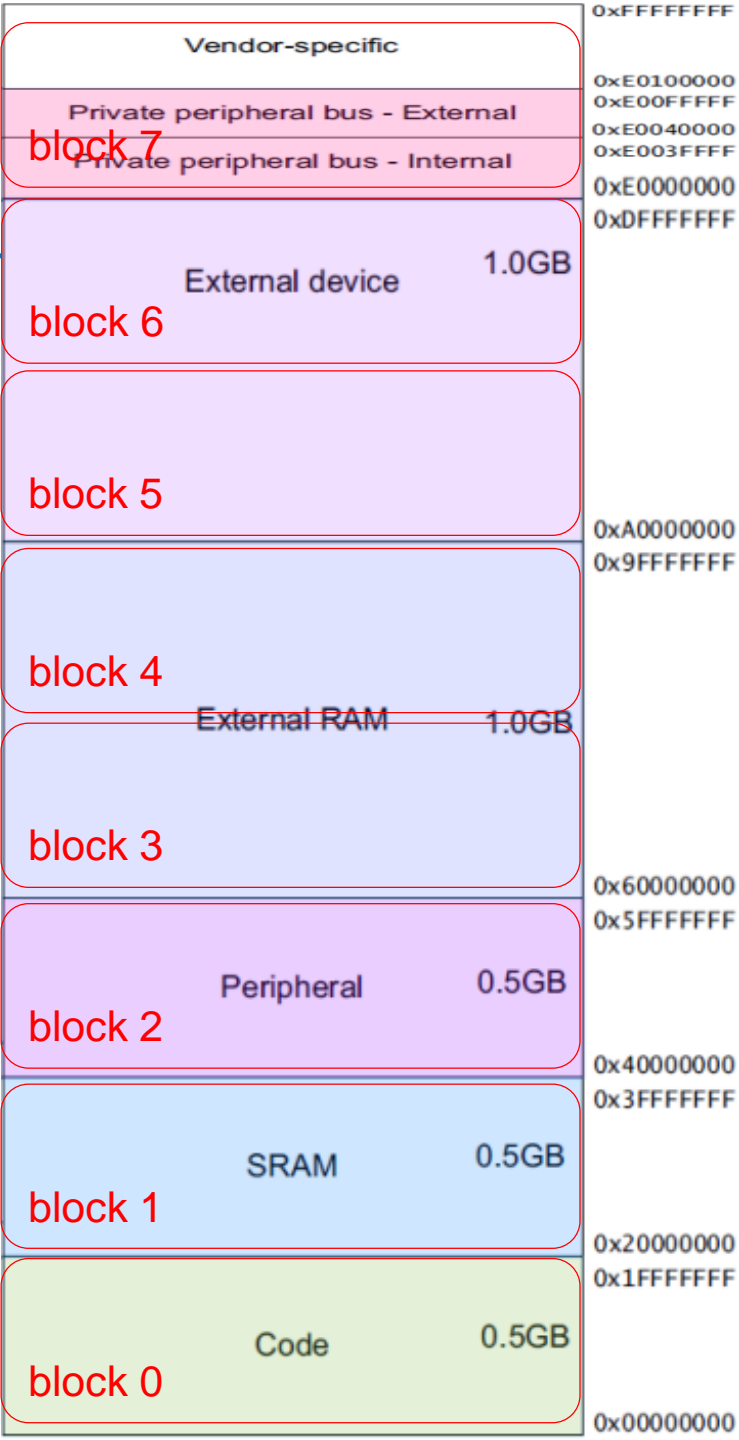
- Low Registers (R0 – R7)
 - Can be accessed by any instruction
- High Register (R8 – R12)
 - Can only be accessed by some instructions
- Stack Pointer (R13)
 - Cortex-M3 supports two stacks
 - Main SP (MSP) for privileged access (e.g. exception handler)
 - Process SP (PSP) for application access
- Link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

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



Memory Map

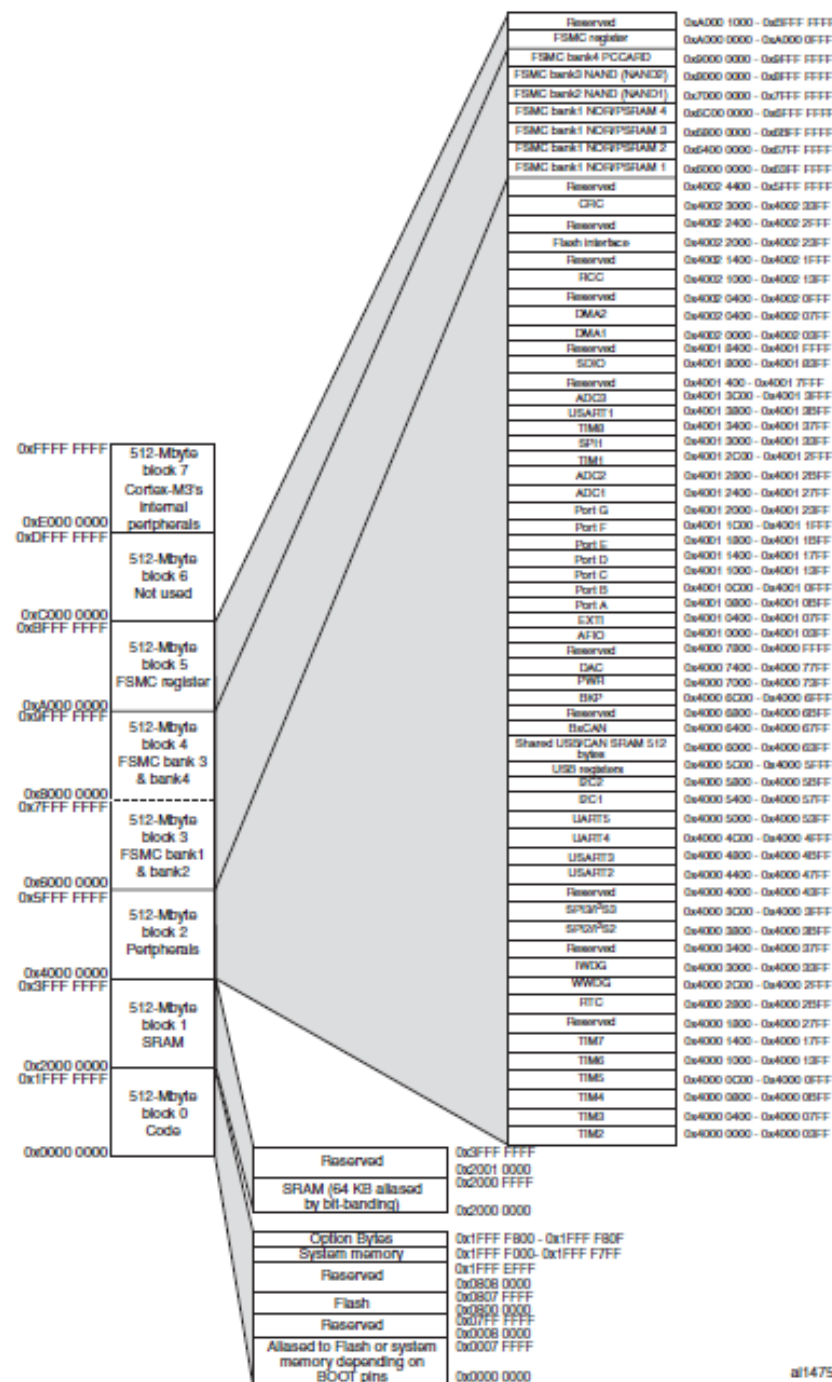
- STM32 Memory is mapped into 8 blocks, each having 512 MB
- We specially take care about the following blocks.
 - Code
 - SRAM
 - Peripheral



Memory Mapped IO

- A technique where both memory and I/O devices use the same address space
 - Memory addresses are assigned to I/O devices.
 - The CPU treats I/O devices like computer memory.
 - Depending on the address, the CPU communicates with either computer memory or I/O devices.

STM32F103 Datasheet, Figure 9

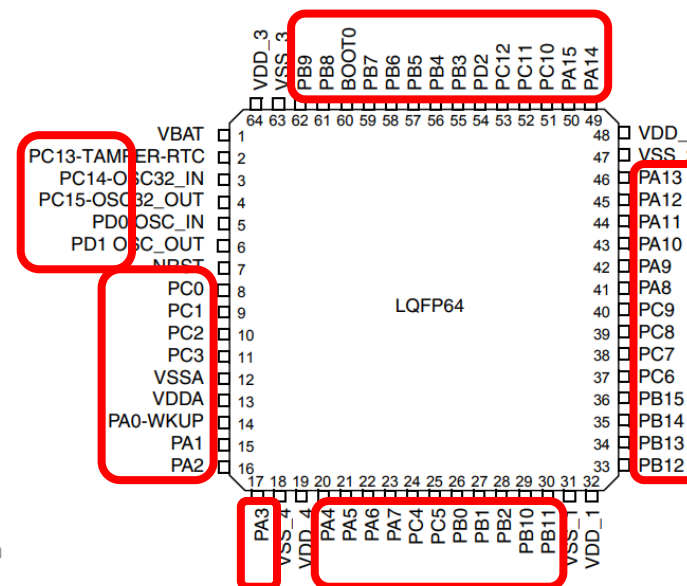
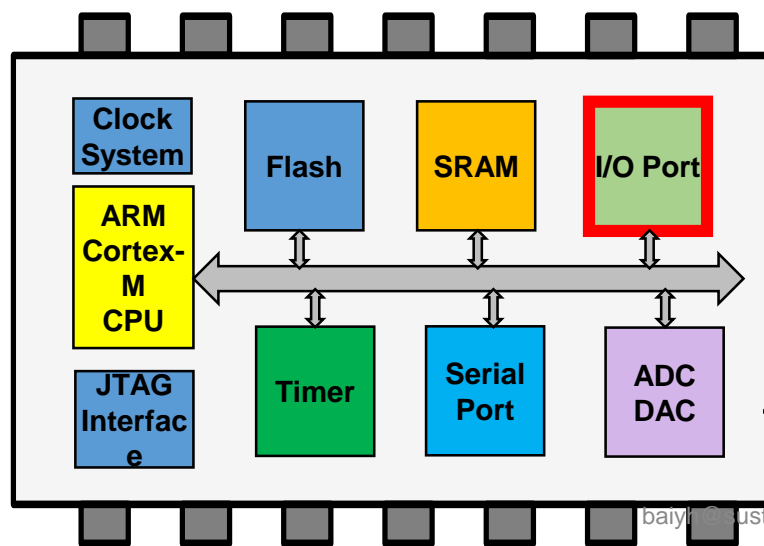


Outline

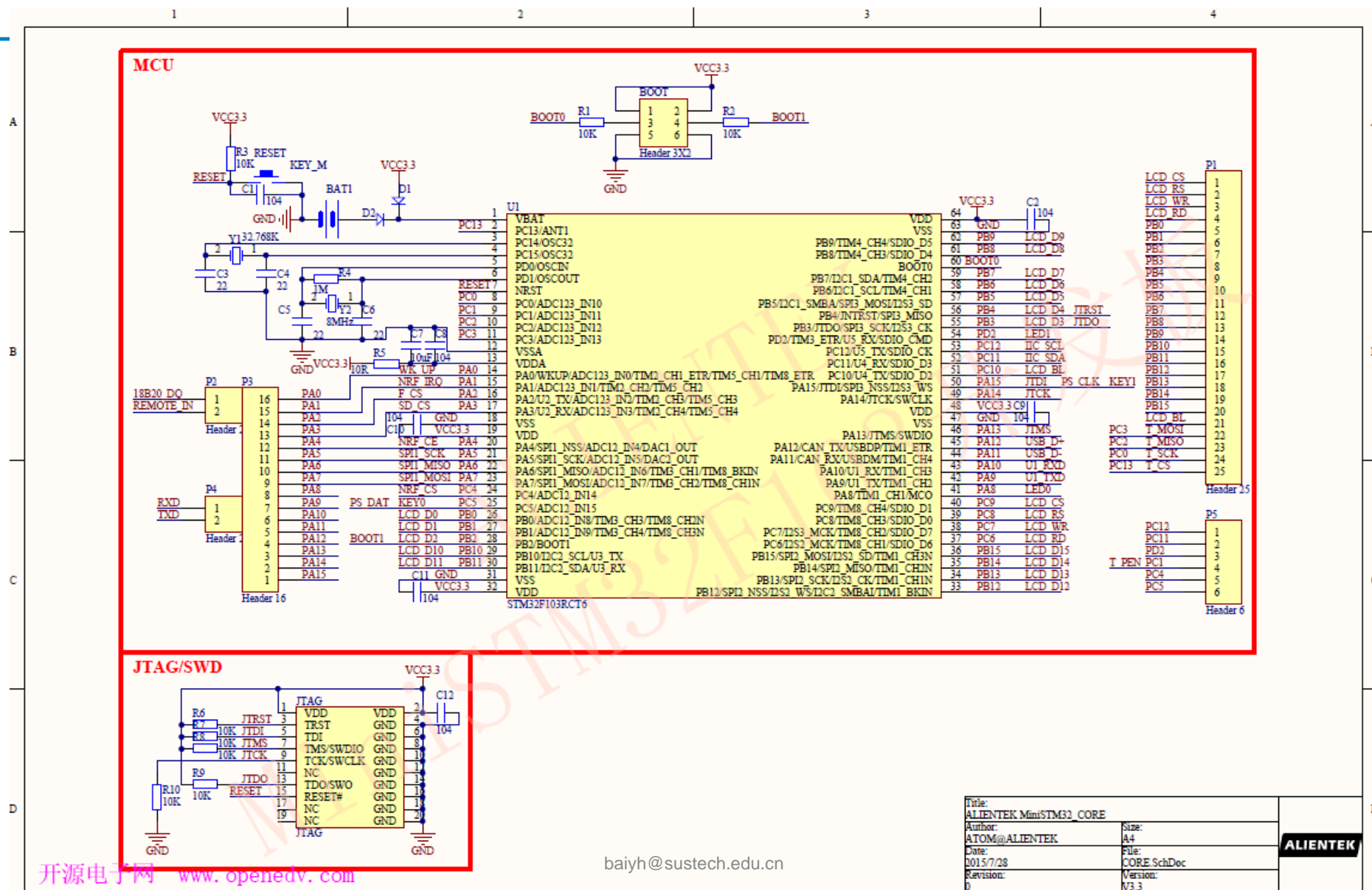
- Recall
- CPU Overview
- CPU Registers & Memory Map
- **GPIO**

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)

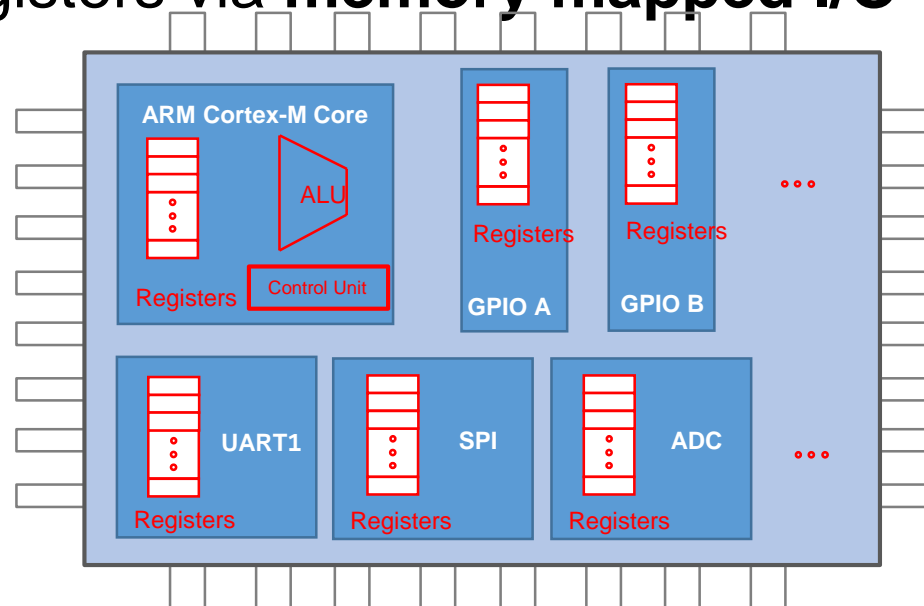


STM32F103 Schematic



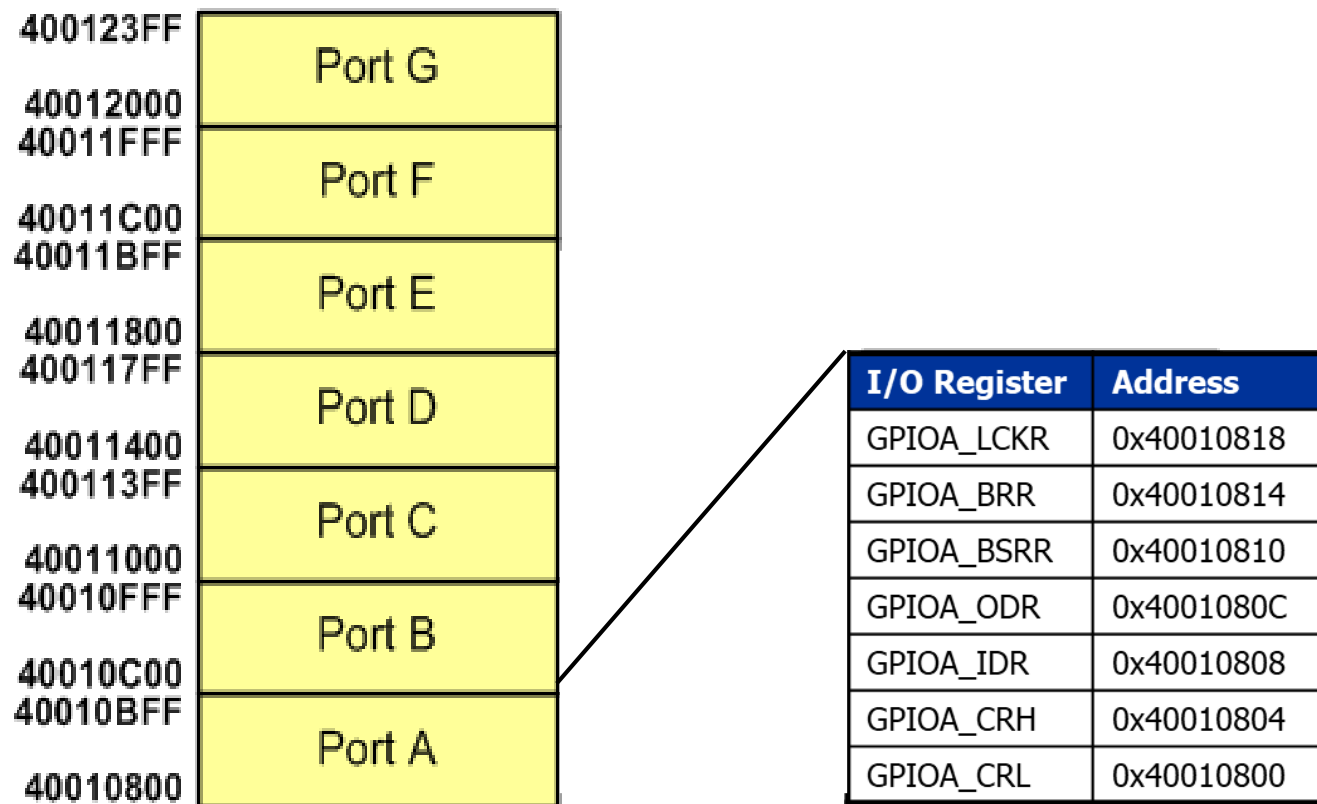
CPU Registers vs GPIO Registers

- Processor can directly access processor registers
 - `ADD r3,r1,r0 ; r3 = r1 + r0`
 - `LDR R0,[R1] ; load value at memory location R1 into 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)

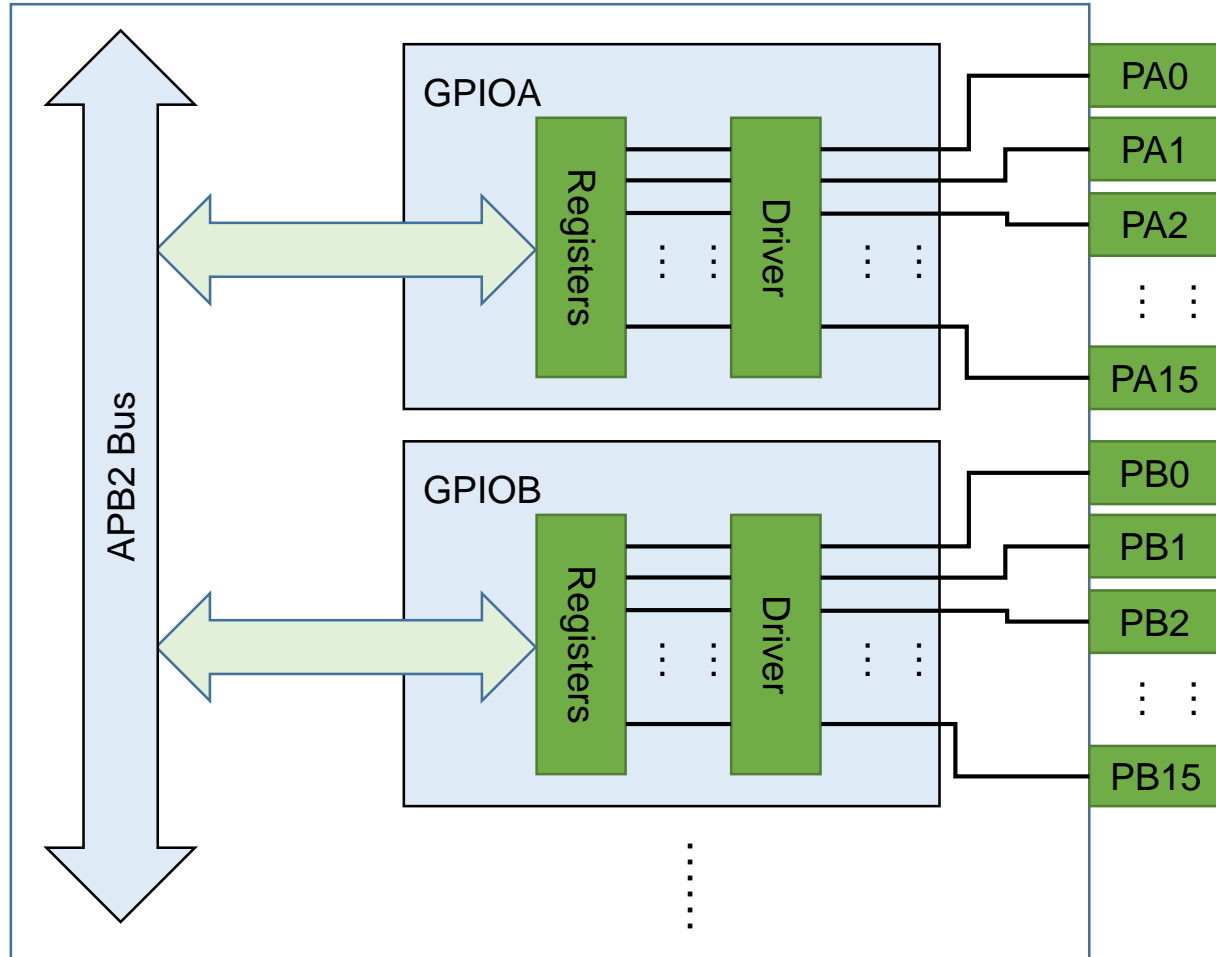


GPIO Register Mapping

- 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.



GPIO Inside



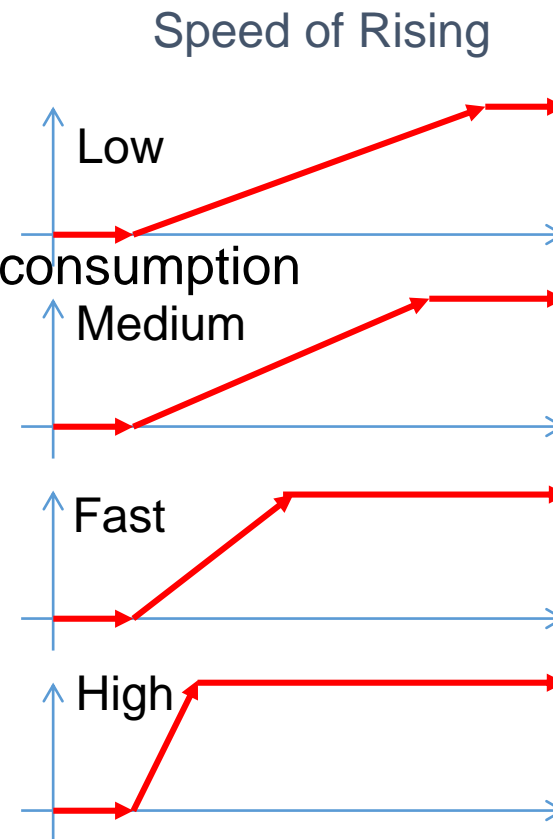
GPIO Mode

- 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



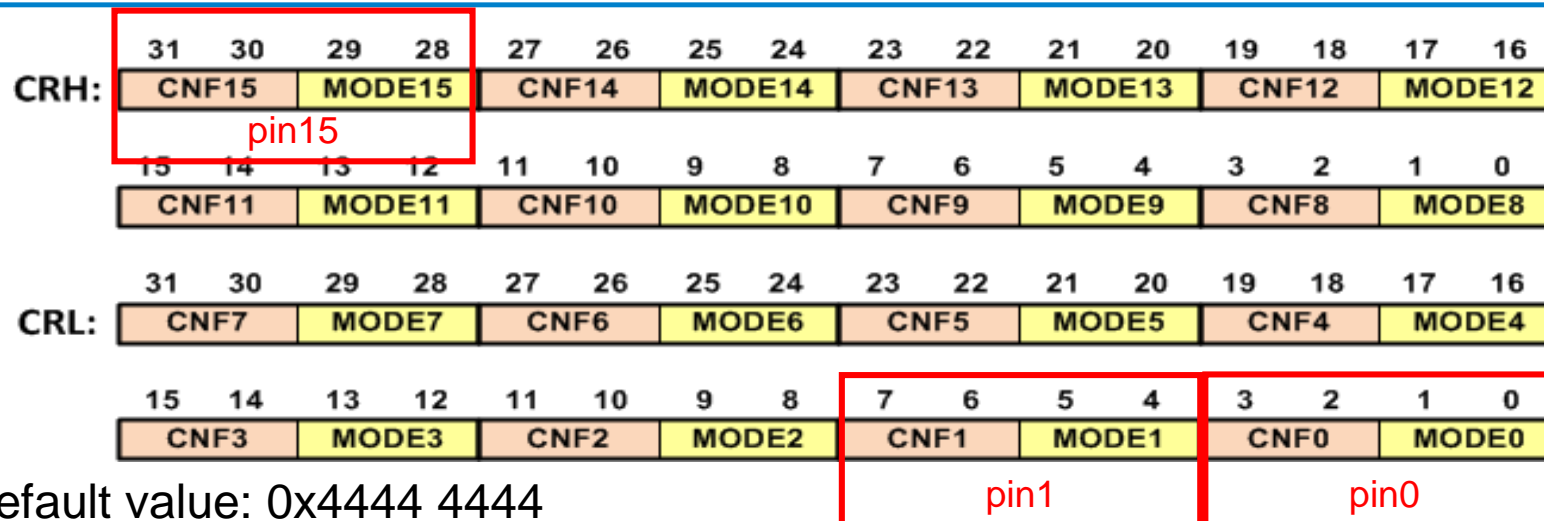
Programming GPIO

- 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)

Note: the modification of ODR will apply the entire GPIO port, you can also use **BSRR** to set only one GPIO pin, which is less risky, see more details in lab.

CRL and CRH (Configuration Registers)

- CRL: lower 8 pins
- CRH: higher 8 pins
- each GPIO pin is configured using 4 bits in CRL/CRH
 - MODE_x
 - CNF_x



Default value: 0x4444 4444

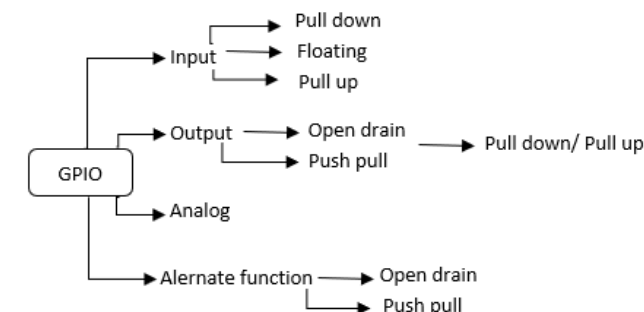
Output (MODE>00)

CNF _x 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

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

Input (MODE=00)

CNF _x 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 Example

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

Output (MODE>00)

CNF _x 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)

CNF _x 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	

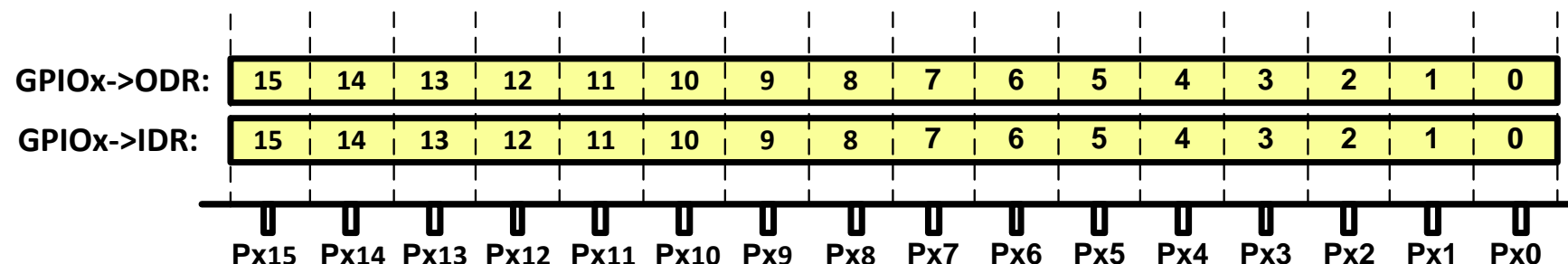
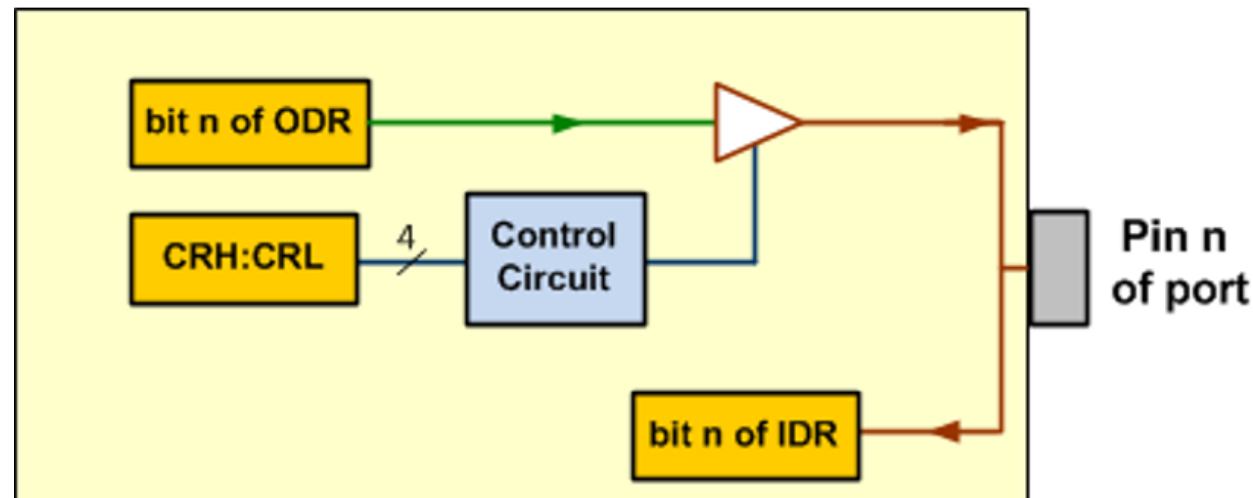
```
GPIOC->CRH &= 0xFFFF00FF; //clear settings of PC11 and PC12
GPIOC->CRH |= 0x00038000; //PC11 as input, PC12 as output
GPIOC->ODR |= 1<<11;      //set PC11 as input with pull-up
```

ODR register is shown in the next slide

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

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

- Higher 16 bits of IDR/ODR are reserved
- each GPIO pin is configured using 1 bit

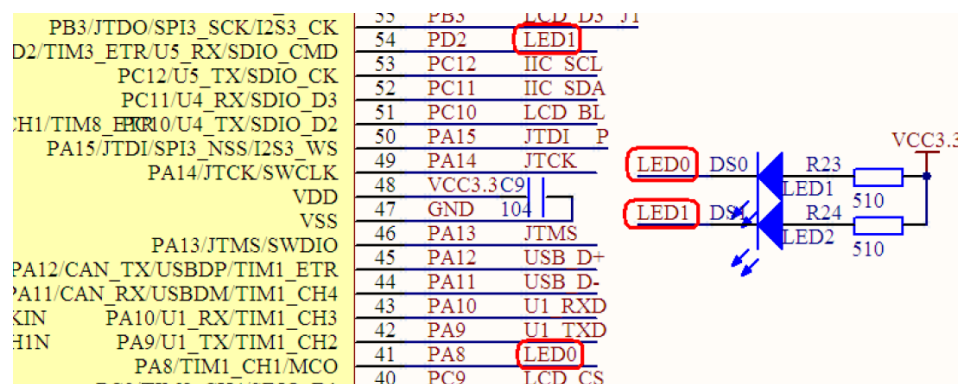


```

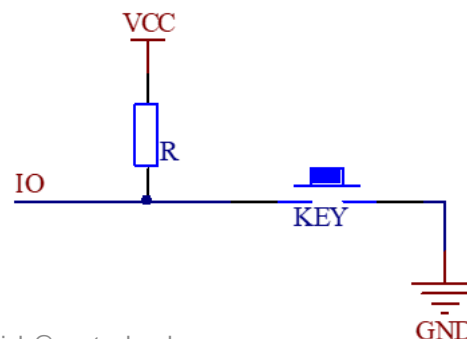
GPIOC->CRH &= 0x0FFFFFFF; //clear settings pf PC15
GPIOC->CRH |= 0x30000000; //set PC15 as output
GPIOC->ODR |= 1<<15; //Set output pin PC15 as high
    
```

IDR and ODR

- For LED0 and LED1 in miniSTM32 board, should we set ODR bit to low or high in order to turn on the LED?



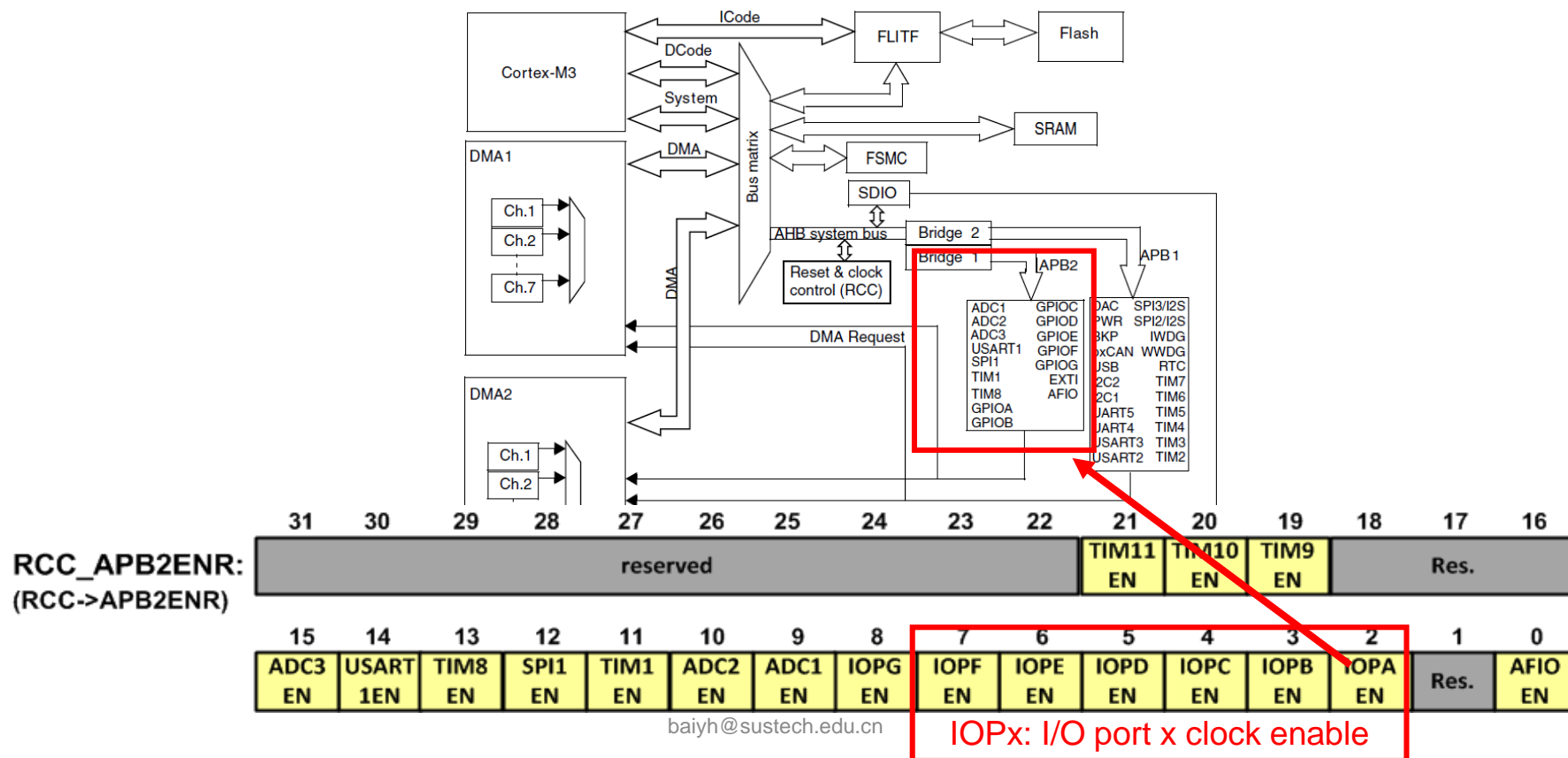
- For KEY in STM32, should we set pull-up or pull-down input mode?



Enabling Clocks

• Example:

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



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);
    }
}
```

$\wedge=$ is often used for toggling bits