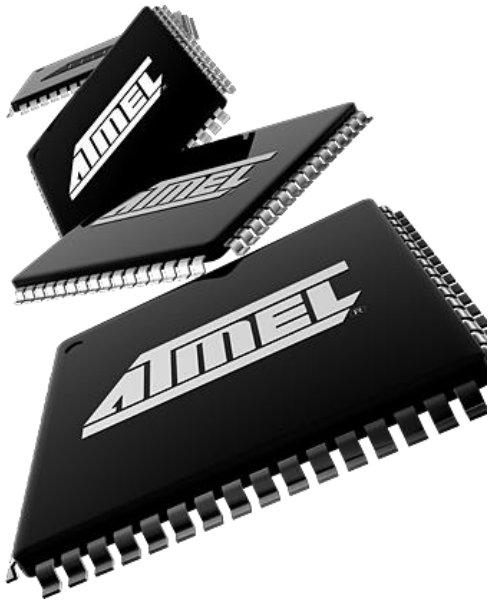
Principles and Applications of Microcontrollers

Yan-Fu Kuo
Dept. of Biomechatronics Engineering
National Taiwan University

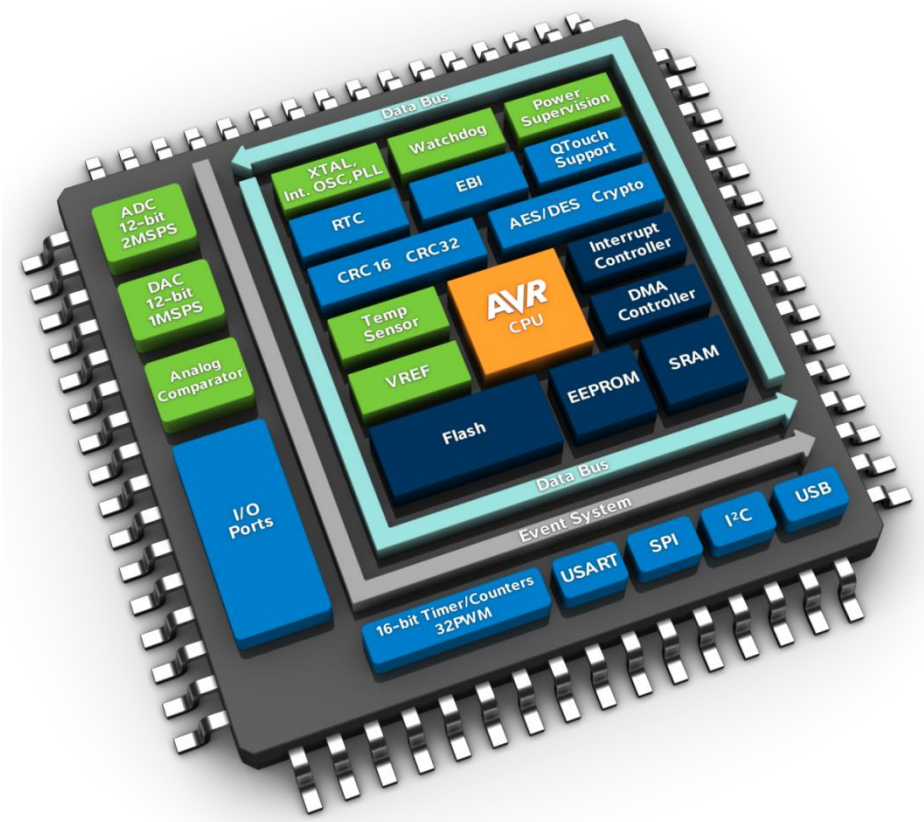
Today:

- AVR CPU
- Assembly introduction

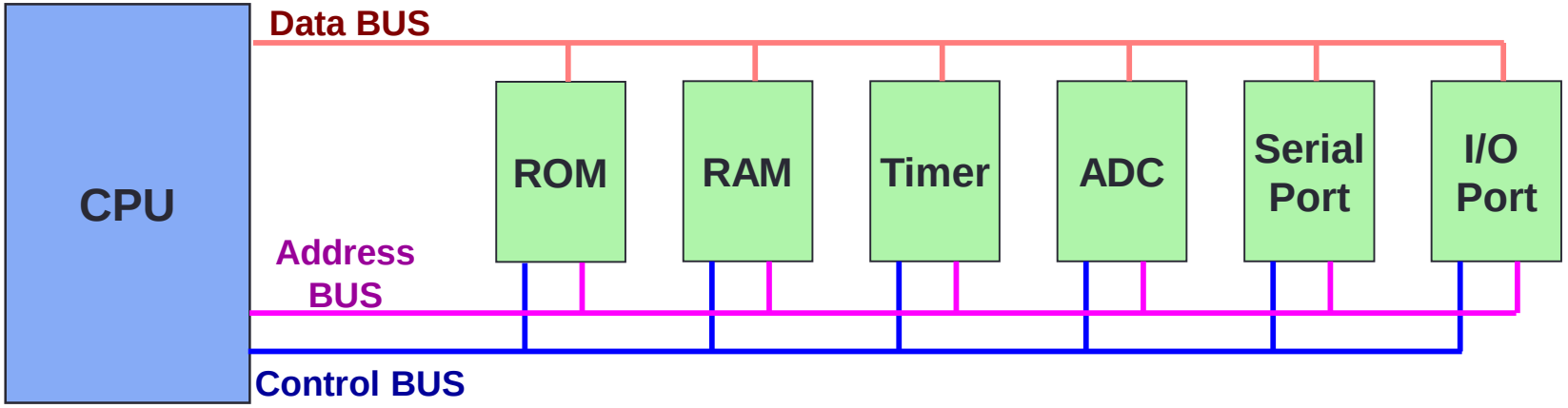


Outline

- AVR CPU architecture
- How computers work
- Assembly
  - Instruction in CPU
  - Accessing I/O
- I/O programming
- Getting started



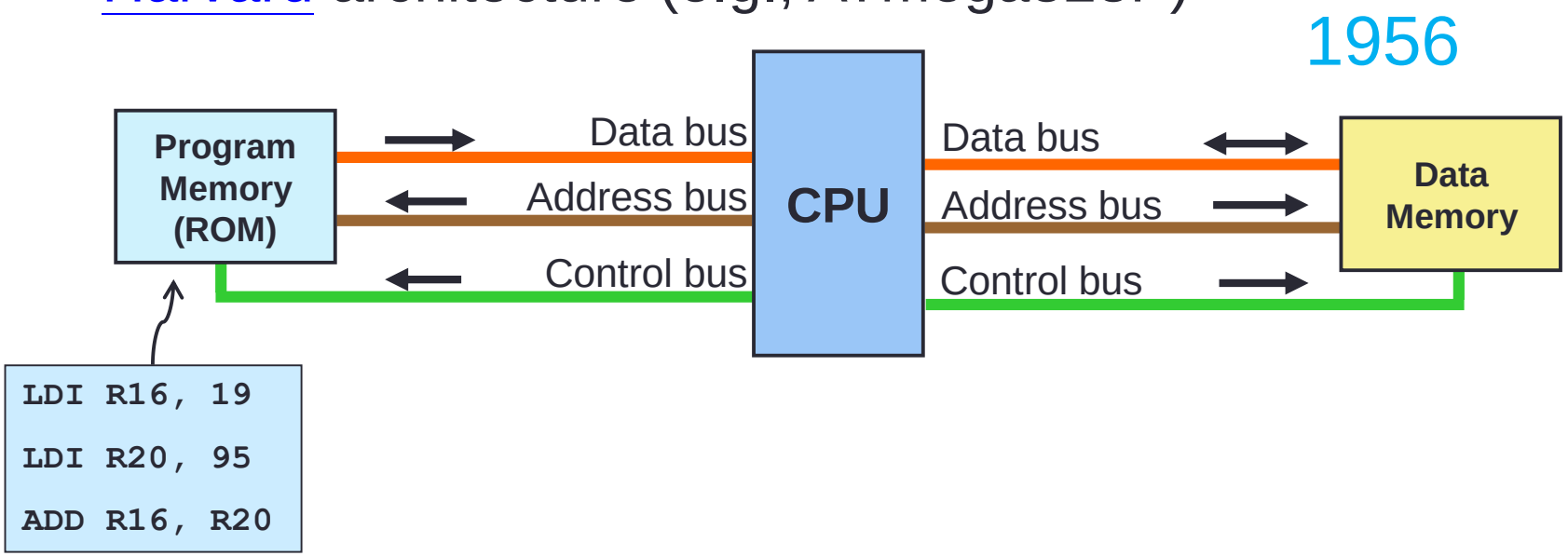
Microcontroller Structure



- We have talked about the peripherals (e.g., I/O, Timer, ADC, Serial)
- It's time to focus on program you wrote and the CPU
- Where is the program stored?

Read Only Memory (ROM)

- Harvard architecture (e.g., ATmega328P)

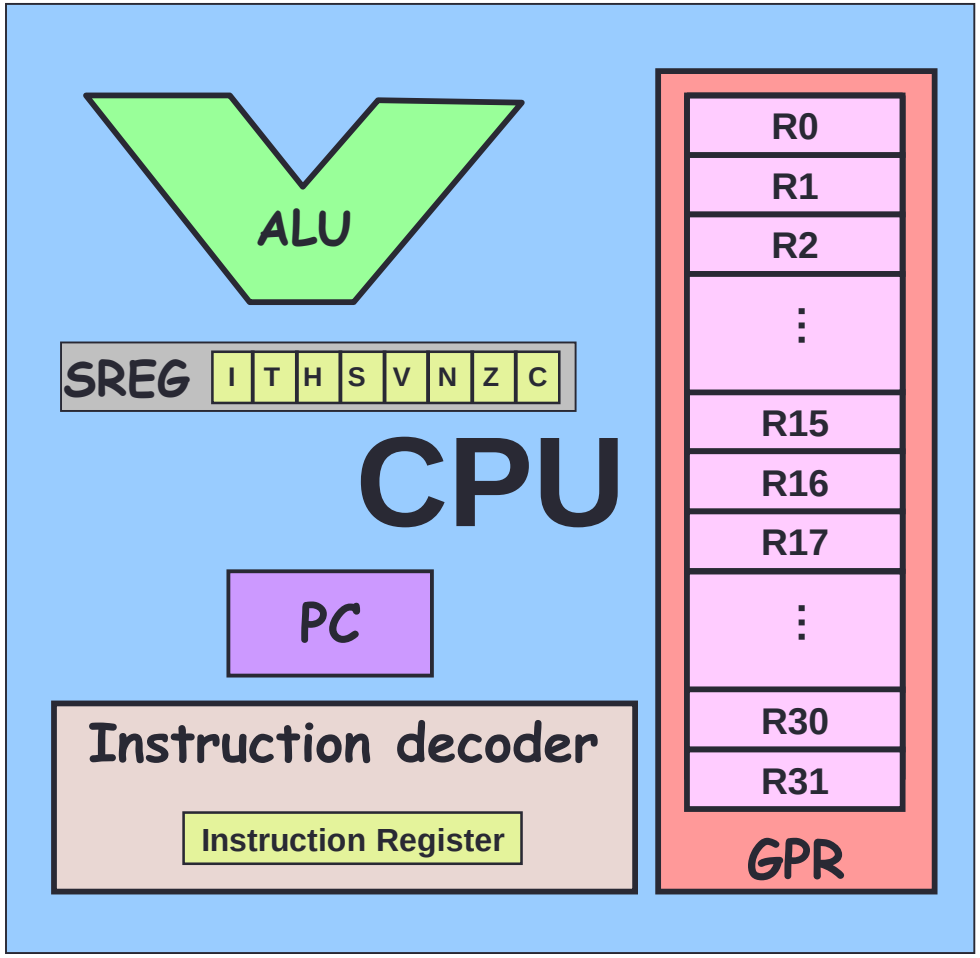


FLASH ROM	EEPROM	SRAM
32K Bytes	1K Bytes	2K Bytes

AVR's CPU

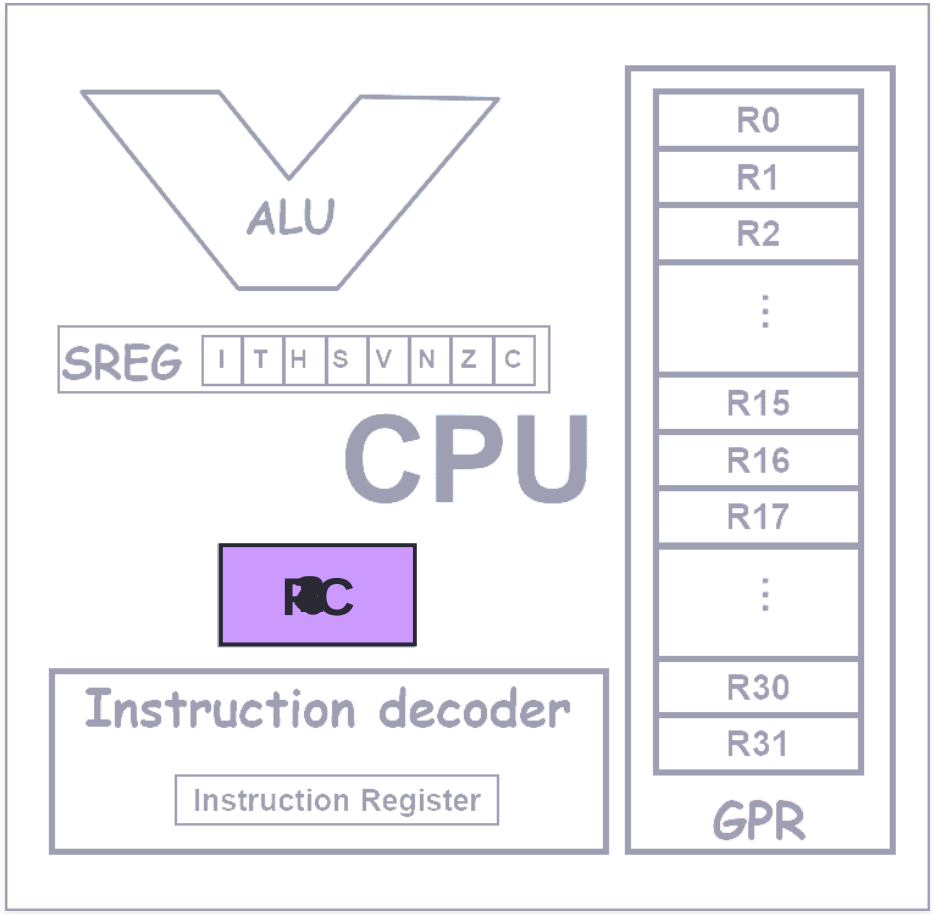
1. Program counter (PC)
2. General purpose registers (GPR) × 32
3. Arithmetic logic unit (ALU)
4. Instruction decoder
5. Status register (SREG)

Note: All registers are 8-bit word length



Program Counter

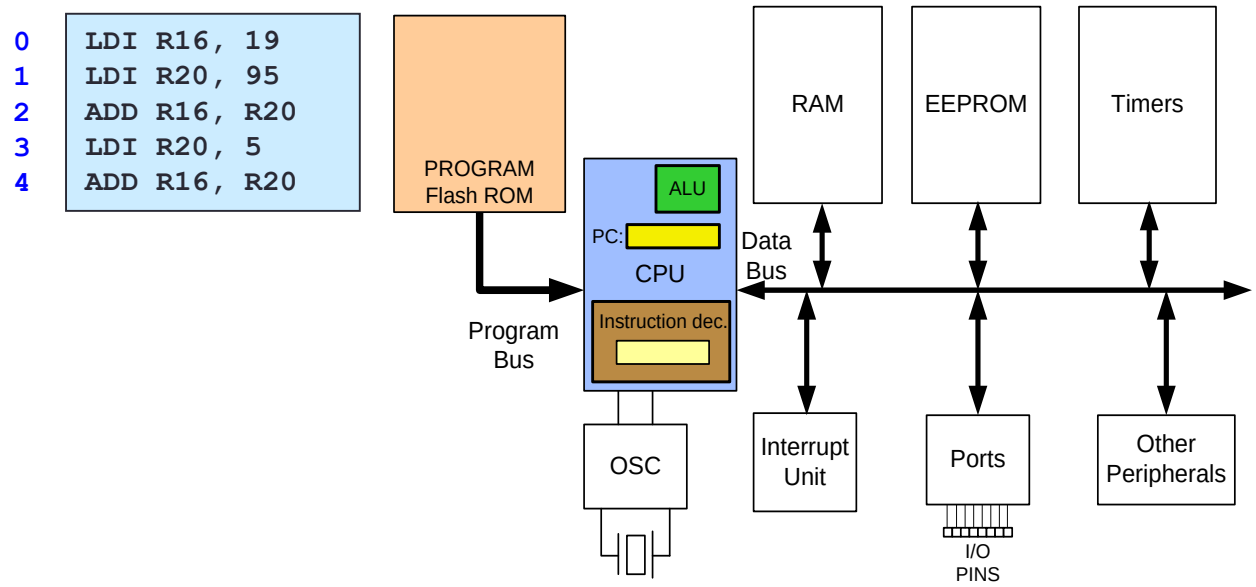
- A pointer to the address of the program ROM



Flash ROM	Address
`LDI R16, 19`	0
`LDI R20, 95`	1
`ADD R16, R20`	2
`LDI R20, 5`	3
`ADD R16, R20`	4

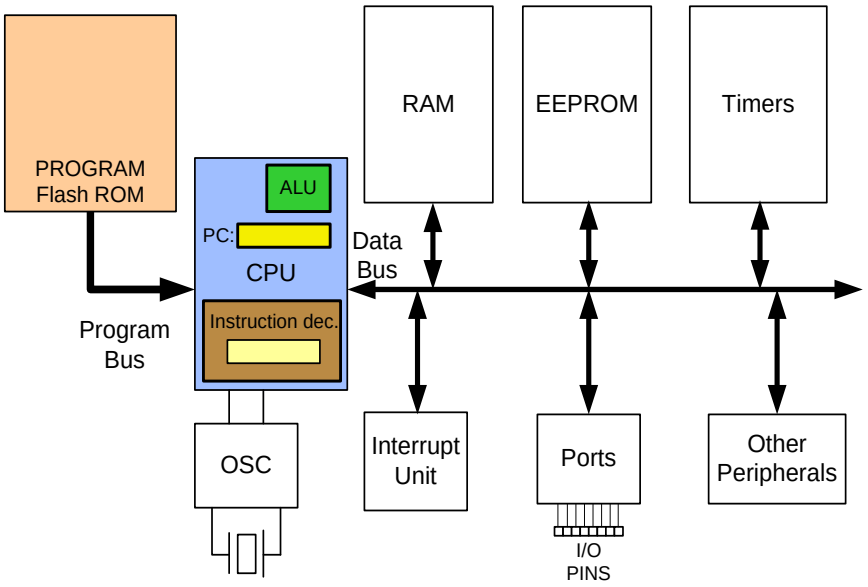
Program Execution Process

1. “Fetch” the machine code at the program counter
2. Program counter +=1
3. Execute the fetched machine code



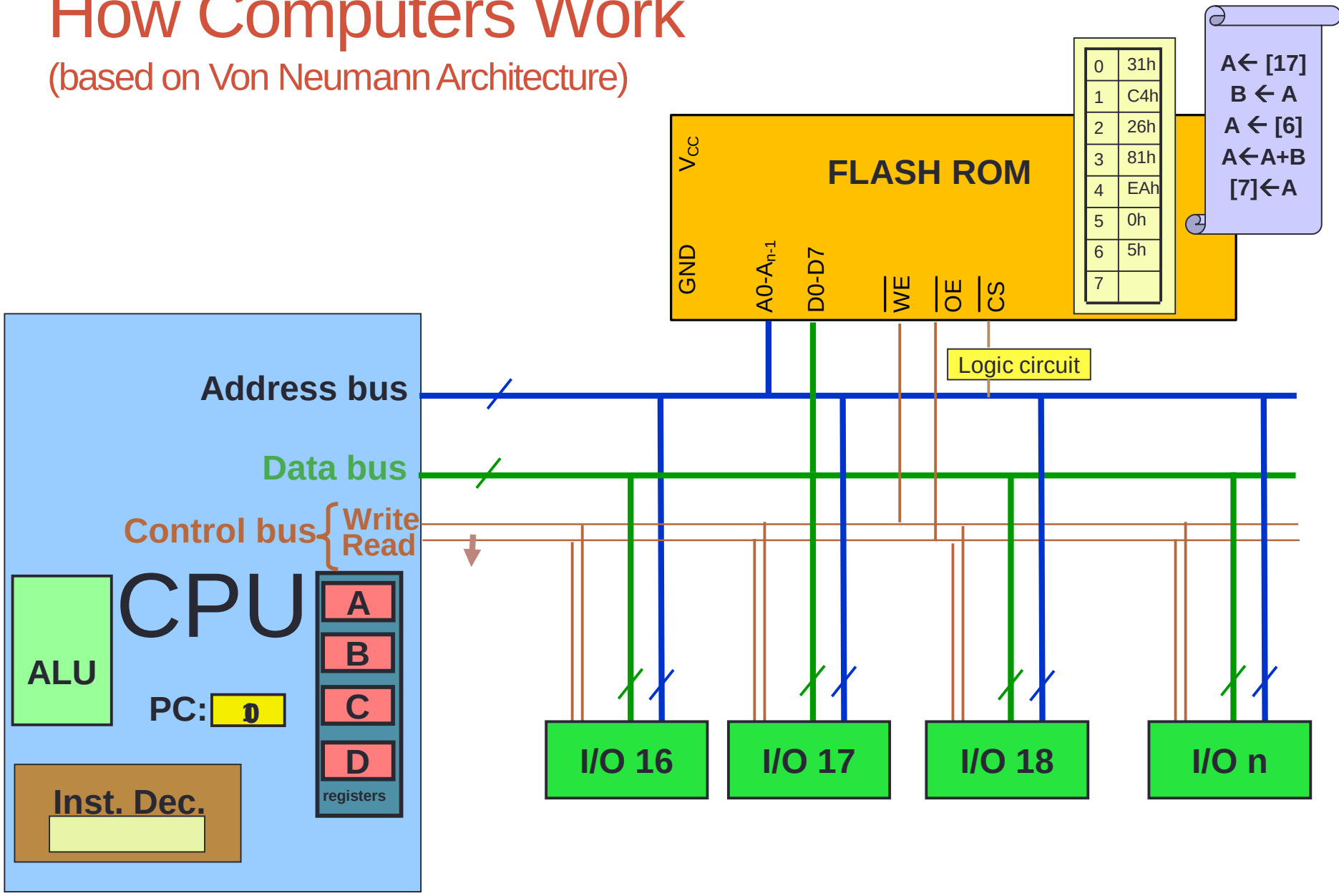
Outline (Cont'd)

- AVR CPU architecture
- How computers work
- Assembly
  - Instruction in CPU
  - Accessing data space
- I/O programming
- Getting started



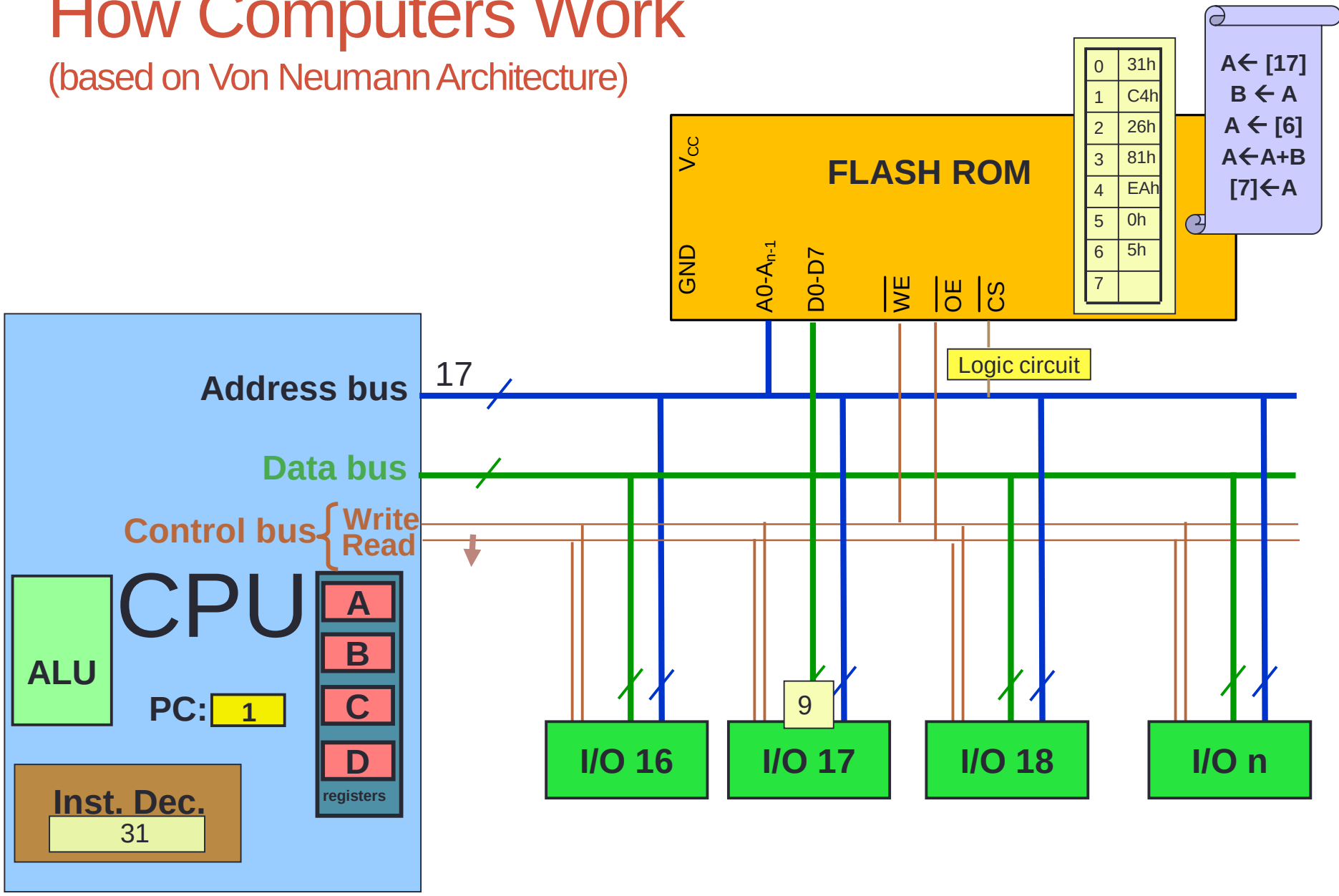
How Computers Work

(based on Von Neumann Architecture)



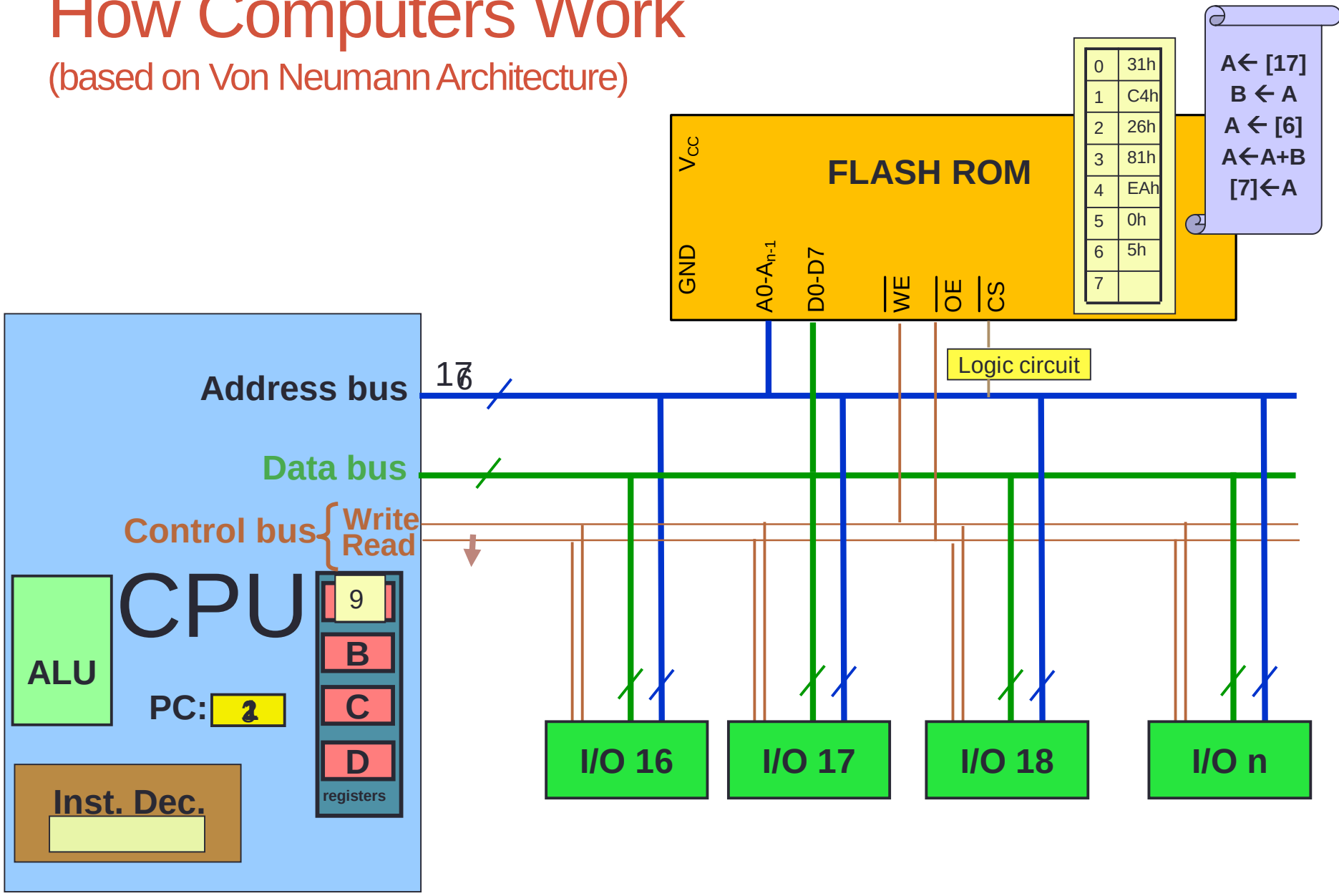
How Computers Work

(based on Von Neumann Architecture)



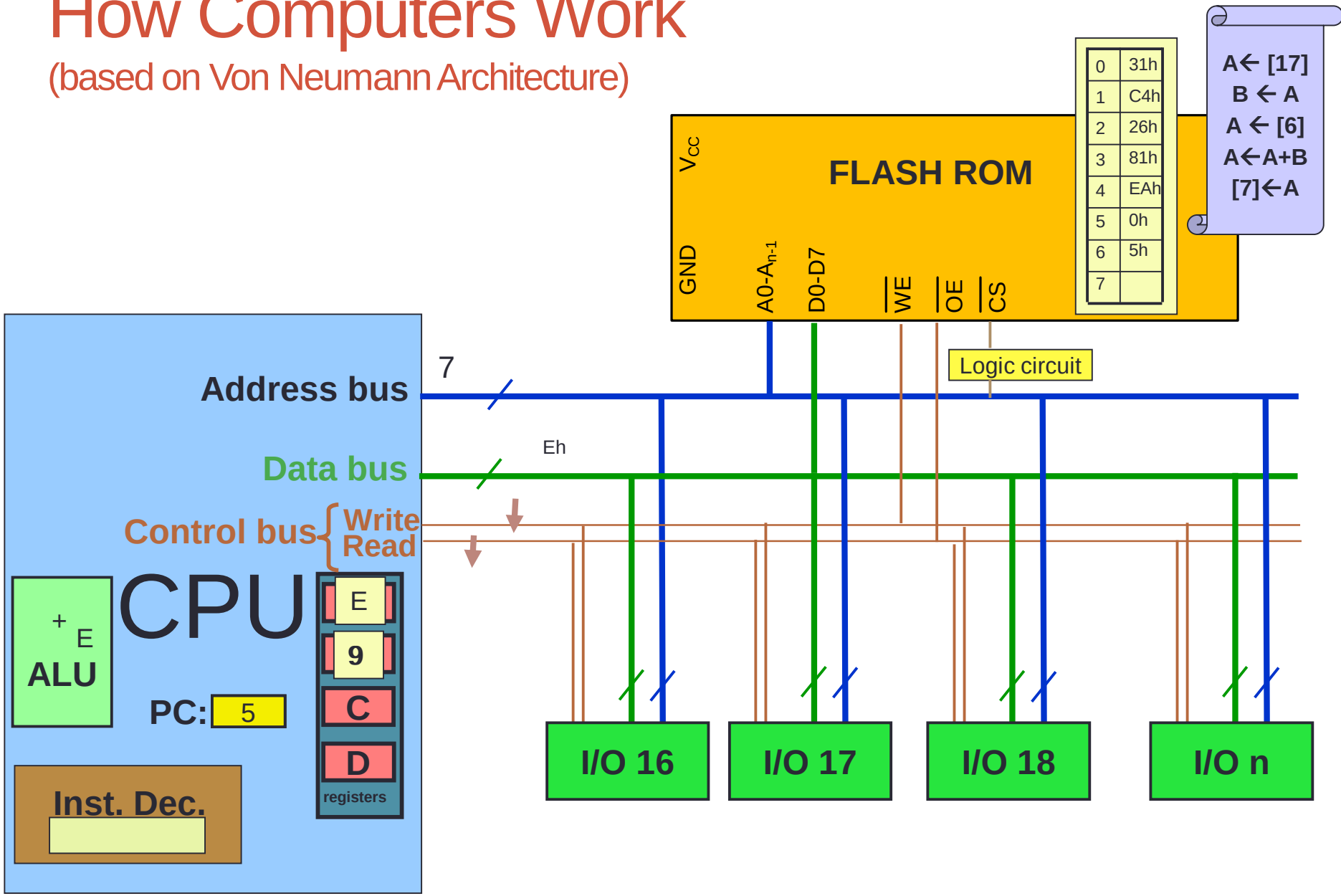
How Computers Work

(based on Von Neumann Architecture)



How Computers Work

(based on Von Neumann Architecture)

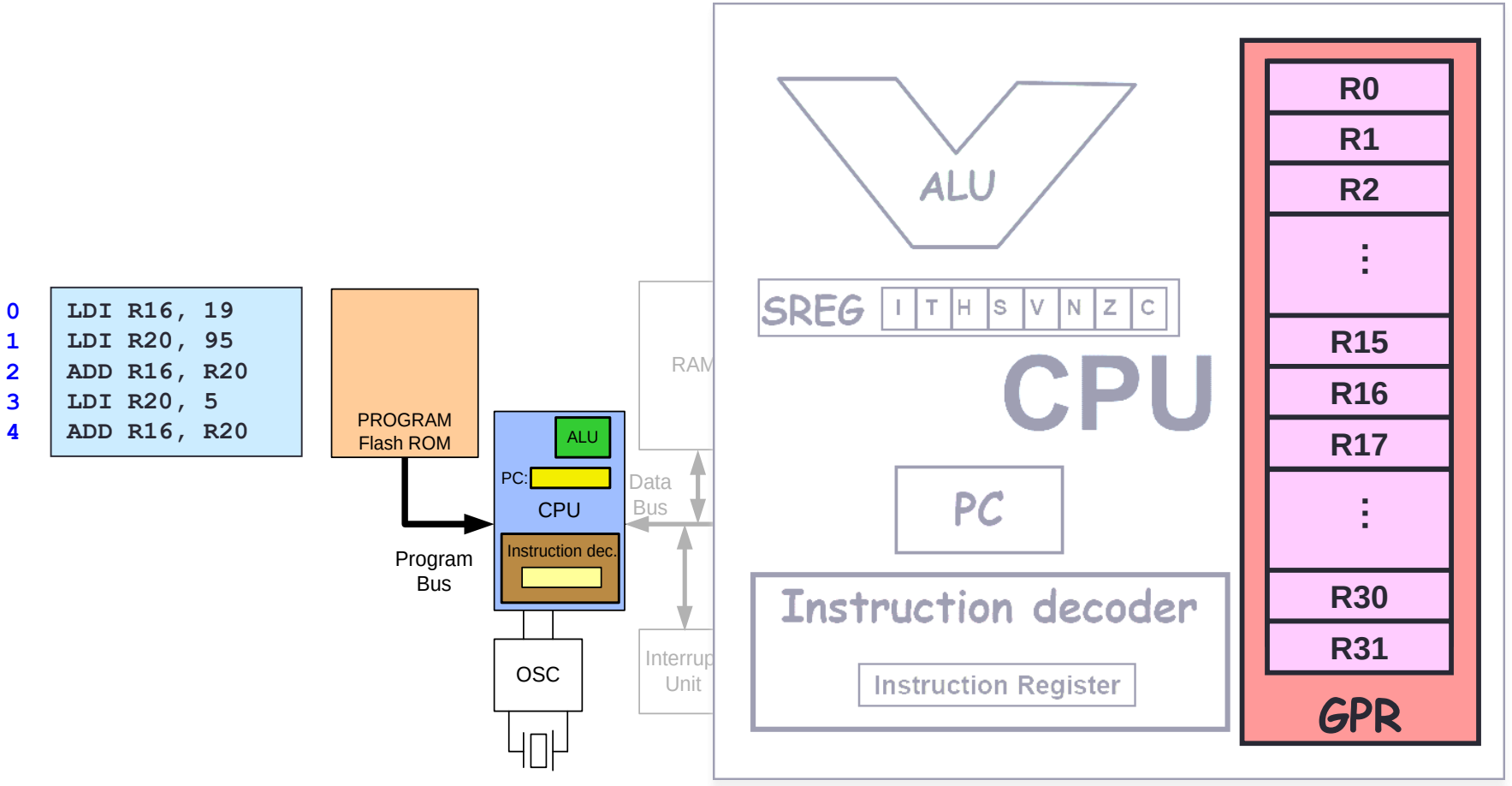


Outline (Cont'd)

- AVR CPU architecture
- How computers work
- Assembly
  - Instruction in CPU
  - Accessing I/O
- I/O programming
- Getting started

Let's Focus on CPU and ROM

- How do I assign a number to a GPR?



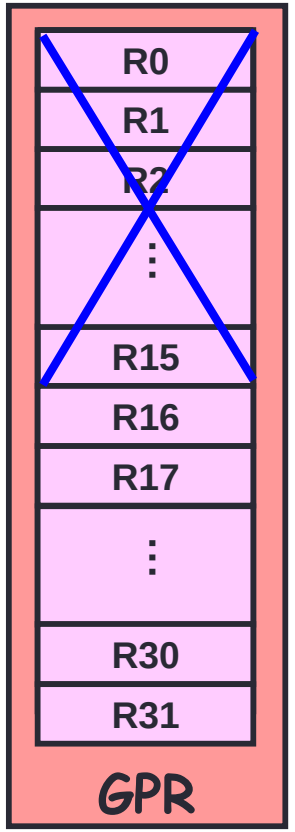
Instruction – LDI

- LDI (**L**oa**d** **I**mmediate) – loading values into the general purpose registers
- Syntax: **`LDI Rd, k`**

  where $16 \le d \le 31$, $0 \le k \le 255$

- Equivalent to: Rd = k in C language
- Example:
  - `LDI R16, 53` `;R16 = 53`
  - `LDI R23, $27` `;R23 = 0x27`
  - `LDI R30, 0x7F2` `;R30 = 0x7F2`

Is there any problem in the example?

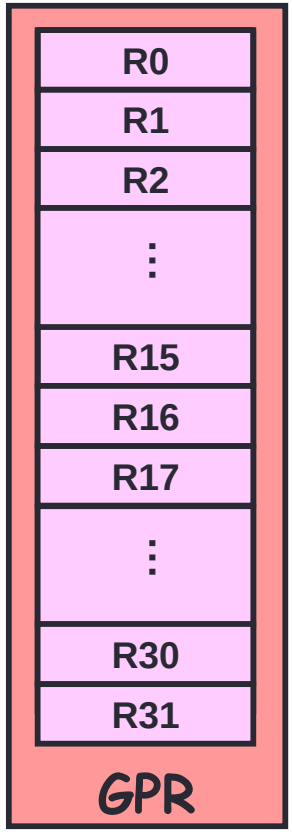


Instruction – ADD

- ADD – add and store values in the general purpose registers
- Syntax: **`ADD Rd, Rs`**

  where $0 \le d \le 31$, $0 \le s \le 31$

- Equivalent to: Rd = Rd + Rs in C language
- Example:
  - **`ADD R25, R9`** `;R25 = R25 + R9`
  - **`ADD R17, R30`** `;R17 = R17 + R30`



Sample Programs

- Write a program that calculates 19 + 95

```
LDI R16, 19   ;R16 = 19
LDI R20, 95   ;R20 = 95
ADD R16, R20  ;R16 = R16 + R20
```

- Write a program that calculates 19 + 95 + 5

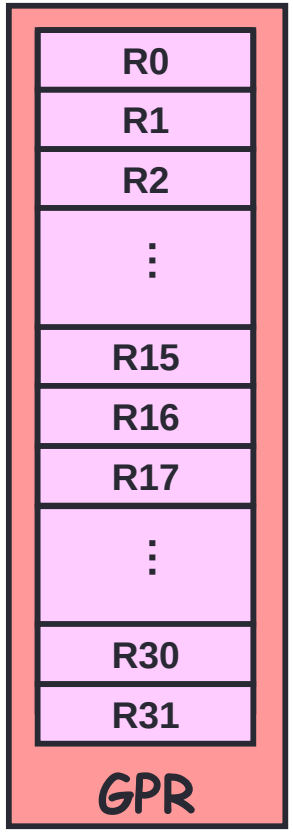
```
LDI R16, 19   ;R16 = 19
LDI R20, 95   ;R20 = 95
ADD R16, R20  ;R16 = R16 + R20
LDI R20, 5    ;R20 = 5
ADD R16, R20  ;R16 = R16 + R20
```

Instruction – SUB

- SUB – **sub**tract and store values in the general purpose registers
- Syntax: **`SUB Rd, Rs`**

  where $0 \le d \le 31$, $0 \le s \le 31$

- Equivalent to: Rd = Rd - Rs in C language
- Example:
  - **`SUB R25, R9`** `;R25 = R25 - R9`
  - **`SUB R17, R30`** `;R17 = R17 - R30`



Instruction – INC & DEC

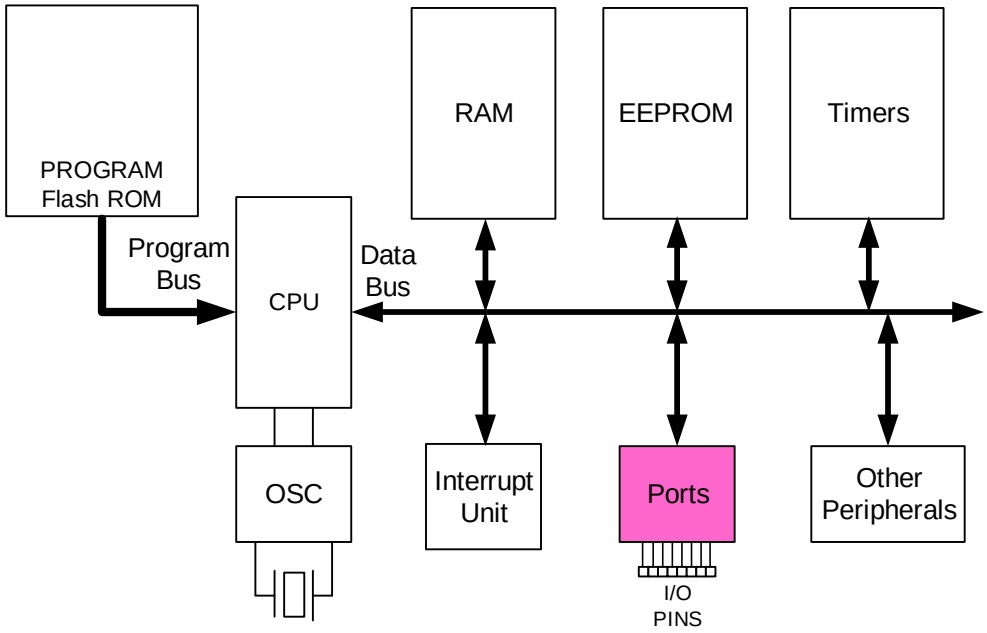
- INC – **inc**rement the contents of a register by one
- Syntax: `INC Rd`
  where 0 ≤ d ≤ 31
- Equivalent to: Rd = Rd + 1 in C language

- DEC – **dec**rement the contents of a register by one
- Syntax: `DEC Rd`
  where 0 ≤ d ≤ 31
- Equivalent to: Rd = Rd - 1 in C language

- Example:
  - `INC R2 ;R2 = R2 + 1`

Outline (Cont'd)

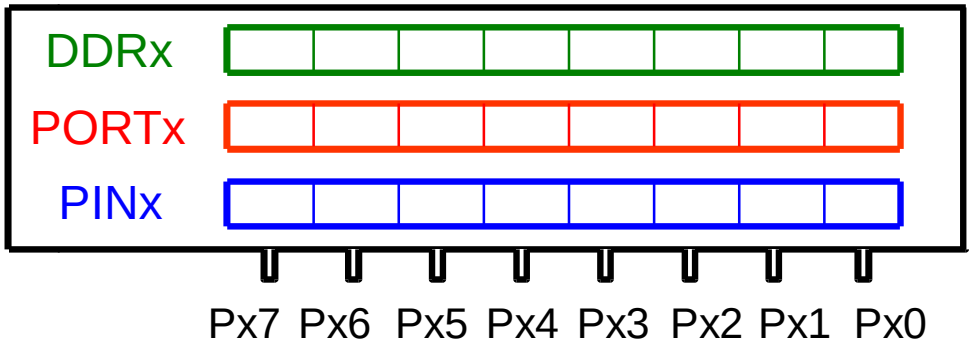
- AVR CPU architecture
- How computers work
- Assembly
  - Instruction in CPU
  - Accessing I/O
- I/O programming
- Getting started

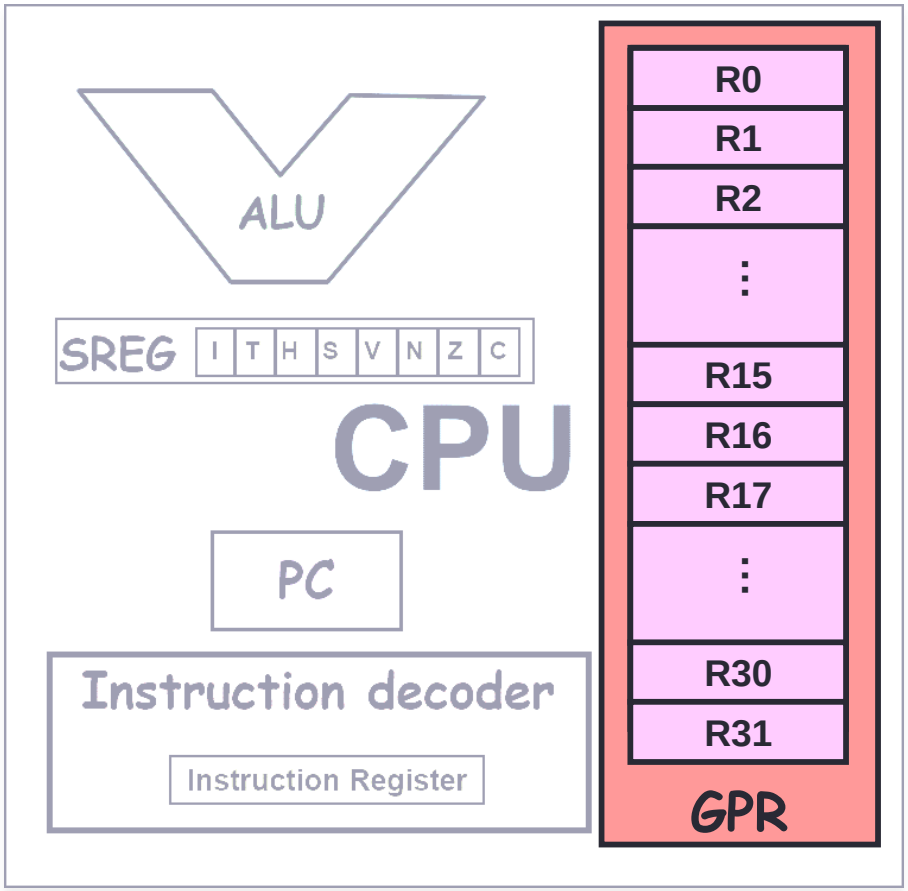


Review – I/O Registers

- Each port is associated with three 8-bit registers:
  - DDRx: data direction
  - PORTx: output
  - PINx: input

  where x = B, C, or D





Accessing I/O Registers – IN & OUT

- IN – load the reading from an I/O register to a GPR
- Syntax: **IN Rd, A**

  where 0 ≤ d ≤ 31, A is the name of an I/O register
- Example:
  - **IN R1,**
- OUT – sto
- Syntax: **OU**

  where 0 ≤
- Example:
  - **OUT DD**

Example: Write a program that adds the contents of the PINC to the contents of PIND and stores the result to PIND

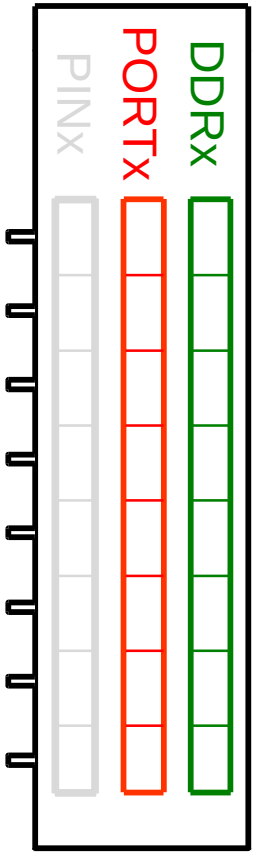
Solution:

```
IN R20, PINC   ;R20 = PINC
IN R21, PIND   ;R21 = PIND
ADD R20, R21   ;R20 = R20 + R21
OUT PIND, R20  ;PIND = R20
```

Note: IN and OUT only work for I/O registers

Review: Output Mode

- DDRx = 1
- PINx is NOT used
- PORTx contains the output voltages
  - 1: logic HIGH – 5V
  - 0: logic LOW – 0V



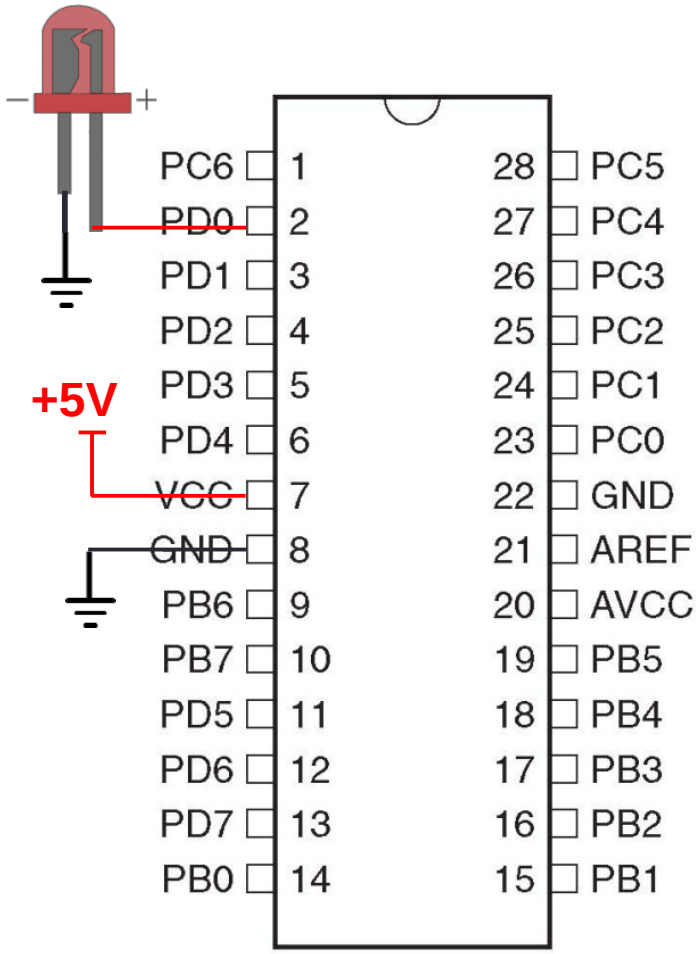
Example: Turn on An LED

- Turn on an LED connected to PD0
- Pseudo program code:

```
DDRD=0b00000001;

PORTD=0b00000001;
```

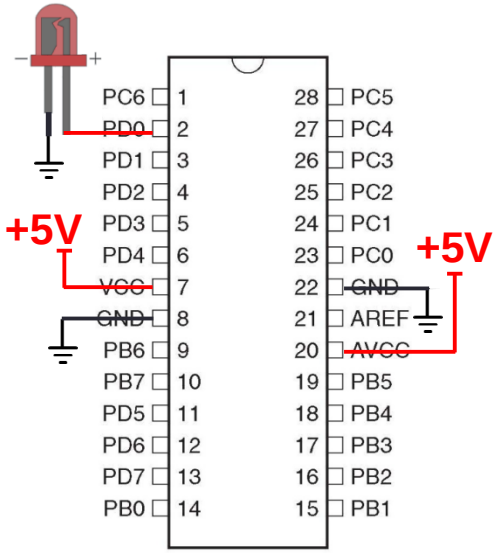
- Assembly program code:

```
LDI R20, 0b00000001

OUT DDRD, R20

OUT PORTD, R20
```



Example: Flash An LED

- Flash the LED at PD0
- Program code:

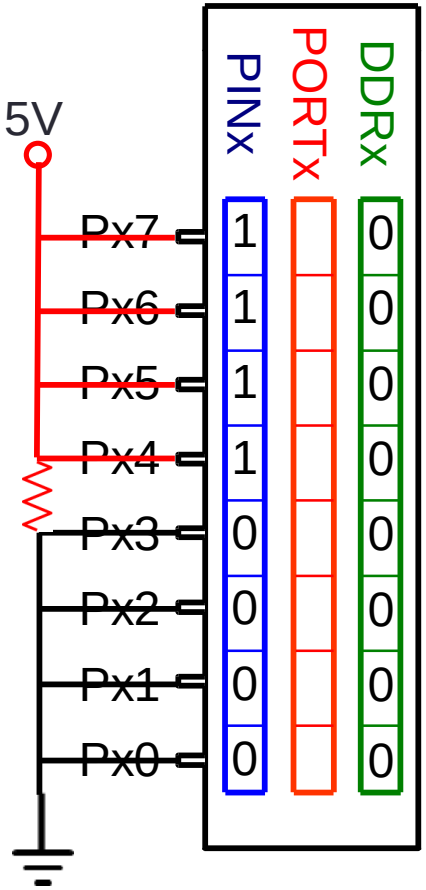



```
        LDI R20, 0b00000001
        LDI R21, 0b00000000
        OUT DDRD, R20
L1:     OUT PORTD, R20
        CALL DELAY
        OUT PORTD, R21
        CALL DELAY
        JMP L1
```

```
DELAY:LDI R17, 12
L2:   LDI R18, 0xFF
L3:   LDI R19, 0xFF
L4:   DEC R19
      BRNE L4
      DEC R18
      BRNE L3
      DEC R17
      BRNE L2
      RET
```

Review: Input Mode without Pull-up

- DDRx = 0
- PINx contains the input logic signal
  - 1: logic HIGH – 5V
  - 0: logic LOW – 0V
- PORTx is NOT used (if not considering pull-up)



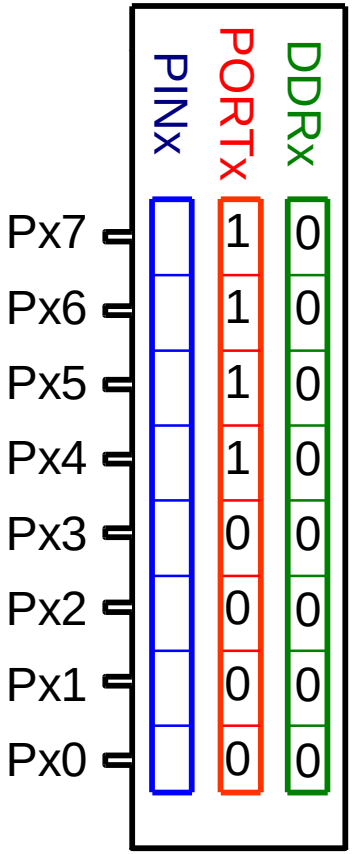
Example: Input Mode without Pull-up

- Make all the pins of Port B input
- Make all the pins of Port D output
- Send the reading from Port B to Port D, after adding a value 5 to it

```
LDI R16, 0x00     ;R16 = 00000000 (binary)
OUT DDRB, R16     ;make Port B an input port
LDI R16, 0xFF     ;R16 = 11111111 (binary)
OUT DDRD, R16     ;make Port D an output port
IN R16, PINB      ;read data from Port B to R16
LDI R17, 5
ADD R16, R17      ;add 5 to it
OUT PORTD, R16    ;send it to Port D
```

Review: Input Mode with Pull-up

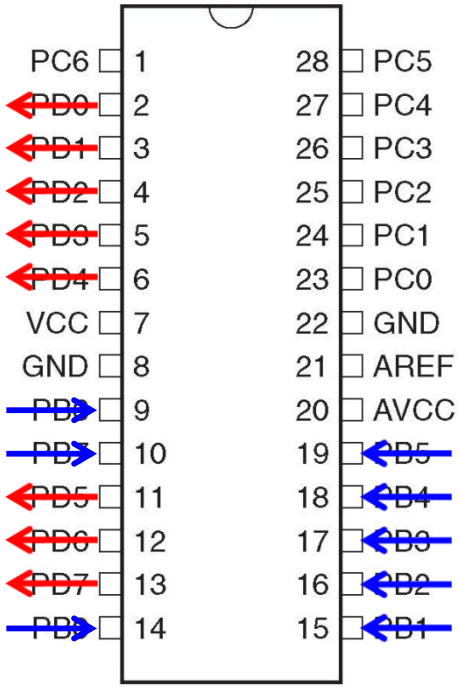
- DDRx = 0
- PORTx determines if pull-up resistors are connected or not
  - 1: pull-up ON
  - 0: pull-up OFF
- PINx contains the input logic signal
  - 1: logic HIGH – 5V
  - 0: logic LOW – 0V



Example: Input with Pull-up

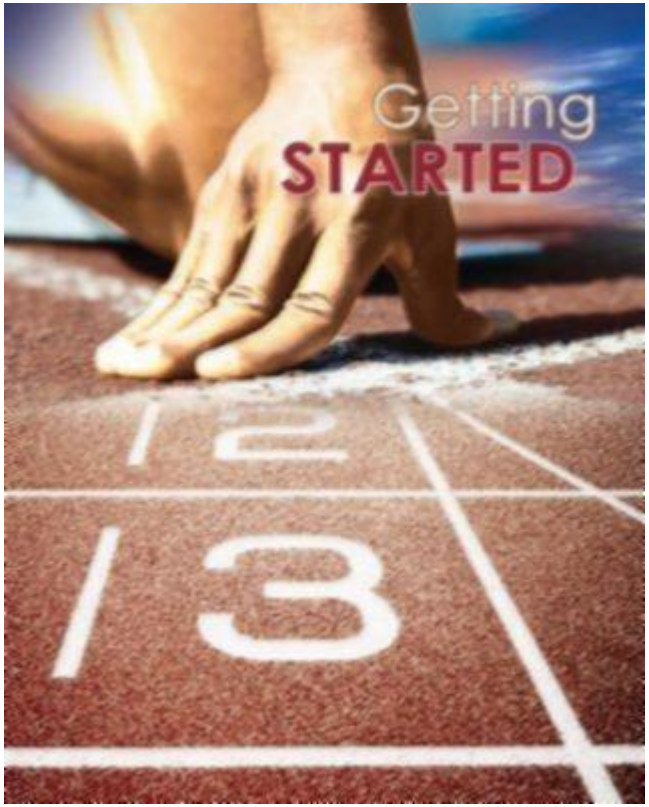
- Read Port B and writes the readings to Port D, with pull-up enabled

```
    LDI   R20, 0x0    ;R20 = 00000000
    OUT   DDRB, R20   ;DDRB = R20
    LDI   R20, 0xFF   ;R20 = 11111111
    OUT   DDRD, R20   ;DDRD = R20
    OUT   PORTB, R20  ;pull-up enabled
    IN    R20, PINB   ;R20 = PINB
    OUT   PORTD, R20  ;PORTD = R20
```



Outline (Cont'd)

- AVR CPU architecture
- How computers work
- Assembly
  - Instruction in CPU
  - Accessing I/O
- I/O programming
- Getting started



Reference

- ATmega328P data sheet
- AVR 8-bit instruction set
- AVR910: In-System Programming
- M. A. Mazidi, S. Naimi, and S. Naimi, *The AVR Microcontroller and Embedded Systems: Using Assembly and C*, Prentice Hall, 2010