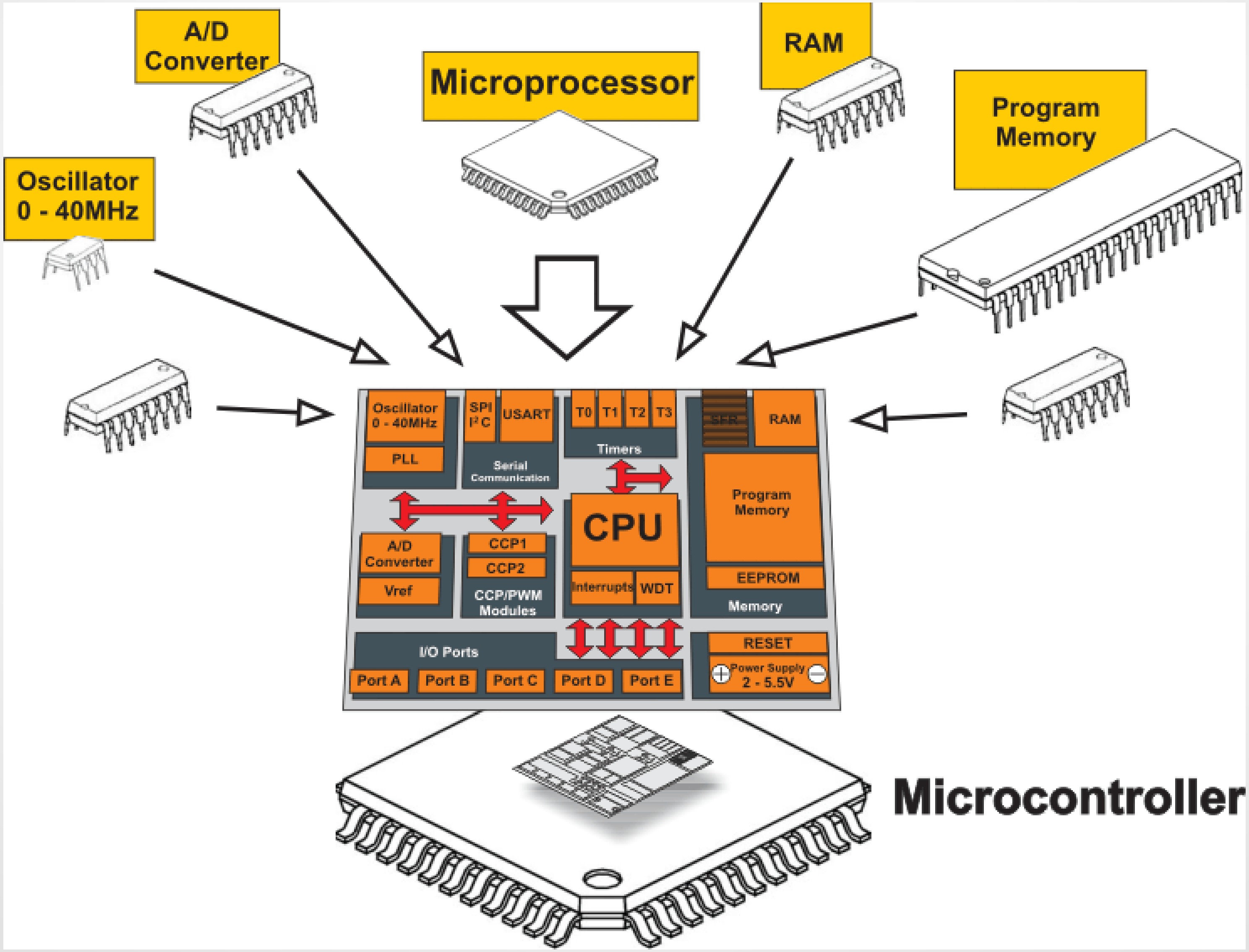
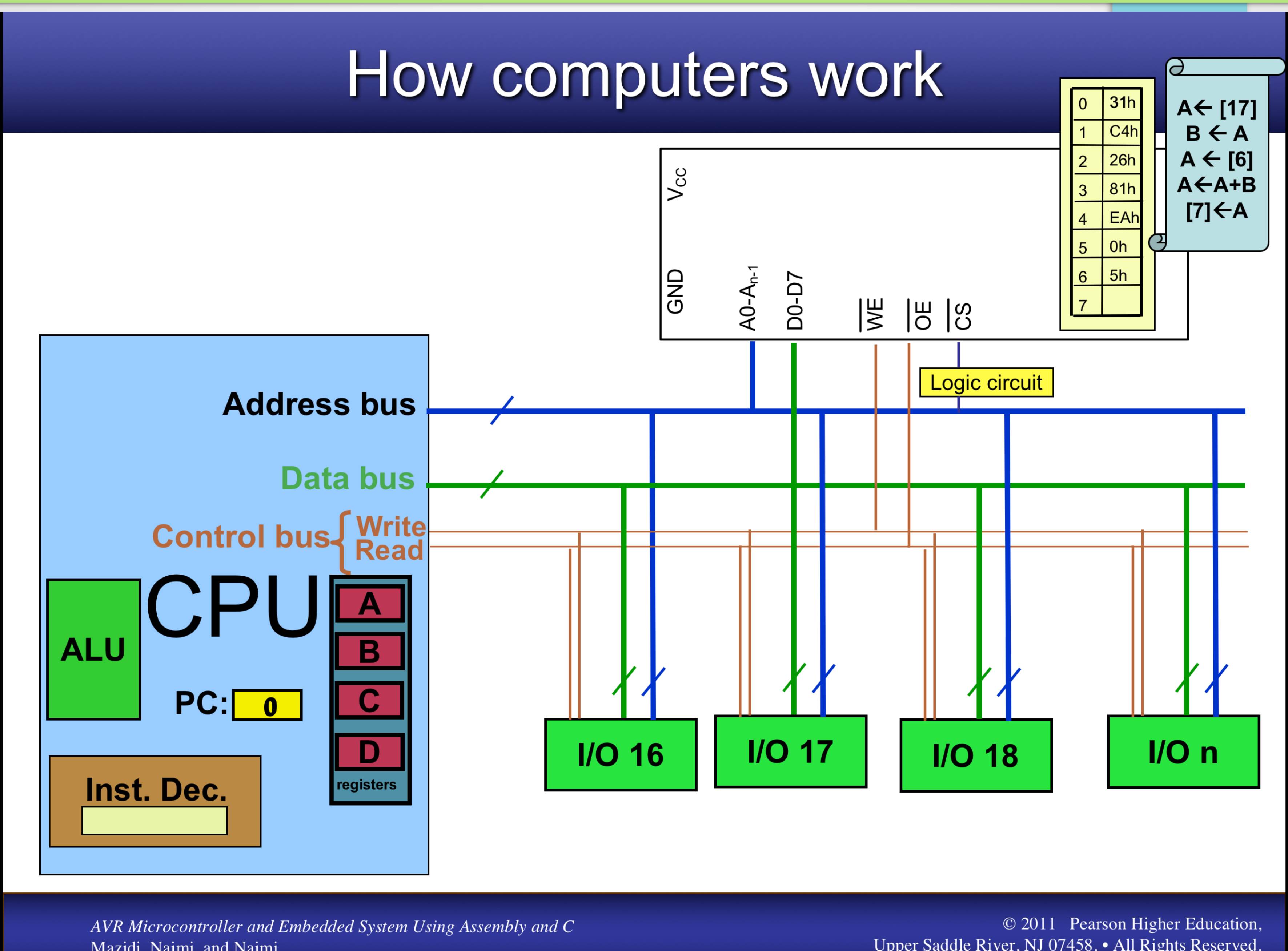
I/O Ports

I/O Ports

Microcontroller V's Microprocessor



Remember this?



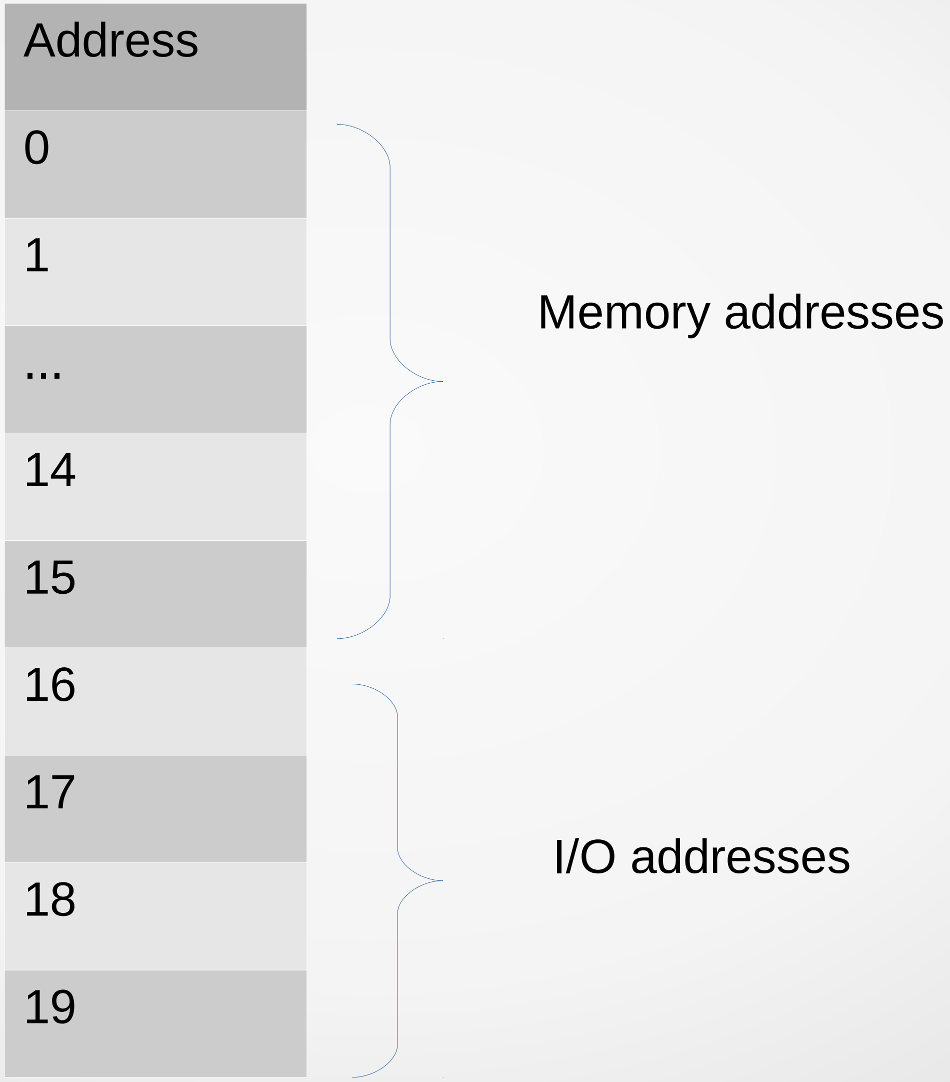
How computers work

0	31h
1	C4h
2	26h
3	81h
4	EAh
5	0h
6	5h
7	

A← [17]
B ← A
A ← [6]
A←A+B
[7]←A

AVR Microcontroller and Embedded System Using Assembly and C
Mazidi, Naimi, and Naimi

Memory Map

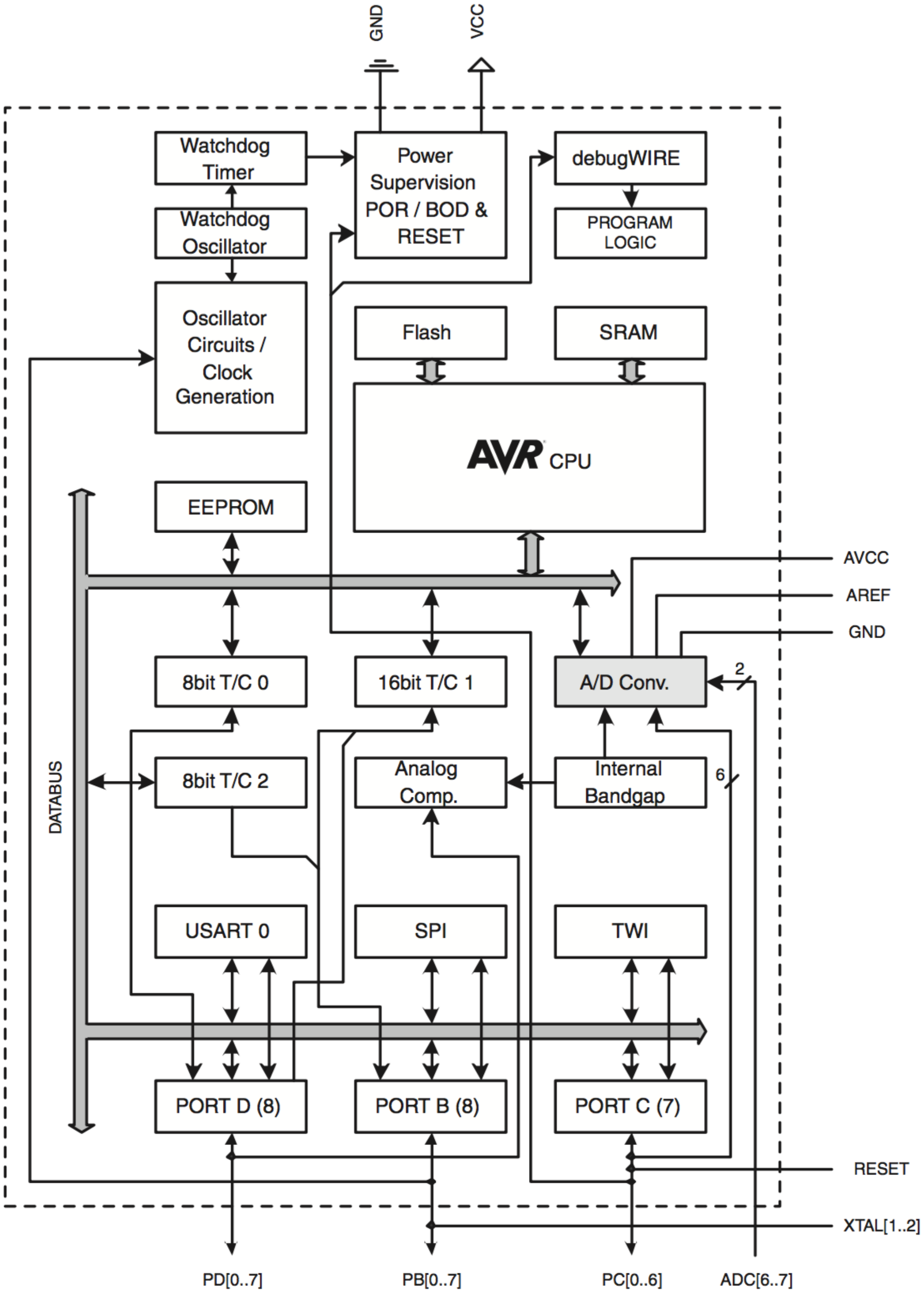


Address
0
1
...
14
15
16
17
18
19

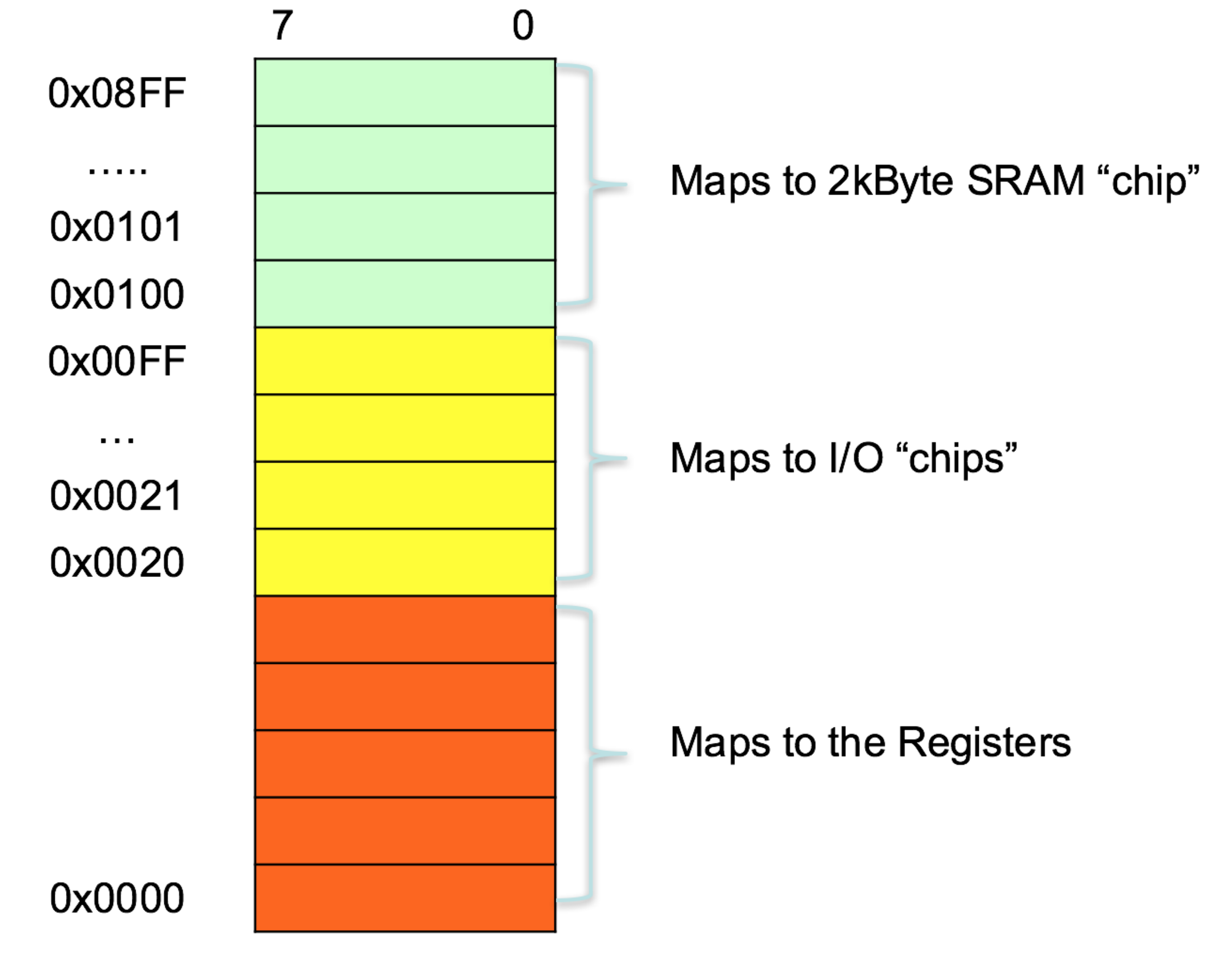
Memory addresses

I/O addresses

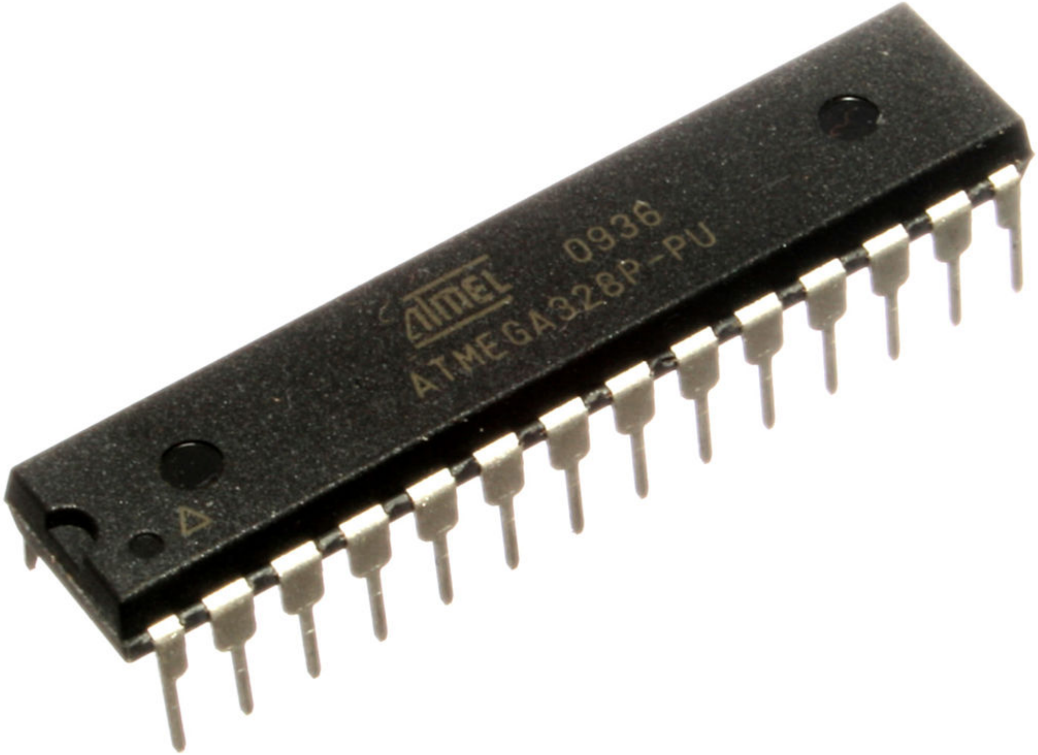
Figure 2-1. Block Diagram



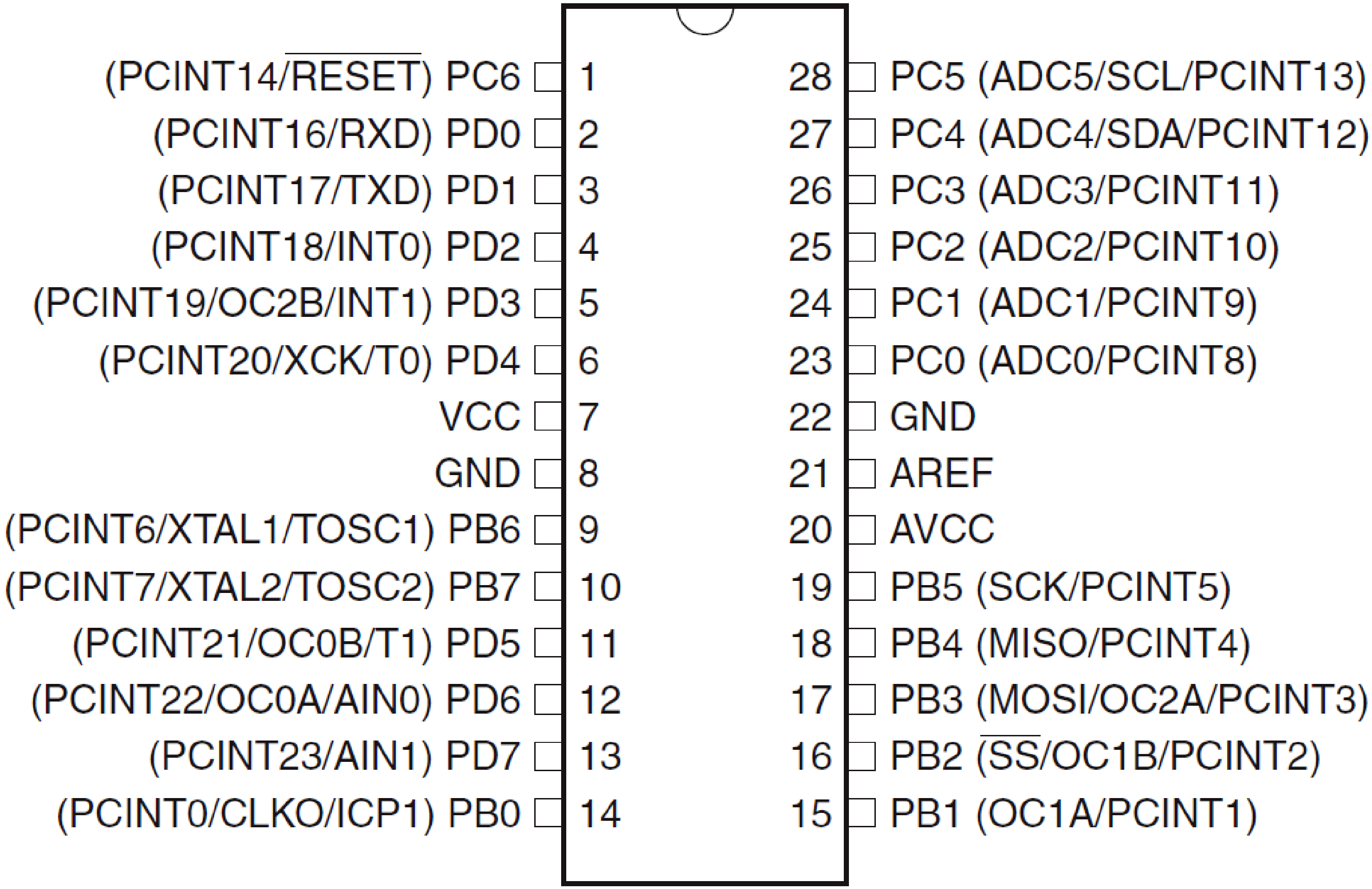
AVR Memory Map



atmega328p

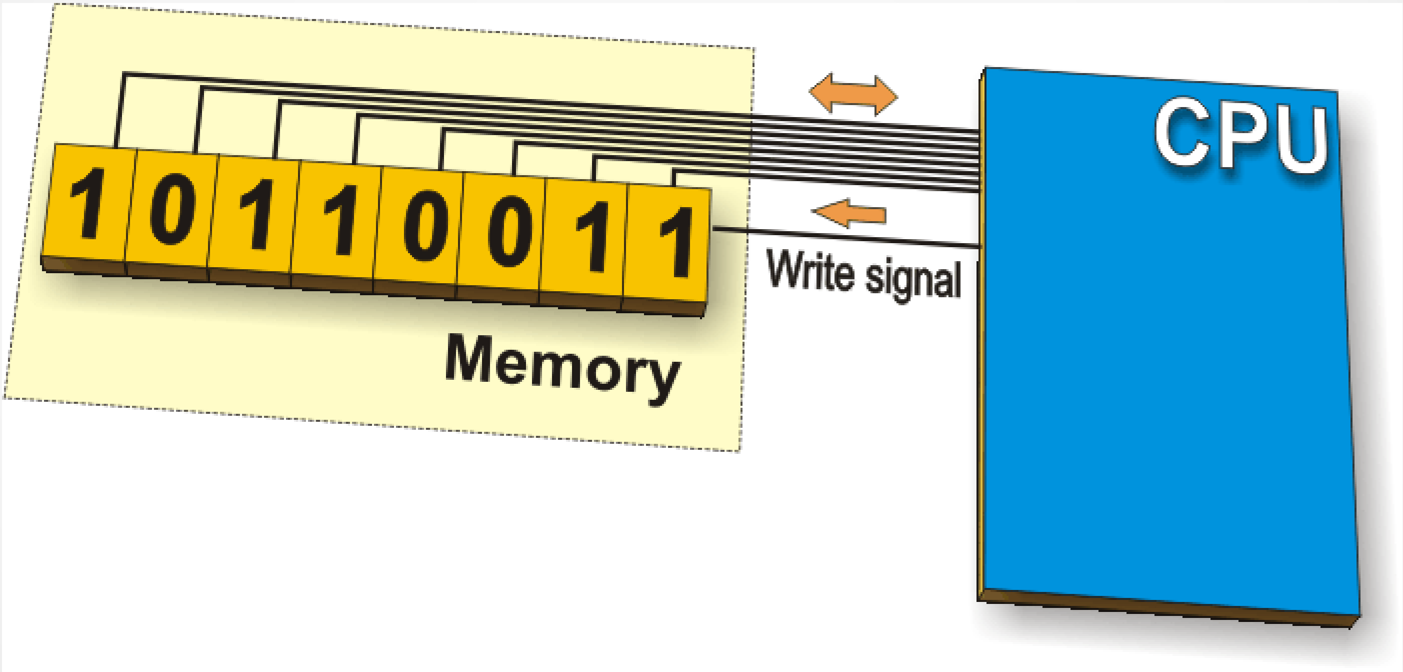


atmega328p

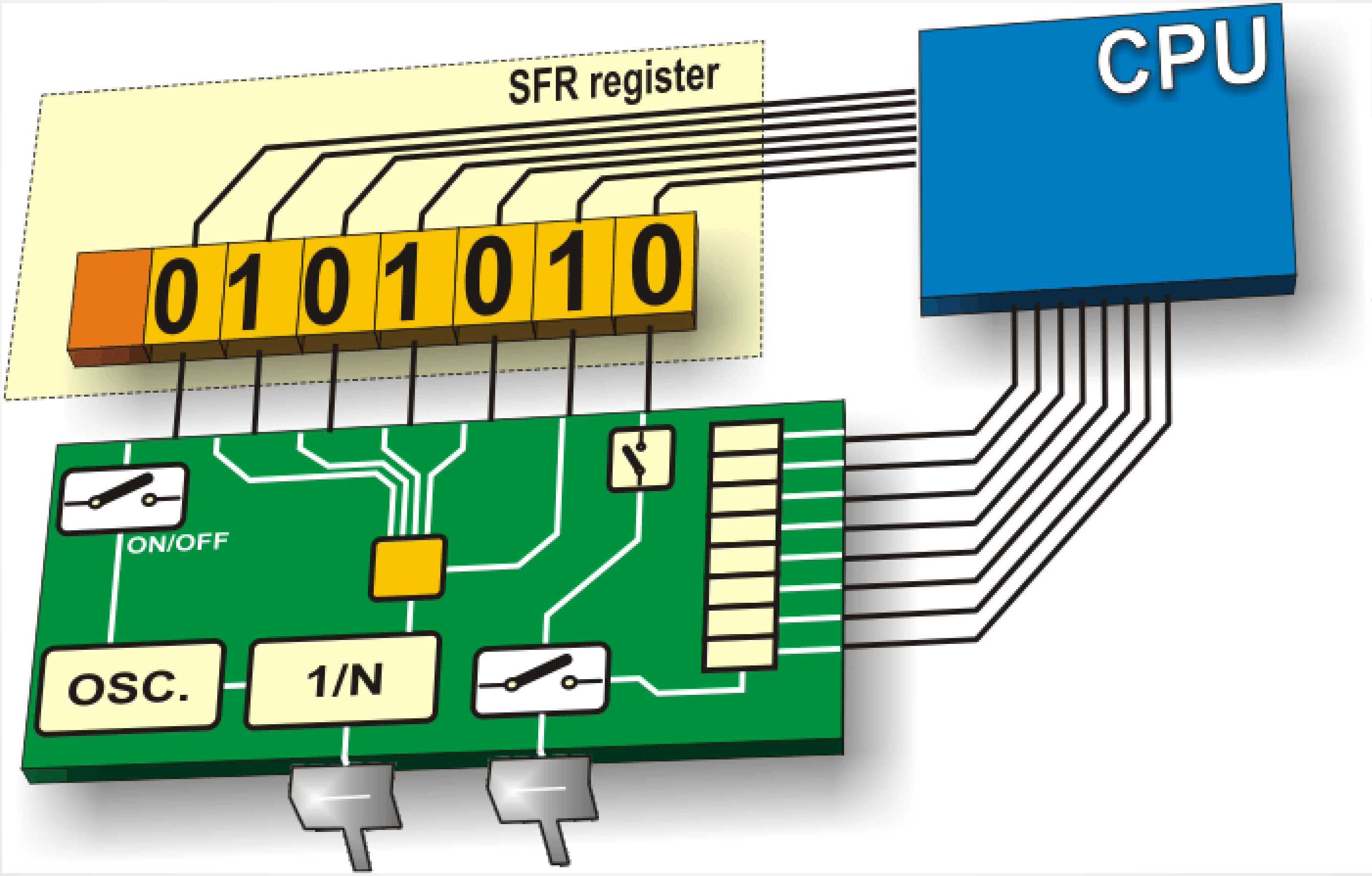


(PCINT14/$\overline{\text{RESET}}$) PC6	1	28	PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)
VCC	7	22	GND
GND	8	21	AREF
(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC
(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)
(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1) PD7	13	16	PB2 ($\overline{\text{SS}}$/OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)

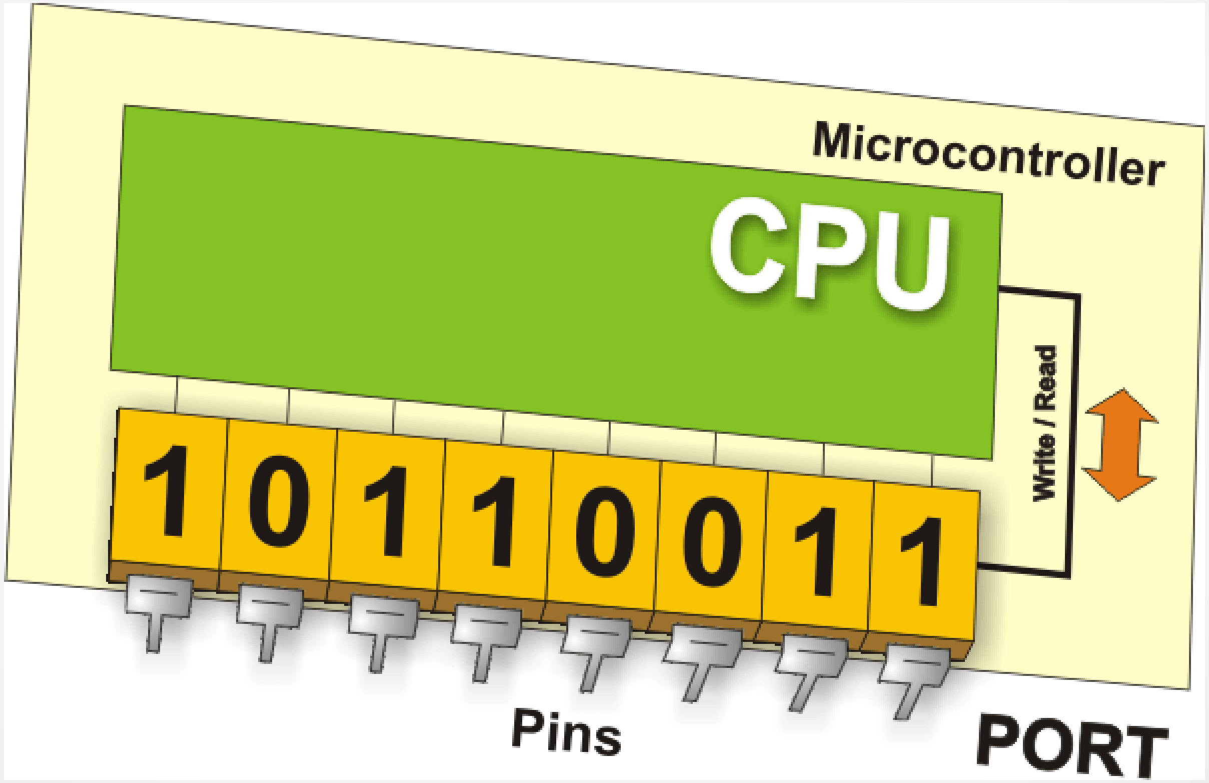
Reading and writing to memory



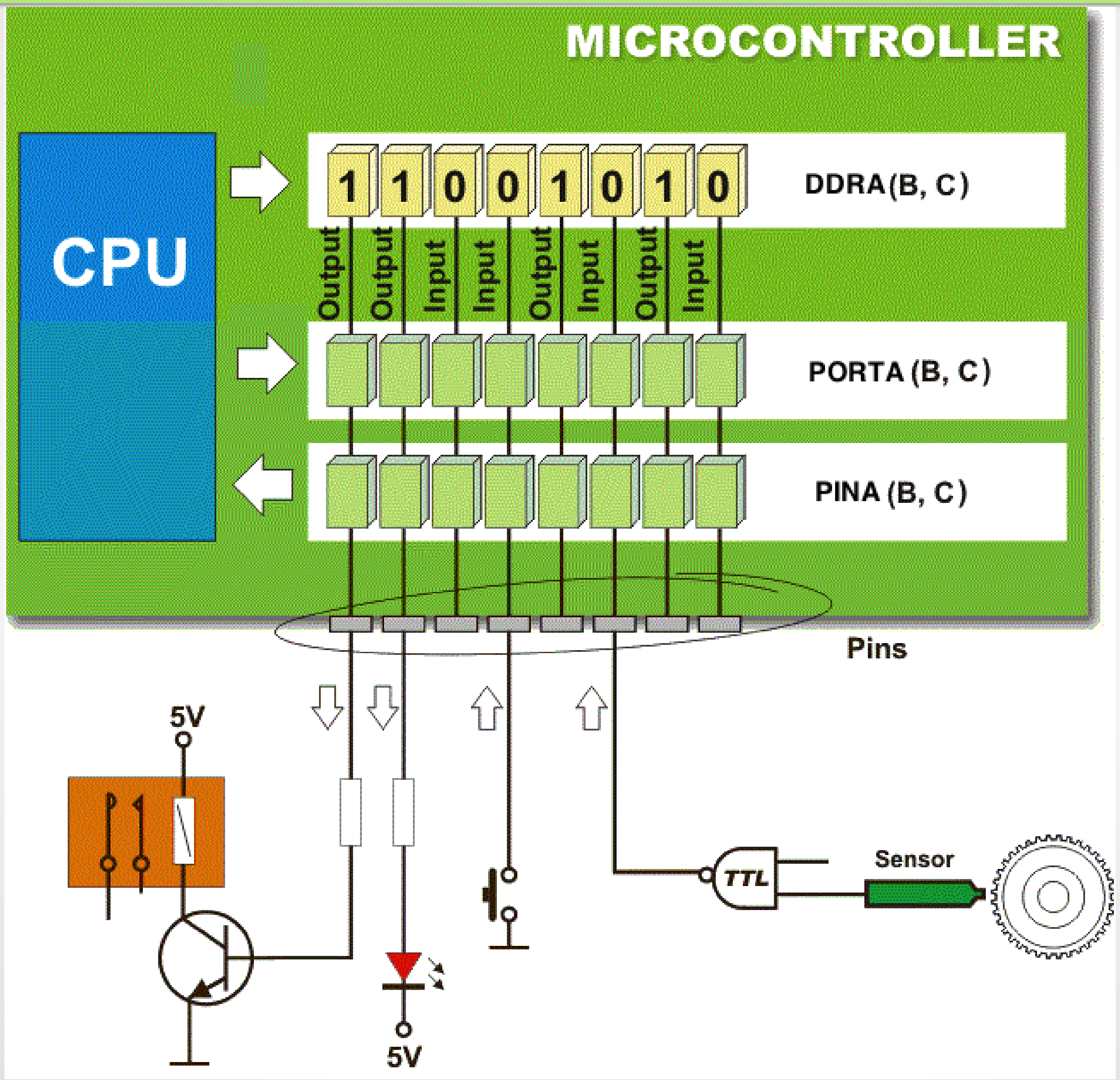
SFR



PORTS – simplified idea



PORT SFR's


MICROCONTROLLER
CPU
1 1 0 0 1 0 1 0
DDRA(B, C)
Output
Output
Input
Input
Output
Input
Output
Input
PORTA (B, C)
PINA (B, C)
Pins
5V
5V
TTL
Sensor

Port control registers

Bit pos	7	6	5	4	3	2	1	0
DDRB								

Sets the direction of the pin

	7	6	5	4	3	2	1	0
PORTB								

Logic values written into these bits appear on the pins as voltages
(if corresponding pin is an output pin)

	7	6	5	4	3	2	1	0
PINB								

The bits in here will tell you the logic level (high or low) on the pin
(assuming the pin is an input pin)

Port control registers - example

Bit pos	7	6	5	4	3	2	1	0
DDRB	1	0	1					

Sets the direction of the pin

	7	6	5	4	3	2	1	0
PORTB	1		0					

Logic values written into these bits appear on the pins as voltages
(if corresponding pin is an output pin)

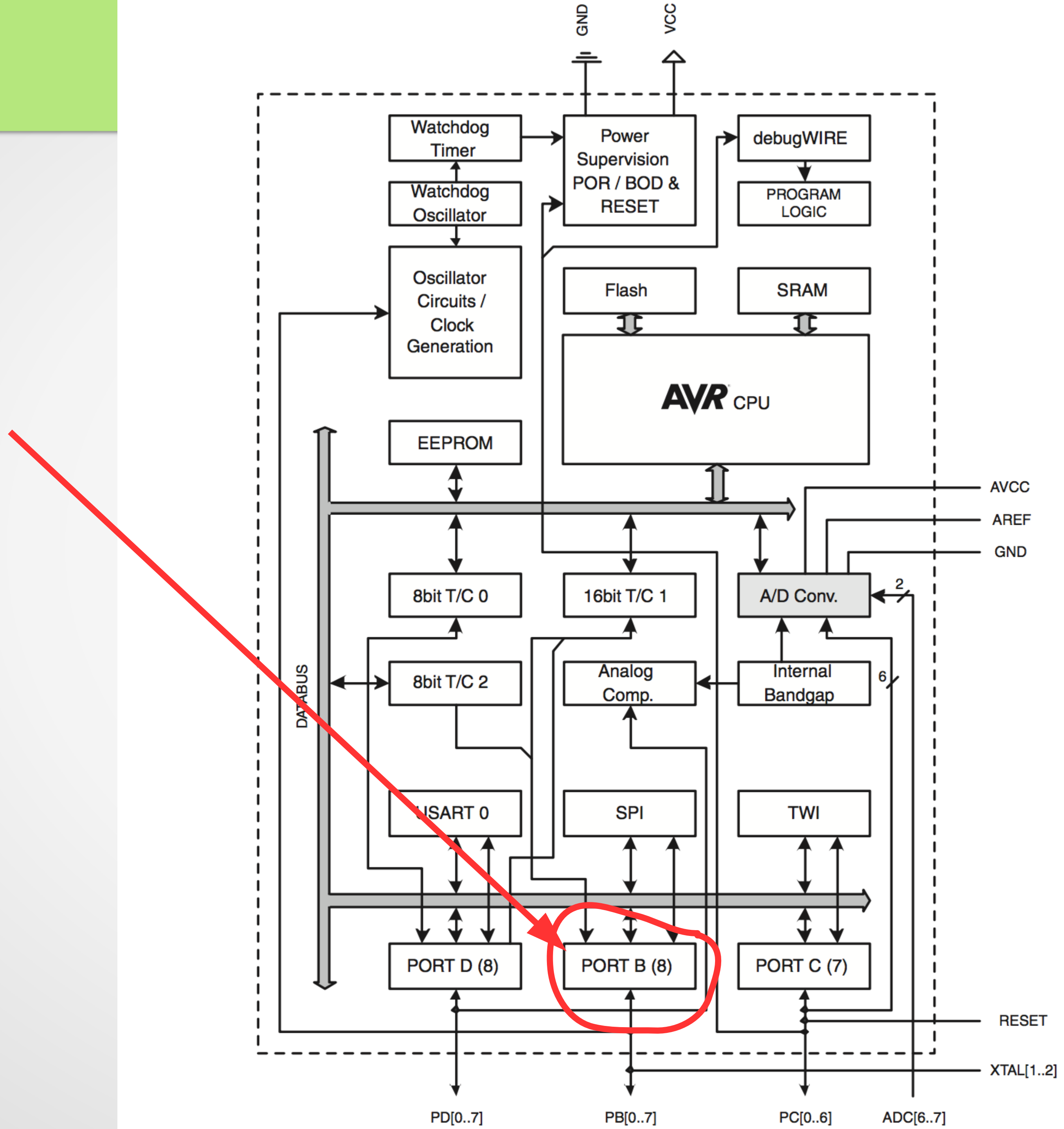
	7	6	5	4	3	2	1	0
PINB		1 or 0						

The bits in here will tell you the logic level (high or low) on the pin
(assuming the pin is an input pin)

DDRB,
PORTB
and
PINB

are all in
here!

Figure 2-1. Block Diagram



Write to input pin?

- What happens if I write to an input pin?
  - Enables pull-up resistor on pin
  - Function of PORTB depends on direction of pin (DDRB)

	7	6	5	4	3	2	1	0
DDRB					0			

	7	6	5	4	3	2	1	0
PORTB					1			

	7	6	5	4	3	2	1	0
PINB								

Pull-up resistors

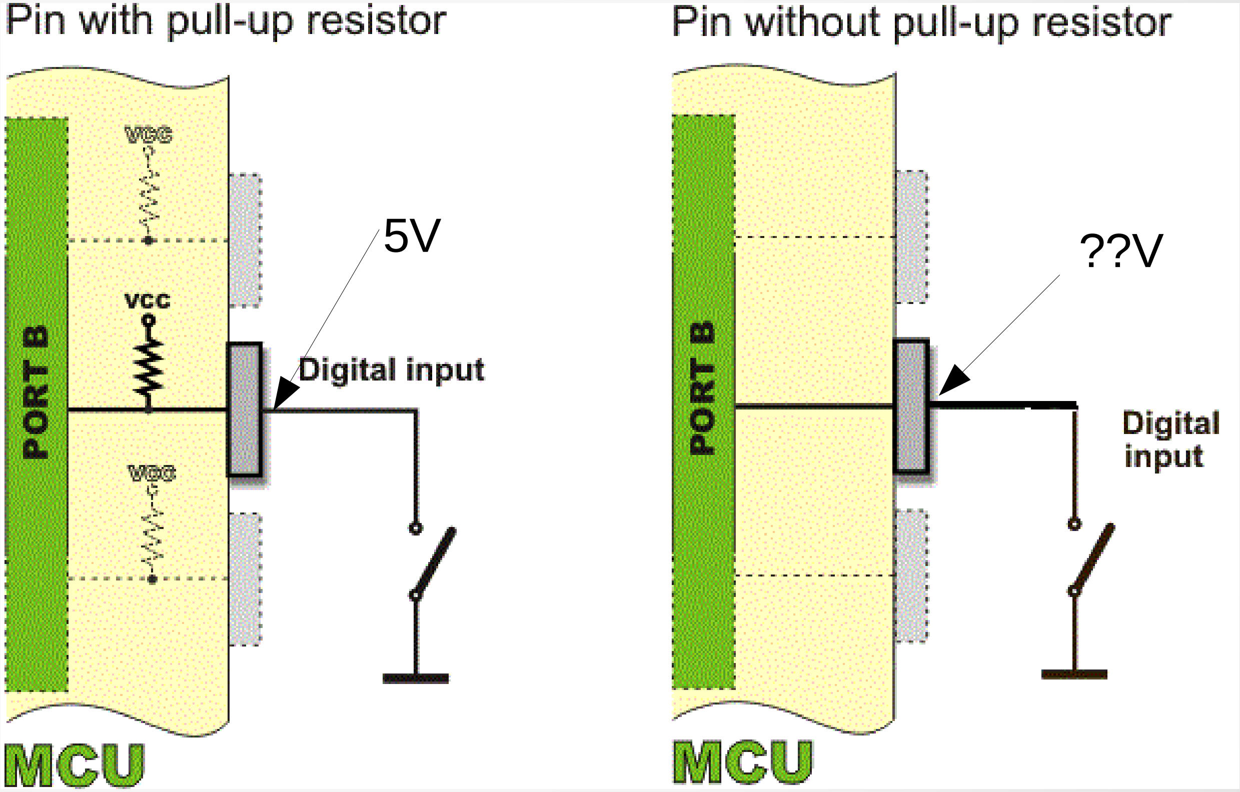
Pull-up resistors stop unconnected pins “floating”

Floating logic levels are bad news in a digital system!

Value read in corresponding bit in PIN register would flicker...

On AVR you can switch on and off the pull-up resistor on an input pin using the corresponding PORT bit.

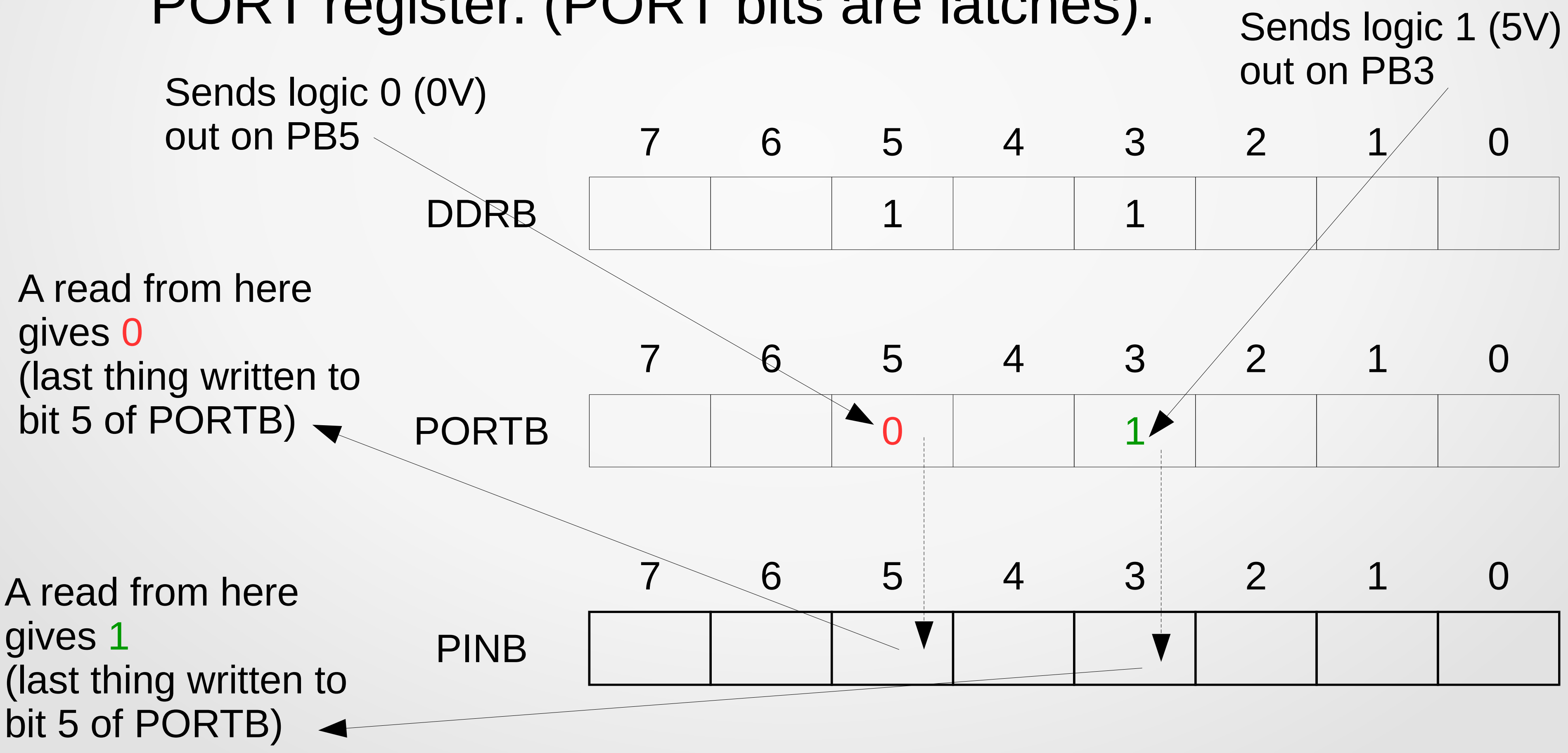
What PORT register does depends on direction of pin (DDR reg setting)



Don't always need pull-ups and sometimes don't want them.... usually do...

Read from output pin?

- What happens if I read from an output pin?
  - Just get last thing written to the corresponding bit in the PORT register. (PORT bits are latches).



Sends logic 1 (5V) out on PB3

Sends logic 0 (0V) out on PB5

	7	6	5	4	3	2	1	0
DDRB			1		1			
PORTB			0		1			
PINB								

A read from here gives 0 (last thing written to bit 5 of PORTB)

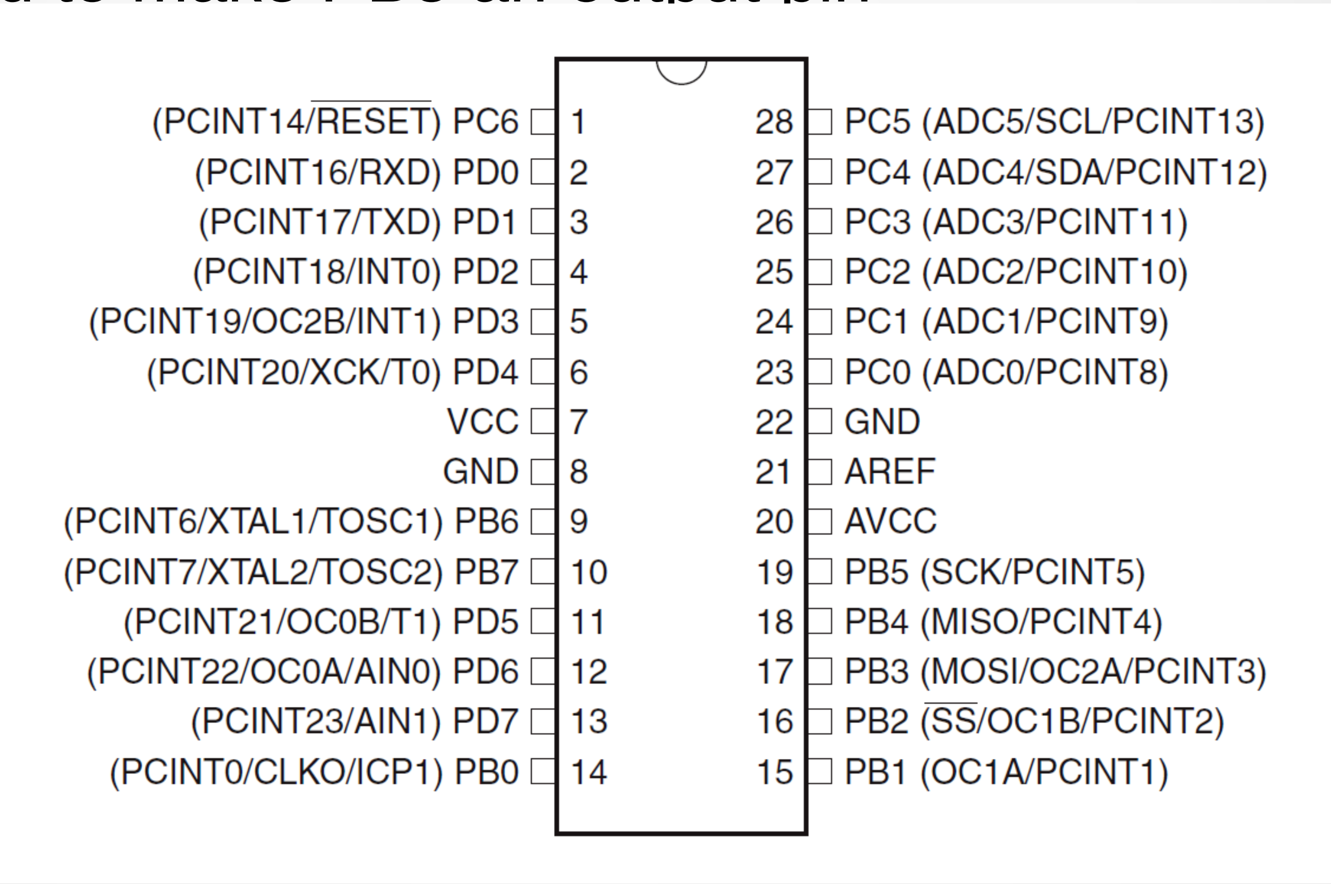
A read from here gives 1 (last thing written to bit 5 of PORTB)

Blinking an LED

- Equivalent to “Hello World!” program in embedded programming
- To blink an LED connected to a microcontroller we need to first physically wire up and LED to an I/O pin and then in our program:
  1) Set the direction of the pin to be an output pin
  2) Send a logic '1' (5V) out on the pin
  3) Delay for a short period
  4) Send a logic '0' (0V) out on the pin
  5) Delay for a short period
  6) Loop to 2.

Blinking an LED

- On the Arduino board there is an LED already connected to PB5
- So we need to make PB5 an output pin



Making PB5 an output pin

- To make PB5 an output pin we need to set bit 5 in DDRB to be a 1 – make the pin an output pin.

	7	6	5	4	3	2	1	0
DDRB			1					

Making PB5 an output pin

	7	6	5	4	3	2	1	0
DDRB			1					

- DDRB = 0b00100000;
- DDRB = 0x20;
- Side effect?

	7	6	5	4	3	2	1	0
DDRB	0	0	1	0	0	0	0	0

Making the voltage on PB5 5V

	7	6	5	4	3	2	1	0
PORTB			1					

- PORTB = 0b00100000;
- PORTB = 0x20;
- Side effect?

	7	6	5	4	3	2	1	0
PORTB	0	0	1	0	0	0	0	0

Making the voltage on PB5 0V

	7	6	5	4	3	2	1	0
PORTB			0					

- PORTB = 0b00000000;
- PORTB = 0x00;
- Side effect?

	7	6	5	4	3	2	1	0
PORTB	0	0	0	0	0	0	0	0

Making the code delay

- Our chip runs at 16MHz
- That's one clock cycle every 1/16000000 seconds
  - 1/16000000 = 0.0000000625 seconds
  - 1 clock cycle every 62.5 nano seconds
- Our chip can execute one machine instruction every clock cycle
- If we down add delays between switching on and off the LED we won't see it – in fact the LED won't have time to switch on or off...

Making the code delay

- We use a library function provided with our compiler
  - The avrgcc compiler
- _delay_ms(500);
  - Causes our code to delay (pause) at this point for 500 milliseconds
  - Also have _delay_us(xx) function which causes a delay for the number of microseconds proscribed.

Making the LED blink

```
#include <avr/io.h>
#include <util/delay.h>


int main(void)
{
   //Setup code that runs once goes here
   //Equivalent to setup() in Arduino

   DDRB = 0x20; //0b00100000


   //Code here loops “forever”
   //Similar to loop() in Arduino

   while (1)
   {
      PORTB = 0x20; //0b00100000
      _delay_ms(500);

      PORTB = 0x00;
      _delay_ms(500);
   }
}
```