Last Time

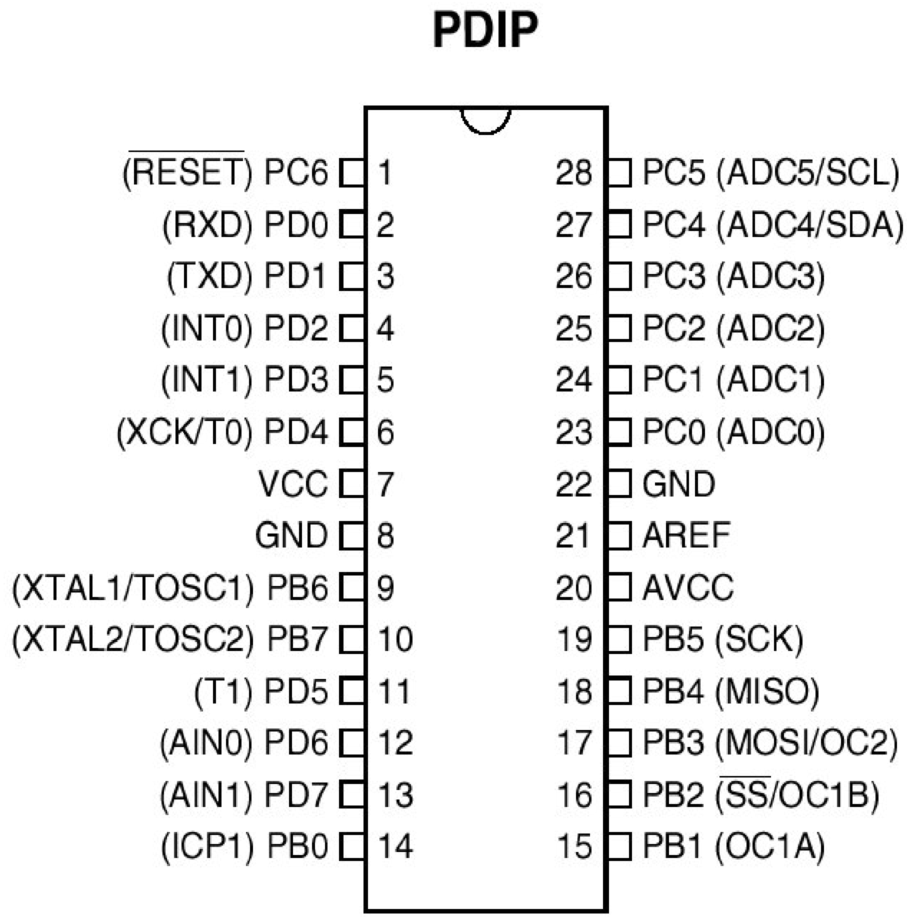
- Resistors
- Diodes
- Transistors

Today

- A bit more on transistors
- Atmel microcontroller basics

Atmel Mega8 Basics

- Complete, stand-alone computer
- Ours is a 28-pin package
- Most pins:
  - Are used for input/output
  - How they are used is configurable



Key Features

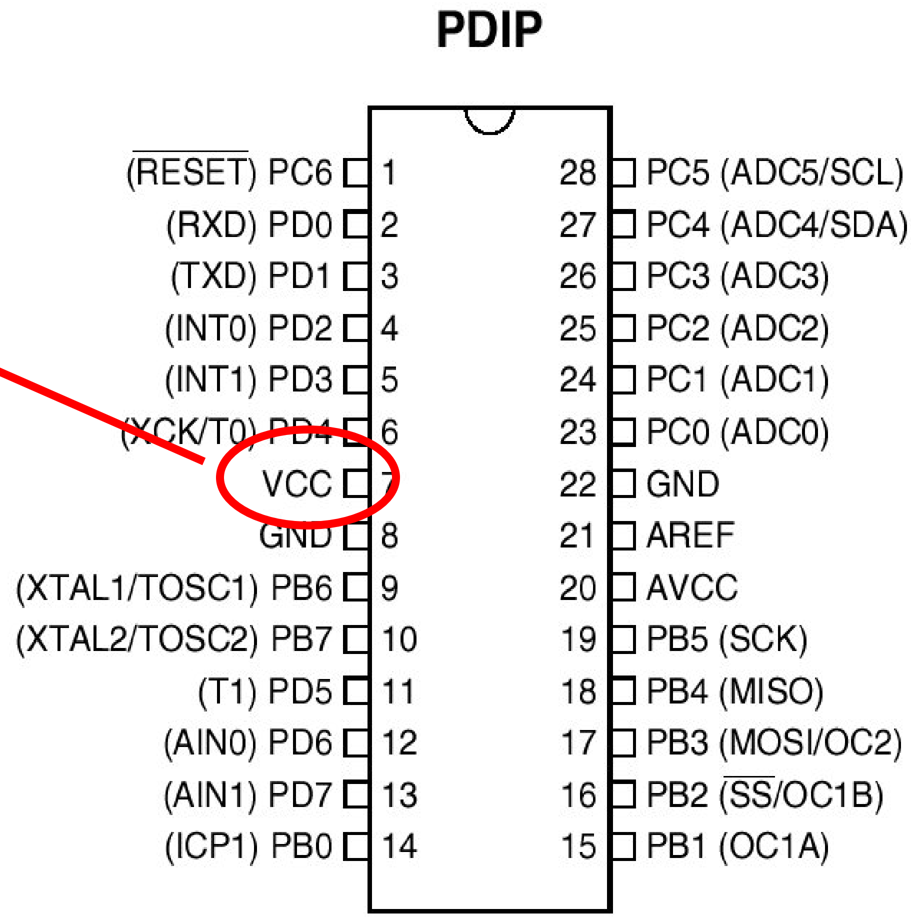
- Up to 16 MIPS (single cycle for most instructions)
- ~23 digital pins: configurable as inputs or outputs
- 6 channel, 10-bit analog-to-digital converter
- Serial communication support: RS232, SPI, I2C
- 3 counter/timers (2 8-bit; 1 16-bit)
- Internal/external interrupt support
- Brown-out detection
- Internal oscillator (1 MHz)
- Bootloader support
- Sleep mode
- Watchdog timer

Interrupt Sources

- External pins: state change; falling/rising edge
- Timer/counters: when counter overflows
- Communication peripherals
- Brown out
- Analog to digital conversion complete

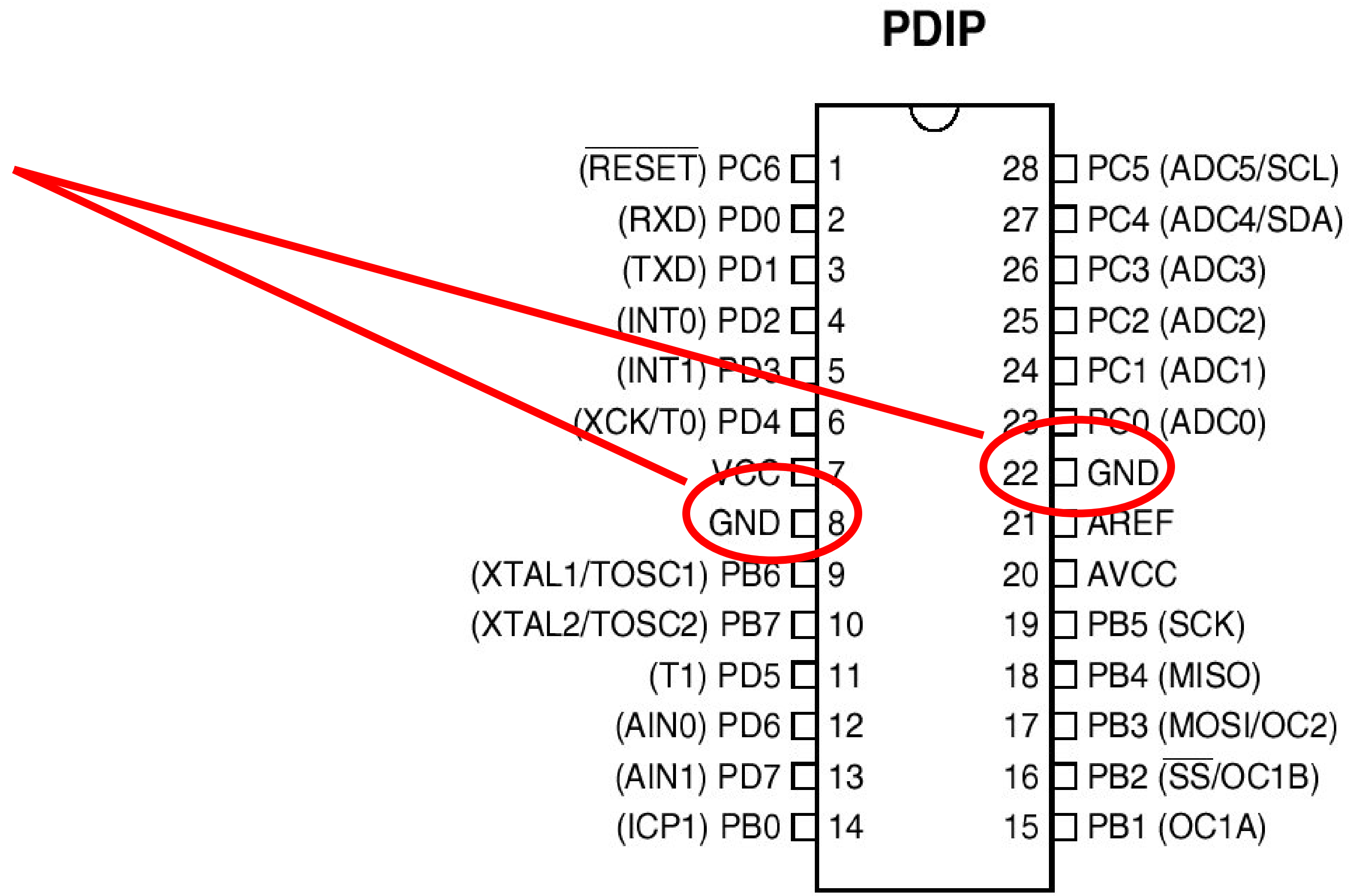
Atmel Mega8 Basics

Power (we will use +5V)



Atmel Mega8 Basics

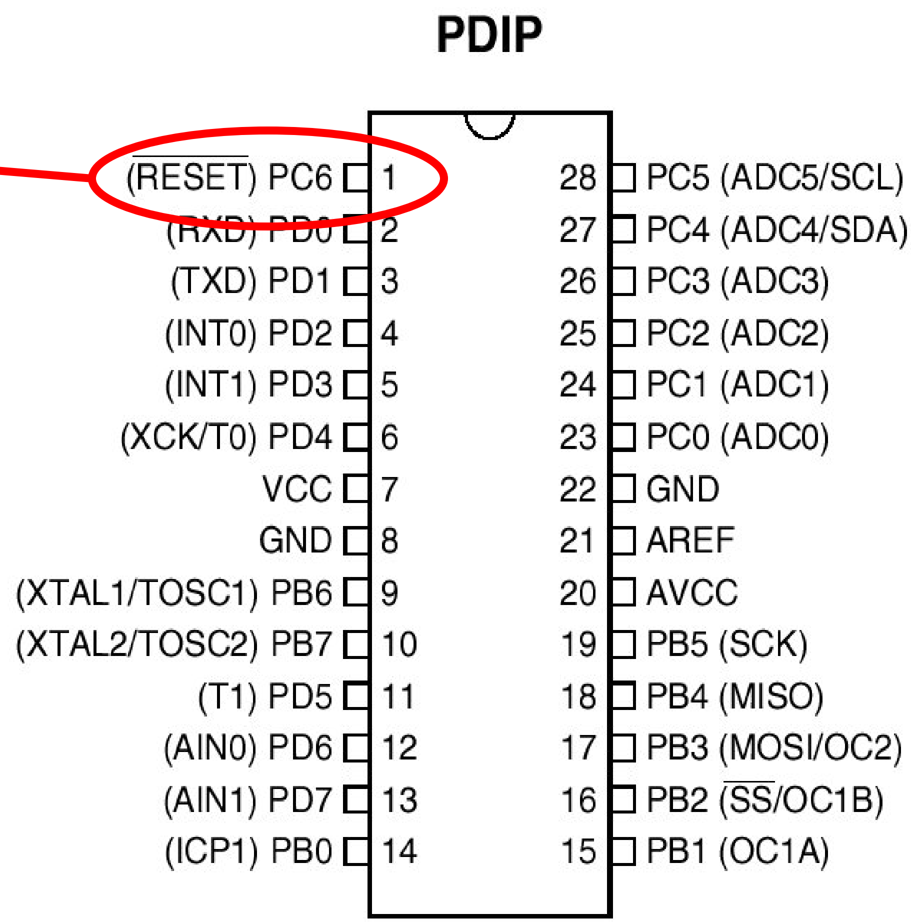
Ground



Atmel Mega8 Basics

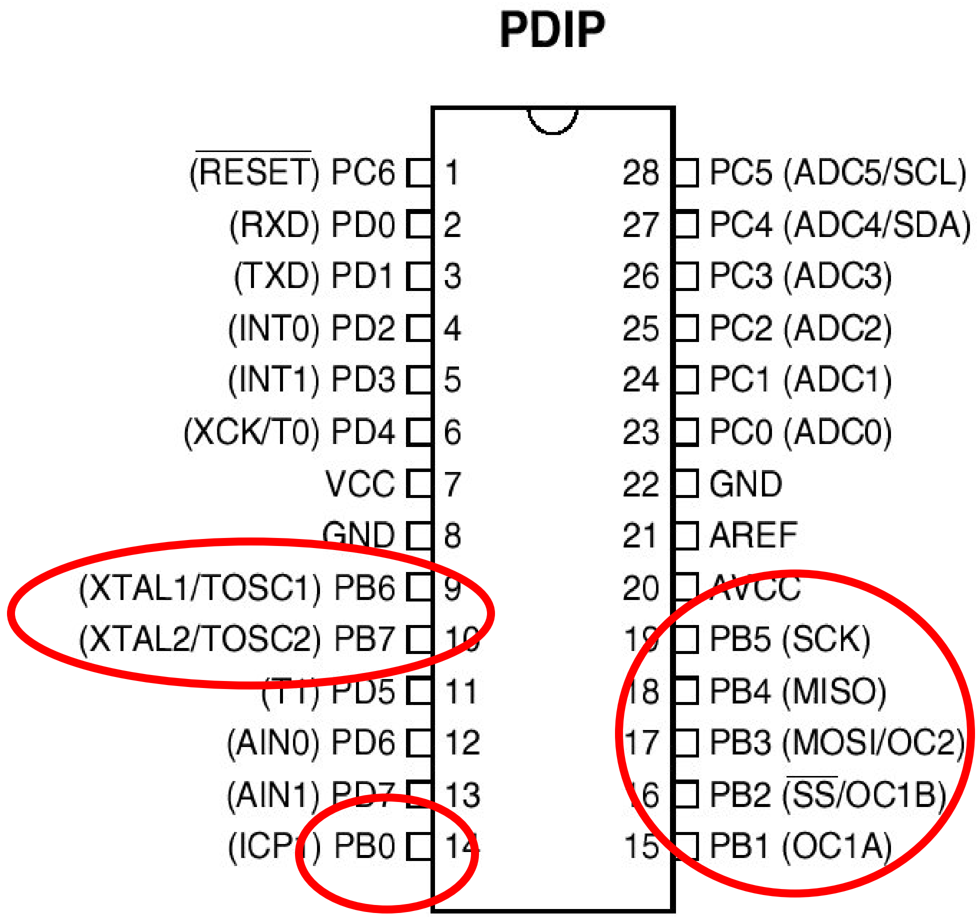
Reset

- Bring low to reset the processor
- In general, we will tie this pin to high through a pull-up resistor (10K ohm)



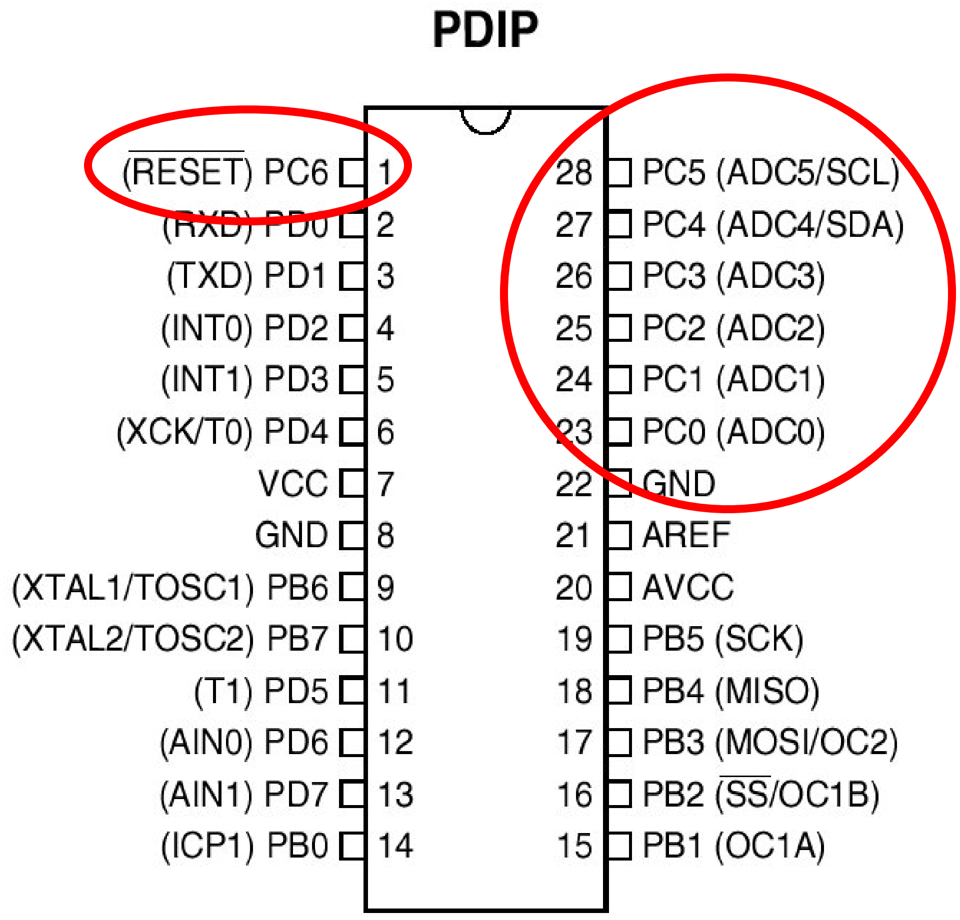
Atmel Mega8 Basics

PORT B



Atmel Mega8 Basics

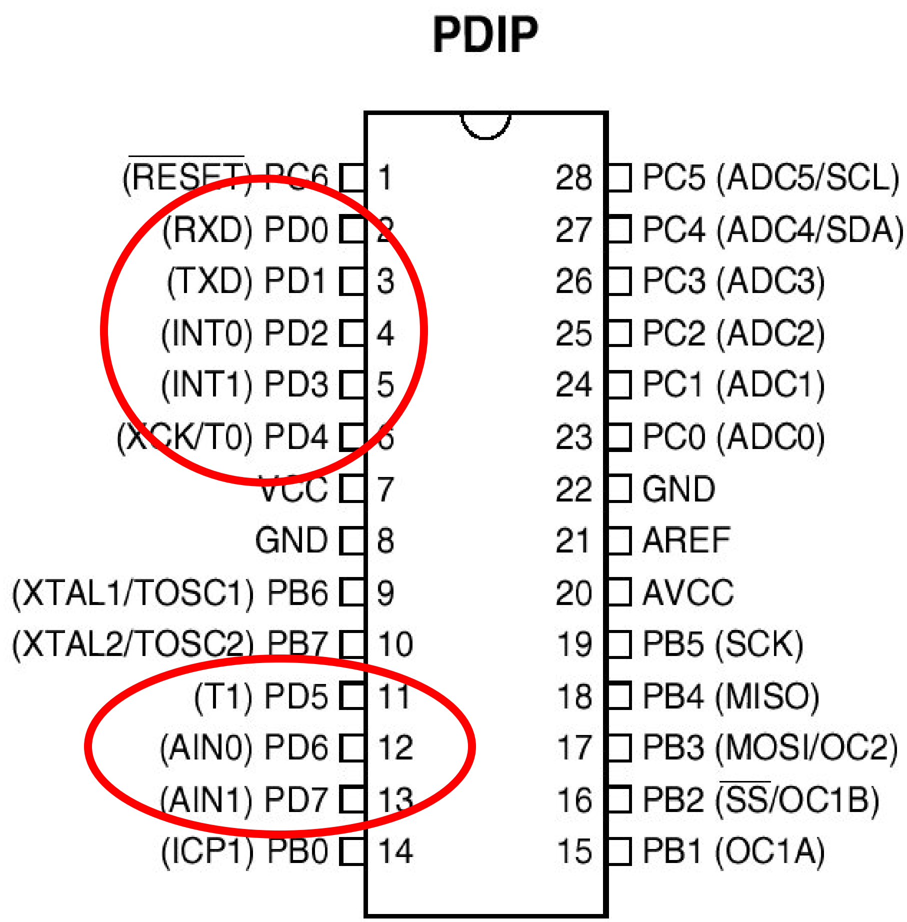
PORT C



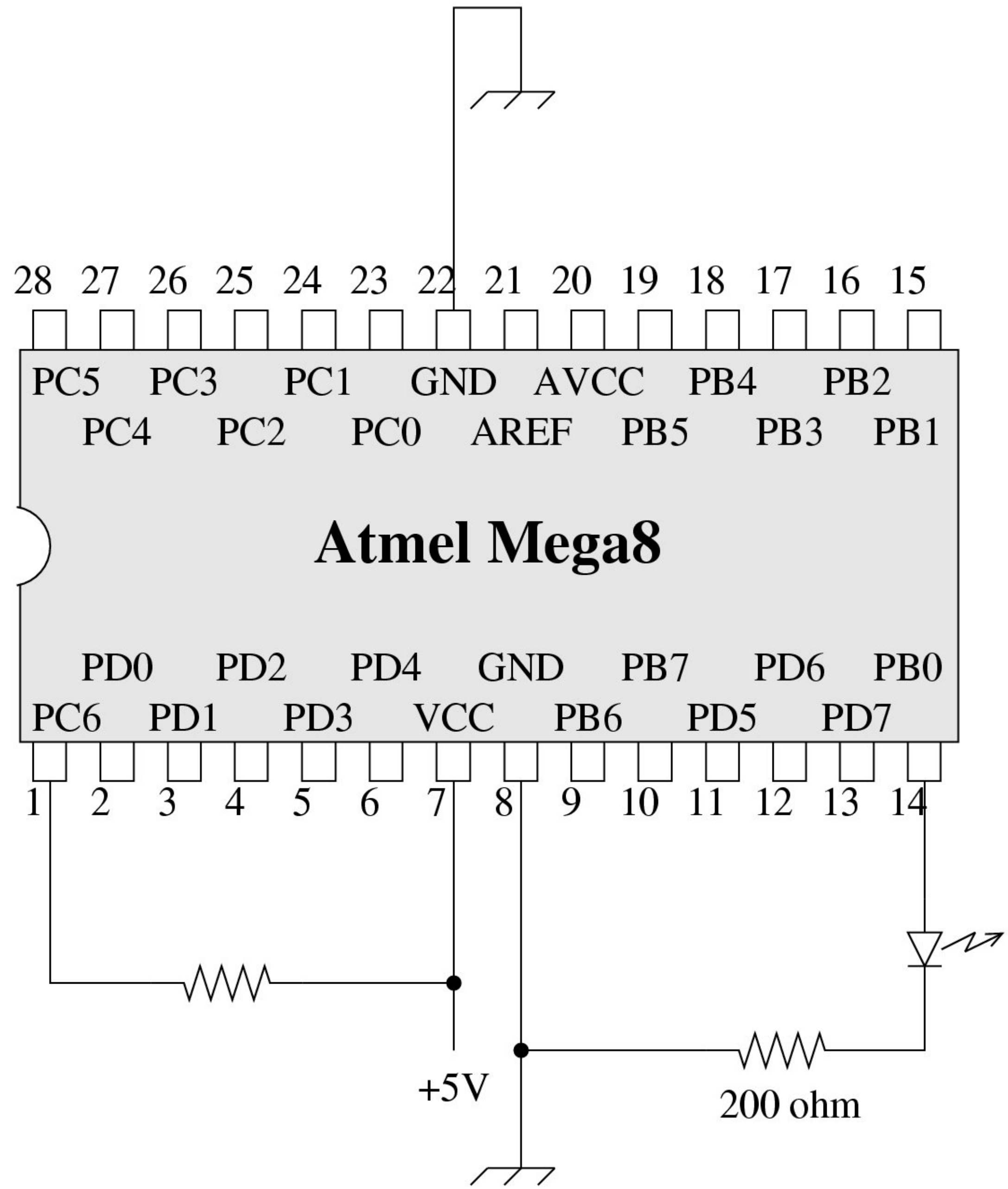
Atmel Mega8 Basics

PORT D

(all 8 bits are available)



A First Circuit



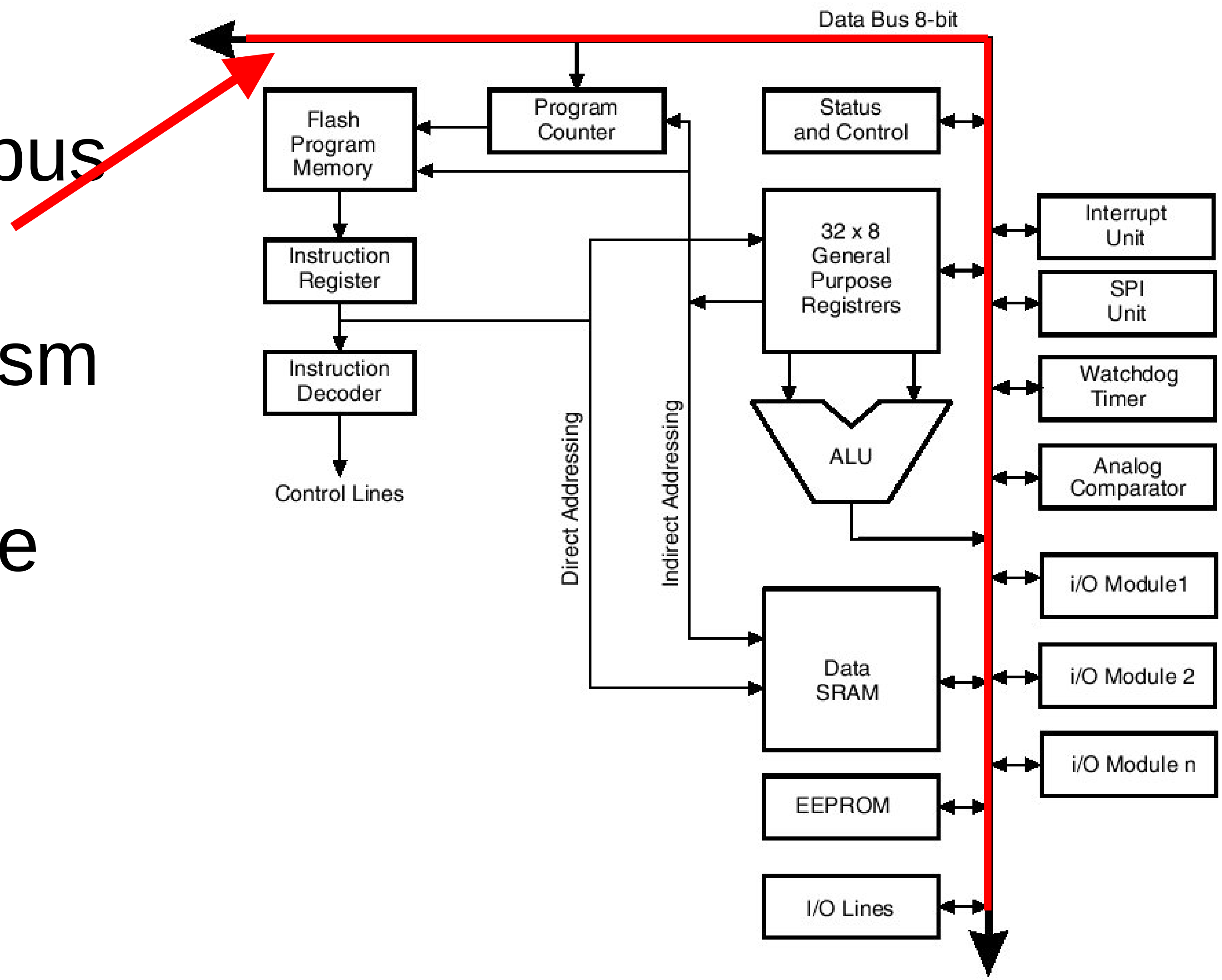
Common Special-Purpose Registers

- Program counter
- Status register
- Instruction register
- Stack pointer
- Peripheral control is all done through registers

Atmel Mega8

8-bit data bus

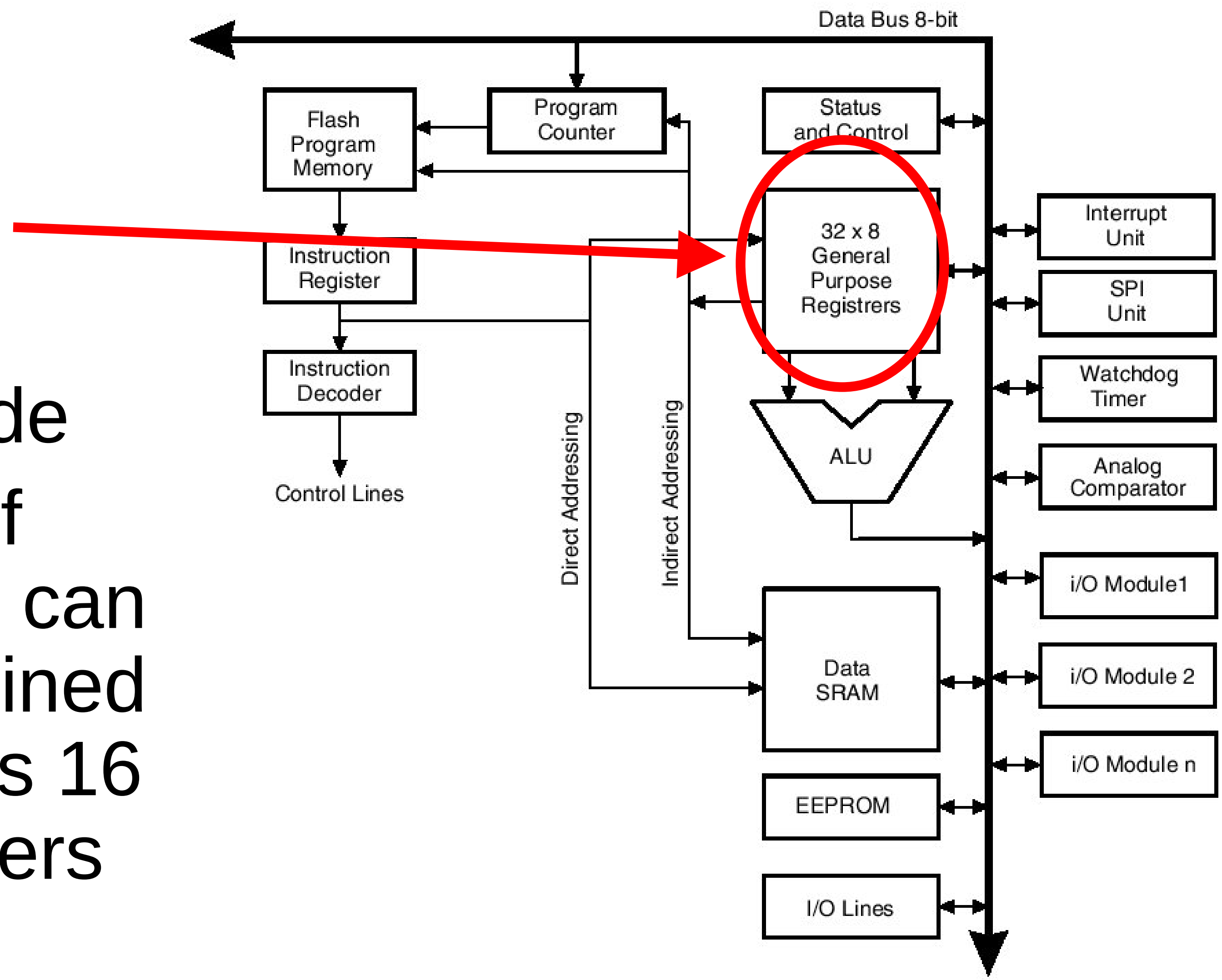
- Primary mechanism for data exchange



Atmel Mega8

32 general purpose registers

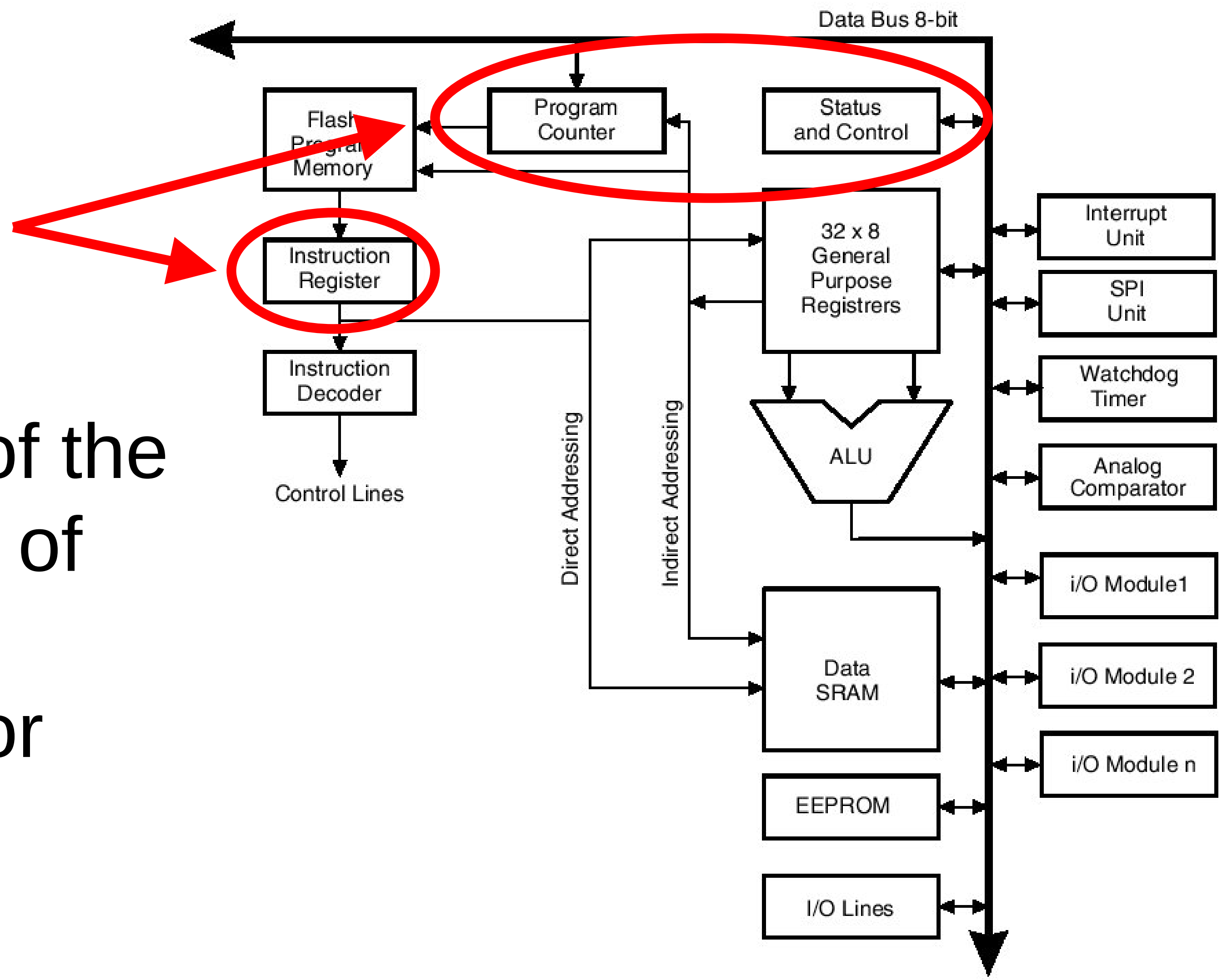
- 8 bits wide
- 3 pairs of registers can be combined to give us 16 bit registers



Atmel Mega8

Special purpose registers

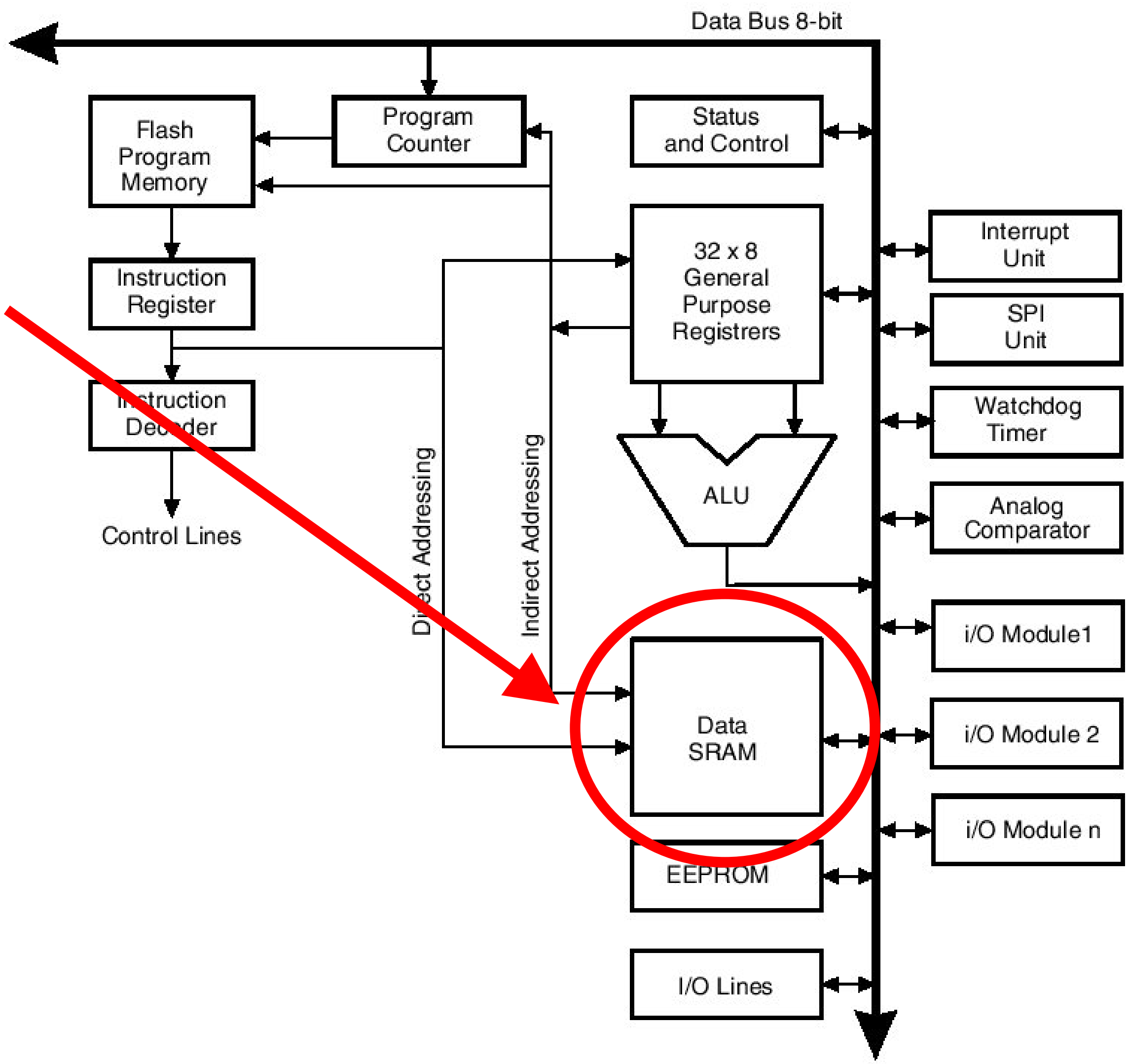
- Control of the internals of the processor



Atmel Mega8

Random Access Memory (RAM)

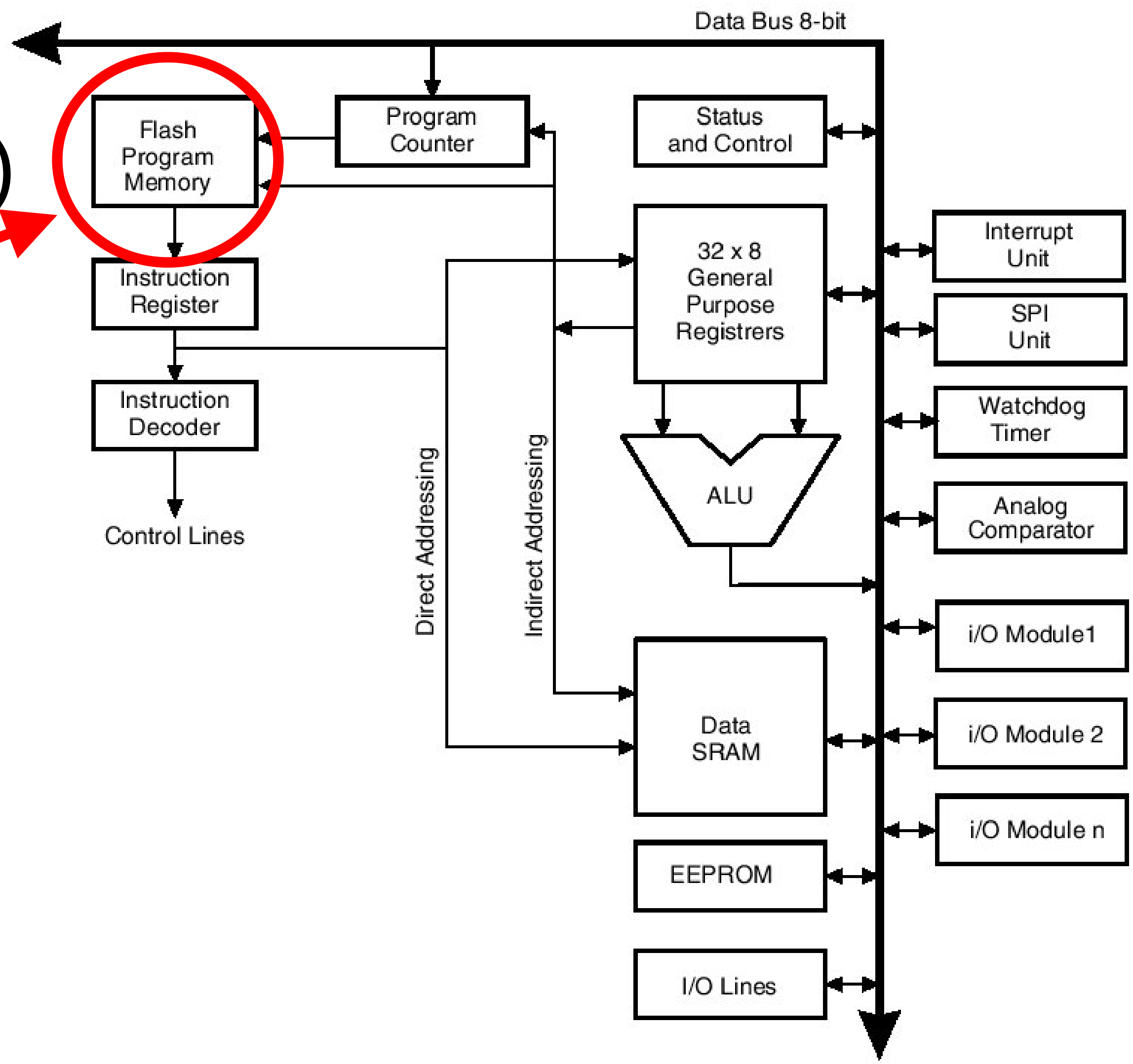
- 1 KByte in size
- Stack is stored here



Atmel Mega8

Flash (EEPROM)

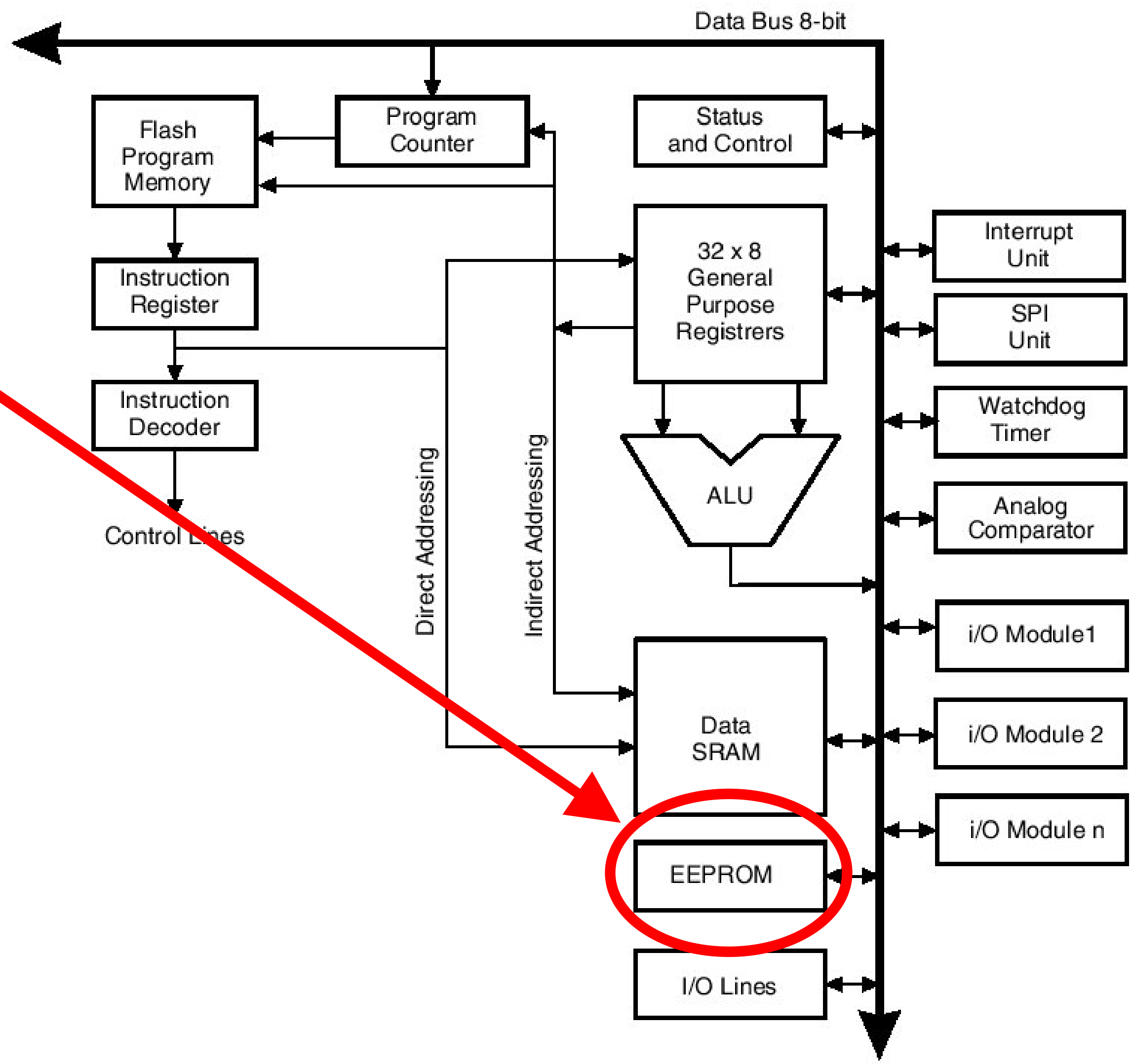
- Program storage
- 8 KByte in size
- 16 bit words



Atmel Mega8

EEPROM

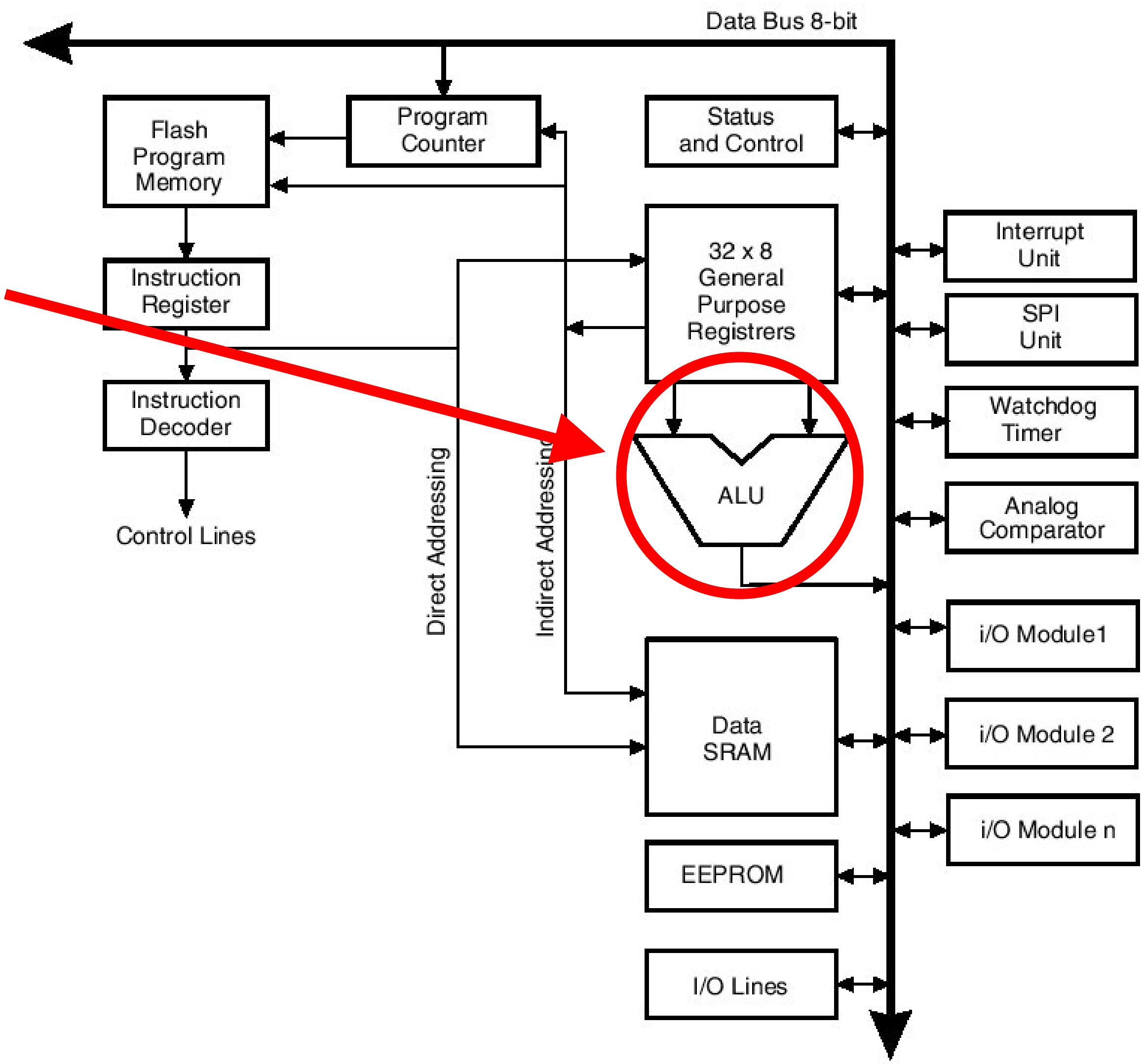
- Permanent data storage



Atmel Mega8

Arithmetic Logical Unit

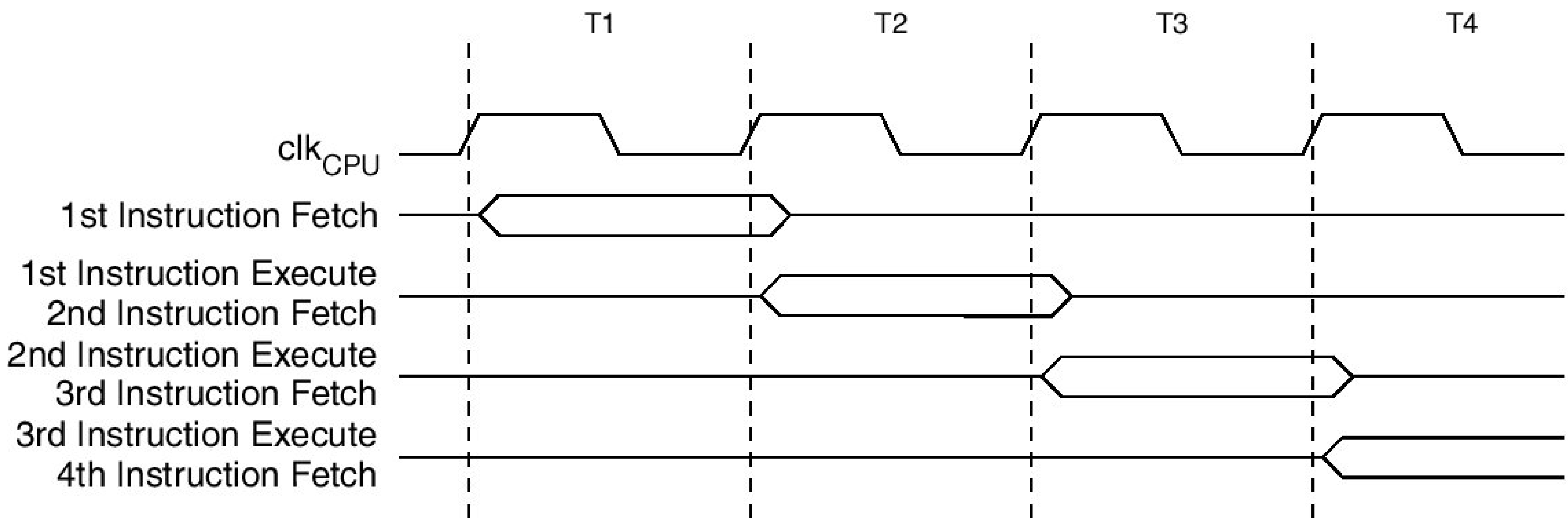
- Data inputs from registers
- Control inputs not shown (derived from instruction decoder)



Processors in the Atmel Family

- Memory/program size
- Different numbers and types of I/O pins
- Custom support for other communication protocols (e.g., CANbus)

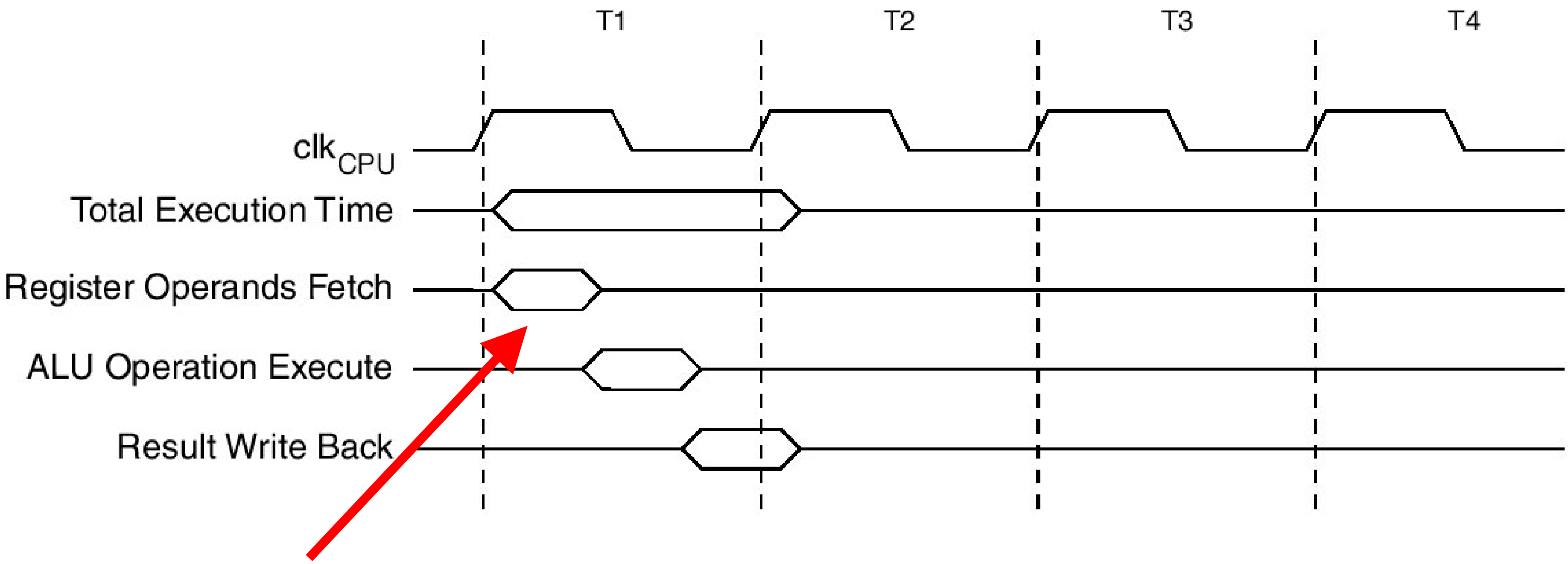
Instruction Fetch/Execution Cycle



From Atmel Mega8 spec

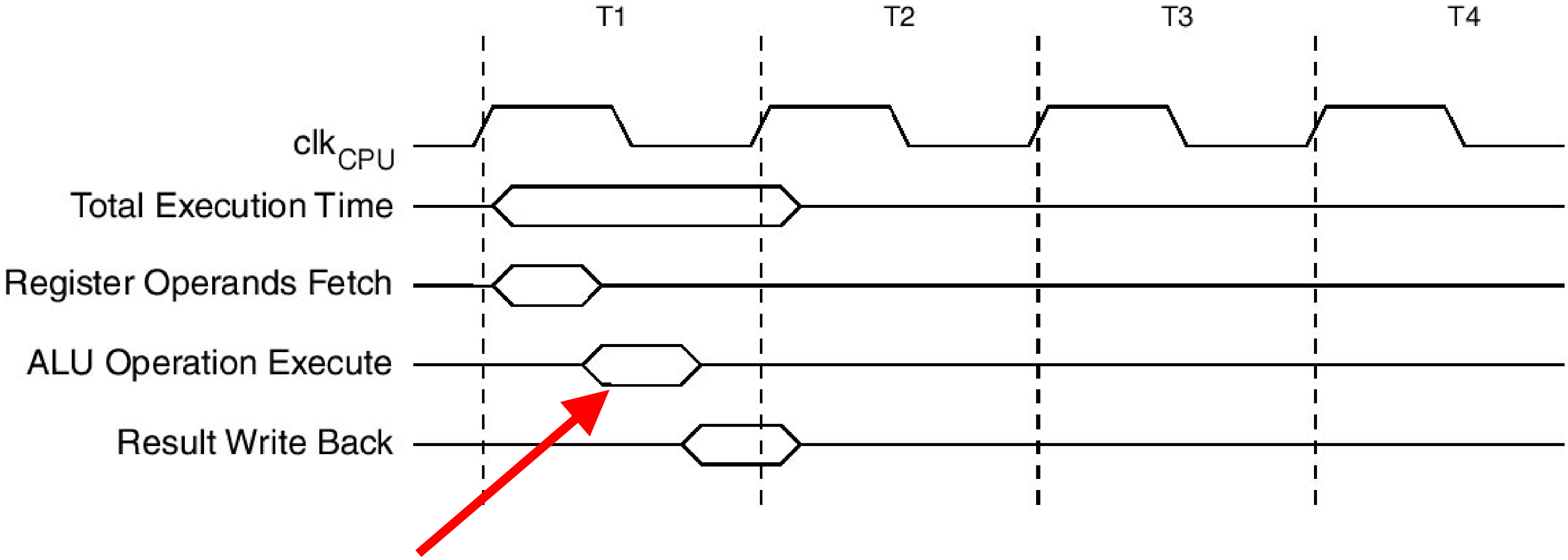
- While one instruction is being executed, the next is already being fetched from memory
- In many cases: each step happens on a single clock cycle

Instruction Execution Cycle



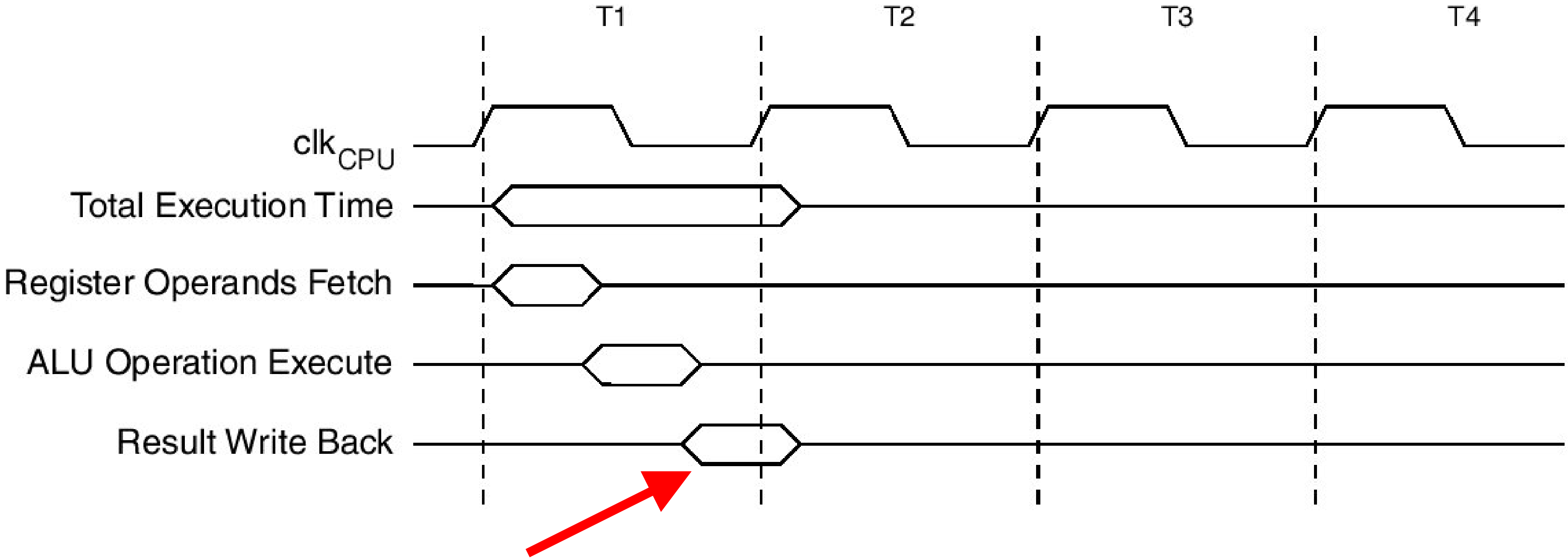
Address the registers and wait for the values to become available

Instruction Execution Cycle



Perform the operation dictated by the instruction

Instruction Execution Cycle

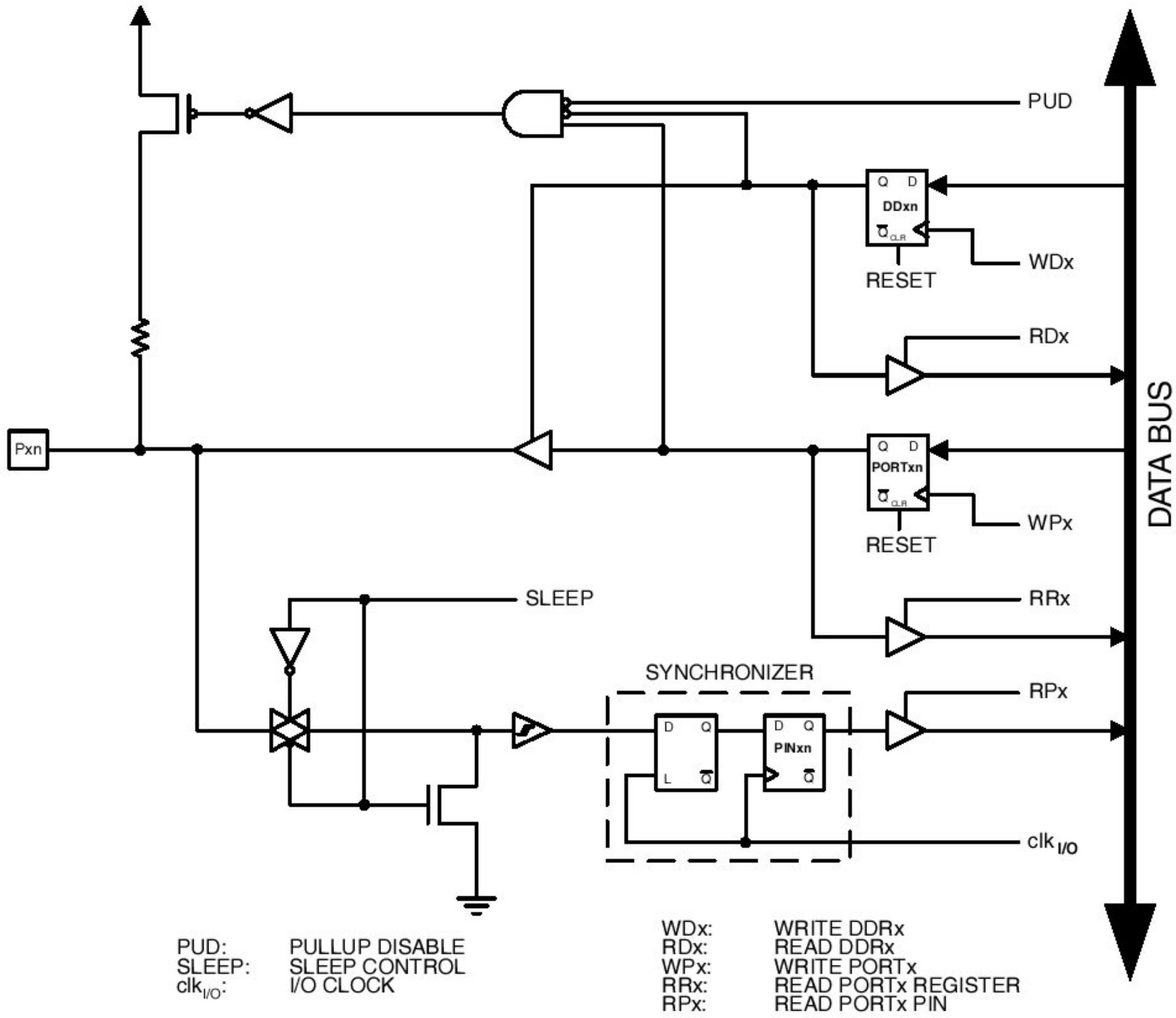


Result stored in destination register

Status register state changed

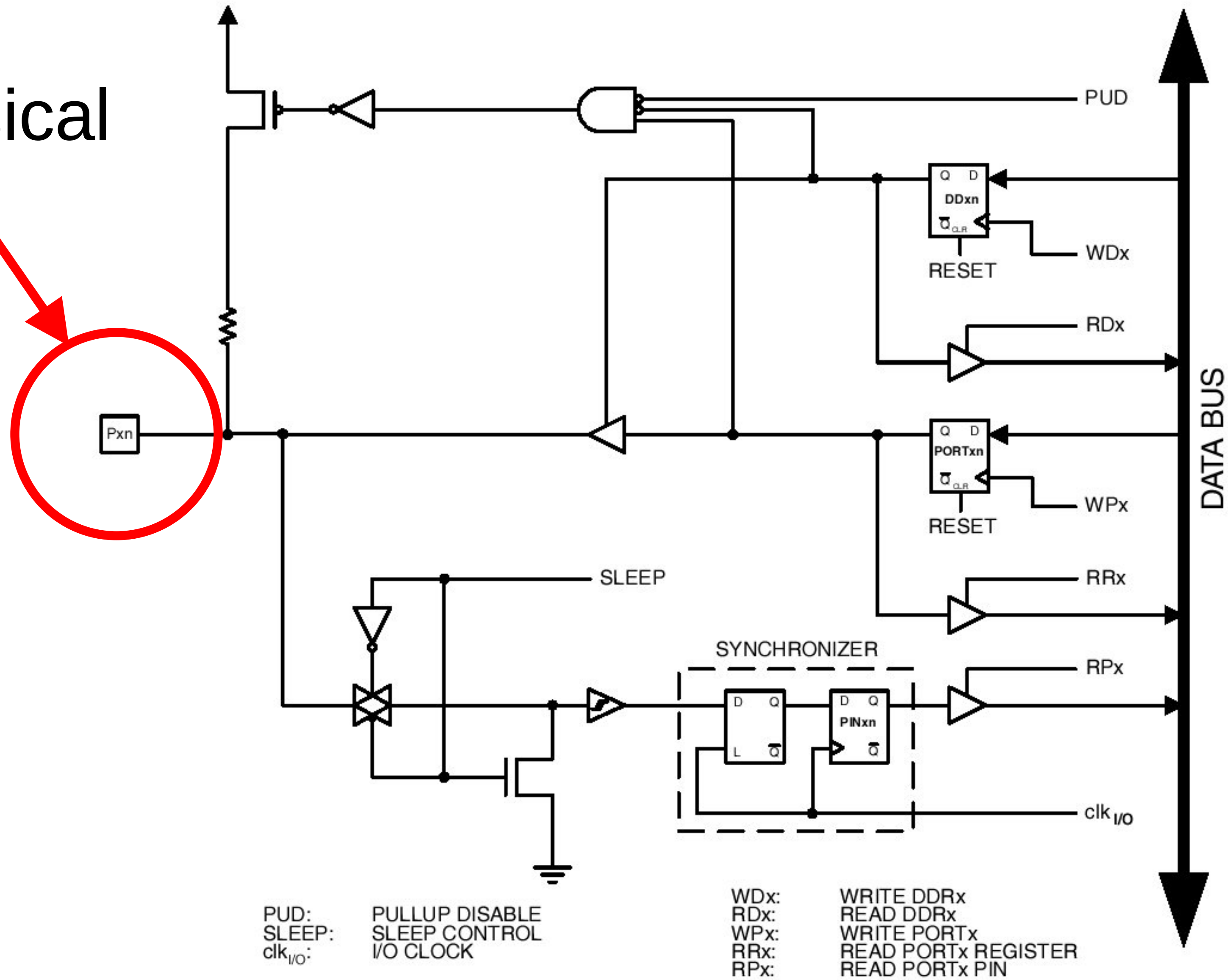
I/O Pin Implementation

Single bit of PORT B



I/O Pin Implementation

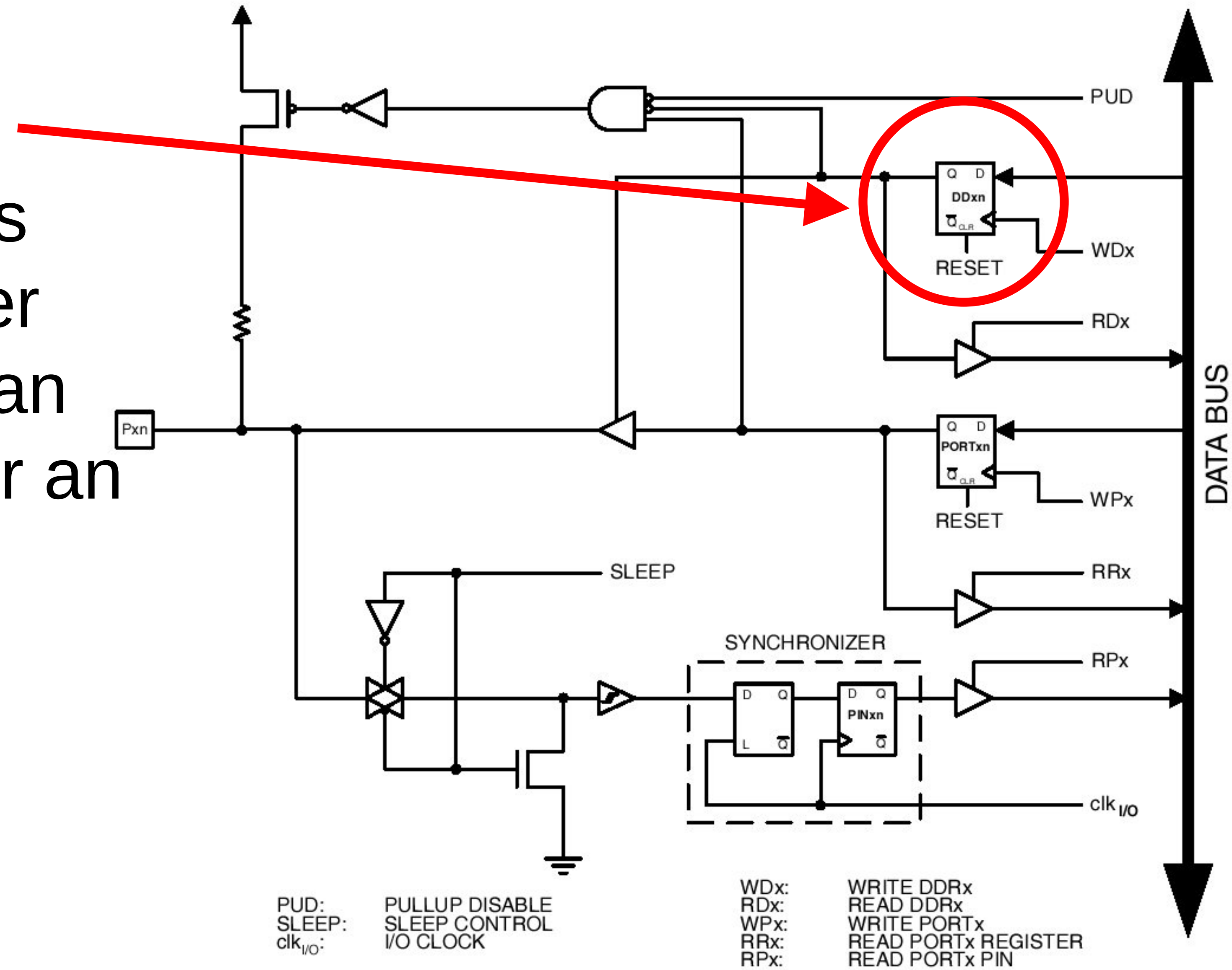
The physical pin



I/O Pin Implementation

DDRB

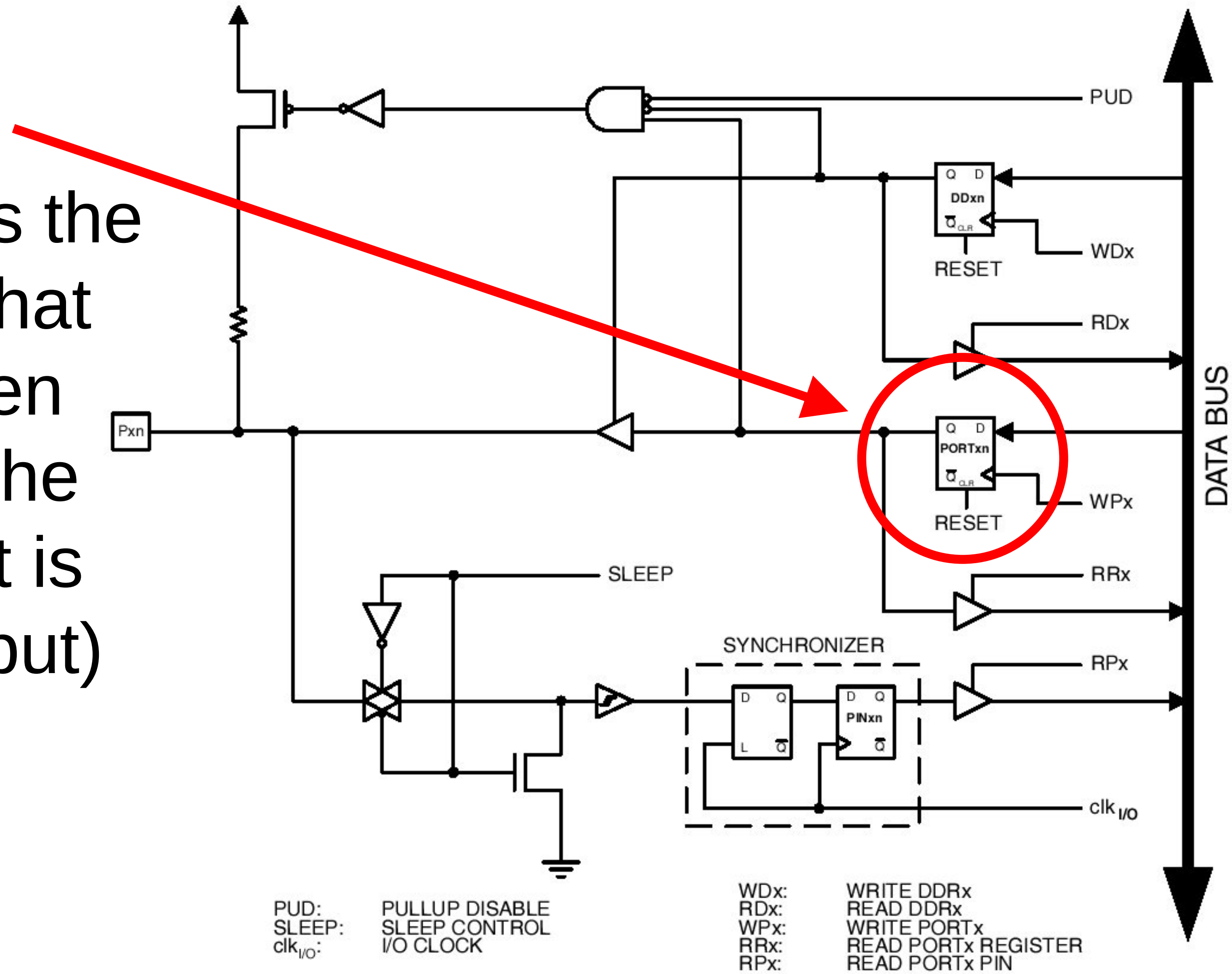
- Defines whether this is an input or an output



I/O Pin Implementation

PORTB

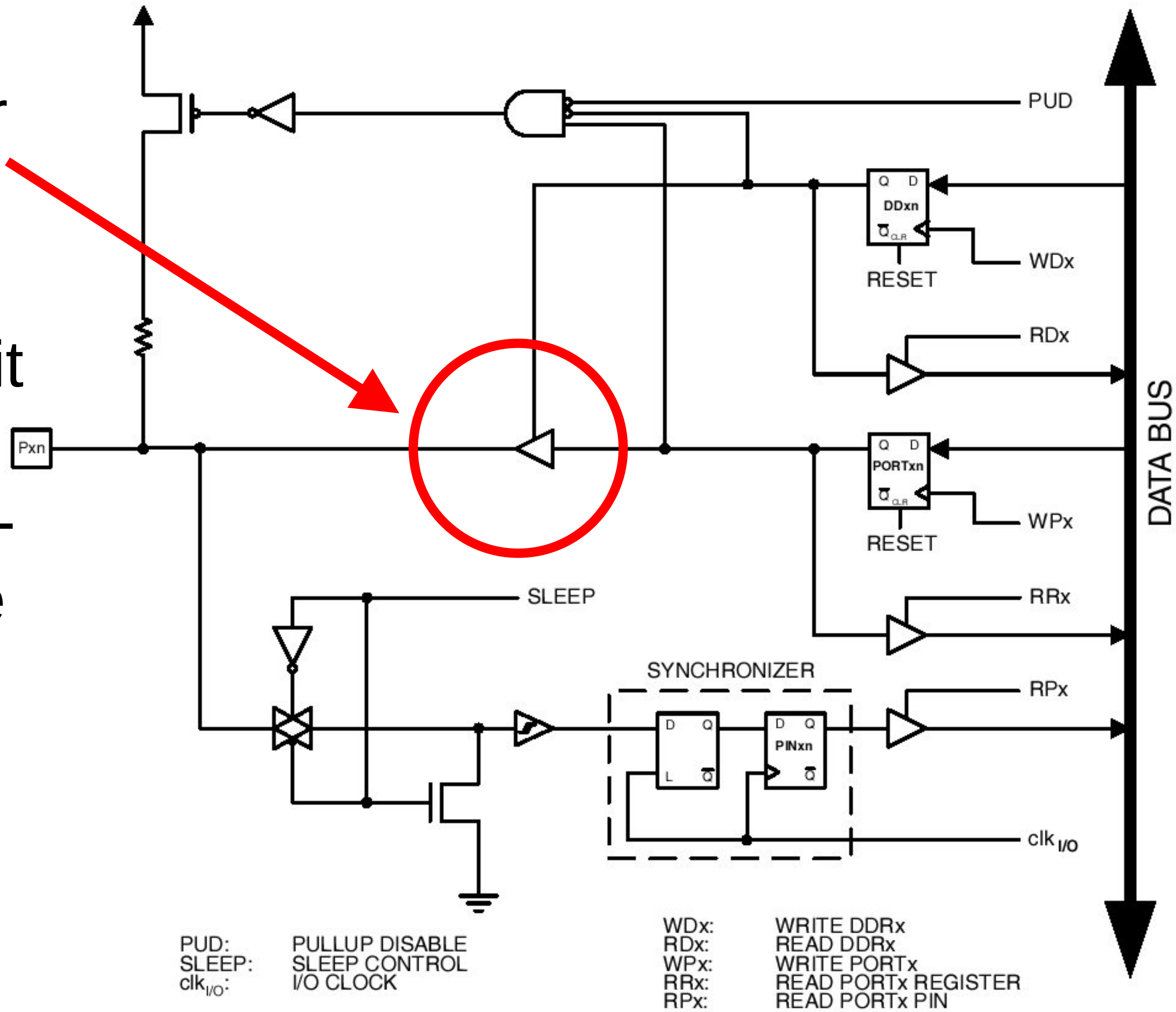
- Defines the value that is written out to the pin (if it is an output)



I/O Pin Implementation

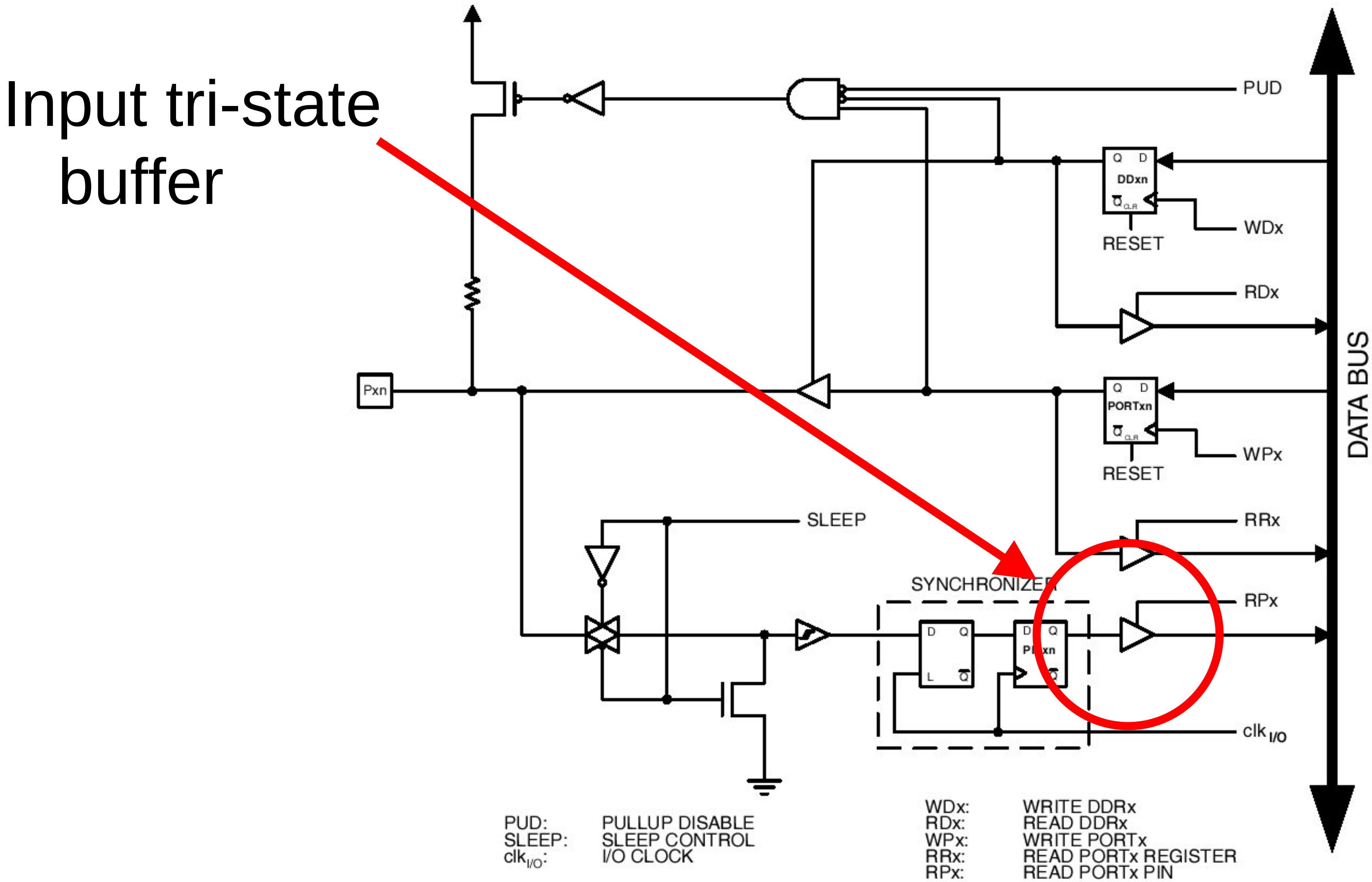
Tristate buffer

- When this pin is an output pin, it allows the PORTB flip-flop to drive the pin



I/O Pin Implementation

Input tri-state buffer



Bit Manipulation

PORTB is a register

- Controls the value that is output by the set of port B pins
- But – all of the pins are controlled by this single register (which is 8 bits wide)

- In code, we need to be able to manipulate the pins individually

Bit-Wise Operators

If A and B are bytes, what does this code mean?

```
C = A & B;
```

The corresponding bits of A and B are ANDed together

Bit-Wise Operators

If A and B are bytes, what does this code mean?

```
C = A & B;
```

Bit-Wise Operators

0 1 0 1 1 1 1 0	A
1 0 0 1 1 0 1 1	B
?	C = A & B

Bit-Wise Operators

0 1 0 1 1 1 1 0	A
1 0 0 1 1 0 1 1	B

C = A & B

Bit-Wise Operators

0 1 0 1 1 1 1 0	A
1 0 0 1 1 0 1 1	B
0	C = A & B

Bit-Wise Operators

0 1 0 1 1 1 1 0	A
1 0 0 1 1 0 1 1	B
1 0	C = A & B

Bit-Wise Operators

0 1 0 1 1 1 1 0	A
1 0 0 1 1 0 1 1	B
0 0 0 1 1 0 1 0	C = A & B

Bit-Wise Operators

Other Operators:

- OR: |
- XOR: ^

Bit Manipulation

Given a byte A, how do we set bit 2 (counting from 0) of A to 1?

Bit Manipulation

Given a byte A, how do we set bit 2 (counting from 0) of A to 1?

```
A = A | 4;
```

Bit Manipulation

Given a byte A, how do we set bit 2 (counting from 0) of A to 0?

Bit Manipulation

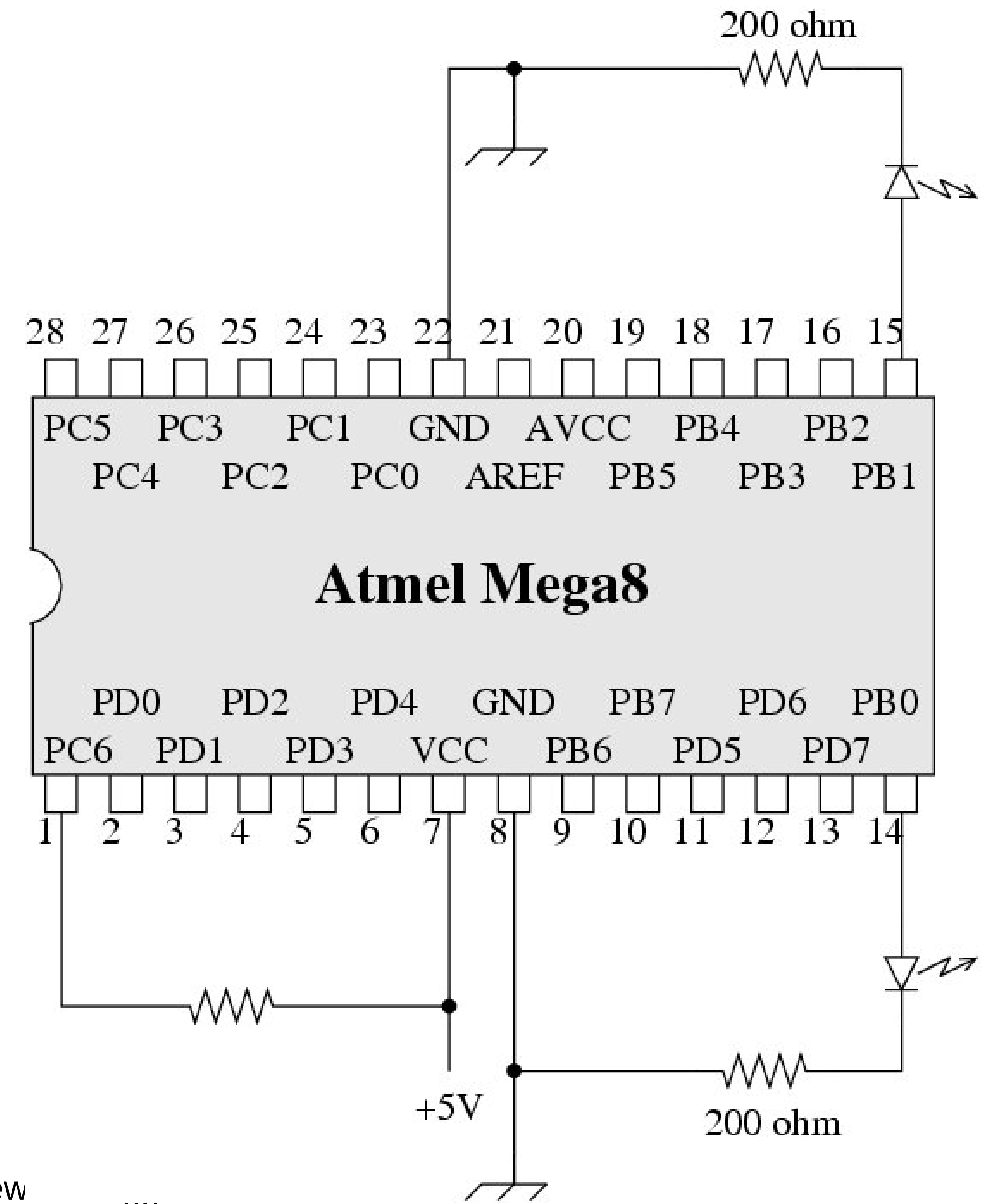
Given a byte A, how do we set bit 2 (counting from 0) of A to 1?

```
A = A & 0xFB;
```

A First Program

Flash the LEDs at a regular interval

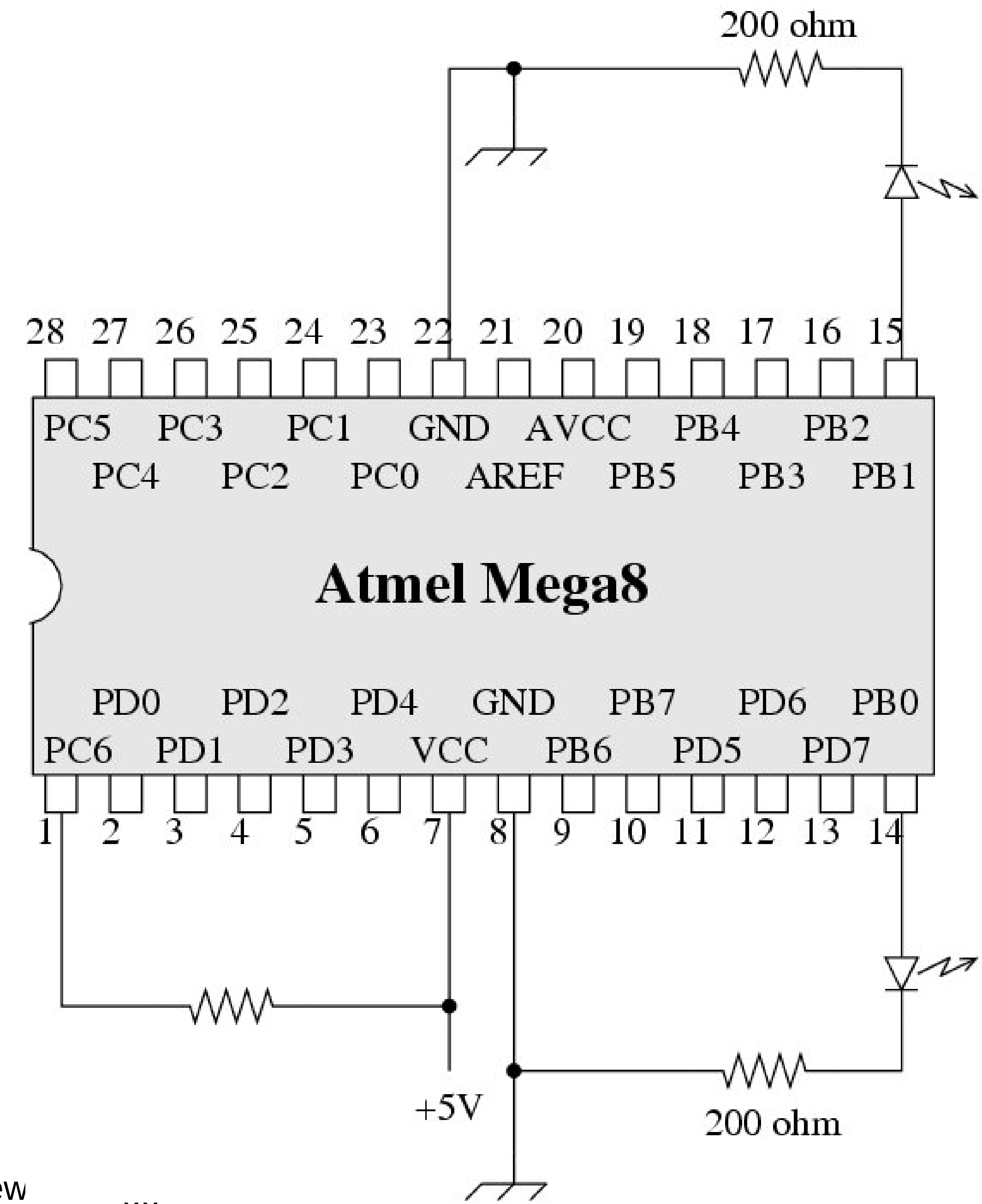
- How do we do this?



A First Program

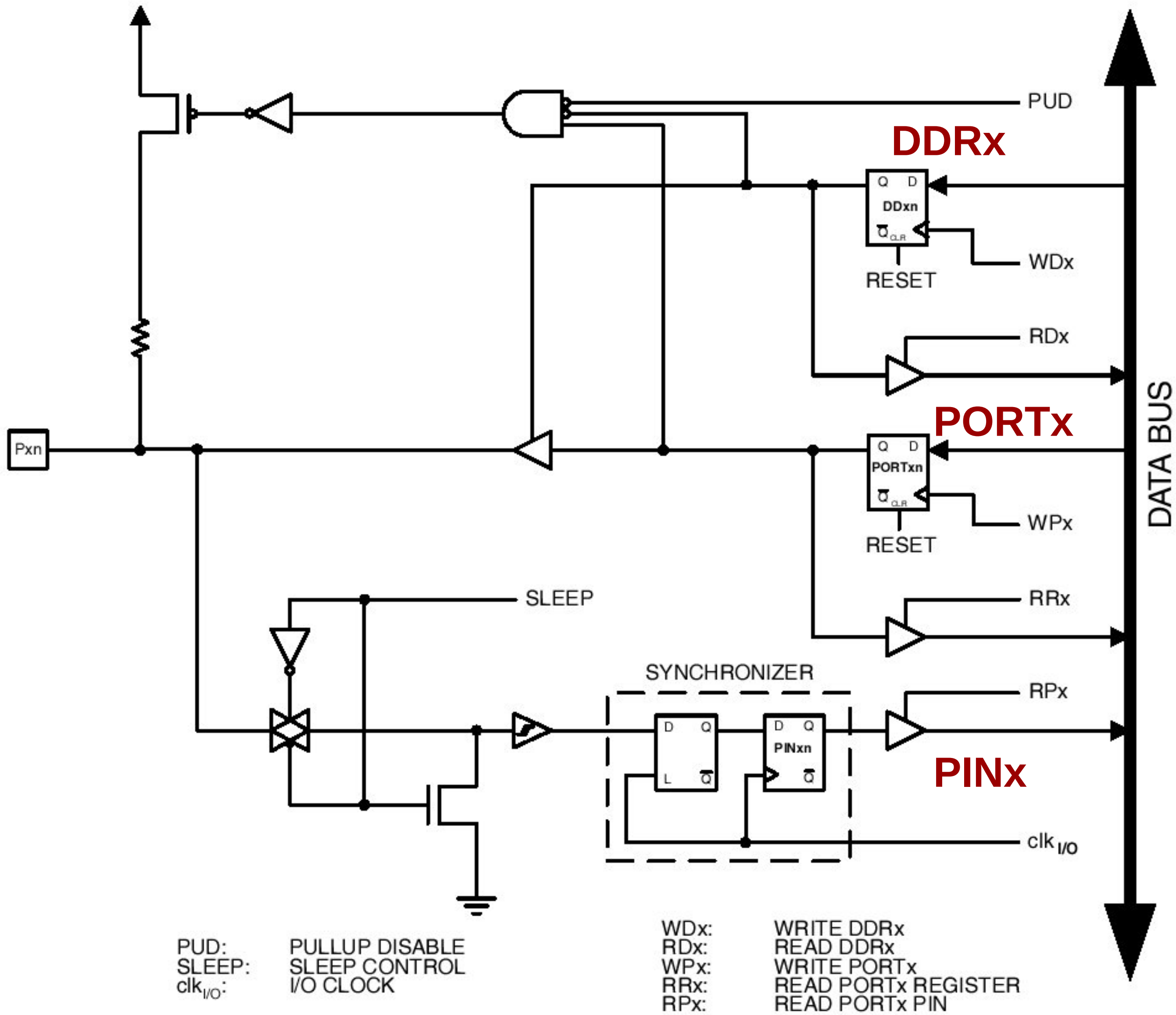
How do we flash the LED at a regular interval?

- We toggle the state of PB0



I/O Pin Implementation

Single bit of
PORT B



A First Program

```
main() {
   DDRB = 0x3;   // Set all port B pins as outputs

   while(1) {
        PORTB = PORTB ^ 0x1;   // XOR bit 0 with 1
        delay_ms(500);         // Pause for 500 msec
        }
}
```

A Second Program

```
main() {
   DDRB = 0xFF;   // Set all port B pins as outputs

   while(1) {
        PORTB = PORTB ^ 0x1;   // XOR bit 0 with 1
        delay_ms(500);         // Pause for 500 msec
        PORTB = PORTB ^ 0x2;   // XOR bit 1 with 1
        delay_ms(250);
        PORTB = PORTB ^ 0x2;   // XOR bit 1 with 1
        delay_ms(250);
   }
}
```

What does this program do?

A Second Program

```
main() {
   DDRB = 0xFF;   // Set all port B pins as outputs

   while(1) {
        PORTB = PORTB ^ 0x1;   // XOR bit 0 with 1
        delay_ms(500);         // Pause for 500 msec
        PORTB = PORTB ^ 0x2;   // XOR bit 1 with 1
        delay_ms(250);
        PORTB = PORTB ^ 0x2;   // XOR bit 1 with 1
        delay_ms(250);
   }
}
```

Flashes LED on PB1 at 1 Hz

on PB0: 0.5 Hz

More Bit Masking

- Suppose we have a 3-bit number (so values 0 … 7)
- Suppose we want to set the state of B3, B4, and B5 with this number (B3 is the least significant bit)

- How do we express this in code?

Bit Masking

```
main() {
   DDRB = 0xF8;   // Set pins B3, B4, B5, B6, B7 as outputs

       :
       :

  unsigned short val;  // A short is 8-bits wide

   val = command_to_robot;

   PORTB = (PORTB & 0xC7)        // Set the current B3-B5 to 0s
       | ((val & 0x7))<<3);      // OR with new values (shifted
                                 //  to fit within B3-B5
}
```

Bit Masking

```
main() {
  DDRB = 0xF8;    // Set pins B3, B4, B5, B6, B7 as outputs

      :
      :

  unsigned short val;  // A short is 8-bits wide

  val = command_to_robot;

  PORTB = (PORTB & 0xC7)          // Set the current B3-B5 to 0s
      | ((val & 0x7))<<3);        // OR with new values (shifted
                                  //  to fit within B3-B5)
}
```

B3-B7 are outputs; all others are still inputs (could be different depending on how other pins are used)

Bit Masking

```
main() {
   DDRB = 0xF8;    // Set pins B3, B4, B5, B6, B7 as outputs

       :
       :

  unsigned short val;  // A short is 8-bits wide

   val = foobar;

   PORTB = (PORTB & 0xC7)           // Set the current B3-B5 to 0s
       | ((val & 0x7))<<3);         // OR with new values (shifted
                                    //  to fit within B3-B5
}
```

"Mask out" the current values of pins B3-B5 (leave everything else intact)

Bit Masking

```
main() {
   DDRB = 0xF8;   // Set pins B3, B4, B5, B6, B7 as outputs

       :
       :

  unsigned short val;  // A short is 8-bits wide

   val = foobar;

   PORTB = (PORTB & 0xC7)           // Set the current B3-B5 to 0s
         | ((val & 0x7))<<3);       // OR with new values (shifted
                                    //  to fit within B3-B5
}
```

Substitute an arbitrary value into these bits

Bit Masking

```
main() {
   DDRB = 0xF8;   // Set pins B3, B4, B5, B6, B7 as outputs

        :
        :

  unsigned short val;  // A short is 8-bits wide

   val = foobar;

   PORTB = (PORTB & 0xC7)          // Set the current B3-B5 to 0s
        | ((val & 0x7))<<3);       // OR with new values (shifted
                                   //  to fit within B3-B5
}
```

And use the result to change the output state of port B

Reading the Digital State of Pins

Given: we want to read the state of PB6 and PB7 and obtain a value of 0 … 3

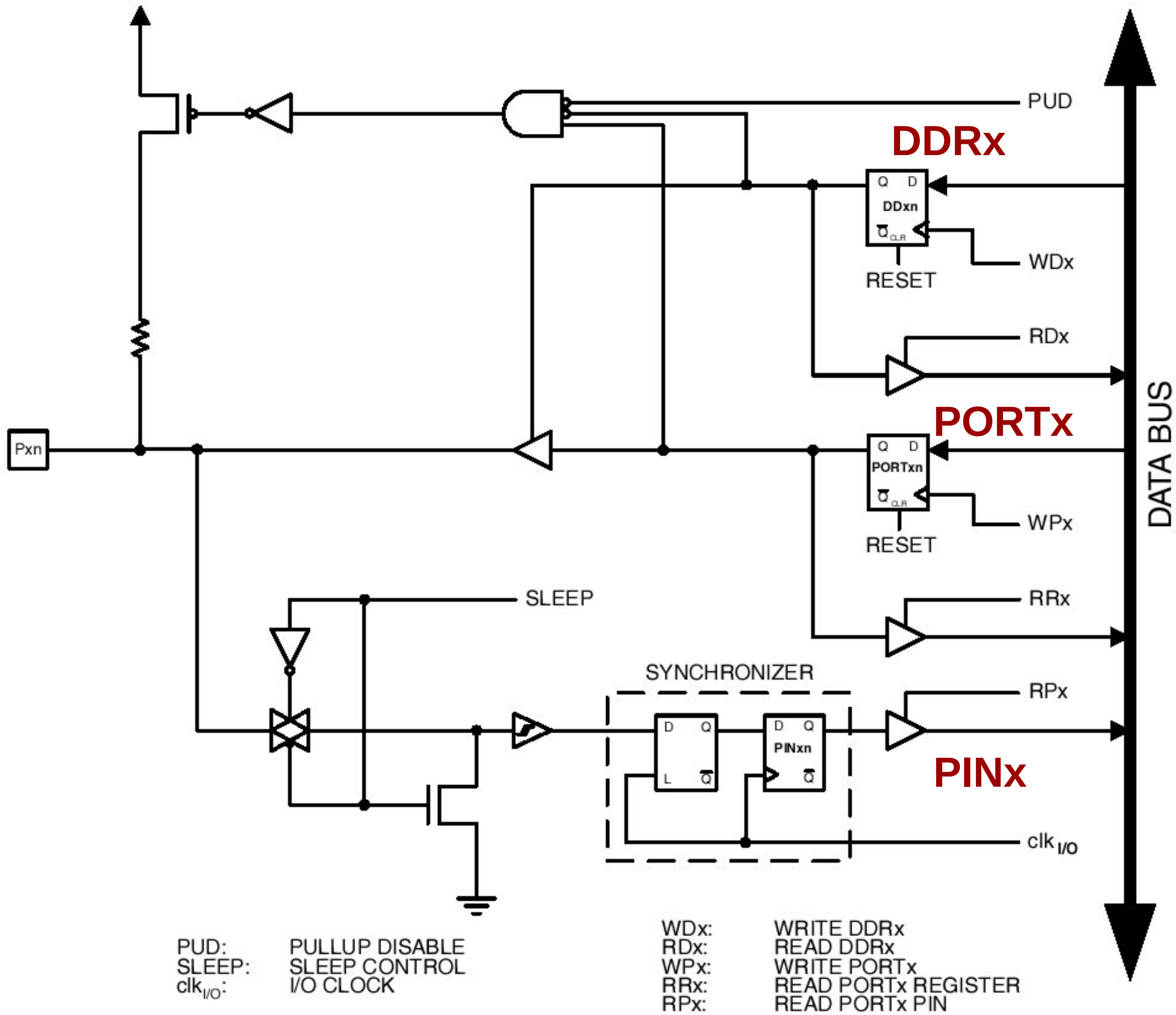
- How do we configure the port?
- How do we read the pins?
- How do we translate their values into an integer of 0 .. 3?

Reading the Digital State of Pins

```
main() {
   DDRB = 0x38;    // Set pins B3, B4, B5 as outputs
                   //   All others are inputs (suppose we care
                   //   about bits B6 and B7 only (so a 2-bit
                   //    number)
       :
       :

  unsigned short val, outval;  // A short is 8-bits wide

   val = PINB;

   outval = (val & 0xC0) >> 6;
}
```

I/O Pin Implementation

Single bit of PORT B



Reading the Digital State of Pins

```
main() {
  DDRB = 0x38;    // Set pins B3, B4, B5 as outputs
                  //   All others are inputs (suppose we care
                  //   about bits B6 and B7 only (so a 2-bit
                  //    number)
       :
       :

  unsigned short val, outval;  // A short is 8-bits wide

  val = PINB;

  outval = (val & 0xC0) >> 6;
}
```

B6 and B7 are configured as inputs

Reading the Digital State of Pins

```
main() {
   DDRB = 0x38;    // Set pins B3, B4, B5 as outputs
                   //   All others are inputs (suppose we care
                   //   about bits B6 and B7 only (so a 2-bit
                  //     number)
         :
         :

  unsigned short val, outval;  // A short is 8-bits wide

   val = PINB;

   outval = (val & 0xC0) >> 6;
}
```

Read the value from the port

Reading the Digital State of Pins

```
main() {
   DDRB = 0x38;    // Set pins B3, B4, B5 as outputs
                   //   All others are inputs (suppose we care
                   //   about bits B6 and B7 only (so a 2-bit
                   //    number)
        :
        :

  unsigned short val, outval;  // A short is 8-bits wide

   val = PINB;

   outval = (val & 0xC0) >> 6;
}
```

"Mask out" all bits except B6 and B7

Reading the Digital State of Pins

```
main() {
   DDRB = 0x38;    // Set pins B3, B4, B5 as outputs
                   //   All others are inputs (suppose we care
                   //   about bits B6 and B7 only (so a 2-bit
                   //    number)
       :
       :

  unsigned short val, outval;  // A short is 8-bits wide

   val = PINB;

   outval = (val & 0xC0) >> 6;
}
```

Right shift the result by 6 bits – so the value of B6 and B7 are now in bits 0 and 1 of "outval"

Port-Related Registers

The set of C-accessible register for controlling digital I/O:

	Directional control	Writing	Reading
Port B	DDRB	PORTB	PINB
Port C	DDRC	PORTC	PINC
Port D	DDRD	PORTD	PIND

A Note About the C/Atmel Book

The book uses C syntax that looks like this:

```
PORTA.0 = 0;      // Set bit 0 to 0
```

This syntax is not available with our C compiler. Instead, you will need to use:

```
PORTA &= 0xFE;
```

or

```
PORTA &= ~1;
```

or

```
PORTA = PORTA & ~1;
```

Putting It All Together

- Program development:
  - On your own laptop
  - We will use a C "crosscompiler" (avr-gcc and other tools) to generate code on your laptop for the mega8 processor
- Program download:
  - We will use "in circuit programming": you will be able to program the chip without removing it from your circuit

Physical Interface for Programming

AVR ISP

Physical Interface for Programming

AVR ISP

USB connection to your laptop

Physical Interface for Programming

AVR ISP

Header connection will connect to your circuit (through an adapter)

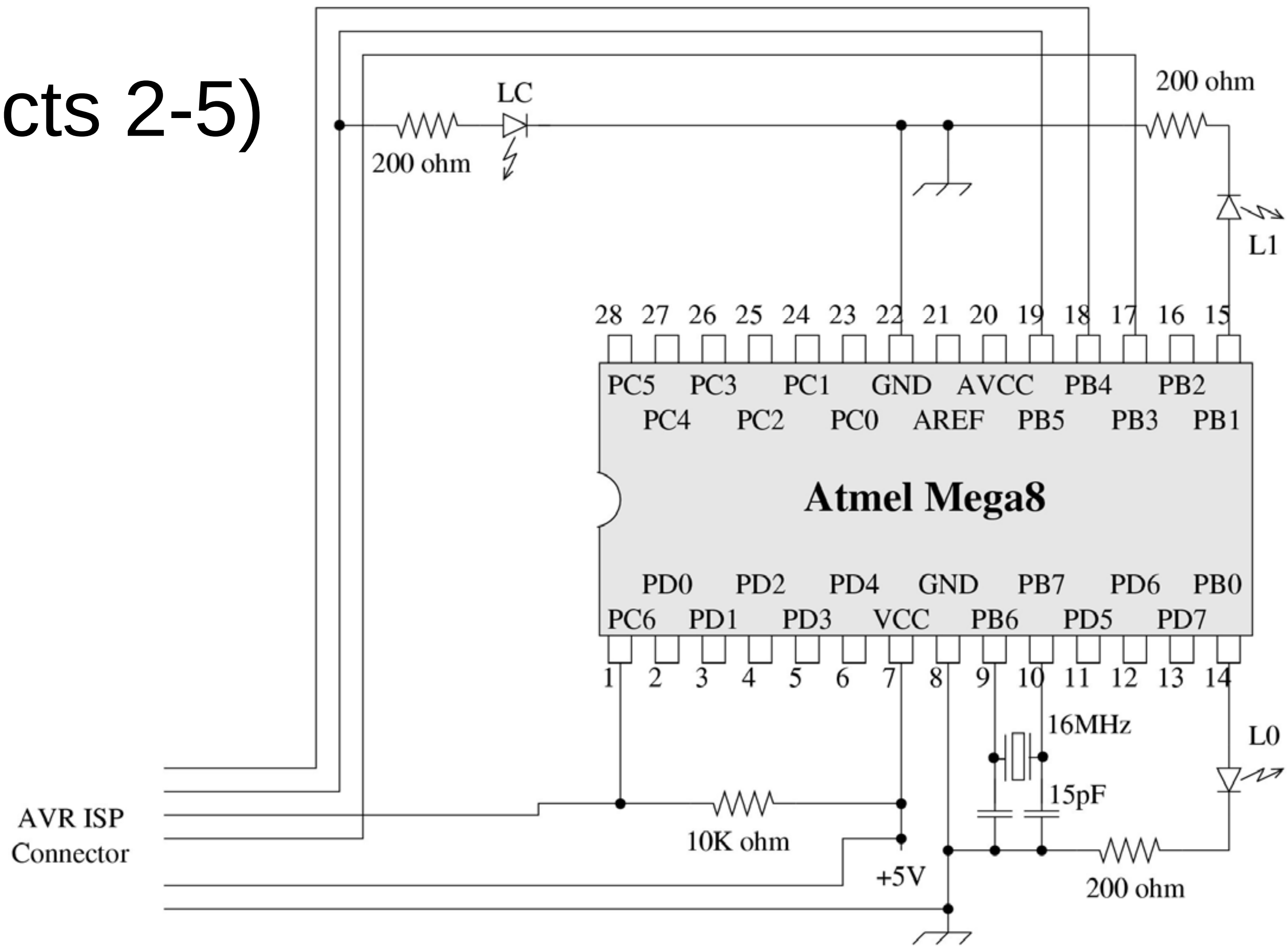
Be careful when you plug your circuit in (check before powering)

AVR ISPs are Cranky

- When things are plugged in and powered, you should see two green LEDs on the ISP (on most units)
- One red: usually means that your circuit is not powered
- Orange: the programmer is confused
  - Could be due to your circuit not being powered at 5V
  - Could be due to other problems
  - Check power and reboot the ISP

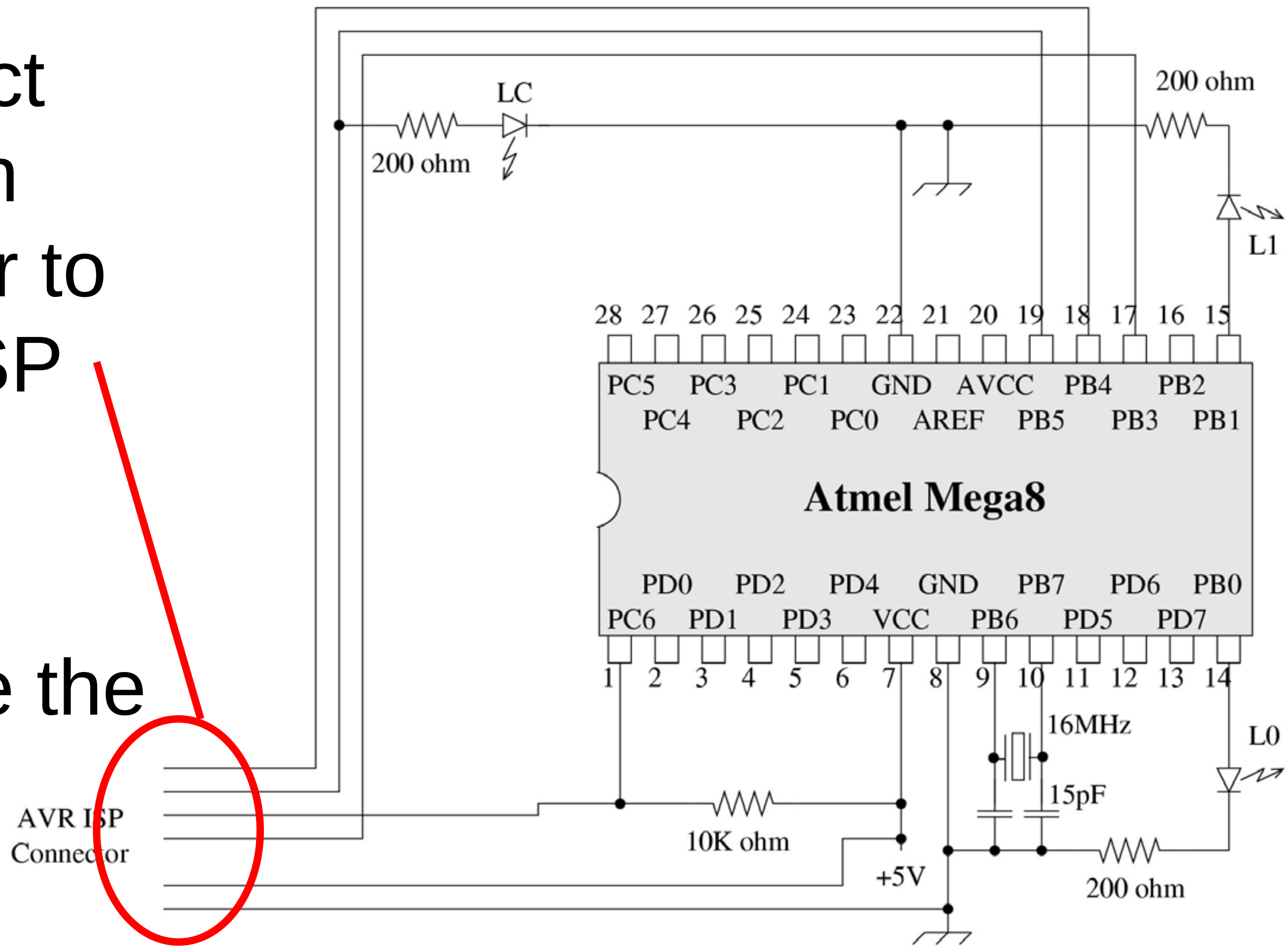
A More Complicated Circuit

(for projects 2-5)



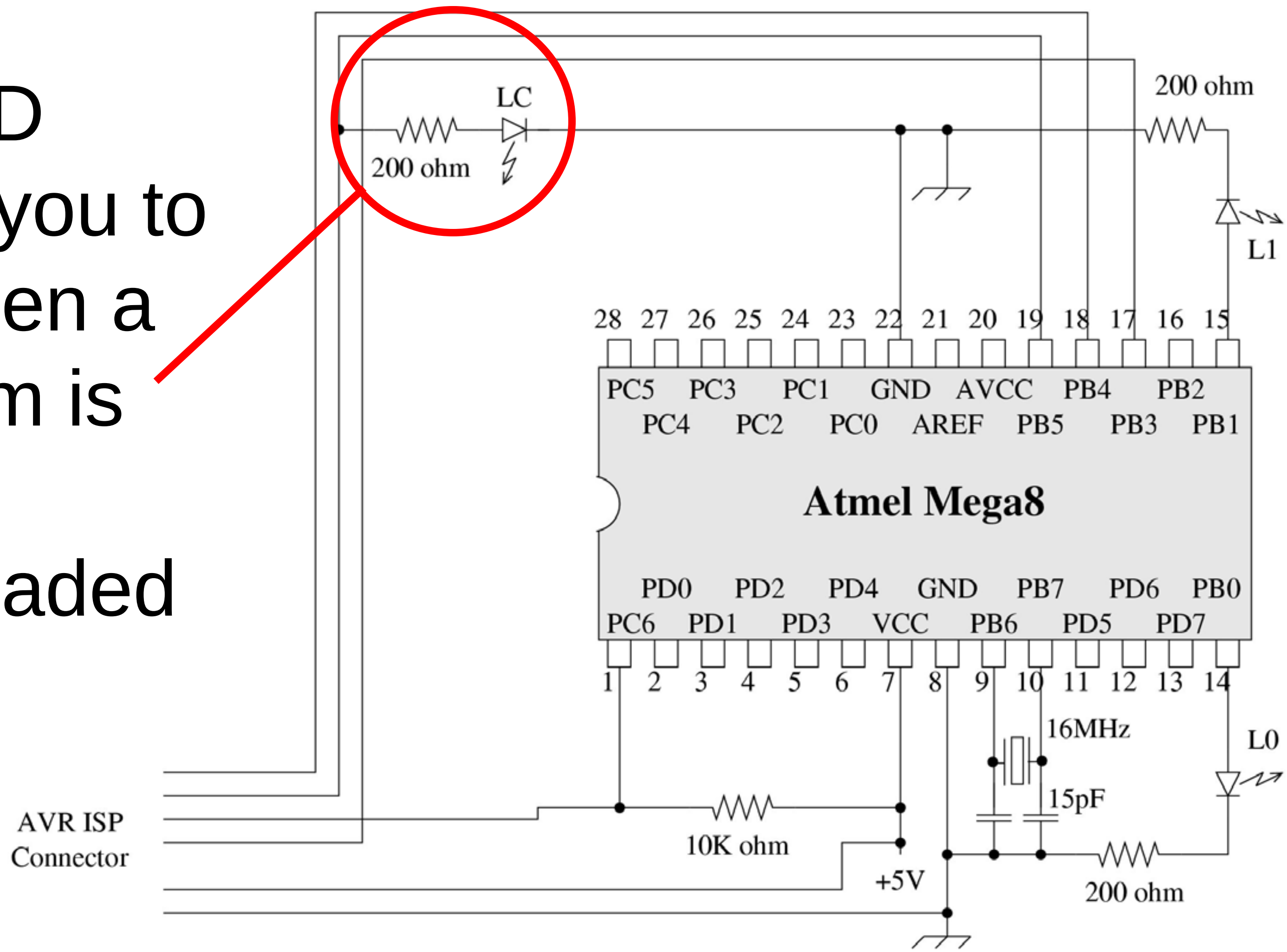
A More Complicated Circuit

- Connect through adapter to AVR ISP
- Do not reverse the pins!



A More Complicated Circuit

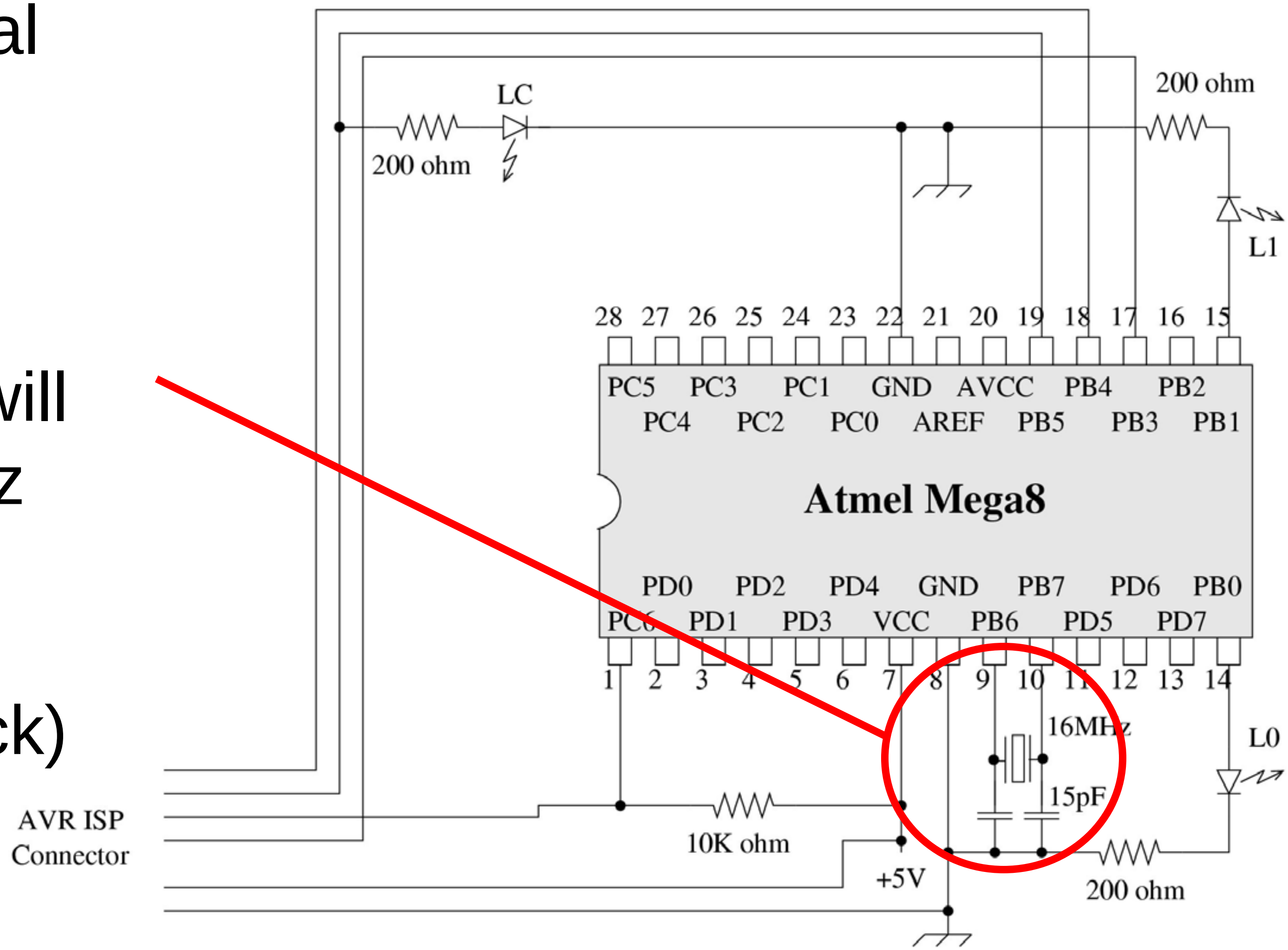
- Extra LED allows you to see when a program is being downloaded



A More Complicated Circuit

16 MHz crystal

- Optional!
- Without it, your processor will run at 1MHz (in general, we will use 16MHz clock)



Compiling and Downloading Code

- We will work through the details next Thursday. Before then:
  - See the Atmel HowTo (pointer from the schedule page)
  - Windoze: Install AVR Studio and WinAVR
  - OS X: Install OSX-AVR
  - Linux: Install binutils, avr-gcc, avr-libc, and avrdude
    - This works well now

Compiling and Downloading Code

- Once the chip is programmed, the AVR ISP will automatically reset the processor; starting your program

Hints

- Use LEDs to show status information (e.g., to indicate what part of your code is being executed)
- Have one LED blink in some unique way at the beginning of your program
- Go slow:
  - Implement and test incrementally
  - Insert plenty of pauses into your code (e.g., with delay_ms())