Last Time

Microprocessors Basics

- Busses
- Memory behavior
- Arithmetic Logical Units (ALUs)
- Fetch-Execute cycle
- Registers: general-purpose versus special-purpose

Today

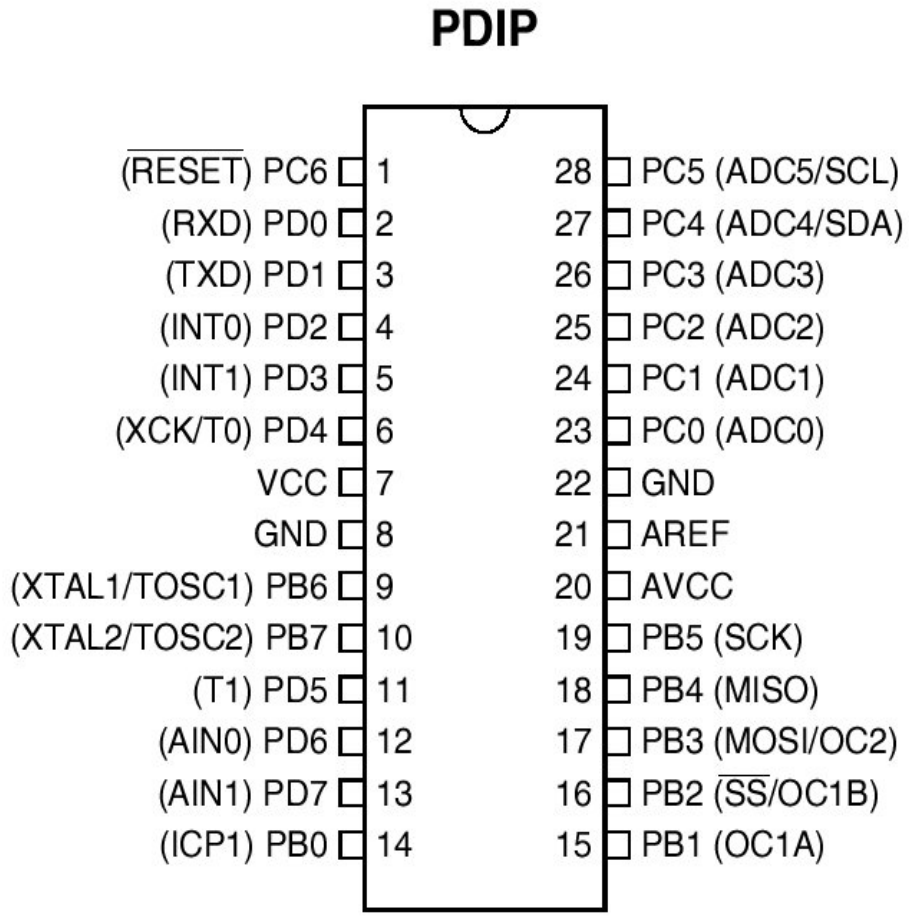
- Bit Masking
- Atmel Mega8 practicalities
  - Circuit design
  - Coding
  - Programming

Administrivia

- Lab 1 Due today @5:00
  - 2 groups left to demonstrate
  - Group report
  - Personal reports
- Pointers to circuit drawing programs are on D2L
  - Please add your own pointers

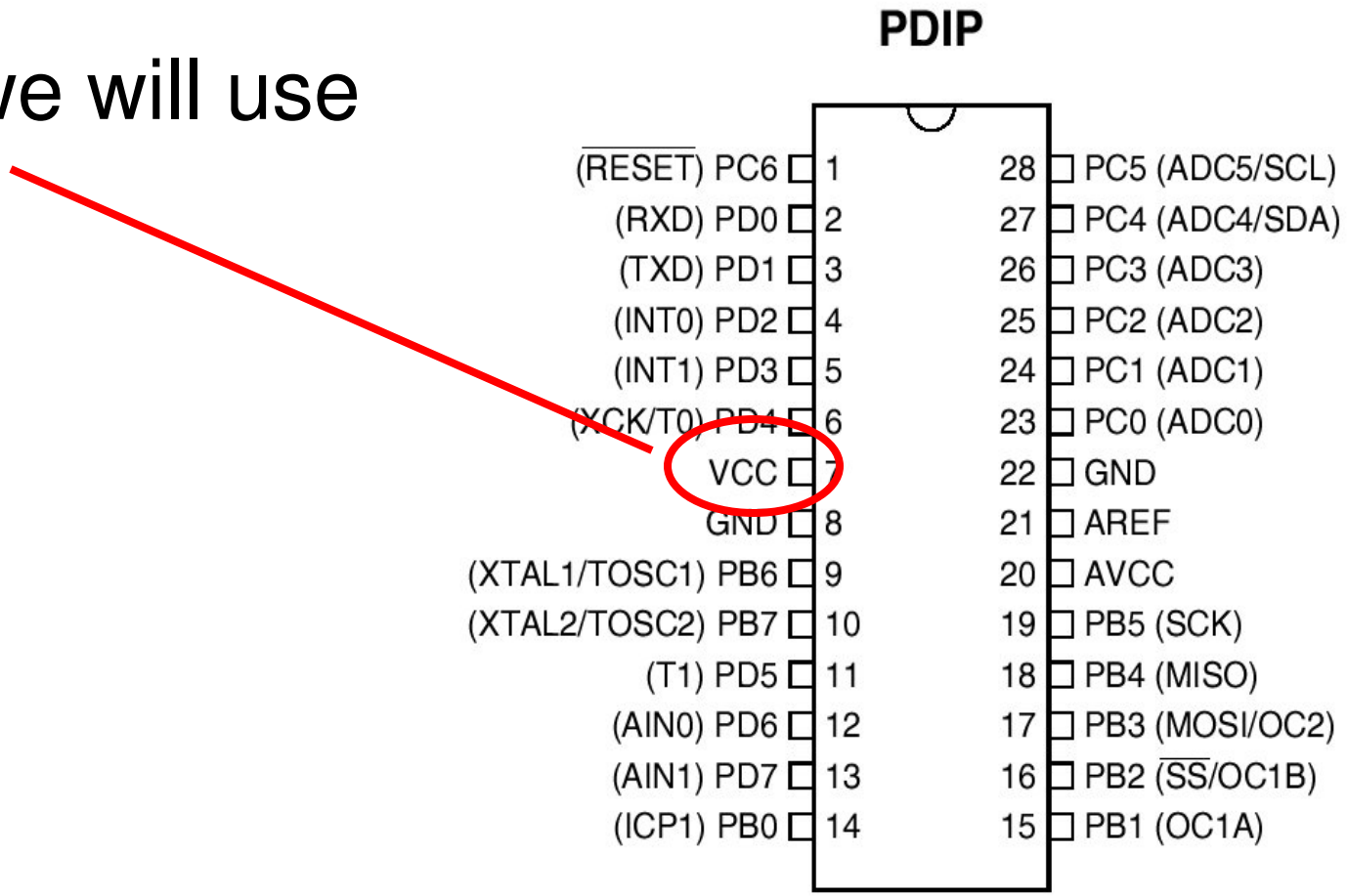
Atmel Mega8 Basics

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



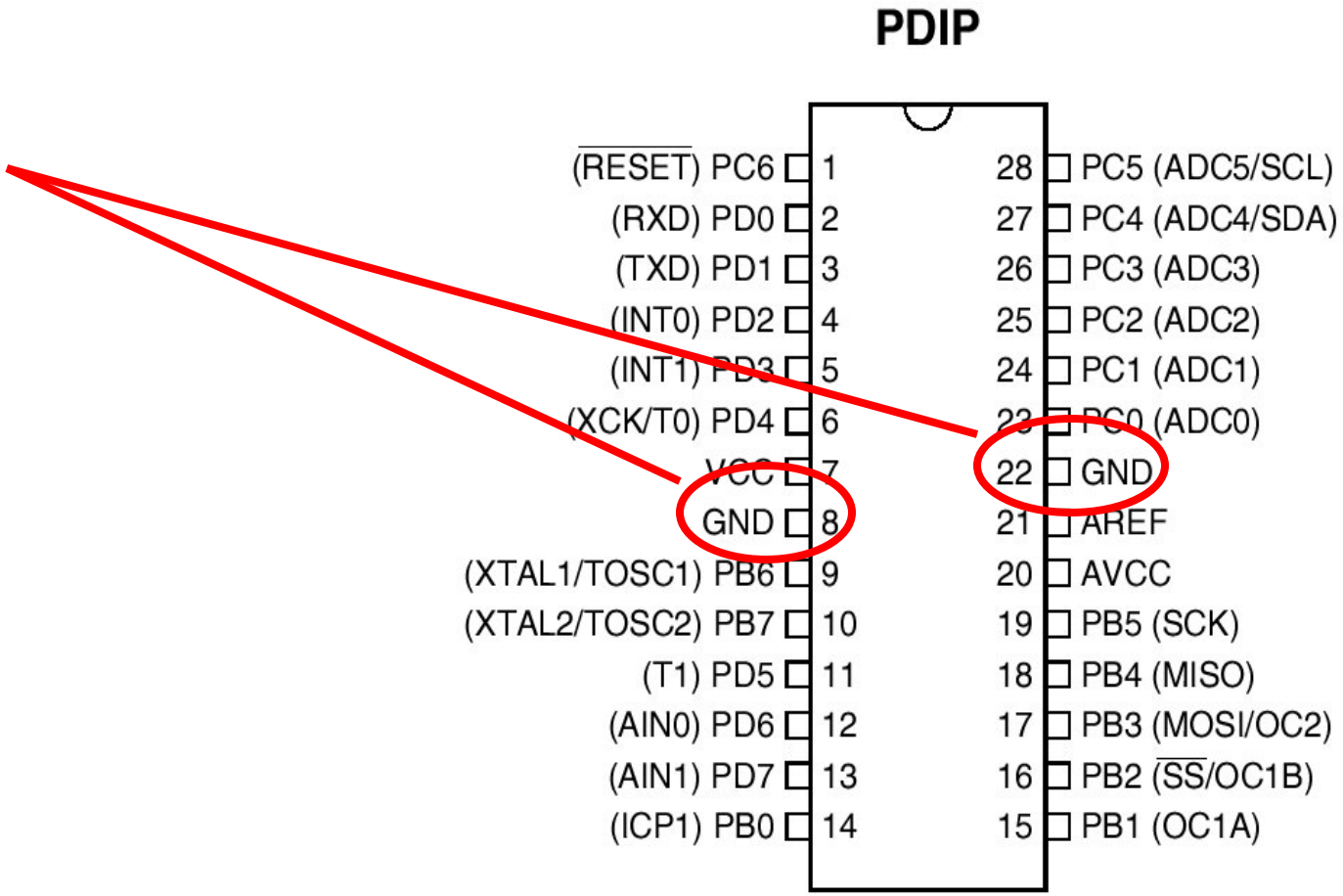
Atmel Mega8 Basics

Power (we will use +5V)



Atmel Mega8 Basics

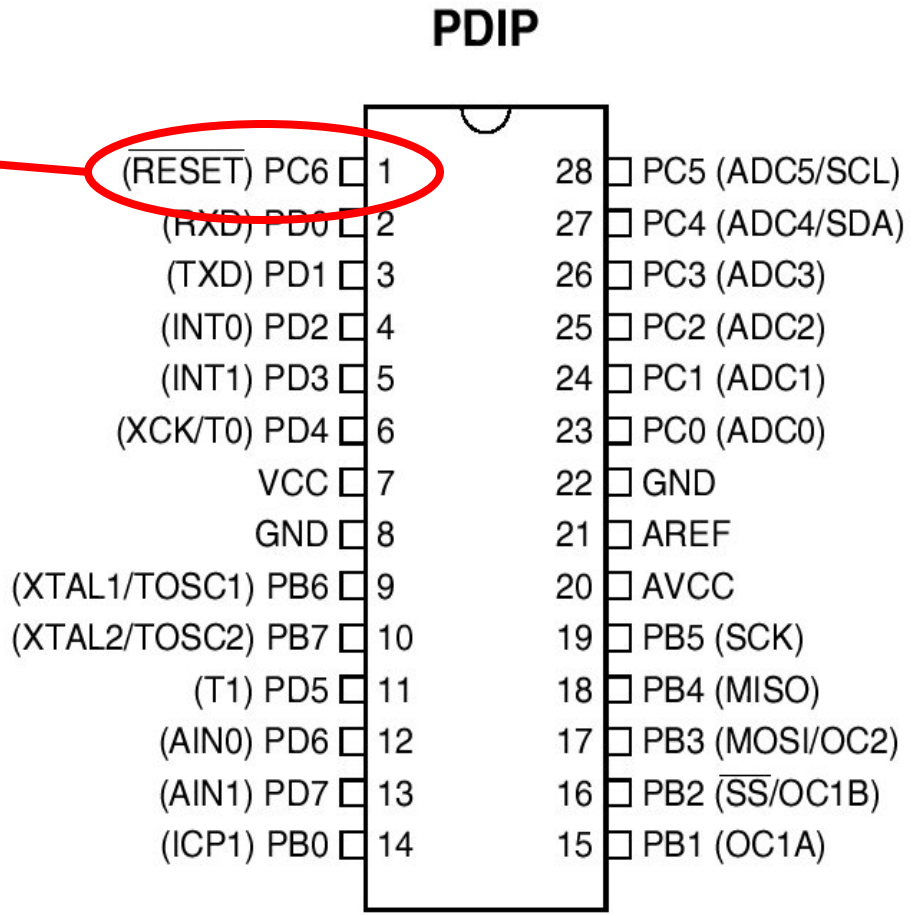
Ground



Atmel Mega8 Basics

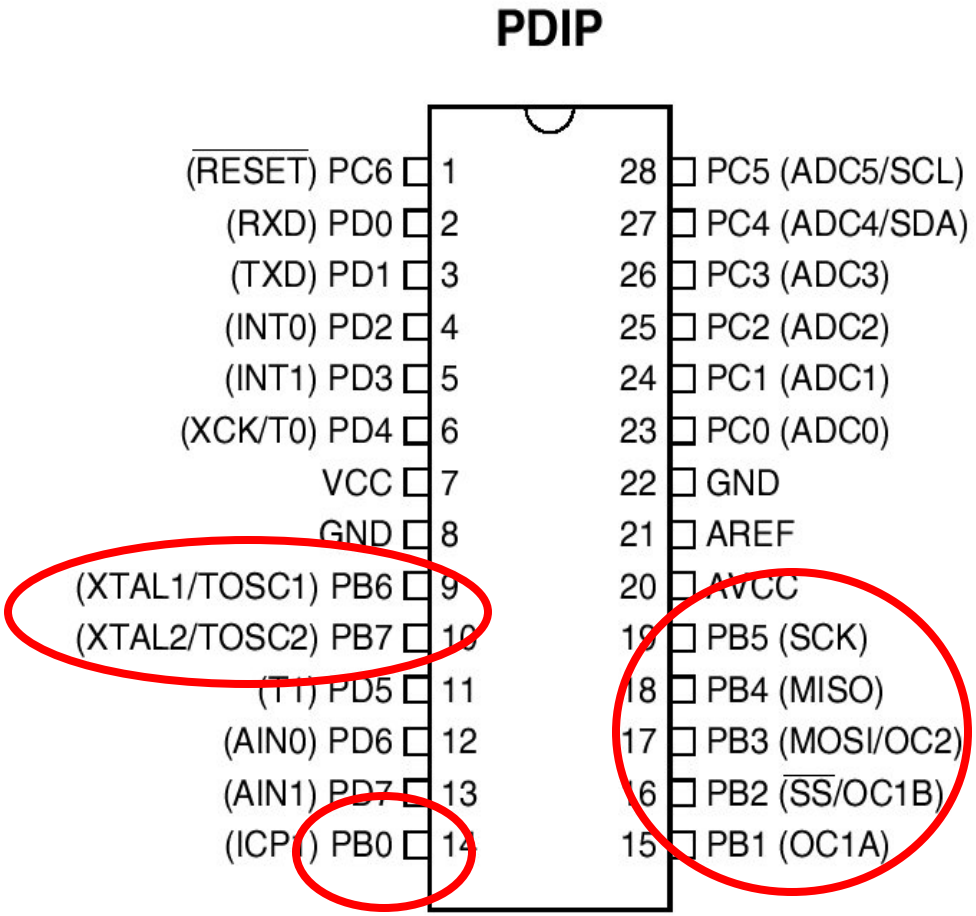
Reset

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



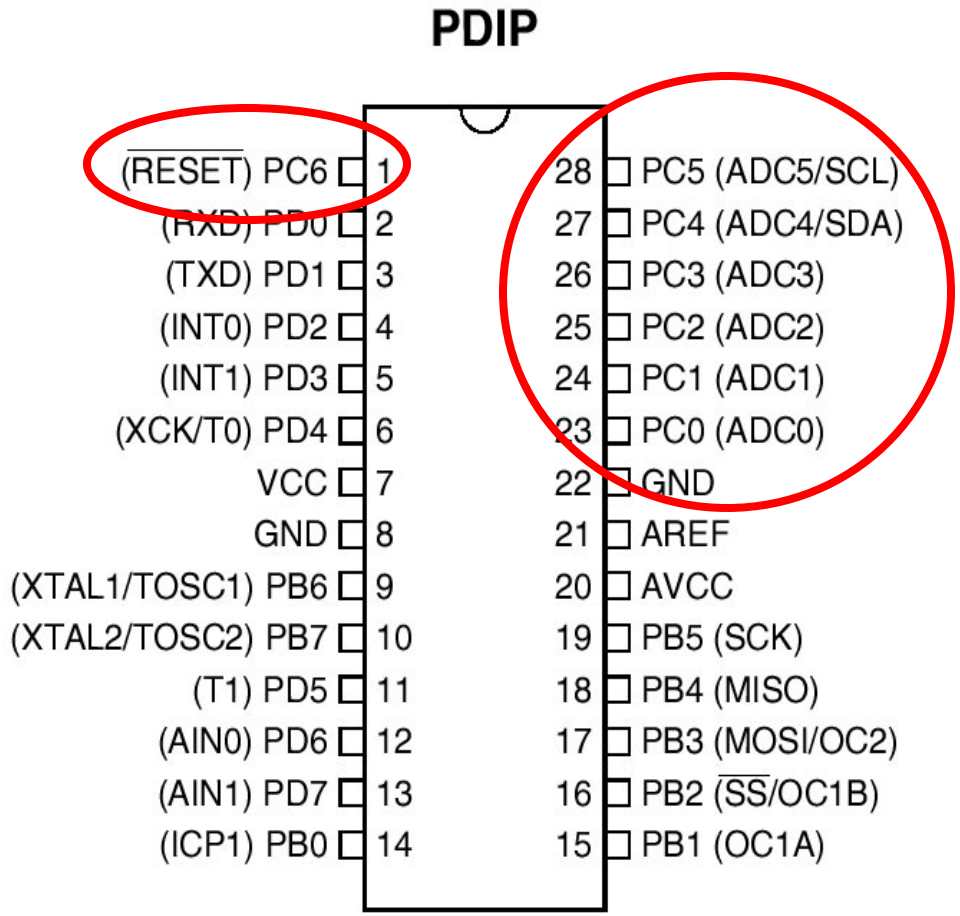
Atmel Mega8 Basics

PORT B



Atmel Mega8 Basics

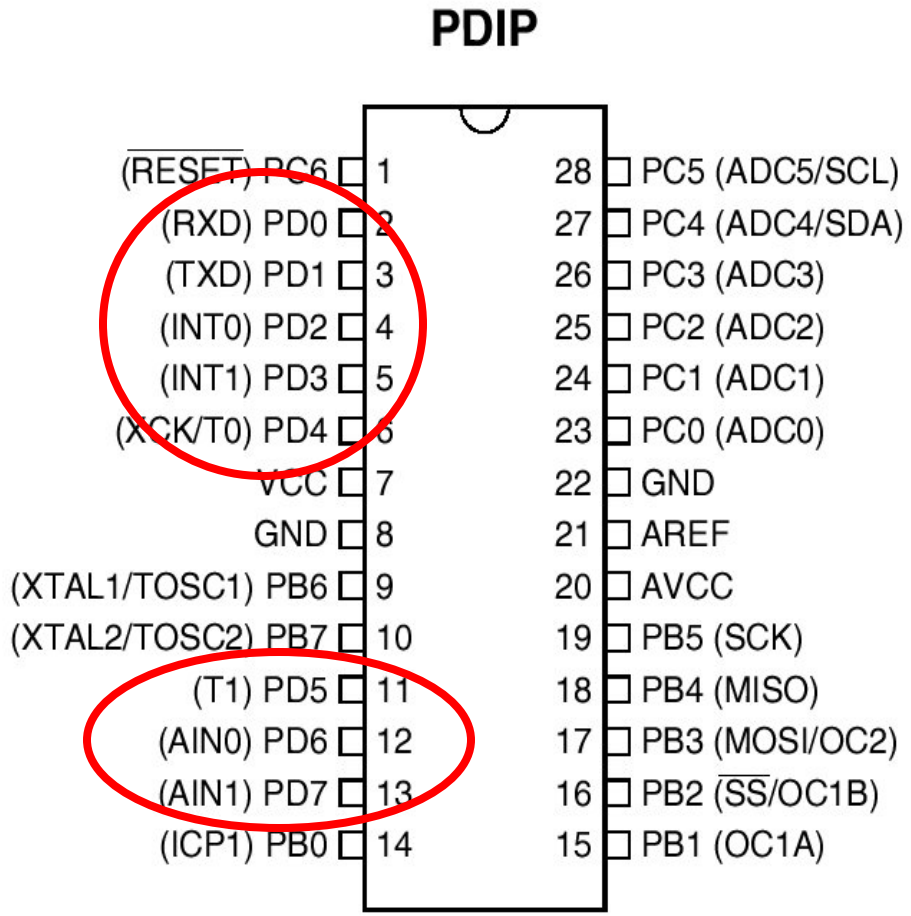
PORT C



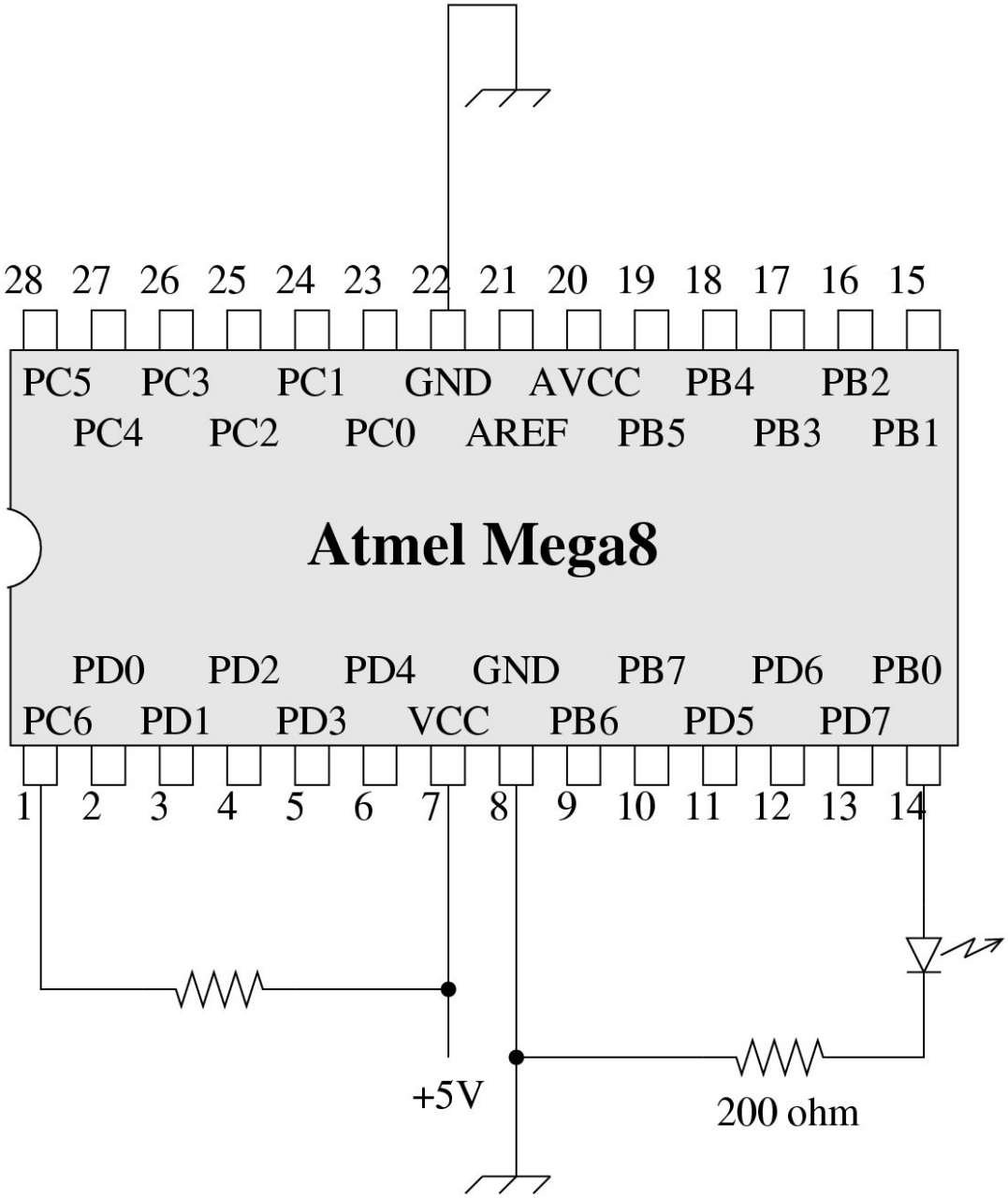
Atmel Mega8 Basics

PORT D

(all 8 bits are available)

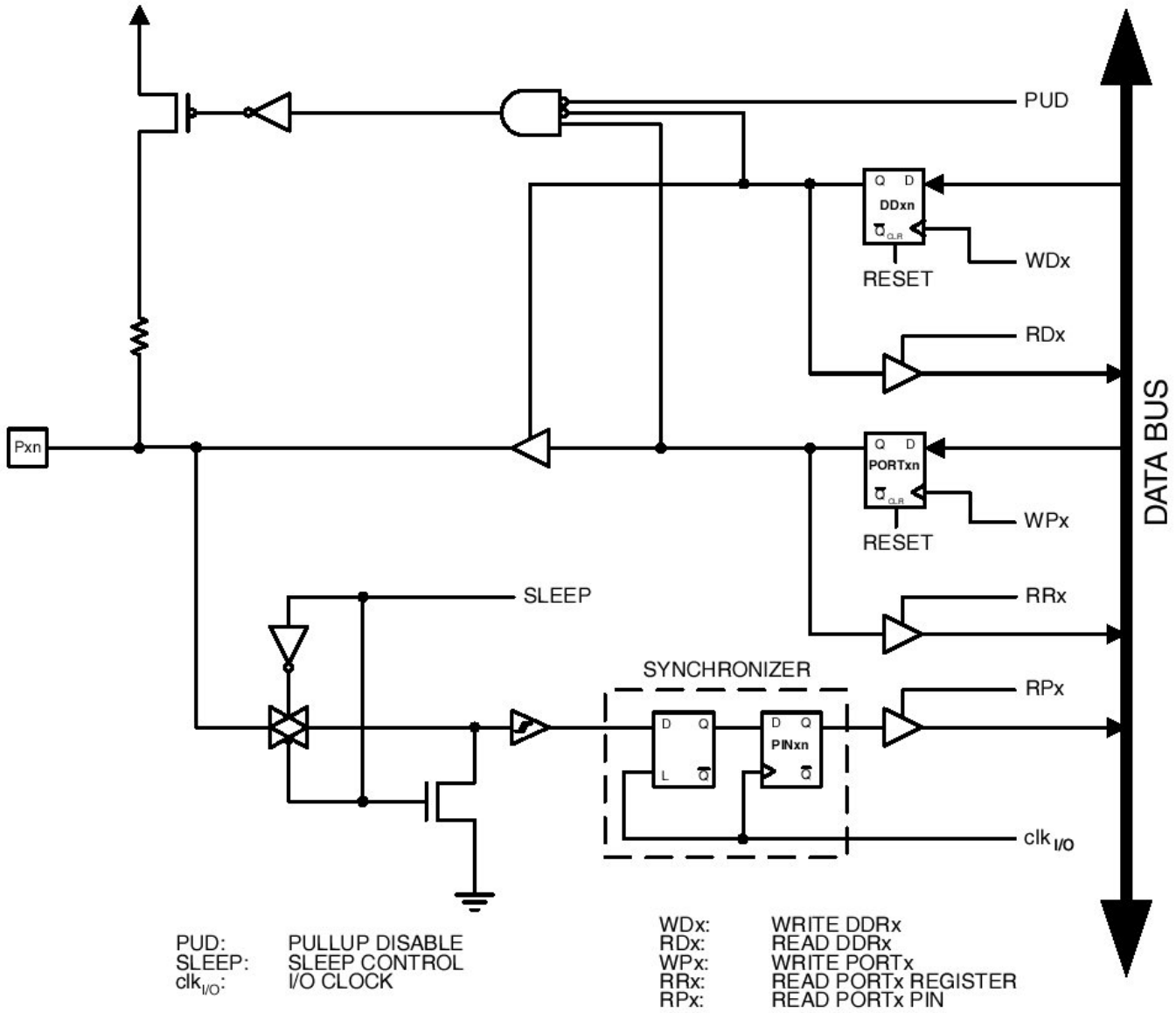


A First Circuit



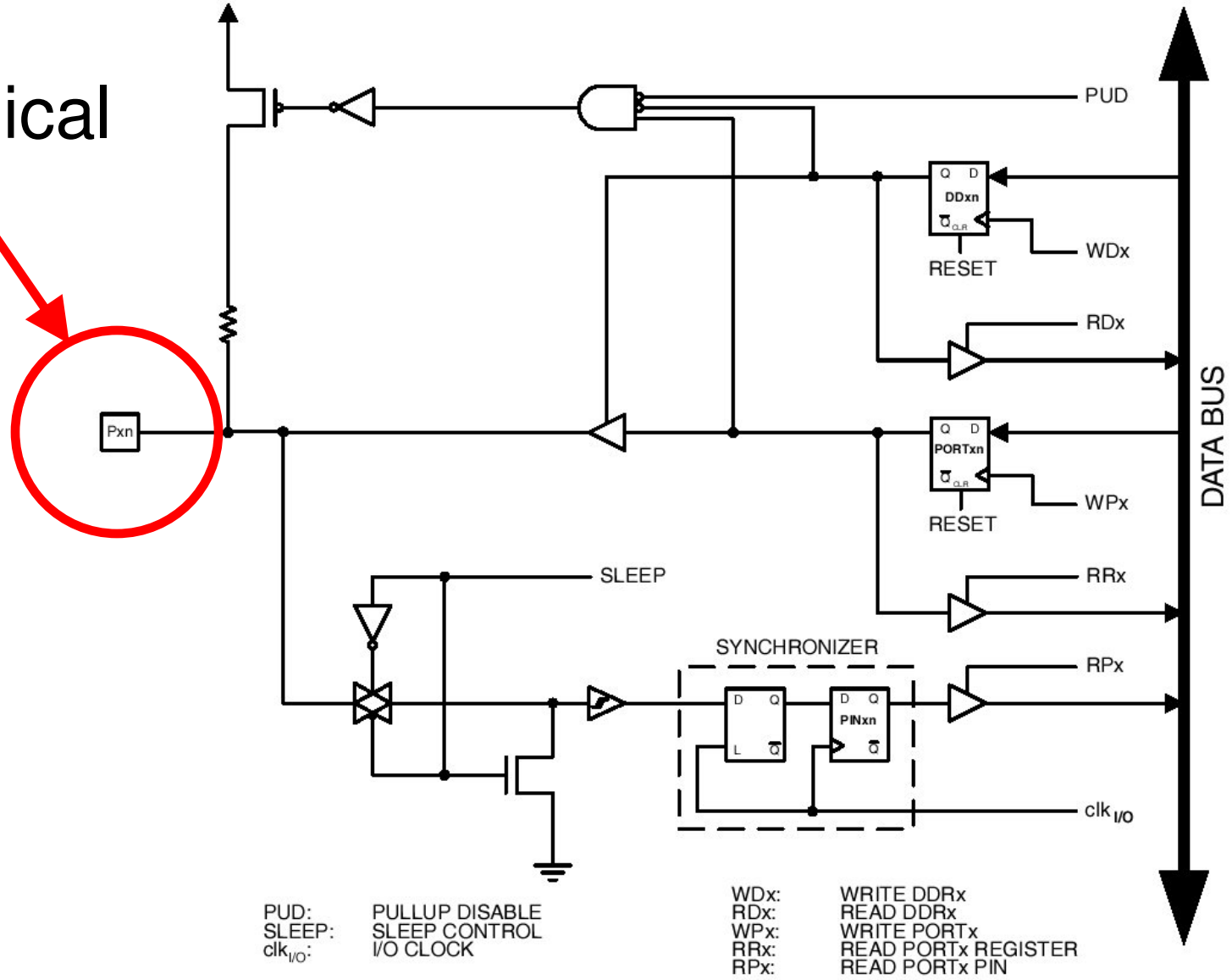
I/O Pin Implementation

Single bit of PORT B



I/O Pin Implementation

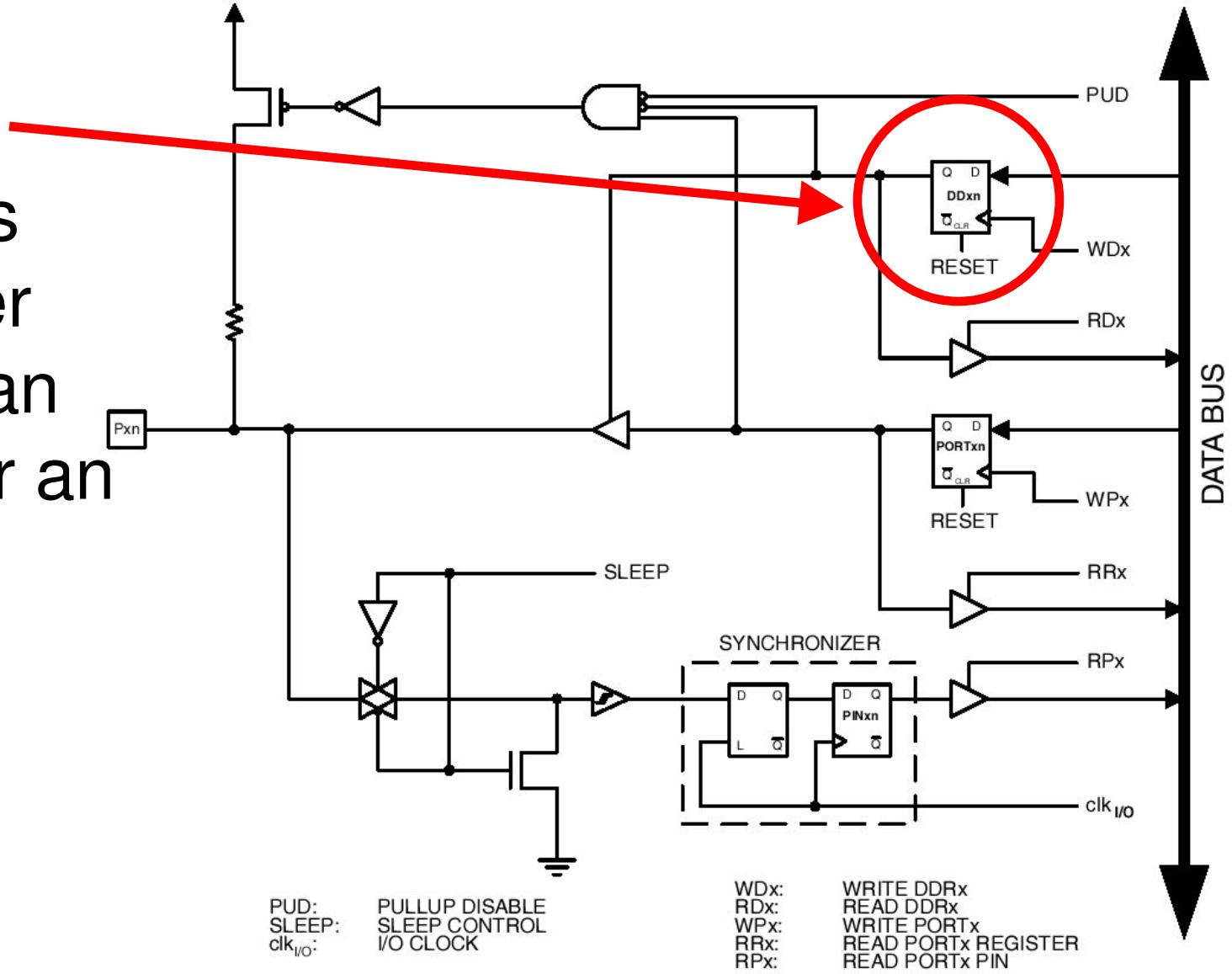
The physical pin



I/O Pin Implementation

DDRB

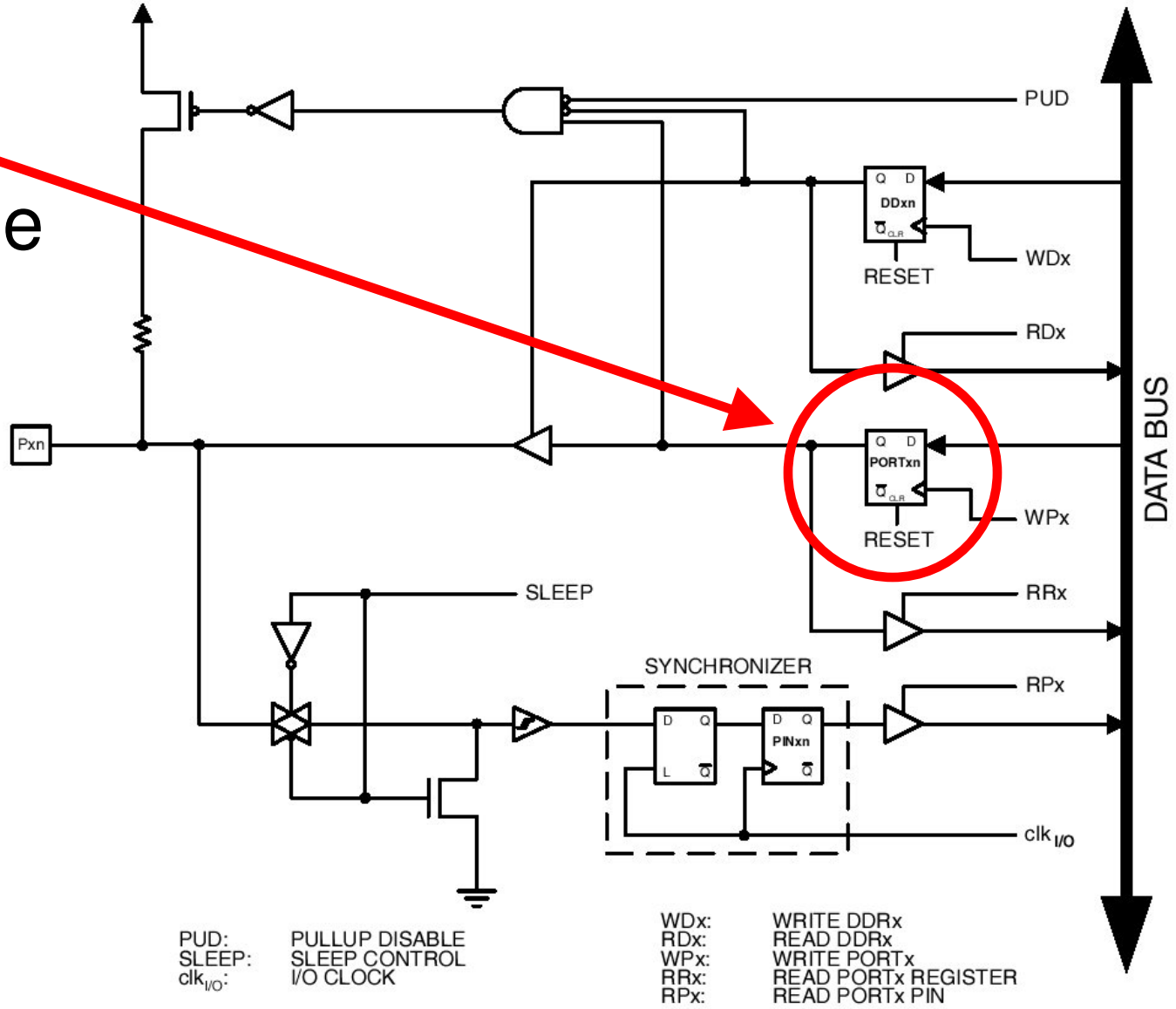
- Defines whether this is an input or an output



I/O Pin Implementation

PORTB

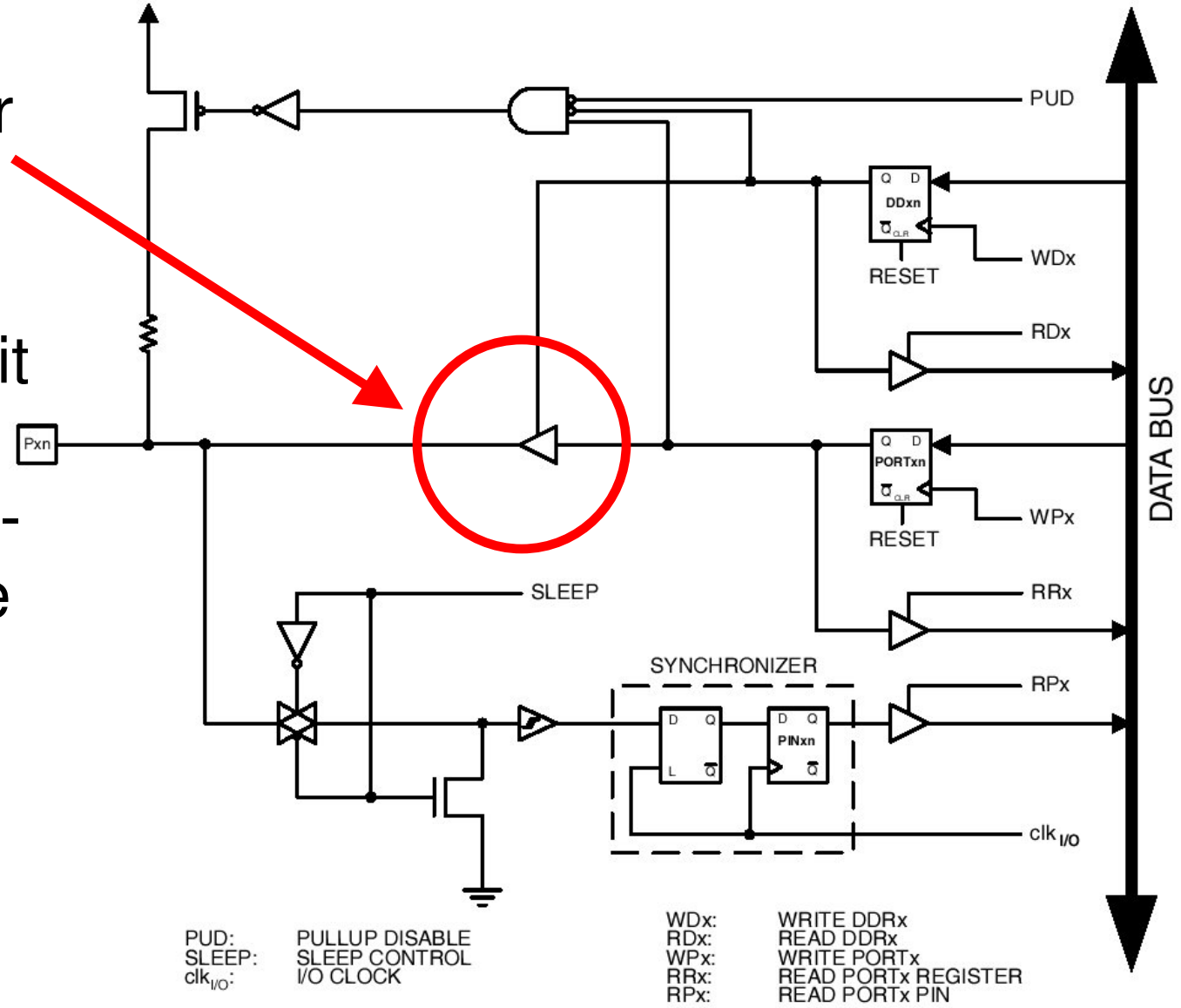
- Defines the value that is written out to the pin (if it is an output)



I/O Pin Implementation

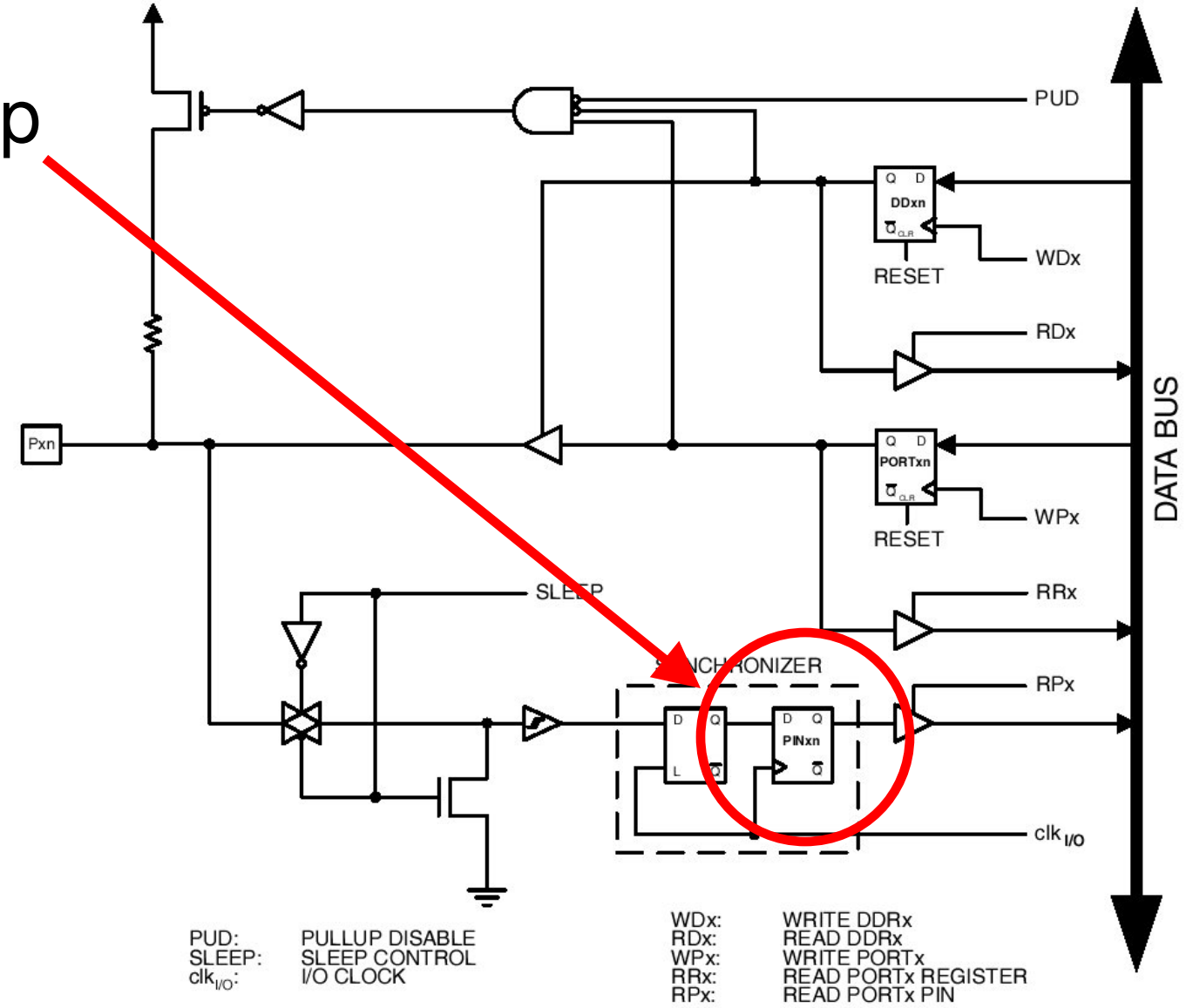
Tristate buffer

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



I/O Pin Implementation

Input flip-flop



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

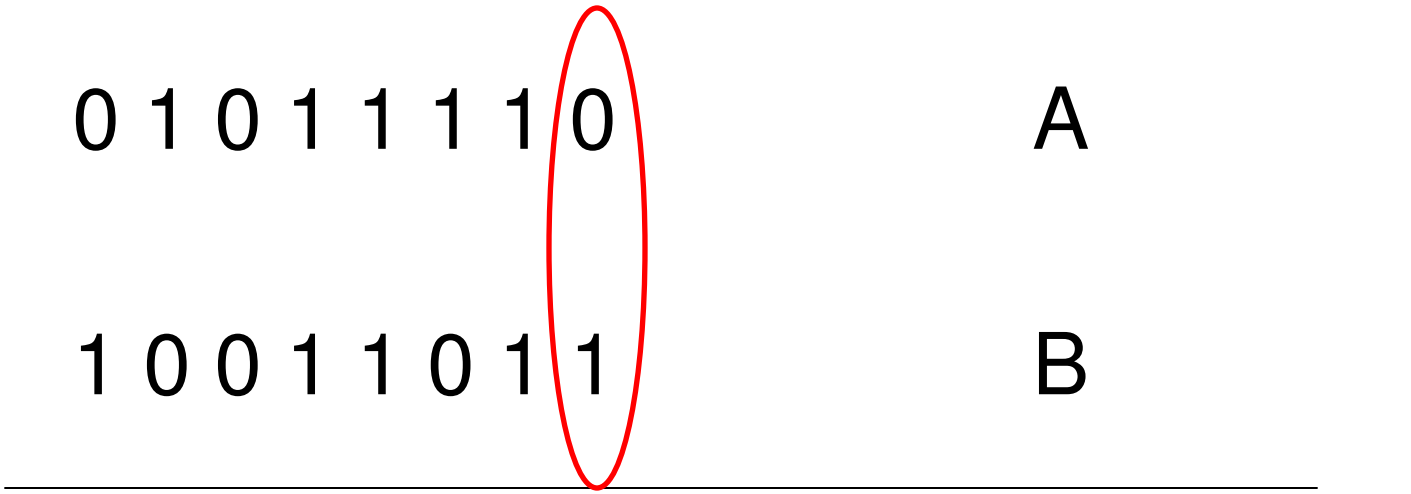
If A and B are bytes, what does this code mean?

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

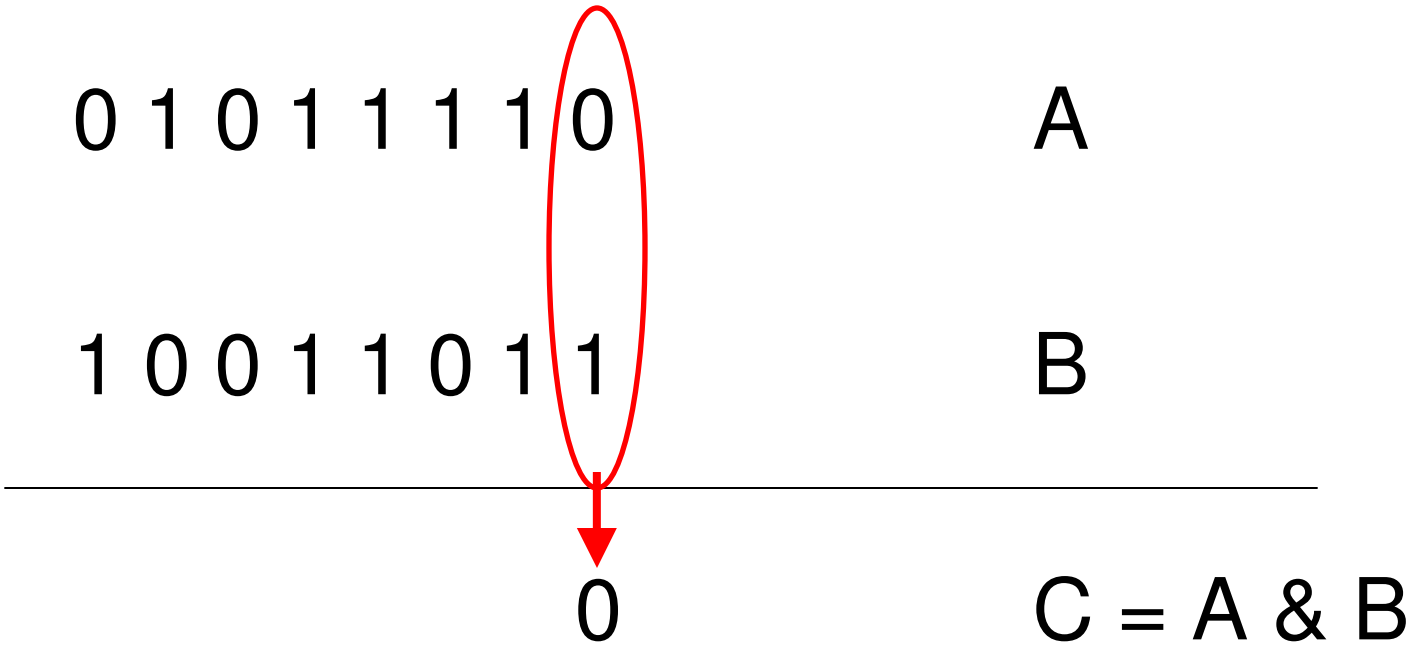
Bit-Wise Operators

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

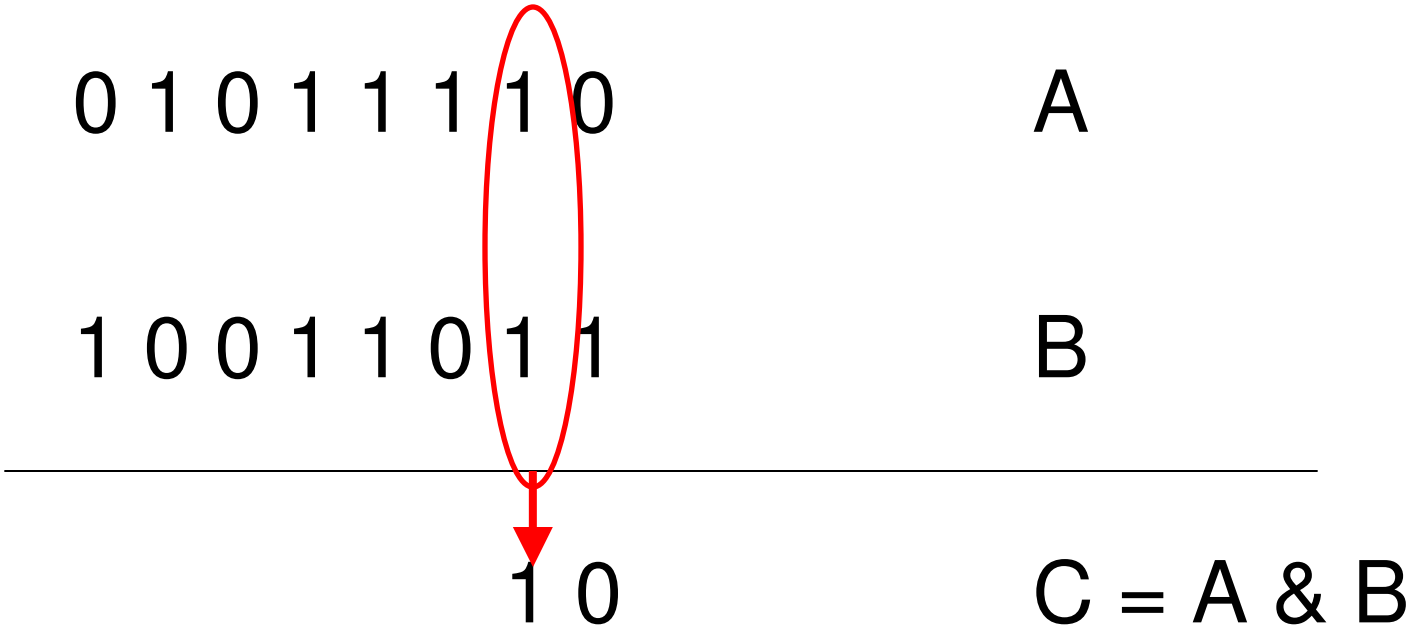
Bit-Wise Operators



Bit-Wise Operators



Bit-Wise Operators



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

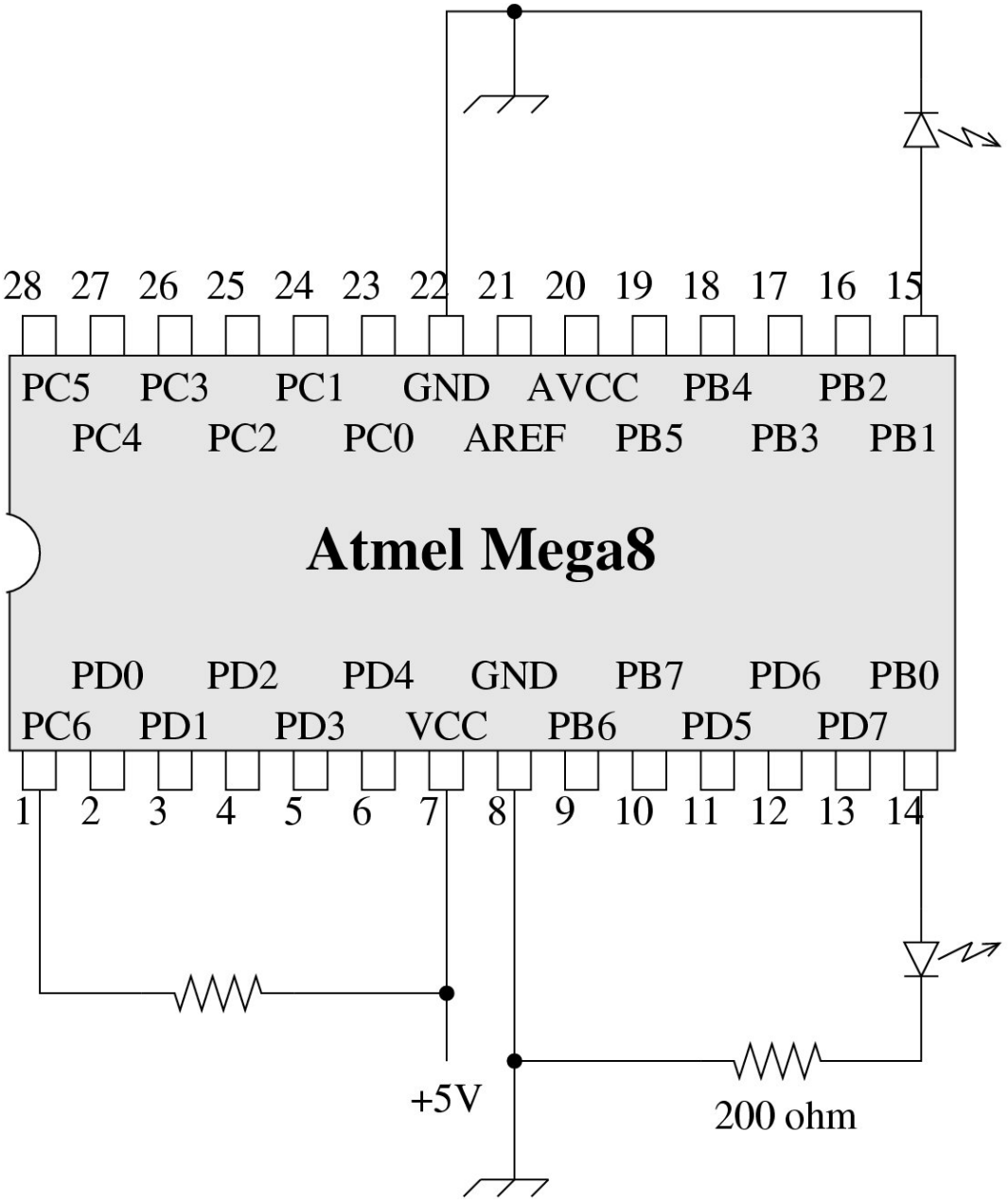
Given a byte A, how do we set bit 2 (counting from 0) of A to 1?

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

A First Program

Flash the LEDs at a regular interval

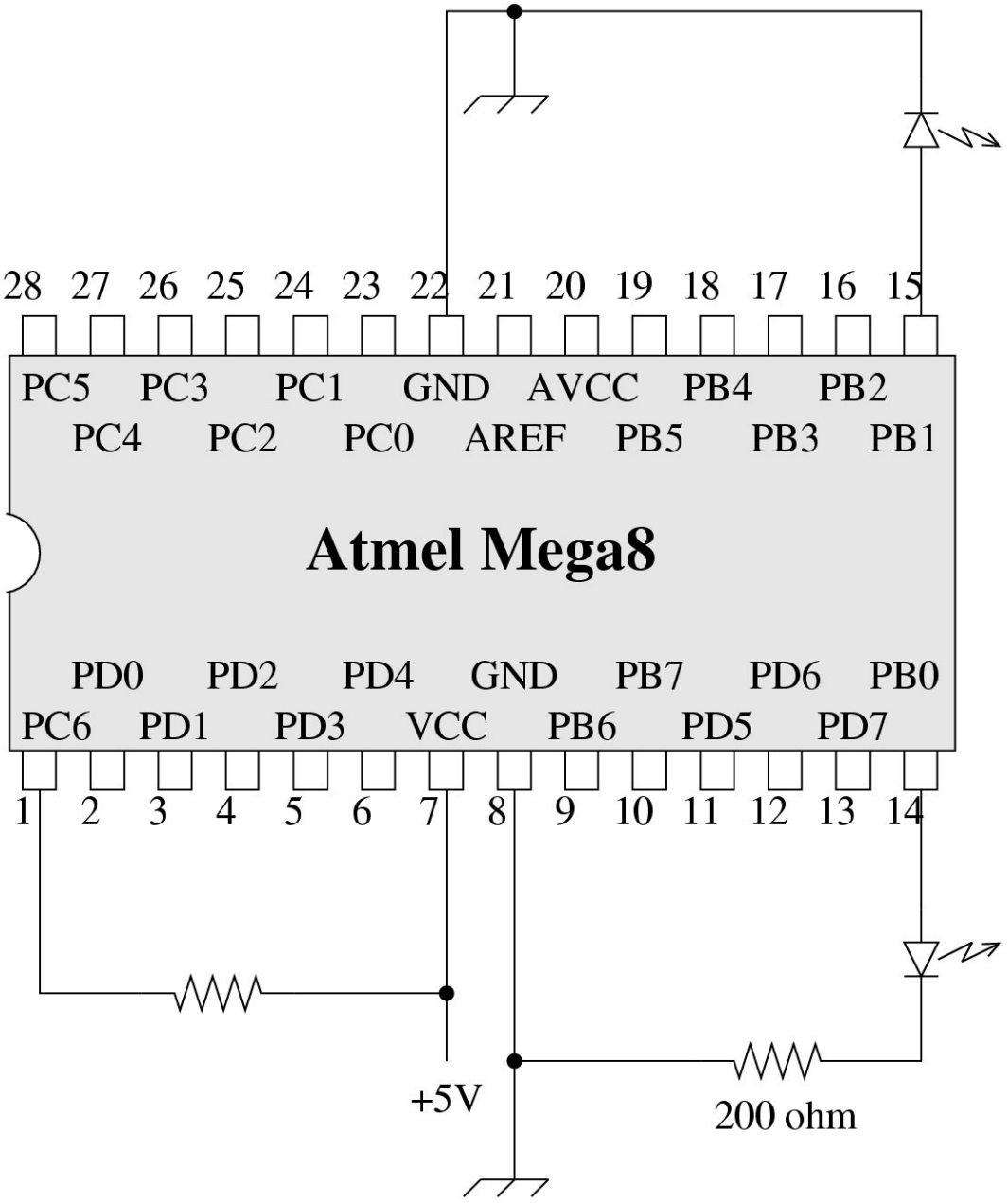
- How do we do this?



A First Program

How do we flash the LED at a regular interval?

- We toggle the state of PB0



A First 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
       }
}
```

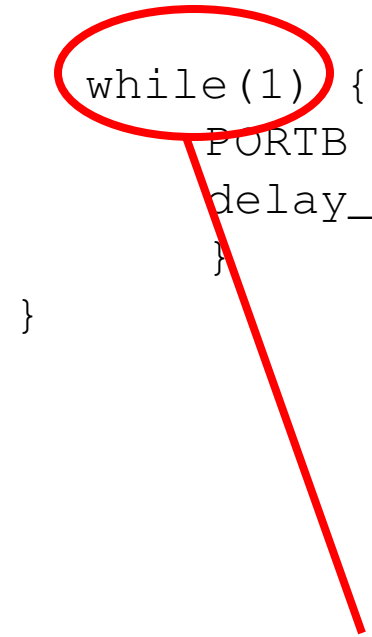
A First Program

```
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
        }
}
```

A predefined “variable” (register) that controls whether the port B pins are digital inputs or outputs (more on this later)

A First Program

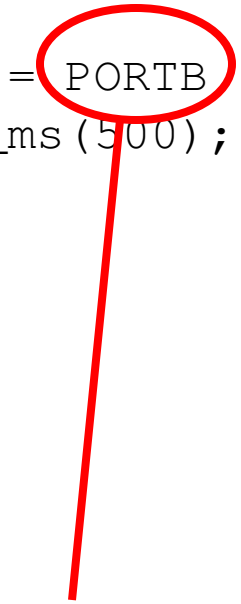
```
main() {
   DDRB = 0xFF;    // Set all port B pins as outputs

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



Loop forever

A First 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
        }
}
```



Another predefined variable: the value being written to port B

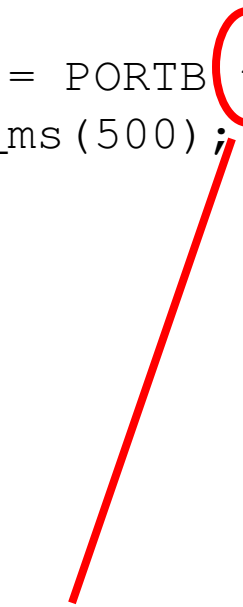
A First Program

```
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
        }
}
```

Change the value being written to port B

A First Program

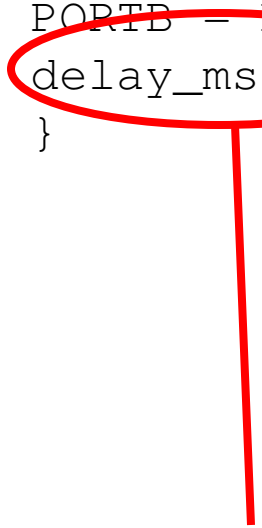
```
main() {
   DDRB = 0xFF;    // Set all port B pins as outputs

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



Bit-wise XOR operator

A First 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
        }
}
```



Program pauses for 500 msec. This function is defined elsewhere.

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 2 Hz
on PB0: 1 Hz

Last Time

Atmel microcontroller

- I/O pins
- Digital port implementation
- Digital output in code

Bit-wise operators

Today

- A “bit” more on bit masking/manipulation
- Digital input in code
- Practical issues in programming your mega8
- Homework 3
- Finite state machines

Administrivia

- Homework 4 due tonight at 5:00
- Thursday: Finite state machines and project 2
- Tuesday (next week): FSM continued and midterm review
- Thursday: midterm

- Spring break: the lab will be open on a limited basis (let me know if you want access)

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 in Practice

- Suppose you have connected your 3 robot control lines to B3, B4, and B5.
- Suppose also that you have connected your 2 turret control lines to B6 and B7

- Our robot control lines are specified as numbers 0 … 6
- How do we change the state of B3…B5 without changing the turret command?

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
}
```

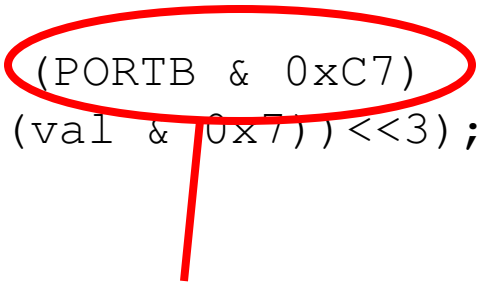
Bit Masking

```
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)

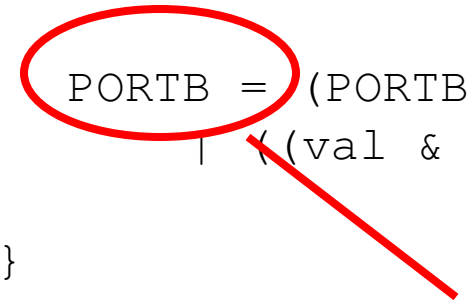
Bit Masking

```
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

```
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;
}
```

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 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 = PORTA & 0xFE;
```

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

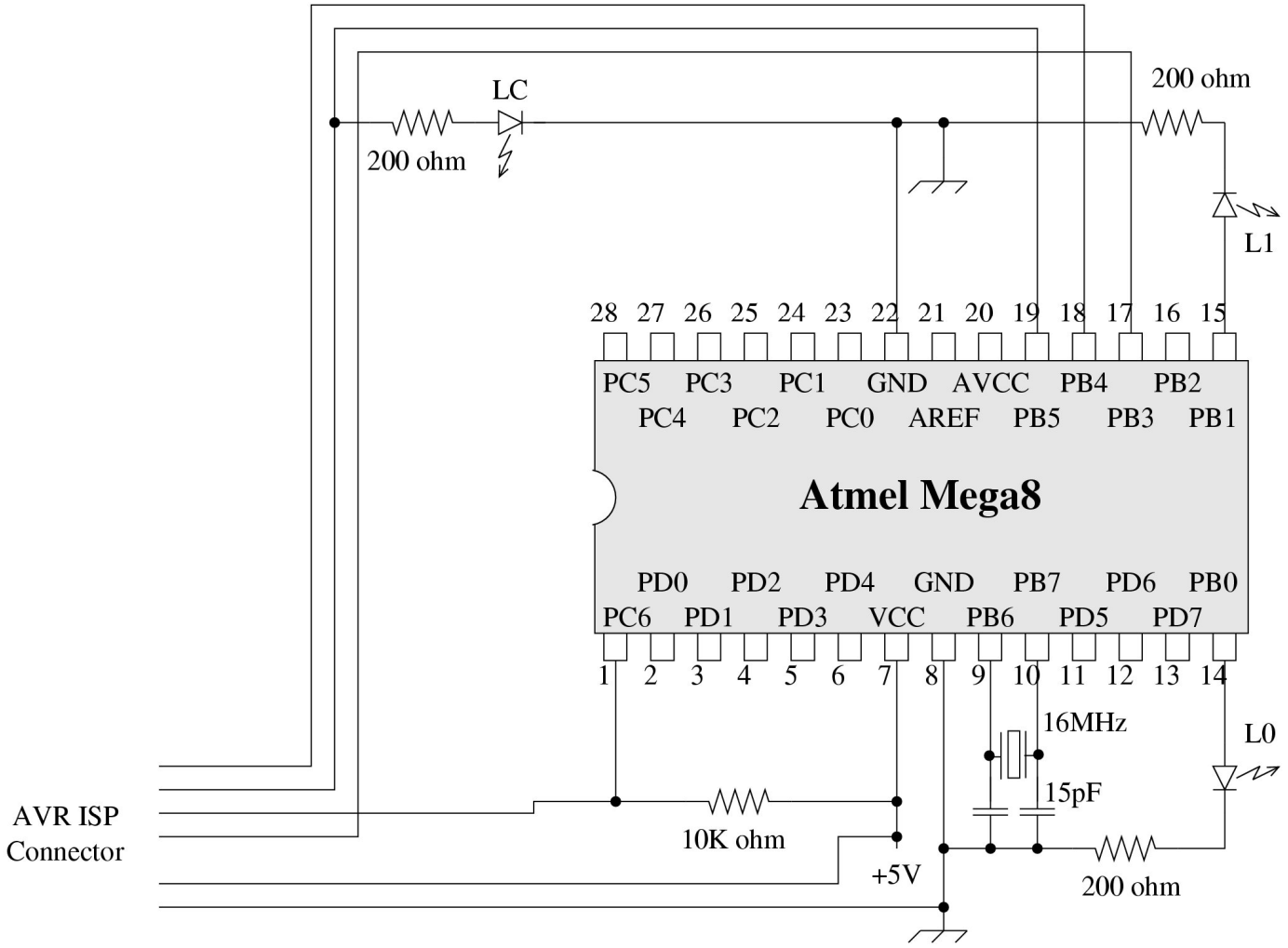
Serial
connection to
your computer

Physical Interface for Programming

AVR ISP

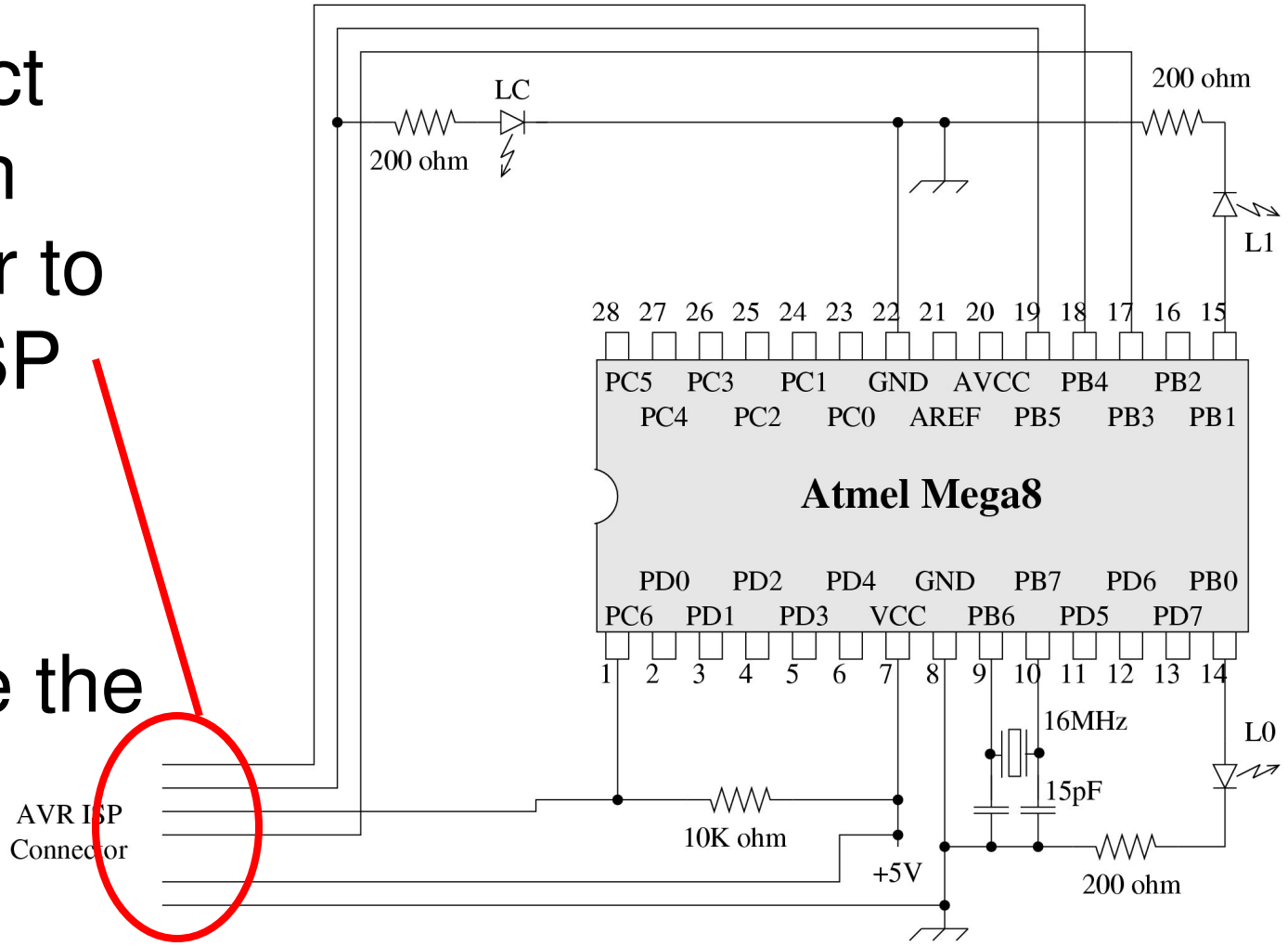
Header connection will connect to your circuit (through an adapter)

A More Complicated Circuit



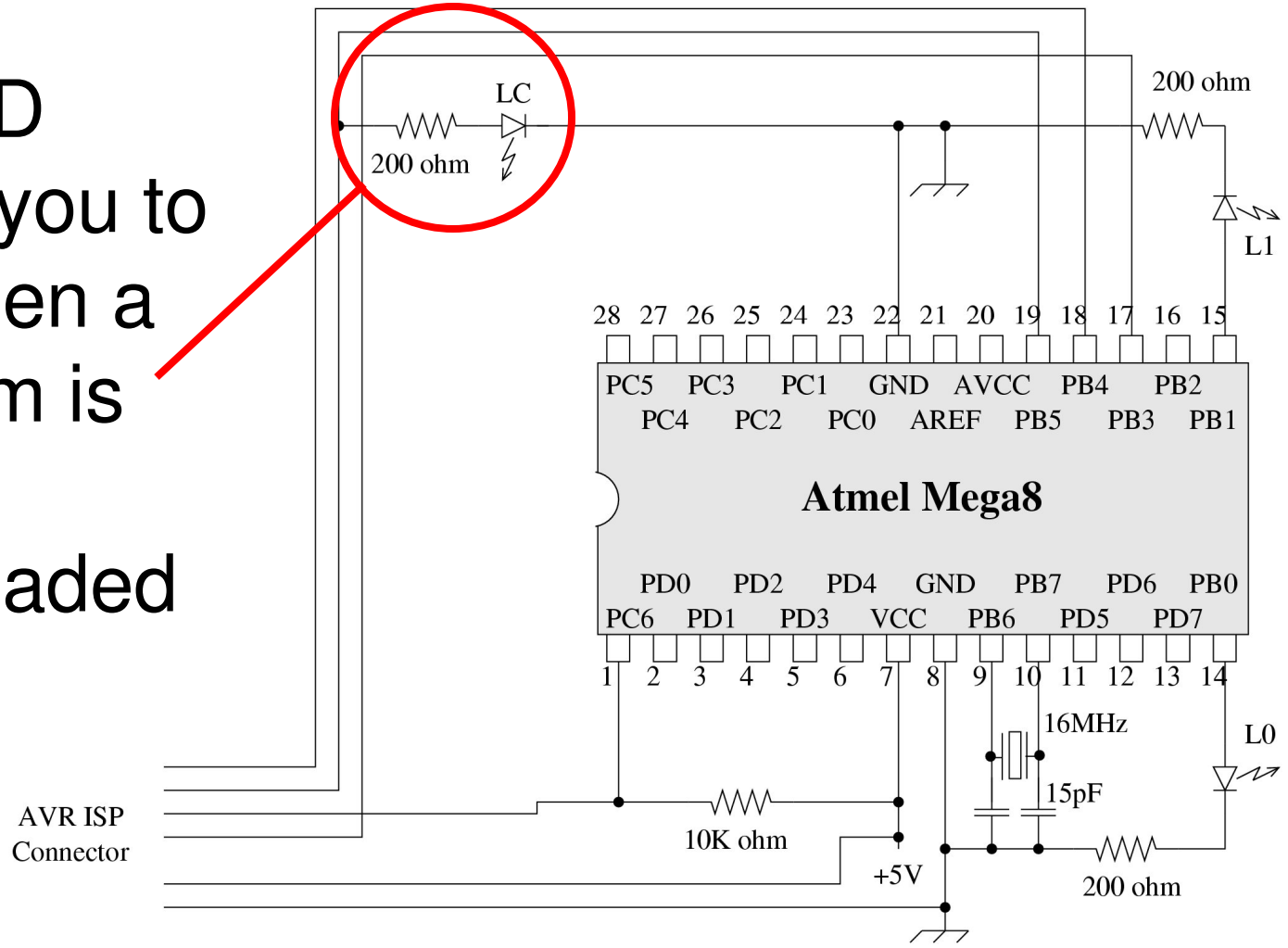
A More Complicated Circuit

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



A More Complicated Circuit

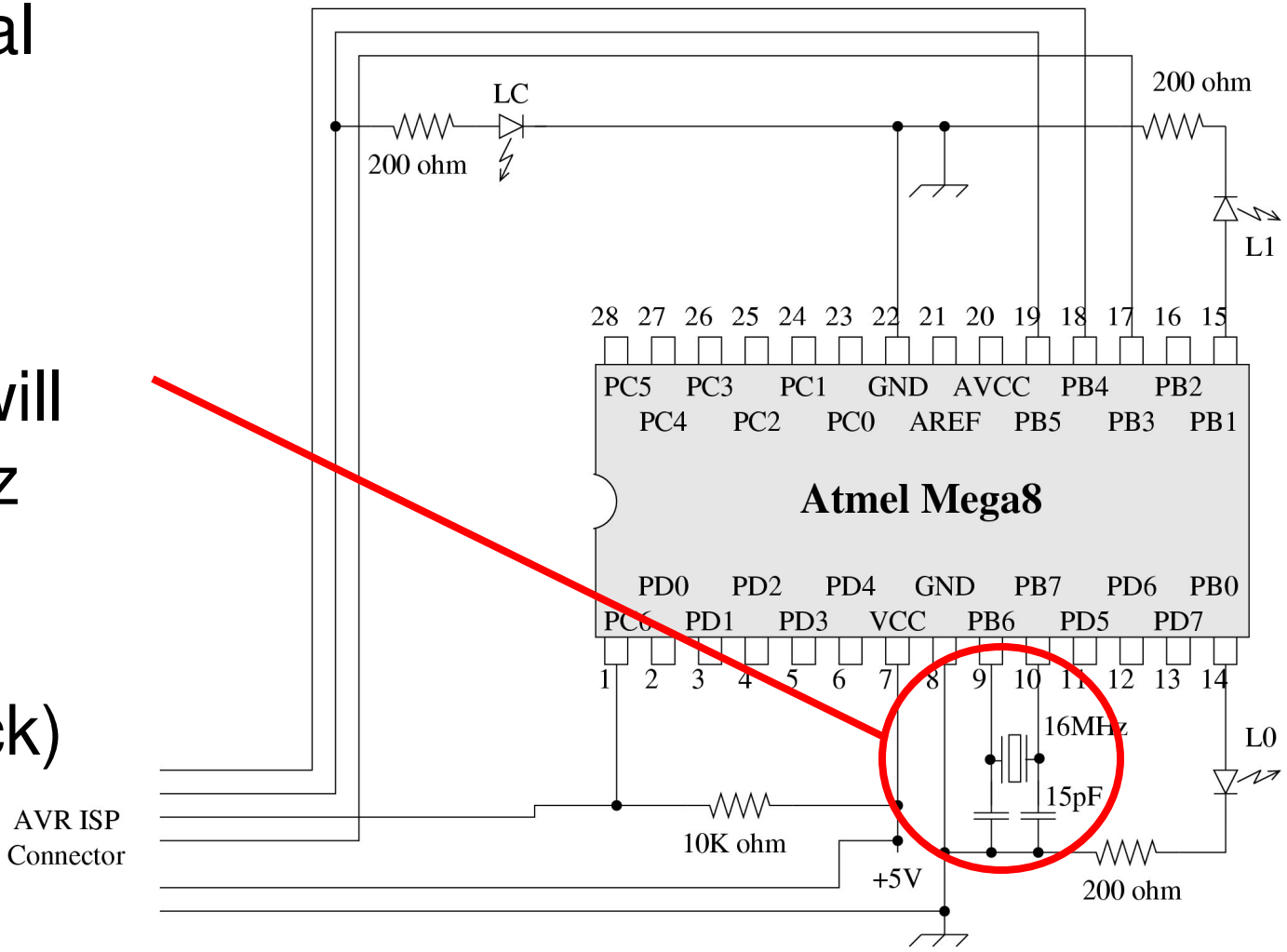
Extra LED allows you to see when a program is being downloaded



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

- Create a “makefile” and a C source file in some personal directory
  - Start with copies from the “Atmel HOWTO” part of the web page
- Change the TARGET line in makefile to match the name of your source file
- Change your source file

Compiling and Downloading Code

- From the windoze menu: select “Start” and then “run”
- Type “cmd”. This will bring up a “shell” interface (a command line)
- “cd” (change directory) to your directory
- Type “make”. Deal with any errors or warnings (both are there for a reason)
- Type “make program”

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 (use delay_ms())

Configuring the Clock

- By default, the mega8 chips are configured for 1 MHz operation
- To reconfigure for 16MHz:
  - Add the crystal + capacitors
  - Use ‘make setfuse’ to tell the chip that it has a crystal (you will only need to do this once)

- If you configure your code to run at 16MHz, and things seem to be running slow, then you probably need a ‘make setfuse’

Final Notes

- You will need some additional circuitry to program your processor
- There are a few more details in the code

-> See the examples posted on the net

Next Time

- Finite state machines
- Project 2