Bare Metal Programming

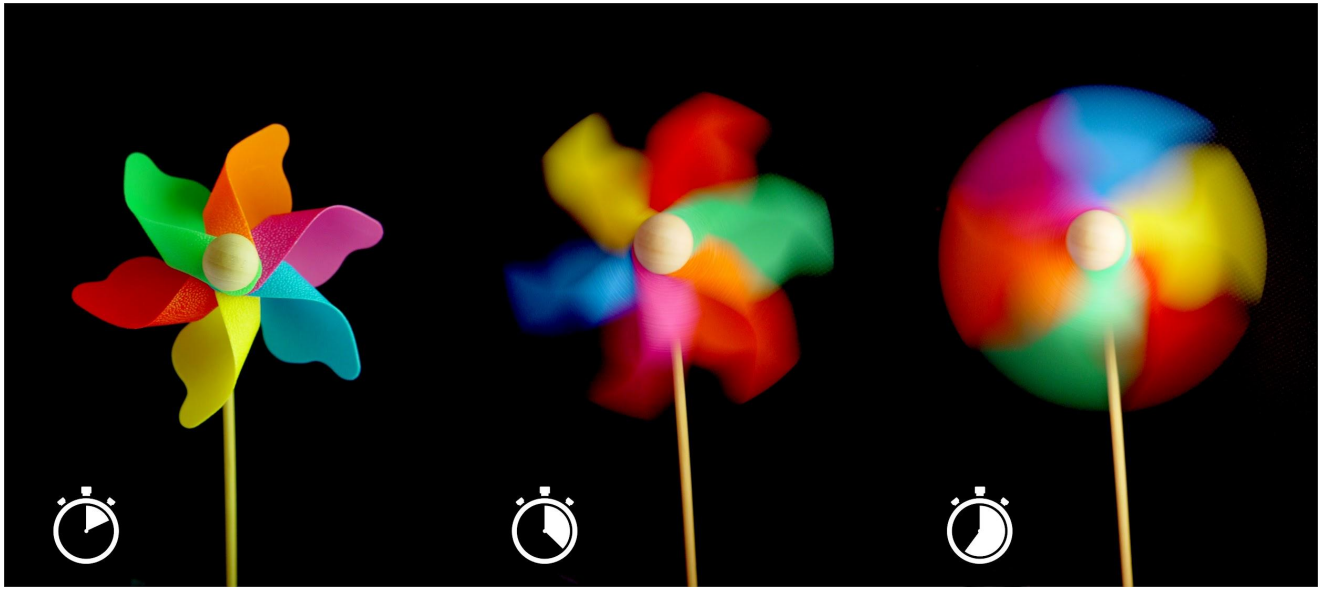
Yes, let’s take a byte out of this apple

(I am sorry I am dad and grandfather and I live for these puns.

This presentation is partially based on the
podcasts by Mitch Davis - Bare-Metal MCU

Bare Metal Programming

Why? Speed and Efficiency

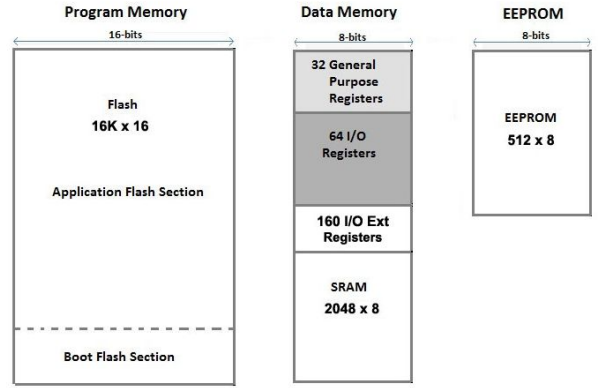


Bare Metal Programming

The Bootloader

The behaviour described above happens thanks to a special piece of code that is executed at every reset of the microcontroller and that looks for a sketch to be uploaded from the serial/USB port using a specific protocol and speed. If no connection is detected, the execution is passed to the code of your sketch.

This little (usually 512 bytes) piece of code is called the "Bootloader" and it is in an area of the memory of the microcontroller – at the end of the address space - that can't be reprogrammed as a regular sketch and had been designed for such purpose.

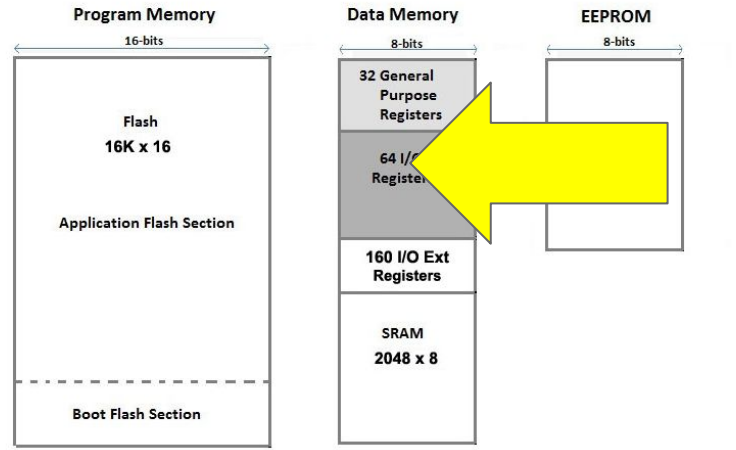


The Memory Map of an ATmega328P

When we mentioned Arduino Bootloader earlier in the class, we spoke about memory.

Specific memory blocks are called registers

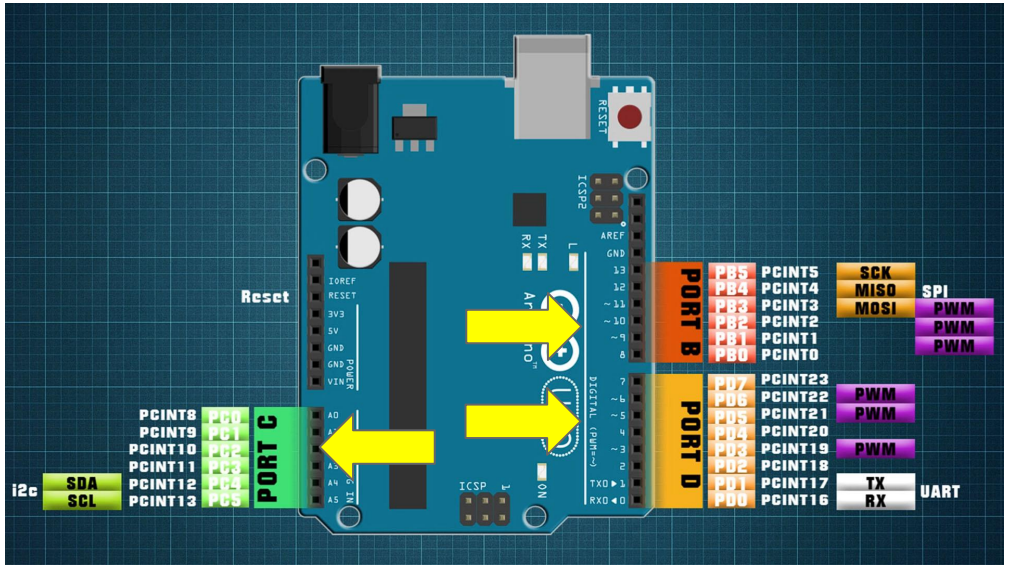
Bare Metal Programming



The Memory Map of an ATmega328P

Registers are memory blocks containing essential data for controlling or setting up microcontrollers.

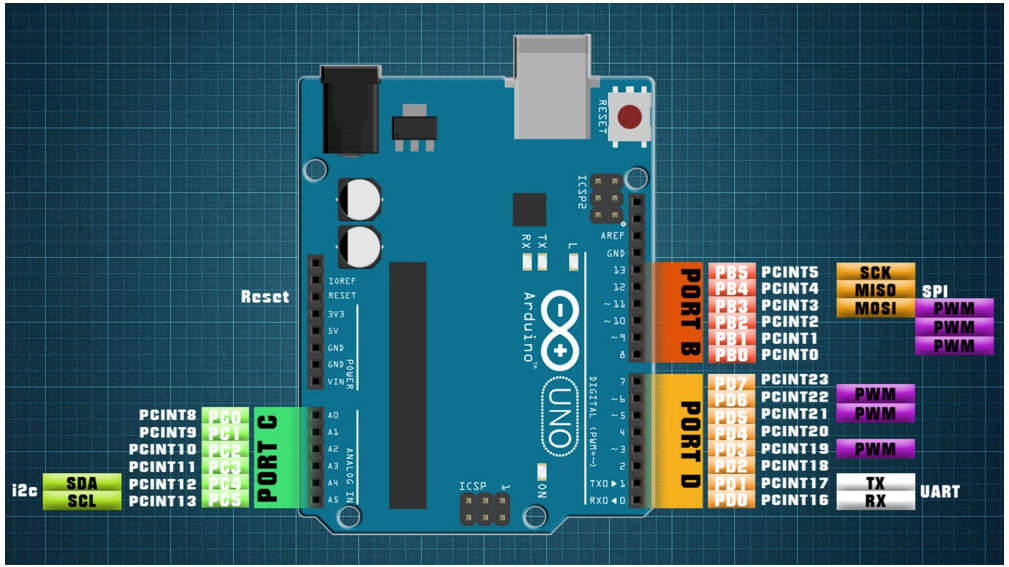
Bare Metal Programming



Each **port is controlled by three registers, which are also defined variables in the arduino language**. The DDR register, determines whether the pin is an INPUT or OUTPUT. The PORT register controls whether the pin is HIGH or LOW, and the PIN register reads the state of INPUT pins set to input with pinMode().

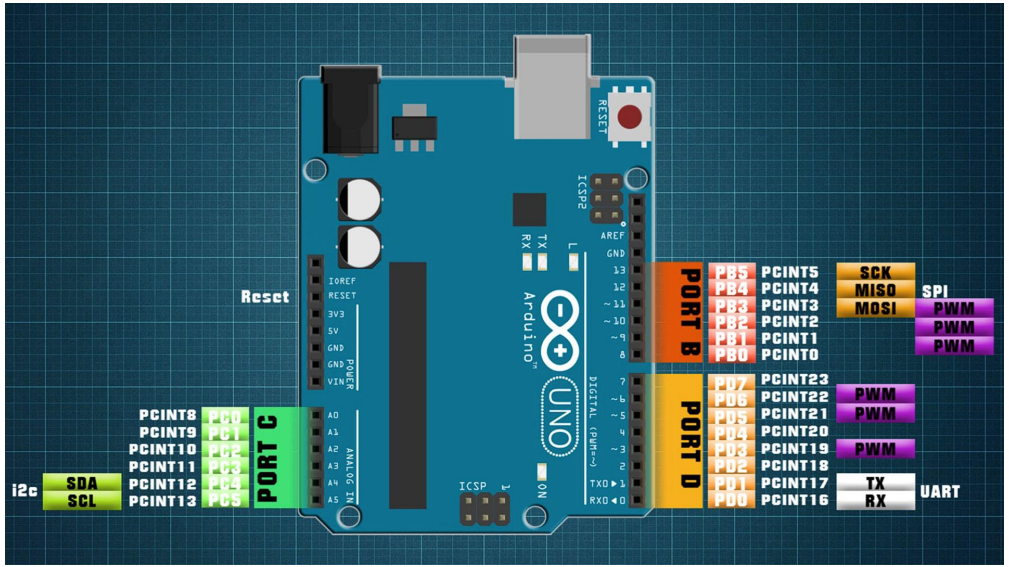
Registers are memory blocks containing essential data for controlling or setting up microcontrollers.

Bare Metal Programming



Each port is controlled by three registers, which are also defined variables in the arduino language. **The DDR register, determines whether the pin is an INPUT or OUTPUT**. The PORT register controls whether the pin is HIGH or LOW, and the PIN register reads the state of INPUT pins set to input with pinMode().

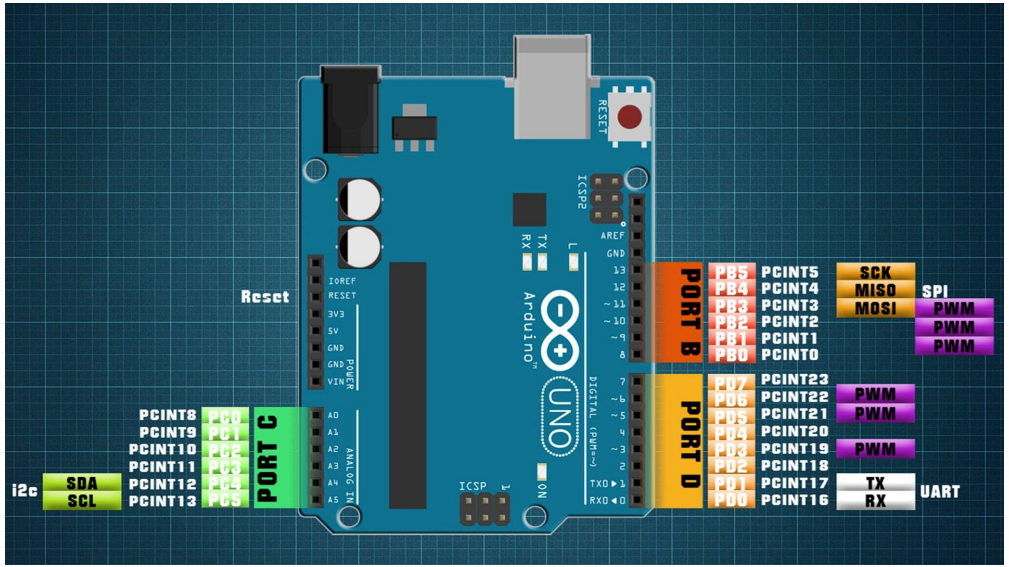
Bare Metal Programming



Each port is controlled by three registers, which are also defined variables in the arduino language. The DDR register, determines whether the pin is an INPUT or OUTPUT. **The PORT register controls whether the pin is HIGH or LOW**, and the PIN register reads the state of INPUT pins set to input with pinMode().

Ok, but what is a register exactly?

Bare Metal Programming



Each port is controlled by three registers, which are also defined variables in the arduino language. The DDR register, determines whether the pin is an INPUT or OUTPUT. The PORT register controls whether the pin is HIGH or LOW, and the **PIN register reads the state of INPUT pins set to input with pinMode()**.

Registers are memory blocks containing essential data for controlling or setting up microcontrollers.

Bare Metal Programming

```
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);   DDR Responsibility
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin,INPUT_PULLUP); DDR Responsibility
  }

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the
buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Each port is controlled by three registers, which are also defined variables in the arduino language. **The DDR register, determines whether the pin is an INPUT or OUTPUT**. The PORT register controls whether the pin is HIGH or LOW, and the PIN register reads the state of INPUT pins set to input with pinMode().

Look at our Arduino code to use a momentary switch with a LED.

Bare Metal Programming

```
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin,INPUT_PULLUP);
  }

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the
buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH); Port Responsibility
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW); Port Responsibility

  }
}
```

Each port is controlled by three registers, which are also defined variables in the arduino language. The DDR register, determines whether the pin is an INPUT or OUTPUT. **The PORT register controls whether the pin is HIGH or LOW,** and the PIN register reads the state of INPUT pins set to input with pinMode().

Look at our Arduino code to use a momentary switch with a LED.

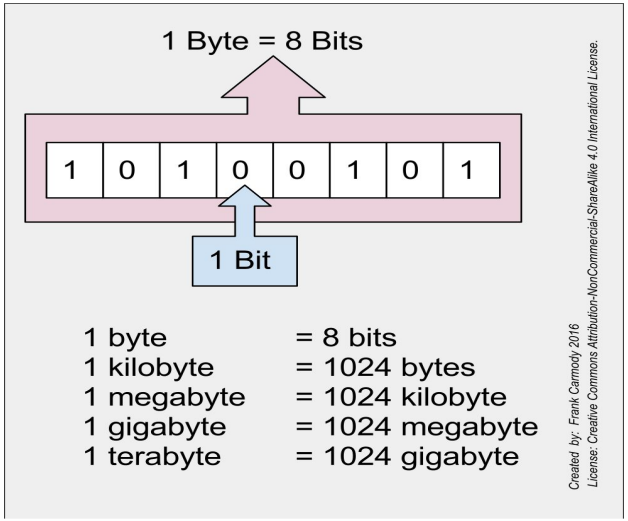
Bare Metal Programming

```
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin,INPUT_PULLUP);
  }

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin); Pin Responsibility

  // check if the pushbutton is pressed. If it is, the
buttonState is HIGH:
  if (buttonState == HIGH) Pin Responsibility
  {
    // turn LED on:      digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Each port is controlled by three registers, which are also defined variables in the arduino language. The DDR register, determines whether the pin is an INPUT or OUTPUT. The PORT register controls whether the pin is HIGH or LOW, and the **PIN register reads the state of INPUT pins set to input with pinMode()**.

Look at our Arduino code to use a momentary switch with a LED.

Bare metal Programming



We need a quick review of bits and bytes.

Bare metal Programming

Representation of a Binary Number

MSB	Binary Digit							LSB
2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
256	128	64	32	16	8	4	2	1

We saw above that in the decimal number system, the weight of each digit from right to left increases by a factor of 10. In the binary number system, the weight of each digit increases by a factor of 2 as shown. Then the first digit has a weight of 1 (2^0), the second digit has a weight of 2 (2^1), the third a weight of 4 (2^2), the fourth a weight of 8 (2^3) and so on.

We need a quick review of bits and bytes.

Bare metal Programming

So for example, converting a **Binary to Decimal** number would be:

Decimal Digit Value	256	128	64	32	16	8	4	2	1
Binary Digit Value	1	0	1	1	0	0	1	0	1

By adding together ALL the decimal number values from right to left at the positions that are represented by a "1" gives us: (256) + (64) + (32) + (4) + (1) = 357_{10} or three hundred and fifty seven as a decimal number.

We need a quick review of bits and bytes.

Bare metal Programming

Please note: The Arduino Port Registers are 8 bit.

0	0	1	1	0	0	1	0
0	0	32	16	0	0	2	0

50 = 0+0+32+16+0+02+0

The decimal equivalent of 00110010 is 50

We need a quick review of bits and bytes.

Bare metal Programming

Please note: The Arduino Port Registers are 8 bit.

Convert 00100010 to decimal

0	0	1	0	0	0	1	0

What is the decimal equivalent?

We need a quick review of bits and bytes.

Bare metal Programming

Please note: The Arduino Port Registers are 8 bit.

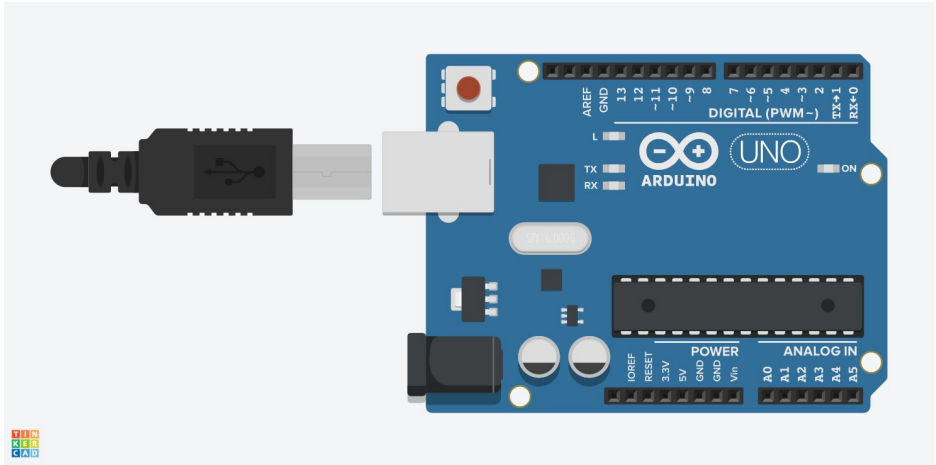
Convert 00100010 to decimal

0	0	1	0	0	0	1	0
0	0	32	0	0	0	2	0

What is the decimal equivalent? 34 = 32 +2

We need a quick review of bits and bytes.

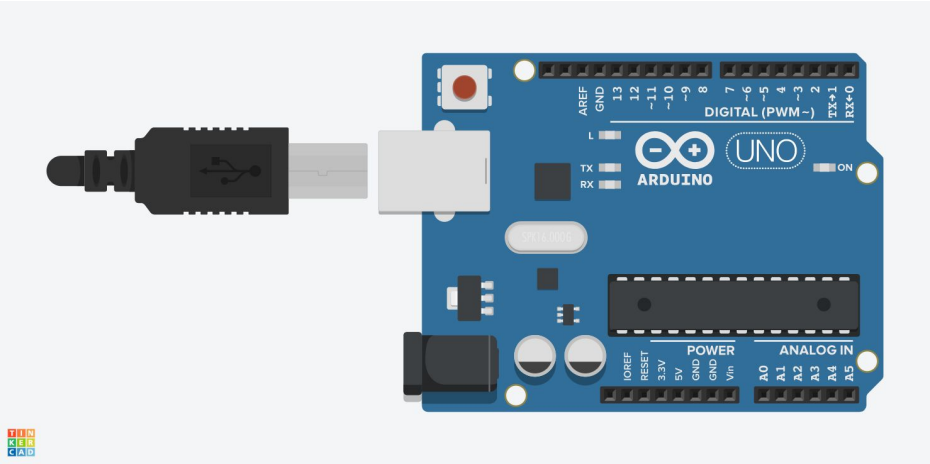
Bare Metal Programming



Port Register

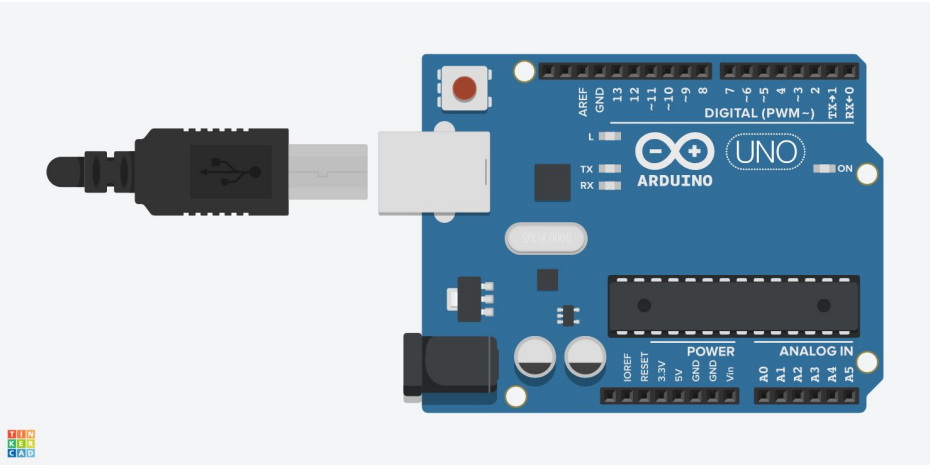
How to:

Bare Metal Programming



Let's return back to the Arduino. PLease review how we would program the BuiltIn LED pin (13) to blink using Arduino code.

Bare Metal Programming



Want to see it work? Please access Circuit in TinkerCad. Select an Arduino and review how to program Pin 13 or the BuiltIn Pin See link

```
// C++ code

void setup()

{

  pinMode(LED_BUILTIN, OUTPUT);

}

void loop()

{

  digitalWrite(LED_BUILTIN, HIGH);

  delay(1000); // Wait for 1000 millisecond(s)

  digitalWrite(LED_BUILTIN, LOW);

  delay(1000); // Wait for 1000 millisecond(s)

}
```

Bare Metal Programming

Port B has an 8 Bit Register

PortB0 Digital Pin 8

PortB1 Digital Pin 9

PortB2 Digital Pin 10

PortB3 Digital Pin 11

PortB4 Digital Pin 12

PortB5 Digital Pin 13 (LED_BUILTIN);

We cannot access PortB6 or PORTB

We want to turn on Pin13

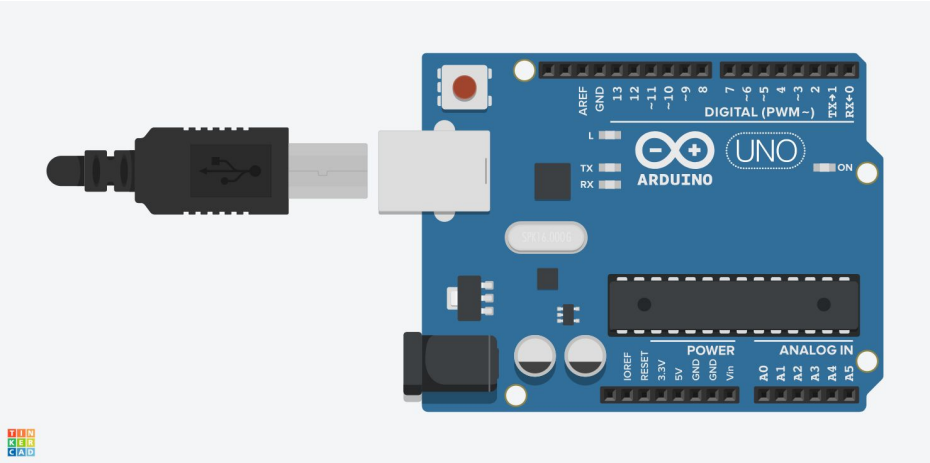
Therefore all the ports are set to Zero except for PortB5

00100000 binary

00320000 = 0+0+32+0+0+0+0

Port	Bin	Dec
B0	0	0
B1	0	0
B2	0	0
B3	0	0
B4	0	0
B5	1	32
B6	0	0
B7	0	0

Bare Metal Programming

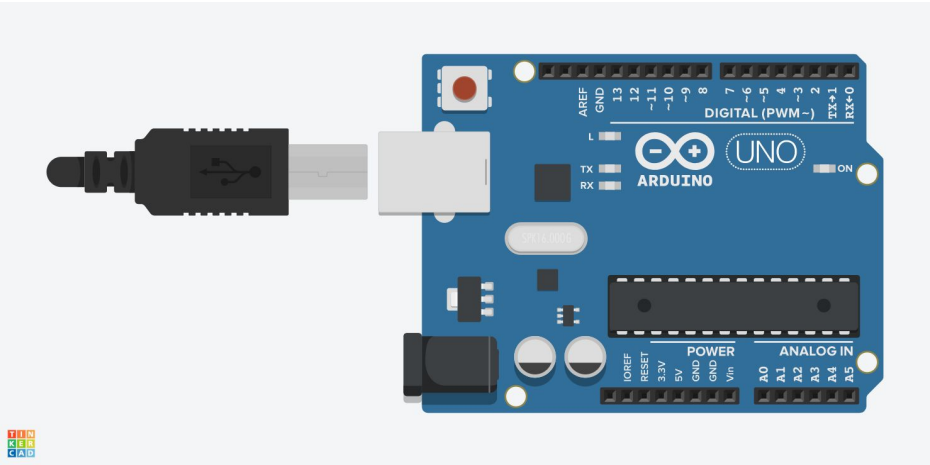


Now use the decimal equivalent to turn on BUILTIN LED

HINT 32

Click here for solution

Bare Metal Programming

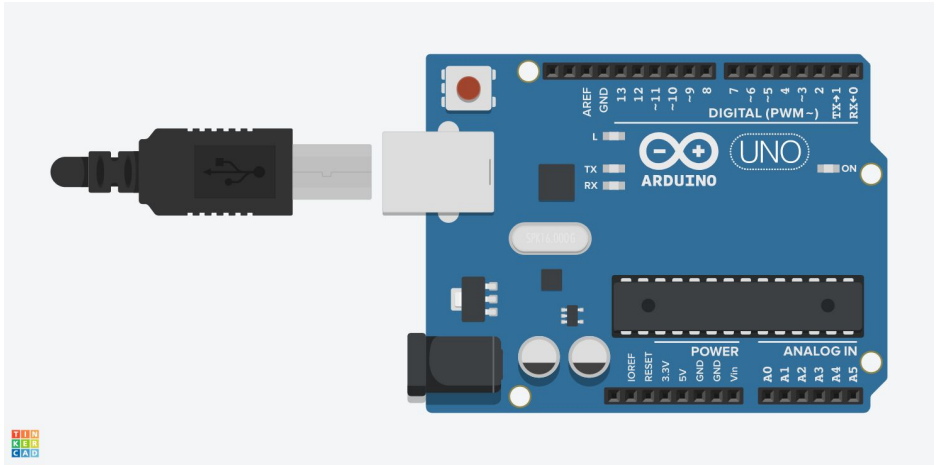


Now use turn on digital **pin 9** using the PORT variable to turn on a LED.

HINT: You will need a breadboard, resistor and LED

Click here for solution

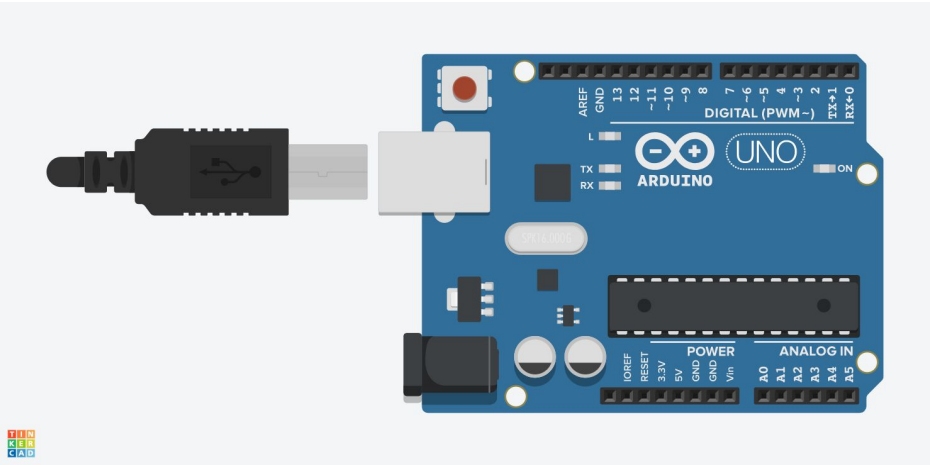
Bare Metal Programming



DDR Register

How to:

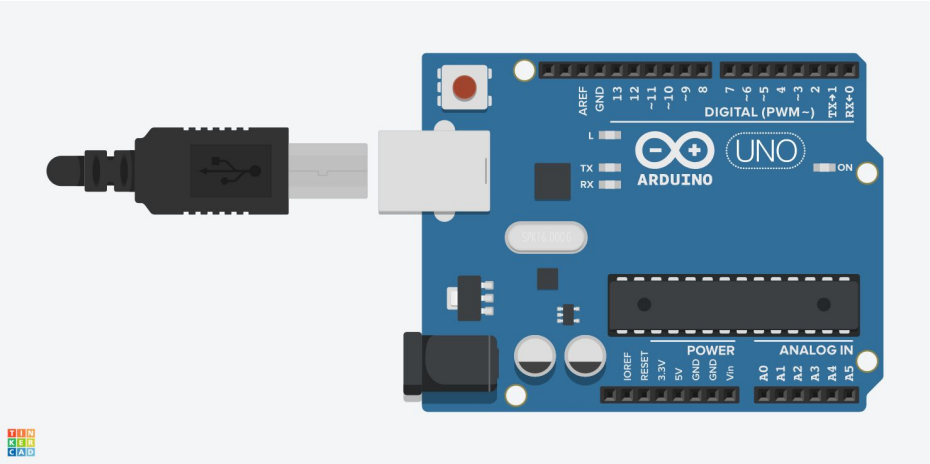
Bare Metal Programming



Click here for solution

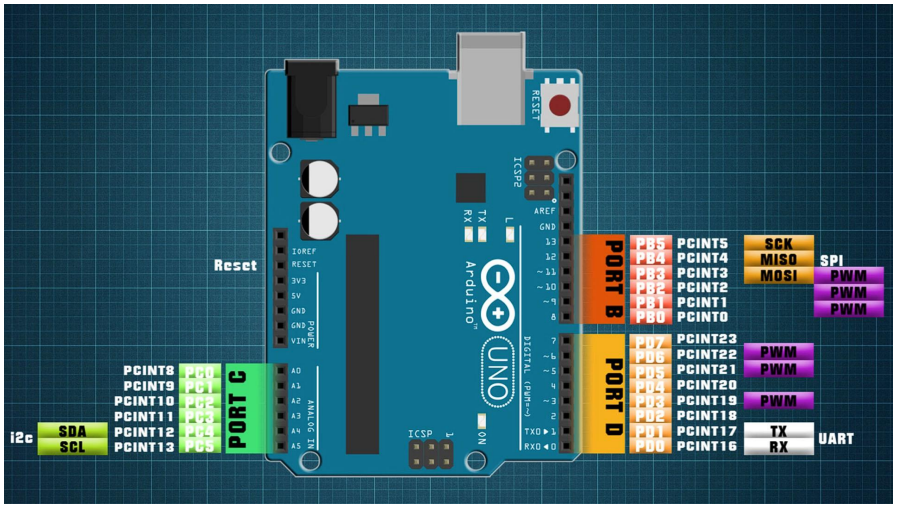
```
void setup()

{

//pinMode(LED_BUILTIN, OUTPUT);

  DDRB = 32;// B010000 sets PB5 as OUTPUT PB5 is 0010000
or 32

}

void loop()

{

  PORTB=32;  //0010000;

  delay(2000); // Wait for 1000 millisecond(s)

  PORTB=0;

  delay(1000); // Wait for 1000 millisecond(s)

}
```

Bare Metal Programming



Now go ahead and change the DDR register variable to suit Pin 9

Bare Metal Programming - memory & Pointers



14.4.2 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

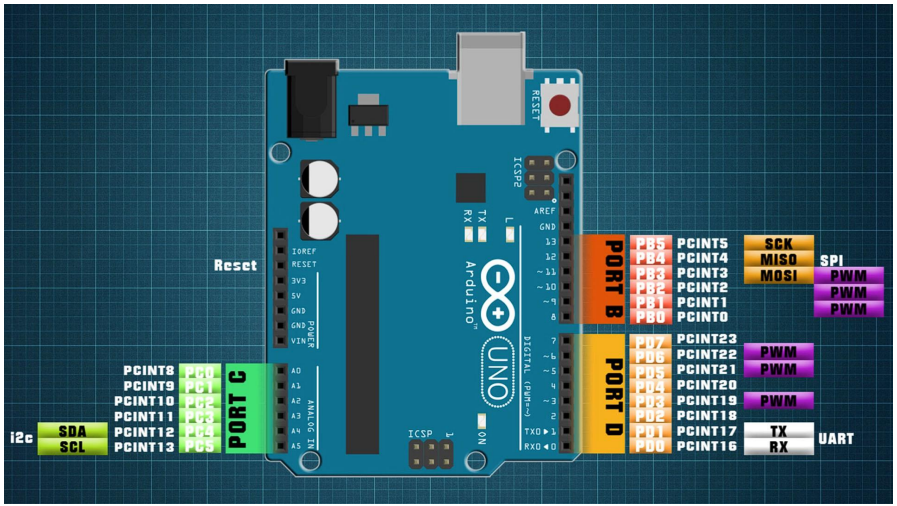
14.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.4.4 PINB – The Port B Input Pins Address[1]

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Bare Metal Programming - memory & Pointers



14.4.2 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.4.3 DDRB – The Port B Data Direction Register

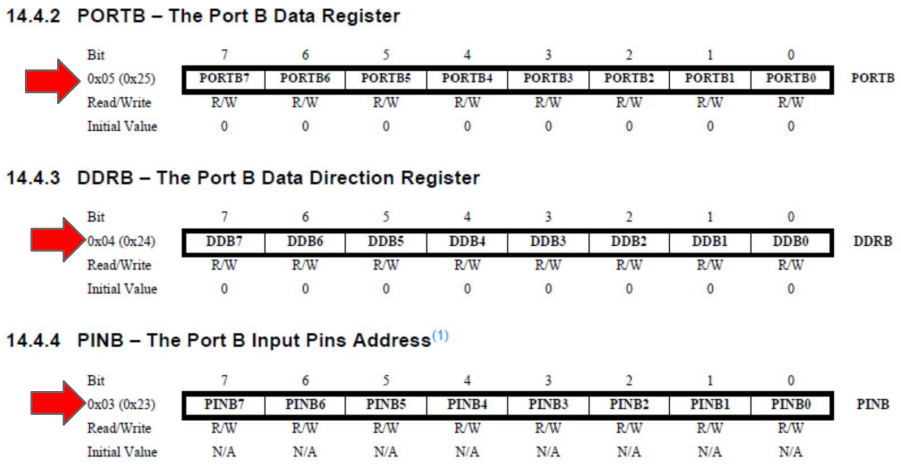
Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.4.4 PINB – The Port B Input Pins Address[1]

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Bare Metal Programming - memory & Pointers

So how do we write 32 to 0x25 (PORTB)?



14.4.2 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

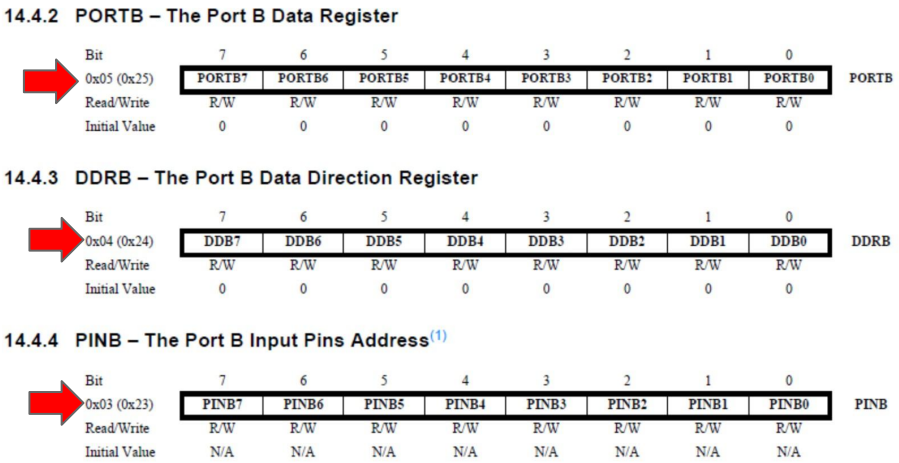
14.4.4 PINB – The Port B Input Pins Address[1]

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Bare Metal Programming - memory & Pointers

So how do we write 32 to 0x25 (PORTB)?

We use **pointers**!



14.4.2 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.4.4 PINB – The Port B Input Pins Address[1]

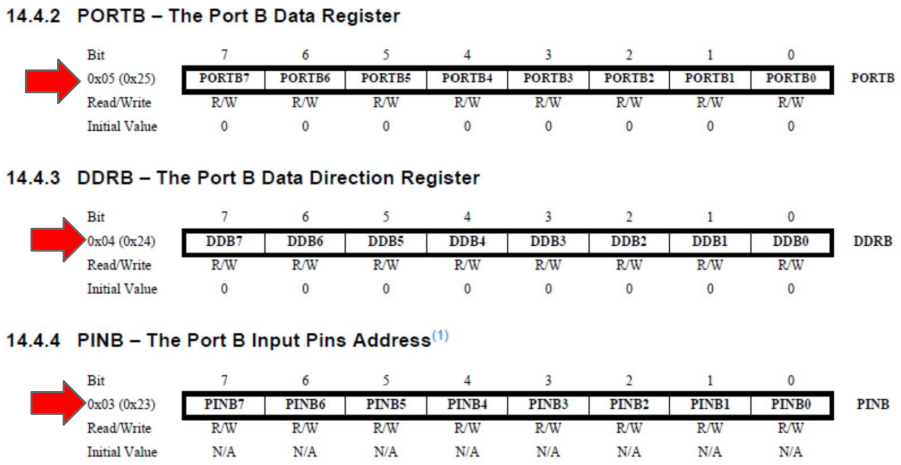
Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Bare Metal Programming - memory & Pointers

So how do we write 32 to 0x25 (PORTB)?

We use **pointers**!

A **Pointer** in the C language is a variable that ***points*** to the memory location of another variable memory address.



14.4.2 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.4.4 PINB – The Port B Input Pins Address[1]

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Bare Metal Programming - memory & Pointers

We use **pointers**!

A **Pointer** in the C language is a variable that ***points*** to the memory location of another variable memory address.

Simple Example

int classicVariable = 16; // create a “classic integer variable

int* pointerToClassicVariable = &classicVariable; //create a pointer to “point” to classicVariable or reference the variable

*** pointerToClassicVariable = 32;** //dereference pointerToClassicVariable and assign 32 to classicVariable

Bare Metal Programming - memory & Pointers

We use **pointers**!

A **Pointer** in the C language is a variable that ***points*** to the memory location of another variable memory address.

So how do we write 32 to 0x25 (PORTB)?

volatile byte* pointerToRegisterB = 0x25; // create a pointer variable called pointerToRegisterB which is pointed to Ox25;

*** pointerToRegisterB = 32;** //dereference pointerToRegisterB and assign 32 to memory location to Ox25

Bare Metal Programming - memory & Pointers

So how do we write 32 to 0x25 (PORTB)?

```
void setup()
{
DDRB = 32;// B010000 sets PB5 as OUTPUT PB5 is 0010000  or 32 );
}
void loop()
{
  volatile byte* pointerToRegisterB = 0x25;
  *pointerToRegisterB = 32;       //PORTB=32;// PORTB is from pinout 32 decimal

  delay(2000); // Wait for 1000 millisecond(s)

  *pointerToRegisterB = 0;  //PORTB=0; //PORTB 0 decimal

delay(1000); // Wait for 1000 millisecond(s)
}
```

Bare Metal Programming - memory & Pointers

So how do we write 32 to 0x25 (PORTB)?

volatile byte* pointerToRegisterB = 0x25; // create a pointer variable called pointerToRegisterB which is pointed to Ox25;

*** pointerToRegisterB = 32;** //dereference pointerToRegisterB and assign 32 to memory location to Ox25

Wow, can we do this all on one line? I don't type well

Bare Metal Programming - memory & Pointers

So how do we write 32 to 0x25 (PORTB)?

volatile byte* pointerToRegisterB = 0x25; // create a pointer variable called pointerToRegisterB which is pointed to Ox25;

* **pointerToRegisterB = 32;** //dereference pointerToRegisterB and assign 32 to memory location to Ox25

Wow, can we do this all on one line? I don't type well

Well, yes we can

* ((volatile byte*) 0x25) = 32;

Ok, I guess if you say so.

Bare Metal Programming - memory & Pointers

So how do we write 32 to 0x25 (PORTB)?

volatile byte* pointerToRegisterB = 0x25; // create a pointer variable called pointerToRegisterB which is pointed to Ox25;

* **pointerToRegisterB = 32;** //dereference pointerToRegisterB and assign 32 to memory location to Ox25

We can so let me show you.

* ((volatile byte*) 0x25) = 32;

We assign the memory address 0x25 to a pointer

Bare Metal Programming - memory & Pointers

So how do we write 32 to 0x25 (PORTB)?

volatile byte* pointerToRegisterB = 0x25; // create a pointer variable called pointerToRegisterB which is pointed to Ox25;

*** pointerToRegisterB = 32;** //dereference pointerToRegisterB and assign 32 to memory location to Ox25

We can say so. Let me show you.

* ((volatile byte*) 0x25) = 32;

We dereference it with * and assign the 0x25 to 32

Bare Metal Programming - memory & Pointers

So how do we write 32 to 0x25 (PORTB)?

volatile byte* pointerToRegisterB = 0x25; // create a pointer variable called pointerToRegisterB which is pointed to Ox25;

*** pointerToRegisterB = 32;** //dereference pointerToRegisterB and assign 32 to memory location to Ox25

We can say so. Let me show you.

* ((volatile byte*) 0x25) = 32;

We dereference it with * and assign the 0x25 to 32

Bare Metal Programming - memory & Pointers

So how do we write 32 to 0x25 (PORTB)?

```
void loop()
{
  /* volatile byte* memoryPointer = (volatile byte*) 0x25;// we are assigning 0x25 memory to memoryPointer
  *memoryPointer=32; */
  * ( (volatile byte*) 0x25) = 32;  //replaces two lines of the above code
    delay(2000); // Wait for 1000 millisecond(s)
 * ( (volatile byte*) 0x25) = 0;
  delay(1000); // Wait for 1000 millisecond(s)
}
```

Bare Metal Programming - memory & Pointers

So how do we write 32 to 0x25 (PORTB)?

```
void loop()
{
  * ( (volatile byte*) 0x25) = 32;  //replaces two lines of the above code
    delay(2000); // Wait for 1000 millisecond(s)
   * ( (volatile byte*) 0x25) = 0;
     delay(1000); // Wait for 1000 millisecond(s)
}
```

This is good. We can make it better by using the **#define**

Bare Metal Programming - memory & Pointers

```
#define blink13    * ( (volatile byte*) 0x25)

void setup()
{
DDRB = 32;
}

void loop()
{
  blink13 = 32;  // replaces  * ( (volatile byte*) 0x25) = 32;
   delay(2000); // Wait for 1000 millisecond(s)
  blink13 = 0; // ( (volatile byte*) 0x25) = 0;
     delay(1000); // Wait for 1000 millisecond(s)
}
```

#define allows us to create blink13 - sorta like a variable

Bare Metal Programming - memory & Pointers

```
#define blink13    * ( (volatile byte*) 0x25)
#define blink13Set    * ( (volatile byte*) 0x24)

void setup()
{

blink13Set = 32; // replaces  DDRB = 32;
}

void loop()
{

  blink13 = 32;  // replaces  * ( (volatile byte*) 0x25) = 32;

   delay(2000); // Wait for 1000 millisecond(s)

  blink13 = 0; // ( (volatile byte*) 0x25) = 0;

      delay(1000); // Wait for 1000 millisecond(s)
}
```

14.4.2 PORTB – The Port E

Bit	7
0x05 (0x25)	PORTB7
Read/Write	R/W
Initial Value	0

14.4.3 DDRB – The Port B

Bit	7
0x04 (0x24)	DDB7
Read/Write	R/W
Initial Value	0

14.4.4 PINB – The Port B In

We replace DDRB with blink13Set assigning 0x24 DDRB memory location

Bare Metal Programming - BIT Masking

We have been using axee, time for a scalpel

0	**0**	**1**	**0**	**0**	**0**	**0**	**0**

Ok, Pin 13 is turned on, but the other pins are turned off!

Bare Metal Programming - Quick register reminder

Port B has an 8 Bit Register

PortB0 Digital Pin 8

PortB1 Digital Pin 9

PortB2 Digital Pin 10

PortB3 Digital Pin 11

PortB4 Digital Pin 12

PortB5 Digital Pin 13 (LED_BUILTIN);

We cannot access PortB6 or PORTB

We want to turn on Pin13

Therefore all the ports are set to Zero except for PortB5

00100000 binary

00320000 = 0+0+32+0+0+0+0

Port	Bin	Dec
B0	0	0
B1	0	0
B2	0	0
B3	0	0
B4	0	0
B5	1	32
B6	0	0
B7	0	0

Bare Metal Programming - BIT Masking

We have been using axe, time for a scalpel

0	**0**	**1**	**0**	**0**	**0**	**0**	**0**

Ok, Pin 13 is turned on, but the other pins are turned off!

This is not very efficient. We just to turn on PB5 or Pin13

Bare Metal Programming - BIT Masking

First we need to get to the truth. The above is a Logic Truth Table and Not Truth Table

OR, AND, XOR - TRUTH TABLES

A	B	OR |	AND &	XOR ^
False	False	False	False	False
False	True	True	False	True
True	False	True	False	True
True	True	True	True	False

NOT ~ TRUTH TABLE

A	B
0	1
1	0

We have been using axe, time for a scalpel

Bare Metal Programming - BIT Masking

We have been using axe, time for a scalpel

OR OPERATION - TURN ON A PIN

	0	0	0	0	1	0	1	0
OR	0	0	1	0	0	0	0	0

0	0	1	0	1	0	1	0

PORTB = PORTB|32 OR PORTB|=32 – turns on Pin 13 while leaving on Pin 9 and Pin 11

Bare Metal Programming - BIT Masking

We have been using axe, time for a scalpel

AND OPERATION - TURN OFF A PIN

	0	0	1	0	1	0	1	0
AND	1	1	0	1	1	1	1	1
	0	0	0	0	1	0	1	0

PORTB = PORTB&223 OR PORTB&=233 - turns off Pin 13 while leaving on Pin 9 and Pin 11

Bare Metal Programming - BIT Masking

We have been using axe, time for a scalpel

AND OPERATION - TURN OFF A PIN

PORTB = PORTB&223 OR PORTB&=233 - turns off Pin 13 while leaving on Pin 9 and Pin 11

233 Really? How am I am going to remember that?

Bare Metal Programming - BIT Masking

We have been using axe, time for a scalpel

AND OPERATION - TURN OFF A PIN

PORTB = PORTB&223 OR PORTB&=233 - turns off Pin 13 while leaving on Pin 9 and Pin 11

233 Really? How am I am going to remember that?

Bit Shifting to the rescue!

Bare Metal Programming - BIT Masking

Bit Shifting

0	0	1	1	1	0	1	0	PORTB
0	1	1	1	0	1	0	0	PORTB <<1
1	1	1	0	1	0	0	0	PORTB <<2
1	1	0	1	0	0	0	0	PORTB <<3

Bare Metal Programming - BIT Masking

Bit Shifting

0	0	0	0	0	0	0	1	PORTB
0	0	0	0	0	0	1	0	PORTB <<1
0	0	0	0	0	1	0	0	PORTB <<2
0	0	0	0	1	0	0	0	PORTB <<3

Bare Metal Programming - BIT Masking

We have been using axe, time for a scalpel

Bit Banging - one bit on

	0	0	0	0	1	0	1	0
OR	0	0	1	0	0	0	0	0
	0	0	1	0	1	0	1	0

1<<5

```
PORTB |= 32
PORTB |=(1<<5)
```

Bare Metal Programming - BIT Masking

Bit Shifting

0	0	0	0	0	0	1	0	1<<2
1	1	1	1	1	1	0	1	~(1<<2)

Bare Metal Programming - BIT Masking

Bit Shifting

	1	0	1	0	1	0	1	0
AND	1	1	0	1	1	1	1	0

~(1<<5)

1	0	0	0	1	0	1	0

PORTB &= 223

PORTB &=~(1<<5)

Bare Metal Programming - BIT Masking

We have been using axe, time for a scalpel

Bit Banging - one bit on

1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	0
1	0	0	0	1	0	1	0

XORR

1	0	0	0	1	0	1	0
1	1	1	1	1	1	1	0
1	0	1	0	1	0	1	0

XORR

PORTB ^ (1<<5) or PORTB ^= (1<<5)

XOR one or other but not both

Bare Metal Programming - BIT Masking

Bare Metal Programming - Putting It All Together

```
#define blink13    * ( (volatile byte*) 0x25)
#define blink13Set    * ( (volatile byte*) 0x24)

void setup()
{

blink13Set  |=(1<<5)    // replaces 32 or B0010000

 }

void loop()
{
   blink13  |= (1<<5);    // replaces 32 or B001000032;  // replaces  32

   delay(2000); // Wait for 2000 millisecond(s)

   blink13  &= ~(1<<5); //

    delay(500); // Wait for 500 millisecond(s)
}
```