Arduino

Tutorial 2
Arduino Uno and Micro-Controller

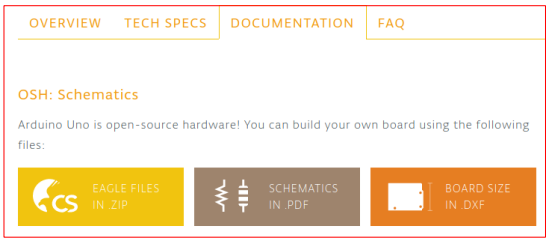
2021 by TeakSoon Ding
for STEMKRAF
https://github.com/teaksoon/stemkraf

Micro-controller and Arduino Uno

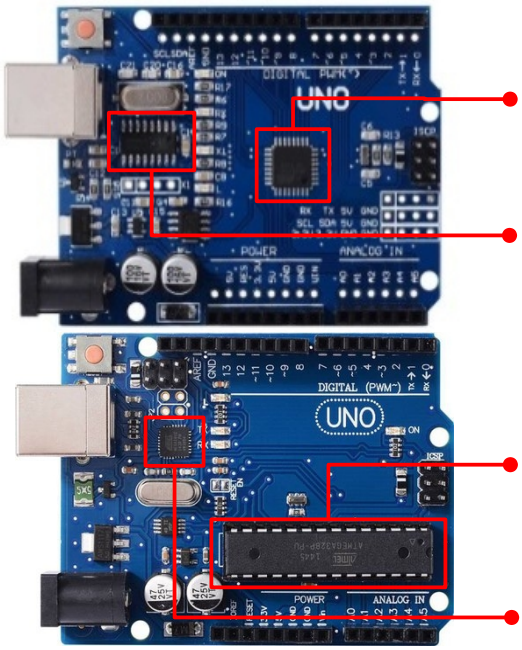
The **Arduino Uno board is a tool** to help us to operate and work with the **Atmega328 micro-controller**

The Arduino Uno board is an Open Source Hardware product. Below is the link to the Official Specification for Arduino Uno R3 board

https://store.arduino.cc/usa/arduino-uno-rev3



From the Open Source Hardware Specification, many Arduino Uno Clone boards were made. Some boards with the exact same design, some with variations. Regardless of which Arduino Uno board, The most important component that all of them must have, is the Atmega328 micro-controller chip.



Atmega328 micro-controller chip (QFP Package) on the Arduino Uno board

USB-TTL Chip (CH340)

Atmega328 micro-controller chip (DIP Package) on this Arduino Uno board

USB-TTL Chip (ATmega16U2)

The picture above shows two Arduino Uno board. They look a little bit different from each other because the components are either having a different physical packaging, brand or are being arranged differently on the board.

The Atmega328 micro-controller chip comes in 3 different physical chip packaging design. Operational wise, they work the same.

Quad Flat Package (QFP)	Dual Inline Package (DIP)	Quad Flat No-Lead (QFN)
		

The Atmega328 micro-controller chip has tiny exposed metal parts outside the chip packaging. They are commonly known as Input/Output "pins". Our micro-controller CPU will be able to manipulate the operations of devices connected to the "pins"

Connecting devices directly to the Atmega328 micro-controller "pins" is very difficult because they are very small, often requires soldering. The Arduino Uno board provide "solderless connectors" to the "pins". Apart from the connectors, the Arduino Uno board also have many components to assist the operations of the Atmega328 micro-controller, such as Voltage Regulator, USB Serial Communication, Oscillators (for System Clock), Test LED and etc...

A micro-controller is like an entire Computer System (Hardware and Software) all packed inside a single chip. Both the Computer System and Micro-controller can be programmed to perform different task.

General comparison for some of the significant difference between the Regular Computer and the Micro-controller.

I/O Connections (Regular Computer vs Micro-controller)

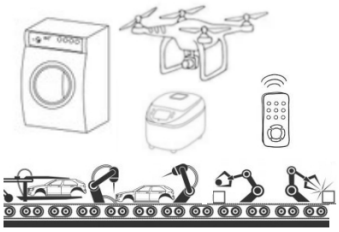
Regular Computers are often connected to standard Input and Output devices. Common computer parts such as Keyboard, Mouse, Monitor, Printer and etc

Micro-controllers are often connected to "raw" Input and Output devices. Electrical parts such as switches, LED, sensors, motors, radio and other specialised components

Programming (Regular Computer vs Micro-controller)

Regular Computers Program often deals with Data Entry, Screen Display and Data Storage/Retrieval, often for human interactions. Mostly information based applications

Micro-controller Program often deals with reading or changing of electrical properties (Voltage, Current and Resistance) at various point of an electrical circuit, creating "Smart Devices" operating with minimal human interactions. Mostly smart electrical equipment.



Micro-controller usage has no limits because it can work with "raw" components of an electrical circuit. It is up to us on how to use it. Some Micro-controllers, like the ARM based micro-controllers has even been programmed to be used like a Regular Computer (for example: Smartphones).

There are many types of micro-controllers in the market from different suppliers. Micro-controllers are often very small, each with their own unique features and capabilities.

The Atmega328 micro-controller is one of the many micro-controllers out there in the market.

Although there are many kinds of micro-controller out there (with different features and abilities), all of them have these few **Main Components.**

1. **CPU - Central Processing Unit**
2. **System Clock - Oscillator**
3. **Memory (Permanent and Non-Permanent)**
4. **Input & Output Interface**
5. **Instruction Set (AVR for Atmega328)**
6. **Program** (This is not supplied by the manufacturer of the micro-controller, we are expected to make this on our own)

All these are inside the single tiny Micro-controller chip

Why Atmega328 micro-controller ?

Atmega328 micro-controller is proven reliable, very cheap, easy to use, easy to learn, a massive amount supporting resources and is easily available.

The micro-controller "pin" holds electrical properties (Electric Current, Voltage, Resistance and Electric Current Flow Direction). We either read or change the electrical properties on the "pin" from our Micro-controller.

LED bulb - Will light-up when there is enough "Voltage" and "Electric Current" with the correct "Electric Current flow direction" at the "pin" that it is connected to.

Tactile Switch - Will **allow or disallow** "Electric Current" flowing through this the tactile Switch.

LDR (SENSOR) - Will change "Resistance" value in the LDR based on the intensity of the light it is exposed to.

From the 3 electrical device example above, we have the following

Electric Current (Ampere) is a measurement of the **"amount of Electric Energy at a point"** in an Electrical Circuit.

Resistance (Ohm) is a measurement of **"obstacles"**, often introduced to reduce the amount of Electric Current in an Electrical Circuit.

Voltage (Volt) is the measurement of **"imbalance"** between the **Negative Charged Atom Side(-ve)** and **Positive Charged Atom Side (+ve)** of a Power Source

Electric Current flow direction is the direction of energy flow. **The flow direction is from +ve terminal to -ve terminal** (the flow direction will influence the operations of some devices such as diode, motor and etc...)

Ohms Law

Voltage = Current X Resistance
Current = Voltage / Resistance
Resistance = Voltage / Resistance

When we work with our Regular Computers, we do not need to know all these because we do not connect "raw" electrical devices to Regular Computers. However, for **micro-controllers**, we have **"raw" electrical devices** connected to the micro-controllers "pins". Programming micro-controller I/O is all about dealing with various electrical properties at the "pins". **We will need to know at least the basics of eletricity, otherwise it will be very difficult for us to write our Program effectively for our micro-controller.**

!!!WARNING !!!

Extra care must be taken on how much Electric Current is flowing through our devices. Every device have a limit of how much Electric Current it can cope (Please refer to device specification provided by the manufacturer). If Electric Current exceed the limit, the device will most likely be destroyed. Same goes to us, if Electric Current flow into our body exceeds our body Electric Current limit, our body will also be destroyed.

We need to understand first how the Electric Energy can suddenly appear with all those electrical properties.

It all started with an **Atom**. Atom is the smallest unit in every physical matter that exist (physical size in pico-meter). A combination of a large number of Atoms makes physical matter visible to our eyes. Atom has **Neutron**, **Proton** (both in the middle, as core) and **Electron** (moving about freely)

For the basics, we only need to know that we have electrons moving from one Atom to another, generating Electric Energy.

In our tutorial example we only see picture of 2 electrons on a chain of single Atom, try to imagine a massive number of them in real-life. For example, a tiny 0.5mm copper wire contains billions of electrons.

An Atom is consist of (Neutron and Proton) in the middle and Electron moving freely on the outside

Number of Electron is Equal To Number of Proton
Neutral Atom (Electrically Neutral)

In some material, Electrons can be removed or added

Number of Electron below is Not Equal To Number of Proton
Atom is now charged (ions) This will become our "Power Source"

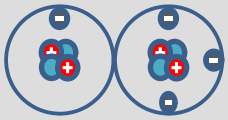
Electron is taken away from Atom

Atom have less Electron than Proton
= Positive Charged Atom = +ve

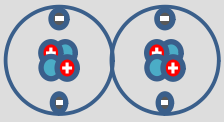
Electron is added to Atom

Atom have more Electron than Proton
= Negative Charged Atom = –ve

What happen if Positive and Negative Charged Atom are placed together ?



Electron from Negative Charged Atom will jump over to the Positive Charged Atom, both previously Charged Atom becomes Neutral again.



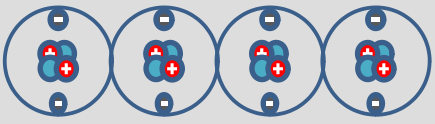
Power Source

+Ve

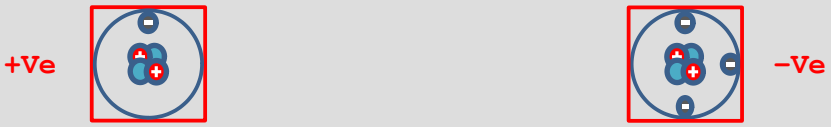
–Ve

Electric Conductor

is a material with Electrons that can be easily removed from its Atom
(mostly metals)

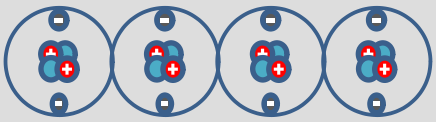


Power Source

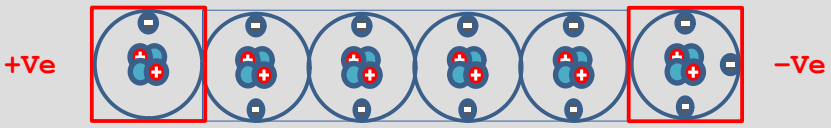


Electric Conductor

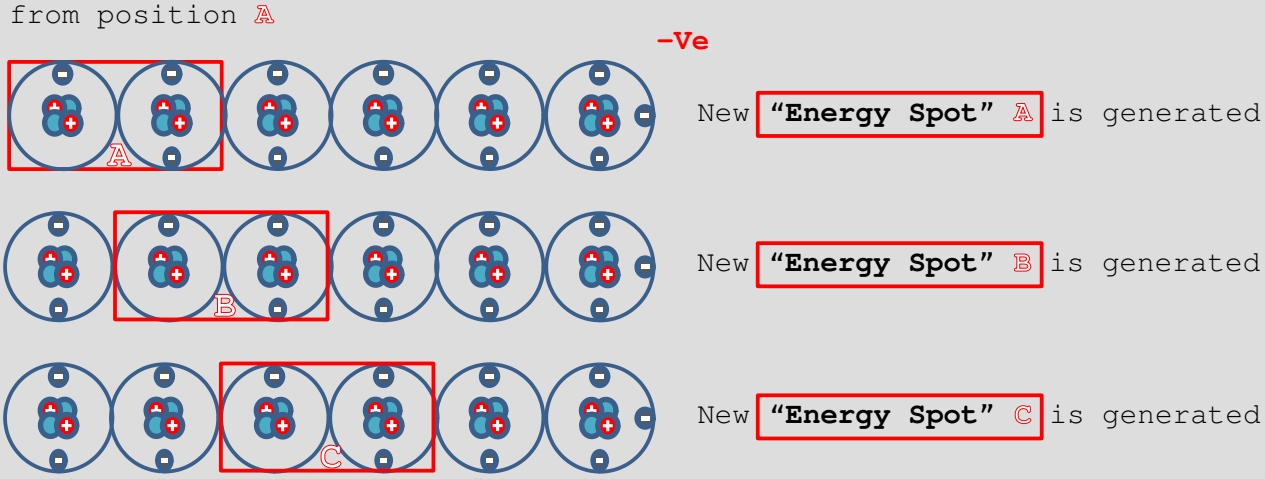
is a material with Electrons that can be easily removed from its Atom (mostly metals)



Place Electric Conductor between the Power Source
There is a path between the +ve and –ve (Known as an Electric Circuit)



Electron starts moving, starting from position A



First, we see **"imbalance"** at both ends of this circuit.

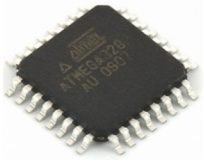
One side have more Electron(-ve), the other side have less Electron(+ve). **The "imbalance" is** Voltage. This "imbalance" will encourage the "Electron" to move to make Charged Atom Neutral.

The **"imbalance"** caused Electron move starting position A to make the Charged Atom there Neutral, then it goes to B, C, and ... until no more "imbalance". Each time Electron moves, **"Energy Spot" or** Electric Current is generated. Anything placed physically on the Energy Spot will receive Electric Current.

The sequence of "Energy spot" appearing at A, then B, then C, then ... is the Electric Current flow direction (+ve to -ve)

Not all Electric Conductors are equal, some material with Electrons that can move easily (less Resistance), some with Electrons that are hard to move(more Resistance).

More Resistance = Less Electric Current can flow
Less Resistance = More Electric Current can flow



1.CPU – Central Processing Unit
2.System Clock – Oscillator
3.Memory
4.Input & Output Interface
5.Instruction Set
6.Program (This is not supplied by the manufacturer of the micro-controller, we are expected to make this on our own)

1.CPU – Central Processing Unit

This is the "Brain" of our micro-controller. The Atmega328 micro-controller CPU will perform task based in "Instruction Sets" on either the memory, arithmetic, logic, control, Input or Output pins.

2.System Clock – Oscillator

The CPU are able to do many task, however some task needs to be performed at a specific time interval. Time interval requires a "Clock", in micro-controller it is known as "System Clock". A "System Clock" can be made from a device that can perform "state change" at a fixed duration, known as "Oscillator".

The Atmega328 micro-controller has "System Clock" built inside the chip. However, the Atmega328 micro-controller can be attached to external device for its "System Clock". The Atmega328 micro-controller that comes with the Arduino Uno board, is connected to a 16Mhz Crystal Oscillator on the Arduino Uno board

3.Memory

Memory inside the micro-controller is consist of many individual "Memory Device", can be in the thousands, millions or even billions. Each "Memory Device" is a physical device that is made from material that can hold two distinct states (for example: Electrical ON/OFF, Magnetic NORTH/SOUTH, and etc.). In general, the active state in each "Memory Device" is represented by either "1" or "0".

The "state" of each "Memory Device" can be retrieved or changed.

4.Input & Output Interface

These are the **exposed metal parts** of the physical micro-controller chip, known as "pins". The micro-controller "pins" are like wires, **one end** is physically connected to the micro-controllers internal circuit inside the chip package, while the **other end** (the exposed part), will be connected to external device.

When external device is connected to the "pin", the device become a part of the micro-controllers internal electrical circuit. There will be an exchange of eletrical elements (Voltage, Current and Resistance) on the "pins" triggering various reactions between the internal circuit and external device, performing task.

5.Instruction Set

The micro-controller CPU will perform a specfic task when a specific instruction code is introduced to it. A CPU can only recognize a fixed number of instruction codes, collectively known as "Instruction Set". Different machine uses different "Instruction Set", for example the Atmega328 micro-controller is designed to use the "AVR Instruction Set". Other popular "Instruction Set" are like "ARM Instruction Set", "x86 Instruction Set"...

6.Program (not supplied by the manufacturer of the micro-controller, we are expected to make this on our own)

Although the Atmega328 micro-controller CPU will execute **"AVR Instruction Set"**, each instruction set can only perform a very simple task.

To perform a complex task, we need many simple task combined. Atmega328 micro-controller **"Program" is a compilation of many AVR Instruction sets, arranged in a specific sequence**, so that the CPU can perform complex task together with the other components connected to the micro-controller.

Very often, we do not see the "Instruction sets" in our "Program" because they are **hidden from us by our Programming Language, the C-Language.**

The Atmega328 micro-controller “Memory” is consist of many “individual Memory Device” all packed inside the micro-controller chip packaging.

Each physical “Memory Device” is made from material or combination of materials that can hold two distinct states. The active state of each “Memory Device” is represented by either “0” or “1”.

The entire “Memory” in the micro-controller can be logically “visualized” as a long sequence of “0”s and “1”, something like the following...

1100110010000110101010101010110101010101010101001110011010000010000000111...
and so on, until whatever physically available.

The each of individal Memory Device that has either “0” or “1” is known as a BIT

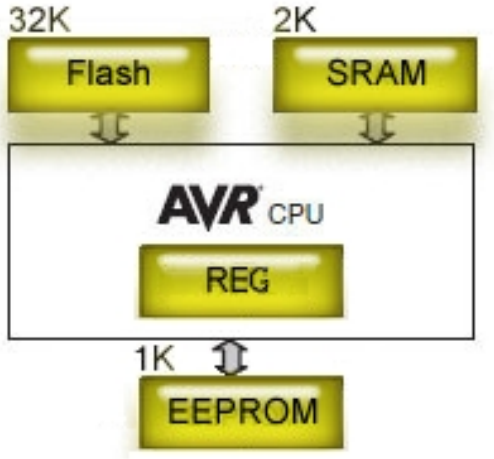
There are 4 separate Memory inside the in the Atmega328 micro-controller chip package for different use.

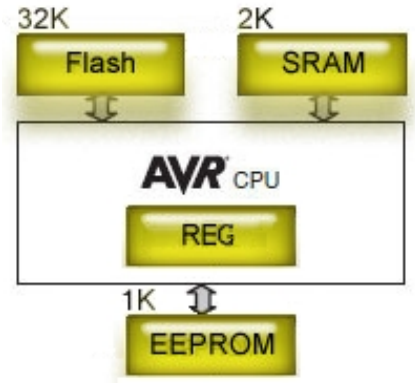
1.Program Memory – Flash Memory

2.CPU Memory – REGISTER

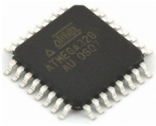
3.Working Memory – SRAM

4.User Storage Memory – EEPROM





There are 4 separate Memory inside the in the Atmega328 micro-controller chip package for different use.



1.Program Memory – Flash Memory

2.CPU Memory – REGISTER

3.Working Memory – SRAM

4.User Storage Memory – EEPROM

1.Program Memory – Flash Memory

This Memory is specially used to store our Program. The micro-controller CPU will automatically execute the Program stored in this Memory whenever the micro-controller is turned-on.

The “Program Memory” memory is “permanent”, when power supply is turned-off, whatever stored in this Memory will not be lost. It will still be there when the power is turned-on again.

2.CPU Memory – REGISTER

This Memory is part of the CPU. CPU Memory is broken down into many blocks of multiple bits. Each block of the CPU Memory is known as a “REGISTER” with a fixed number of bits. The usage of each REGISTER is already fixed.

In Atmega328 micro-controller, each “REGISTER” are mostly consist of 8-bits (That is why the Atmega328 micro-controller is known as an 8-bit micro-controller)

We can only do two things with the Register Memory,
1. We **change** specific bit(s) inside a "REGISTER" in order to trigger some kind of response from the CPU. The CPU may also change a specific bit(s) inside a “REGISTER” for to be pick up by other components.
2. We **retrieve** specific bit(s) from a "REGISTER", each bit or combination of bits in the REGISTER, they indicators of something in the micro-controller.

The “CPU Memory” are “semi-permanent”. Whatever values we put in will be lost when the micro-controller is turned-off. Some values in the Register Memory are “permanent”, in the sense that when micro-controller is turned-on or being “Reset” bits inside all the REGISTER will be set to default values.

3.Working Memory – SRAM

This memory is used in our Program to hold temporary working data to be used at various points of our Program. We can read and change the bit(s) in this Memory in any way we wish from our Program. However, we can only read and change in the smallest block of 8-bits (also known as 1-byte).

The “Working Memory” (**also known as as “SRAM”**) memory is “non-permanent”. This means, when power supply is turned-off, whatever kept in this Memory will be lost. When power is turned-on again, whatever previously there in the “Working Memory” will not be there anymore. We work with new values all over again.

4.User Storage Memory – EEPROM Memory

This memory is used to store permanent information which we intend to read or change in the future. We can read and change the bit(s) in this Memory in any way we wish from our Program. However, we can only read and change in the smallest block of 8-bits (also known as 1-byte).

The “Storage Memory” (**also known as EEPROM**) memory is “permanent”. This means, when power supply is turned-off, whatever stored in this Memory will still be there. When power is turned-on again, we can still read and change whatever kept there previously.

Use the Arduino IDE Software
1.Save as "blink_arduino"
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = blink_arduino
void setup() {
  pinMode(13, OUTPUT);
}
void loop() {
  digitalWrite(13, HIGH);
  delay(500);
  digitalWrite(13, LOW);
  delay(500);
}
```

Watch the LED on Arduino Uno board

1.Program Memory – Flash Memory

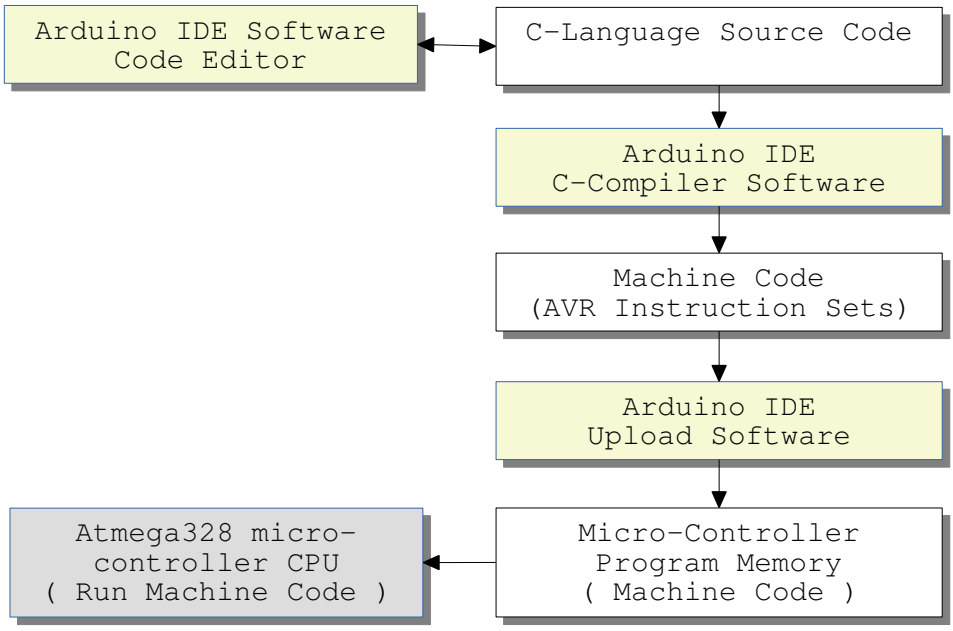
The code inside the Arduino IDE Software is full of "texts" and "symbols". They are entered in specific **sequence, text and symbols and etc, folowing a set of rules (of a Programming Language)**. In this example, our Program is using the **C-Language** rules. The **"text" and "symbols"** that we see on our Arduino IDE Software Code Editor is known as **"C-Language Source Code"**

Memory can only hold "0"s and "1"s, "C-Language Source Code" are not "0"s and "1"s. Our "C-Language Source Code" has to be converted into "0"s and "1"s before it can stored in our Program Memory.

The process of converting instruction codes in "C-Language Source Code" into instruction codes in "1"s and "0"s is called "Compiling". This is done by using a software known as **"C-Compiler"**, which is specially built to **convert "C-Language Source Code"** into instruction codes in "0"s and "1"s.

The instruction codes in "1"s and "0"s is known as **"Machine Code"**

"Machine Code" is the one that gets uploaded and **kept in Program Memory**, and will be **executed by the micro-controller CPU** each time we turn-on our micro-controller



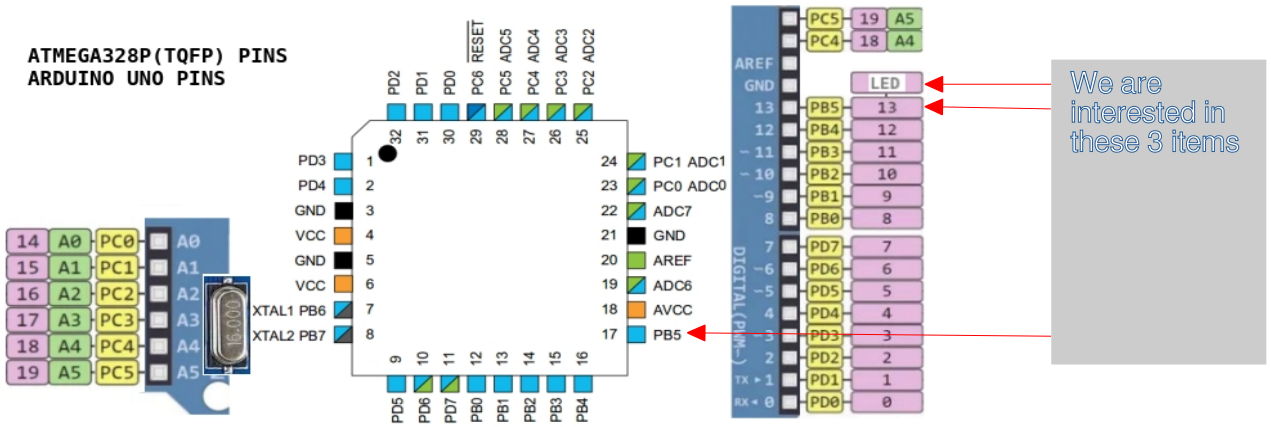
Only one Program can be kept in the **Program Memory**, each time we upload, it will be replaced. This memory however, is permanent.. meaning as long as we did not do a new upload, whatever Program stored there will always be there.

2.CPU Memory – REGISTER

Below is a snapshot of datasheet supplied by the manufacturer of Atmega328 micro-controller, list of "REGISTER" and their bits

ATmega48A/48PA/88A/88PA/168A/168PA/328/328P

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–
0x0 (0x20)	Reserved	–	–	–	–	–	–	–	–



1. The Atmega328 micro-controller **"pin"** labelled as **"PB5"** is connected to **"connector 13"** on the Arduino Uno board

2. The **"connector 13"** on the Arduino Uno board is connected to the **LED** on the Arduino Uno board

According to the REGISTER datasheet above, the **behaviour of the "pin"** labelled as **PB5 on the Atmega328 micro-controller** depends on the state of **Bit-5** in the **REGISTER named DDRB, PORTB and PINB**

Use the Arduino IDE Software
1.Save as "blink_register"
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = blink register
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
int main() {
  DDRB |= 0b00100000;;
  while(1) {
    PORTB |= 0b00100000;;
    _delay_ms(500);
    PORTB &= ~0b00100000;
    _delay_ms(500);
  }
  return 0;
}
```

Watch the LED on Arduino Uno board

```
// Program: t2_blink register
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
int main() {
  DDRB |= 0b00100000;;
  while(1) {
    PORTB |= 0b00100000;;
    _delay_ms(500);
    PORTB &= ~0b00100000;
    _delay_ms(500);
  }
  return 0;
}
```

```
// Program: t2_blink_arduino
void setup() {
  pinMode(13, OUTPUT);
}
void loop() {
  digitalWrite(13, HIGH);
  delay(500);
  digitalWrite(13, LOW);
  delay(500);
}
```

The program “blink_arduino” is what we normally see when people show us Arduino. “blink_arduino” is a C-Language Program coded with the Arduino Library.

The Arduino Library is a collection of C-Language Program codes written by other people. We add them into our Program to simplify our coding. For example, in the “blink_arduino” we have the code “pinMode(13,OUTPUT);” which is exactly the same as “DDRB |= 0b00100000;” which is harder to code.

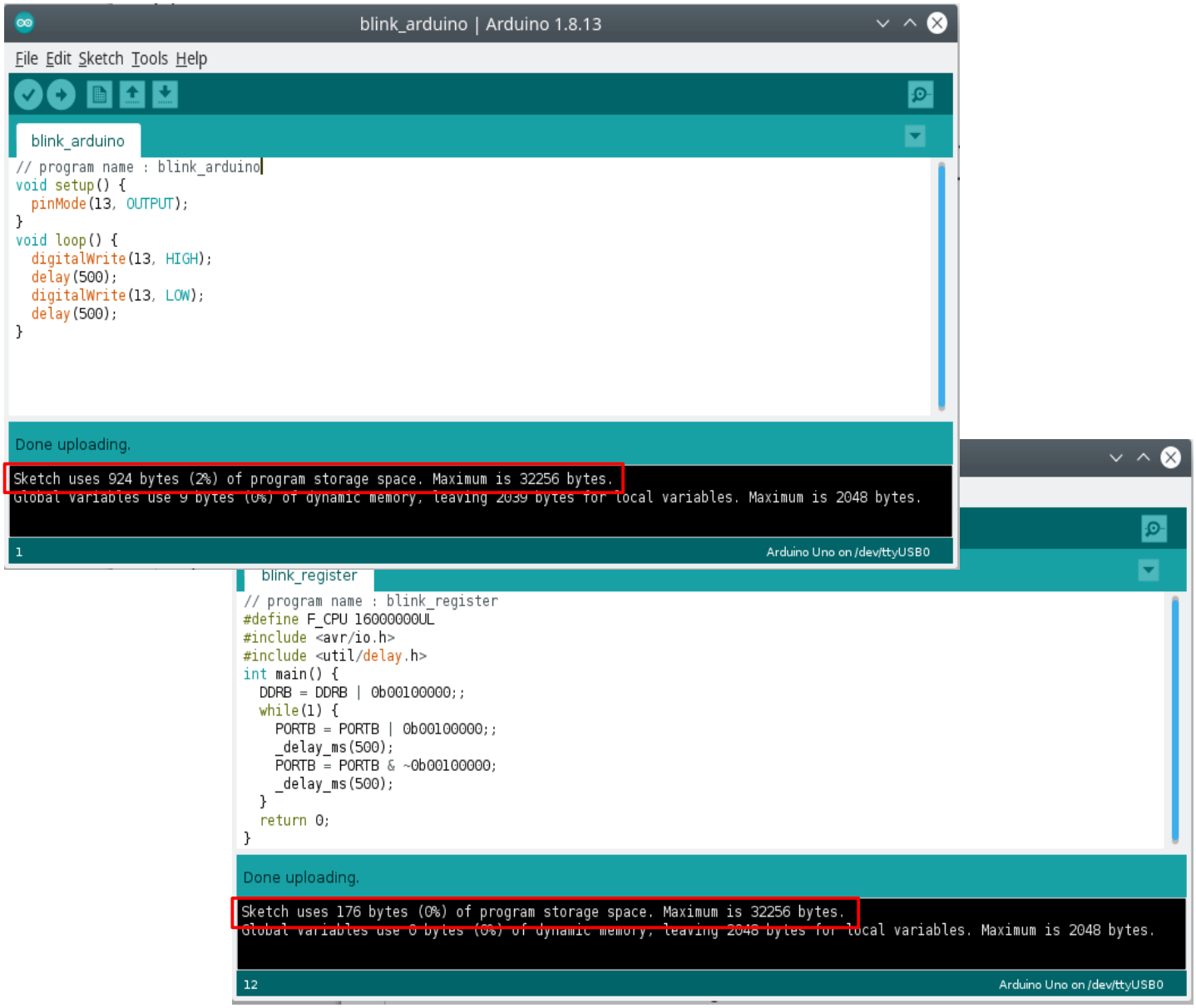
The program “blink_register” is where we work with the REGISTER directly with C-Language without using Arduino Libary.

The Arduino Library can be found in the following folder, **<installation folder>/hardware/arduino/avr/cores/arduino**. There are files in there, **make sure you do not change any of them**, just look at them.

If you wish to see pinMode(13, OUTPUT),digitalWrite(13, HIGH),digitalRead(12) from the Arduino Library, you can find them in the “wiring_digital.c” file in folder mentioned above. You can use any text editor to open the files.

If the Arduino Library and other Libraries make it so easy to Program the Atmega328 micro-controller, then why sometimes we avoid using the Libraries ?

Lets compare these two Program below, by just look at the size of “Machine Code” generated. First Program is using Arduino Library and the second Program using direct Register Memory with our own codes.



As you can see, there is a big difference in “Machine Code” size the generated by the C-Compiler for both Program that does the same task

924 bytes (with Arduino Libraries) vs 176 bytes (without Arduino Libraries)

In this example it is just the “Machine Code” size, there are many other reasons why sometimes we cant use functions from Libraries.

- There will be many occasion where Libraries does not exist for certain devices, we have to make our own code.
- Libraries tends to be for general purpose use, they are not optimized and not good enough, we want to make our own more efficient code.

Using Arduino Library is very much easier. So, to get best of both world, we mix them up. We can use some of the functions from the Arduino Library together with our own codes.

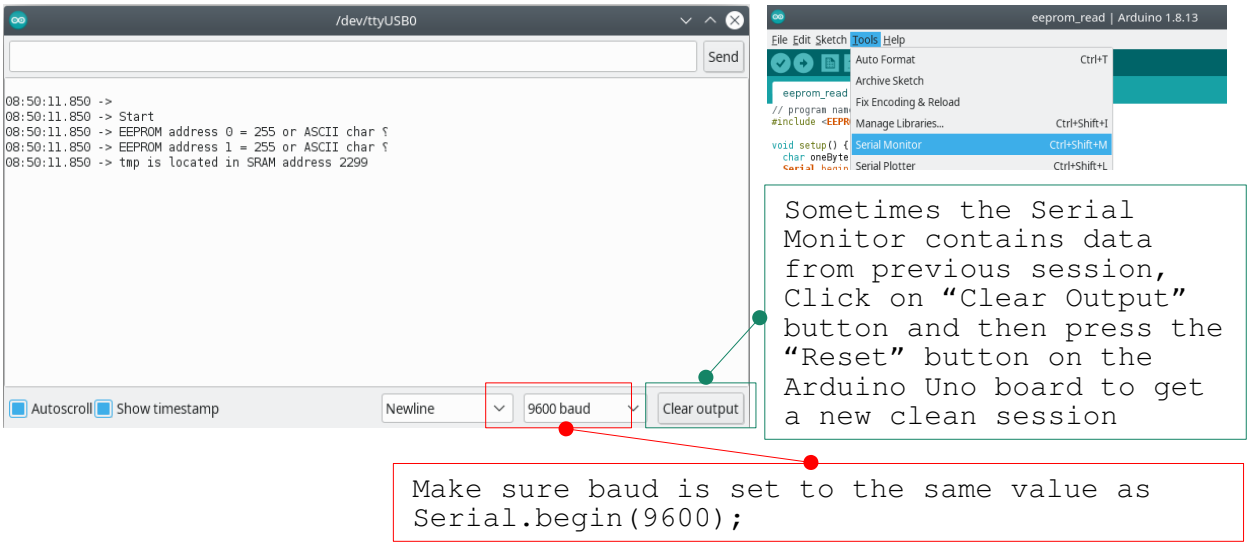
3.Working Memory – SRAM Memory
4.User Storage Memory – EEPROM Memory

Use the Arduino IDE Software
1.Save as “t2 eeprom read”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// Program: t2_eeprom_read
#include <EEPROM.h>

void setup() {
uint8_t tmp;

  Serial.begin(9600);
  Serial.print("\n\nStart");

  tmp = EEPROM.read(0);
  Serial.print("\nEEPROM address 0 = "); Serial.print(tmp);
  Serial.print(" or ASCII char "); Serial.print((char) tmp);

  tmp = EEPROM.read(1);
  Serial.print("\nEEPROM address 1 = "); Serial.print(tmp);
  Serial.print(" or ASCII char "); Serial.print((char) tmp);

  Serial.print("\ntmp is located in SRAM address ");
  Serial.print( (uint16_t) &tmp);
}

void loop() { }
```

This program does not use any LED or other external devices that we can see. Everything is in the Memory. In order to “see” what is inside the “Memory” we will sent the information to a facility in Arduino IDE Software, the “Serial Monitor”

From Arduino IDE Software menu, select **“Tools | Serial Monitor”**



3.Working Memory – SRAM Memory
4.User Storage Memory – EEPROM Memory

```
// Program: t2_eeprom_read
#include <EEPROM.h>

void setup() {
uint8_t tmp;

  Serial.begin(9600);
  Serial.print("\n\nStart");

  tmp = EEPROM.read(0);
  Serial.print("\nEEPROM adddress 0 = "); Serial.print(tmp);
  Serial.print(" or ASCII char "); Serial.print((char) tmp);

  tmp = EEPROM.read(1);
  Serial.print("\nEEPROM address 1 = "); Serial.print(tmp);
  Serial.print(" or ASCII char "); Serial.print((char) tmp);

  Serial.print("\ntmp is located in SRAM address ");
  Serial.print( (uint16_t) &tmp);
}

void loop() { }
```

uint8_t tmp;

"uint8_t" is telling the CPU to reserve an 8-bit memory block and name that memory block as **"tmp"**.

tmp is the name of the reserved Memory Block from the **SRAM Memory** and there are 3 things we can do with it.
1. **change** data stored in "tmp" SRAM Memory block
2. **read** the data stored in "tmp" SRAM Memory block
3. get the **address (location)** of "tmp" in the SRAM Memory

tmp = EEPROM.read(0);
We are reading 1-byte from the EEPROM Memory address 0 and **change the data of tmp** with it. The equal **"="** symbol is the code for assignment.

Serial.print(tmp);
To **read** a reserved SRAM Memory block named **"tmp"**, we simply specify the SRAM Memory name, **tmp**

Serial.print((uint8_t) &tmp);
The symbol **&** in front **tmp** will tell us where the **address of tmp in the SRAM Memory** (this is where the contents of tmp is located)

(char) tmp – converts value of tmp to char datatype
(uint16_t) &tmp – converts address of tmp to unsigned integer 16-bits
(memory address in atmega328 is stored in 16-bits)

If this is a new chip, tmp will receive the value of 255 (that is the default value for a new unused EEPROM Memory).

Upload the next "t2_eeprom_write" program (to put data into EEPROM) and run this program again to see what we put in there with the "t2_eeprom_write" program.

3.Working Memory – SRAM Memory
4.User Storage Memory – EEPROM Memory

Use the Arduino IDE Software
1.Save as "t2 eeprom write"
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// Program: t2_eeprom_write
#include <EEPROM.h>

void setup() {
uint8_t tmp;

  Serial.begin(9600);
  Serial.print("\nStart...");

  tmp = 65;
  EEPROM.write(0, tmp);
  Serial.print("\n\nchange data at Address 0 of EEPROM to 65");
  tmp = 66;
  EEPROM.write(1,tmp);
  Serial.print("\n\nchange data at Address 1 of EEPROM to 66");
}

void loop() { }
```

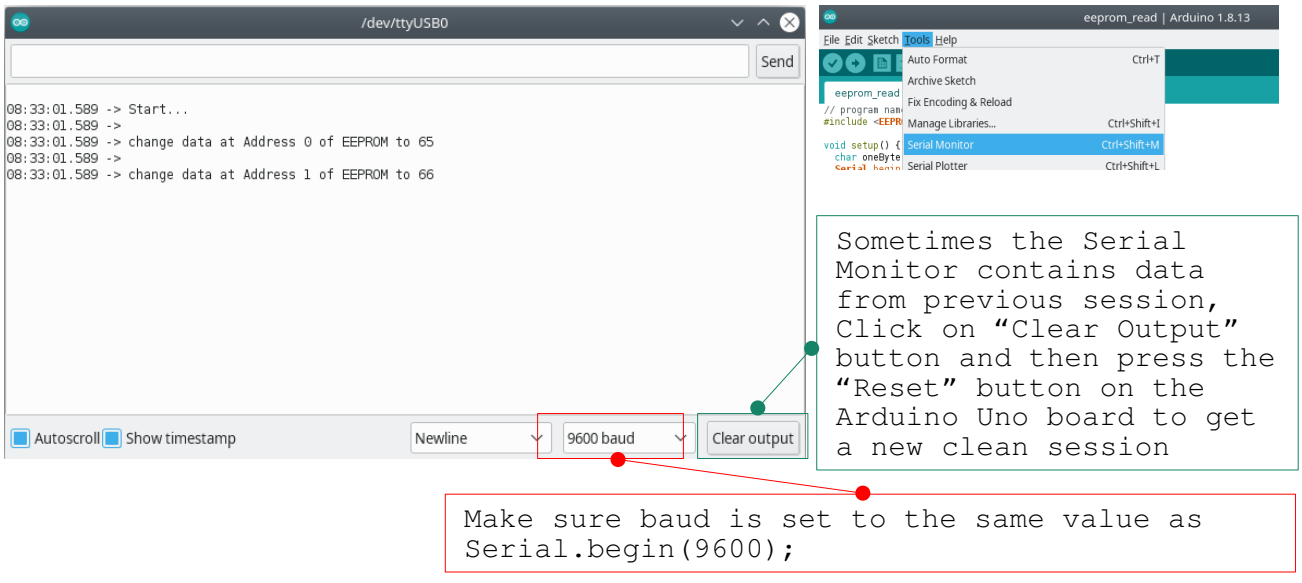
This program does not use any LED or other external devices that we can see. Everything is in the Memory. In order to "see" what is inside the "Memory" we will sent the information to a facility in Arduino IDE Software, the "Serial Monitor"

WARNING!!!

EEPROM has 100,000 erase/write cycle limit. That is alot for normal usage. Our only fear is Programming Errors. where we can have a Program Code that does repeated erase/write to the EEPROM that can hit above 100,000 cycles in seconds. Be very careful with "**EEPROM.write()**"

There are no retrieval limits for EEPROM, we can read from EEPROM as many times as we wish

From Arduino IDE Software menu, select **"Tools | Serial Monitor"**



3.Working Memory – SRAM
4.User Storage Memory – EEPROM

In the previous example Program, we read from the EEPROM and load data from the EEPROM into the SRAM. The data from EEPROM depends on what stored there previously, it could be empty on first time usage.

The EEPROM storage is permanent, will retain the data stored there until it is being replaced.

This this example, we will change the EEPROM with new data. Once this is done, if we were to run the previous example Program again, we will see this new data being read.

Use the Arduino IDE Software
1.Save as "t2_eeprom_write"
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// Program: t2_eeprom_write
#include <EEPROM.h>

void setup() {
uint8_t tmp;

  Serial.begin(9600);
  Serial.print("\nStart...");

  tmp = 65;
  EEPROM.write(0, tmp);
  Serial.print("\n\nchange data at Address 0 of EEPROM to 65");
  tmp = 66;
  EEPROM.write(1,tmp);
  Serial.print("\n\nchange data at Address 1 of EEPROM to 66");
}

void loop() { }
```

Use the "Serial Monitor" to see output

tmp = 65;
EEPROM.write(0, tmp);
This code will take the value from the "tmp" SRAM Memory 65 and place it into the EEPROM, starting from EEPROM address 0.

tmp = 66;
EEPROM.write(1, tmp);
This code will take the value from the "tmp" SRAM Memory 66 and place it into the EEPROM, starting from EEPROM address 1.

Each address position inside the EEPROM is 8-bits, starting from address 0 until whatever that is available.

Theatmega328 has 1024 bytes of EEPROM, meaning address 0 to address 1023

Upload the previous "t2_eeprom_read" Program again.

The "t2_eeprom_read" Program will be able to retrieve the newly updated value inside the EEPROM Memory, address 0 and address 1 of the EEPROM.

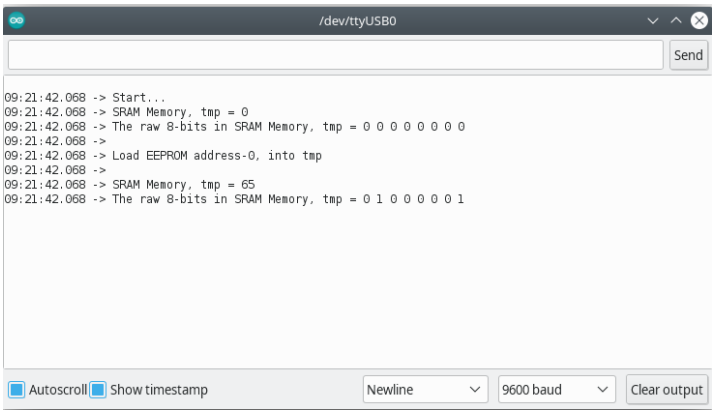
3.Working Memory – SRAM
4.User Storage Memory – EEPROM

In the previous Program, we saw decimal numbers and alphabets being displayed. As we know, Memory can only store “1”s and “0”s. Decimal numbers and alphabets taht we sare are results of a “conversion” done for us when we specify a “datatype” in our program (unit8_t, char, and etc...)

Use the Arduino IDE Software
1.Save as “t2 eeprom read bits”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// Program: t2_eeprom_read_bits
#include <EEPROM.h>

void setup() {
uint8_t tmp = 0;

  Serial.begin(9600);
  Serial.print("\nStart...");

  Serial.print("\nSRAM Memory, tmp = "); Serial.print(tmp);
  Serial.print("\nThe raw 8-bits in SRAM Memory, tmp = ");

  for (int i=7; i>=0; i--) {
    Serial.print((tmp >> i) & 0x1);Serial.print(" ");
  }

  tmp = EEPROM.read(0);
  Serial.print("\n\nLoad EEPROM address-0, into tmp\n");

  Serial.print("\nSRAM Memory, tmp = "); Serial.print(tmp);
  Serial.print("\nThe raw 8-bits in SRAM Memory, tmp = ");

  for (int i=7; i>=0; i--) {
    Serial.print((tmp >> i) & 0x1);Serial.print(" ");
  }
}
void loop() { }
```

Use the “Serial Monitor” to see output



The “1”s and ”0”s is the representation of ON/OFF, that is the representation of the actual physical storage in any Computer Memory.

If you have a very powerful “micro-scope”, you can actually see the physical formation on the physiscal Memory, represented by 1s and 0s.

The uint8_t is an integer datatype, which does automatic conversion from the binary 1s and 0s into a more familiar decimal numbering system.