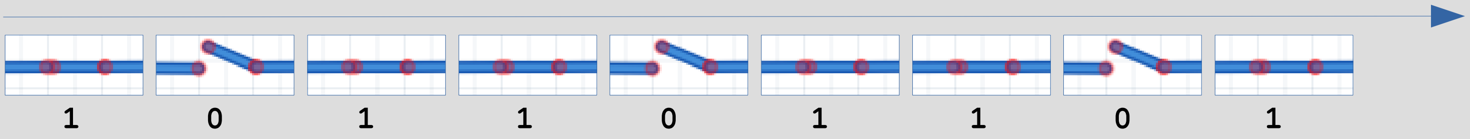
MEMORY

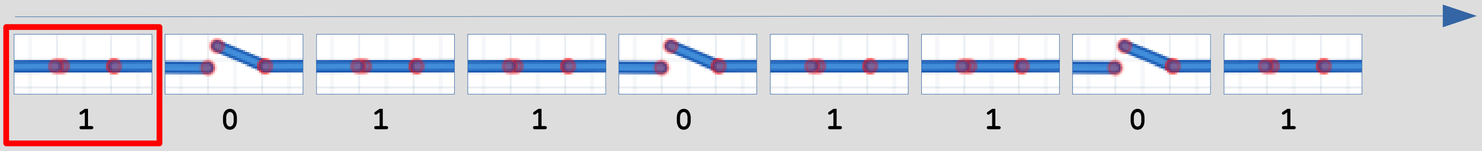
Inside the ATMEGA328 micro-controller chip, is a massive electrical circuit. Bulk of the massive circuit are “switches” made from different types of very tiny transistors. They are mostly used as **“MEMORY”, to store information**



We can visualize the MEMORY as a long sequence of “switches” that can be turned ON or OFF (each represented by 1 or 0)
A Sequence of 1 and 0 in a specific order, represents a piece of information

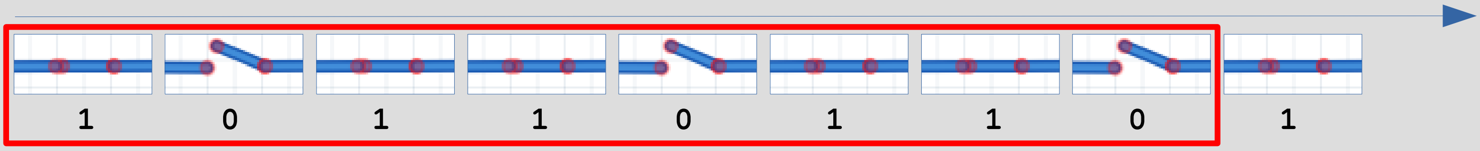
What are BIT and BYTE ?

BIT



Each of the **physical “switch” in the “MEMORY”** known as **a “BIT”**
Each “BIT” can be either **ON(represented by 1)** or **OFF(represented by 0)**

BYTE



A Group of **8-BITS** in sequence is known as **a “BYTE”**

Types of MEMORY in the ATMEGA328P micro-controller

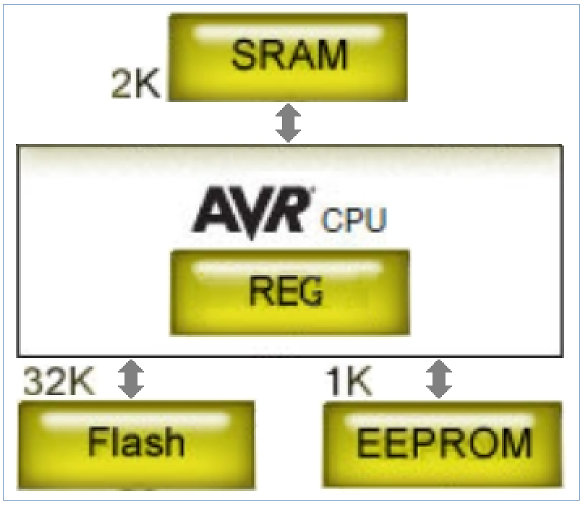
There are 4 TYPES of MEMORY inside the ATMEGA328P micro-controller chip

- **CPU MEMORY (REGISTERS,** our Regular Computer have this same thing)

- **WORKING MEMORY (SRAM,** similar to our Regular Computer RAM, but is inside the chip)

- **PROGRAM MEMORY (FLASH,** similar to our Regular Computer SSD/HARDDISK, but is inside the chip, for permanent storage of PROGRAM)

- **STORAGE MEMORY (EEPROM,** similar to our Regular Computer SSD/HARDDISK, but is inside the chip, for permanent storage of ANY DATA)



2K, 32K and 1K are in BYTE

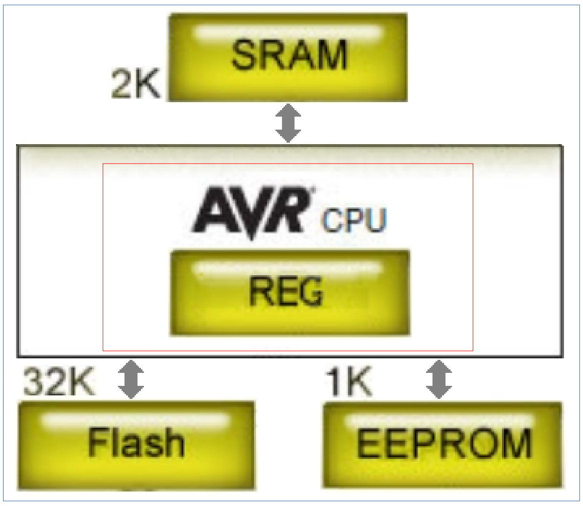
ATMEGA328/Arduino Uno – CPU MEMORY (REGISTER) –
https://github.com/teaksoon/lmaewapm

– CPU MEMORY (REGISTERS, our Regular Computer have this same thing **)**

This "MEMORY" is part of the micro-controller CPU (Central Processing Unit)

This MEMORY can be visualized as a long sequence of BITS. This long sequence of BITS has already been broken into many "BLOCKS of multiple BITS". Each "BLOCK" is known as a "REGISTER"(refer Datasheet for each of them)

The number of BITS in each "REGISTER" can differ. For ATMEGA328P micro-controller, the REGISTERS are mostly 8-BITS. That is why the ATMEGA328P is also known as an 8-BIT MICRO-CONTROLLER



2K, 32K and 1K are in BYTE

The CPU MEMORY (REGISTER) are special, when they are changed they will normally cause the micro-controller perform a task. We can read or update the REGISTERS from our PROGRAM

In our earlier tutorial topic, you probably have seen the following code,

SBI 0x04, 5
"***SBI***" is an insruction from the **AVR INSTRUCTION SET**
"***0x04***" is the address of **an 8-BIT REGISTER (DDRB) from the CPU MEMORY**

When the CPU sees the code "***SBI 0x04, 5***" , it will **turn** the **5th BIT of the DDRB REGISTER to 1. When the 5th bit of DDRB REGISTER in the CPU MEMORY is 1,** the Arduino Pin 13(PB5) will become an **OUTPUT Pin**

NOTE: "SBI" instruction can only turn a BIT to 1, if you wish to turn the 5th bit of DDRB REGISTER to 0, we use a different instruction from the "AVR INSTRUCTION SET" on the same REGISTER. See the following line of code,

CBI 0x04, 5
"***CBI***" is an instruction from the **AVR INSTRUCTION SET**
"***0x04***" is the address of **an 8-BIT REGISTER (DDRB) from the CPU MEMORY**

When the CPU sees the code "***CBI 0x04, 5***" , it will **turn** the **5th BIT of the DDRB REGISTER to 0. When the 5th bit of DDRB REGISTER in the CPU MEMORY is 0,** the Arduino Pin 13(PB5) will become an **INPUT Pin**

Refer to the ATMEGA328P datasheet for the rest of the REGISTER and what each of its BITS is used for

---

If we do not wish to use the "AVR INSTRUCTION SET", we can also manipulate the CPU MEMORY (REGISTER) by using the C-Language PROGRAM Code,

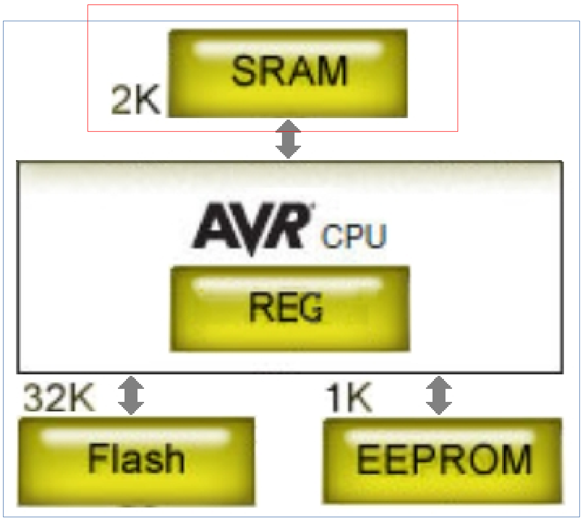
DDRB = DDRB | (1<<5);
C-Language PROGRAM code ***DDRB = DDRB | (1<<5);*** // same as ***SBI 0x04, 5***

DDRB = DDRB & ~(1<<5);
C-Language PROGRAM code ***DDRB = DDRB & ~(1<<5);*** // same as ***CBI 0x04, 5***

– WORKING MEMORY (SRAM, similar to our Regular Computer RAM, but is inside the chip **)**

This MEMORY can be visualized as a long sequence of BITS

The “WORKING MEMORY” is for us to use in our PROGRAM as temporary working storage



2K, 32K and 1K are in BYTE

To use the WORKING MEMORY (SRAM) in our PROGRAM, we will need “Reserve” MEMORY in “BLOCKS of multiple BITS” from the “WORKING MEMORY” with our PROGRAM

The number of BITS in each “Reserved MEMORY” will be determined by the specified “datatype” at the time when we “Reserve” the MEMORY. Each “Reserved MEMORY” will also be given a name, so that we can access to each of them

ATMEGA328P C-Language basic “datatype”
char   = 8-BIT
int   = 16-BIT
long   = 32-BIT
float = 32-BIT
double = 32-BIT (in ATMEGA328P, this is same as float)

The “Reserved MEMORY” is known as either “variable” or “constant”

“variable” – the BITS can be **assigned with initial value** and the initial **value can be changed**

“constant” – the BITS can be **assigned with initial value** and the initial **value cannot be changed**

Code to “Reserve” MEMORY from the “WORKING MEMORY” (known as “variable/constant declaration”) requires 4 parts,
Part1:const Keyword
Part2:datatype
Part3:name
Part4:assign initial value

Part1:const Keyword, followed by a space to “Reserve” a “constant MEMORY”. If we wish to “Reserve” a “variable MEMORY”, simply leave this Part blank

Part2:datatype (char, int, long, float, double or others...), followed by a space

Part3:name

Part4:assign initial value (any numbers) This example value is in binary, just to to show all the bits (“int” has 16-BITS)

```
const int var_name = 0b0000000000000010;
```

```
const int var_name = 2;
```

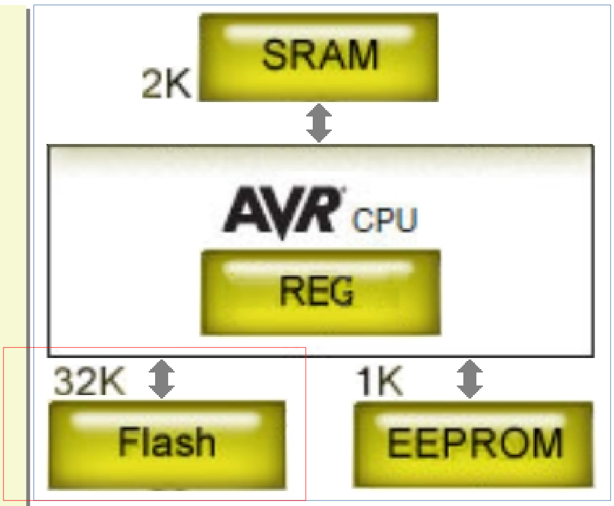
in equivalent decimal number

ATMEGA328/Arduino Uno – PROGRAM MEMORY (FLASH) – https://github.com/teaksoon/lmaewapm

– PROGRAM MEMORY (FLASH, similar to our Regular Computer SSD/HARDDISK, but is inside the chip, for permanent storage of PROGRAM **)**

This MEMORY can be visualized as a long sequence of BITS, The sequence of BITS has already been broken into many BLOCKS of 8-BIT(BYTE)

This "MEMORY" is for us to store our PROGRAM permanently. Even when the micro-controller is powered-off, whatever stored in this memory remains



2K, 32K and 1K are in BYTE

ATMEGA328 PROGRAM MEMORY, Generally we do not care much about the PROGRAM MEMORY apart from monitoring our PROGRAM size so that it does not exceed what is physically available.

We normally just use this MEMORY to store our PROGRAM

To use this MEMORY, all we need to do is: Click "Upload" from our Arduino IDE Sofware, our PROGRAM will be automatically stored into the PROGRAM MEMORY. Our PROGRAM will stay there permanently until it is being replaced by our next "Upload"

PROGRAM MEMORY is a FLASH Memory, it has 10,000 "write-cycle" limit, meaning if this memory is "changed" for more than 10,000 times, new update into this MEMORY can become un-reliable. Since we only Upload PROGRAM into it, very unlikely we will need to upload more than 10,000 times

There is no limit on now many times we can read from this MEMORY

Apart from **our PROGRAM,** the PROGRAM MEMORY can also store another optional PROGRAM known as **"BOOTLOADER"**

- If a "BOOTLOADER" PROGRAM exist, the CPU will run the "BOOTLOADER" PROGRAM first when it is powered-up. Once the "BOOTLOADER" completes, the CPU will continue to run our PROGRAM

- If a "BOOTLOADER" PROGRAM does not exist, the CPU will run our PROGRAM immediately when it is powered-up

When we use the Arduino Uno Board, a "BOOTLOADER" PROGRAM is already exist in our ATMEGA328P micro-controller PROGRAM MEMORY (most likely is the "Optiboot" PROGRAM). The Arduino IDE Software needs to use the "BOOTLOADER" for the "Upload" process when using the USB connection

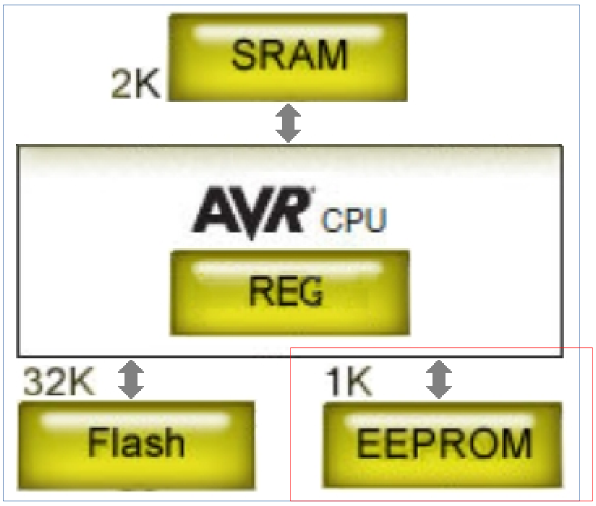
NOTE:
If we wish to use the PROGRAM MEMORY as WORKING MEMORY, we can still do it from our C-Language PROGRAM. However, we normally avoid doing that because of the 10,000 "write-cycle" limit

ATMEGA328/Arduino Uno – STORAGE MEMORY (EEPROM) –
https://github.com/teaksoon/lmaewapm

– STORAGE MEMORY (EEPROM, similar to our Regular Computer SSD/HARDDISK, but is inside the chip, for permanent storage of ANY DATA **)**

This MEMORY can be visualized as a long sequence of BITS. The sequence of BITS has already been broken into many BLOCKS of 8-BIT(BYTE)

This “MEMORY” is for us to store ANY DATA permanently. Even when the micro-controller is powered-off, whatever stored in this memory remains



2K, 32K and 1K are in BYTE

The STORAGE MEMORY (EEPROM) can be manipulated based on each BYTE index position, BYTE-0, BYTE-1, BYTE-2 and so on...

We can update this MEMORY from our C-Language PROGRAM

The EEPROM has a 100,000 “write-cycle” limit, meaning if this memory is “changed” for more than 100,000 times, new update into this MEMORY can become un-reliable.

There is no limit on now many times we can read from this MEMORY