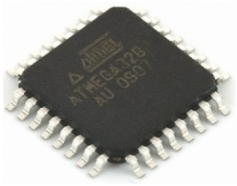
PROGRAM are INSTRUCTIONS from Programmer(Human) to the micro-controller, asking the micro-controller to do task



The micro-controller can only do task according to what it is designed to do

The manufacturer of the micro-controller will prepare a list of task that the micro-controller can perform, the list is known as

"INSTRUCTION SET"

The "INSTRUCTION SET" is consist of many individual "INSTRUCTION" where each "INSTRUCTION" will tell the micro-controller to perform a simple task

The **ATMEGA328 micro-controller** is using the **"AVR INSTRUCTION SET"**,

The picture below (from ATMEGA328P datasheet) shows some of the "INSTRUCTION" from the "AVR INSTRUCTION SET"

31. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
Arithmetic and Logic Instructions					
ADD	Rd, Rr	Add two registers	Rd ← Rd + Rr	Z,C,N,VH	1
ADC	Rd, Rr	Add with carry two registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl, K	Add immediate to word	Rdh: Rdl ← Rdh: Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two registers	Rd ← Rd – Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract constant from register	Rd ← Rd – K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with carry two registers	Rd ← Rd – Rr – C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with carry constant from reg.	Rd ← Rd – K – C	Z,C,N,V,H	1
SBIW	Rdl, K	Subtract immediate from word	Rdh: Rdl ← Rdh: Rdl – K	Z,C,N,V,S	2

There are other types of "INSTRUCTION SET" used by different micro-controllers. Some of these other "INSTRUCTION SET" types you probably have heard of, "ARM", "x86", "RISC-V",etc...

As you can see from the "INSTRUCTION SET" list above, sending a single "INSTRUCTION" to the micro-controller will not be able to do anything useful for us

We will need to combine "many" individual "INSTRUCTION" from the "INSTRUCTION SET", arrange them in a particular order before the micro-controller can perform something useful for us

The combined multiple "INSTRUCTIONS" from the "INSTRUCTION SET" that is arrange in a particular order for the micro-controller to perform various task is known as a **"PROGRAM"**

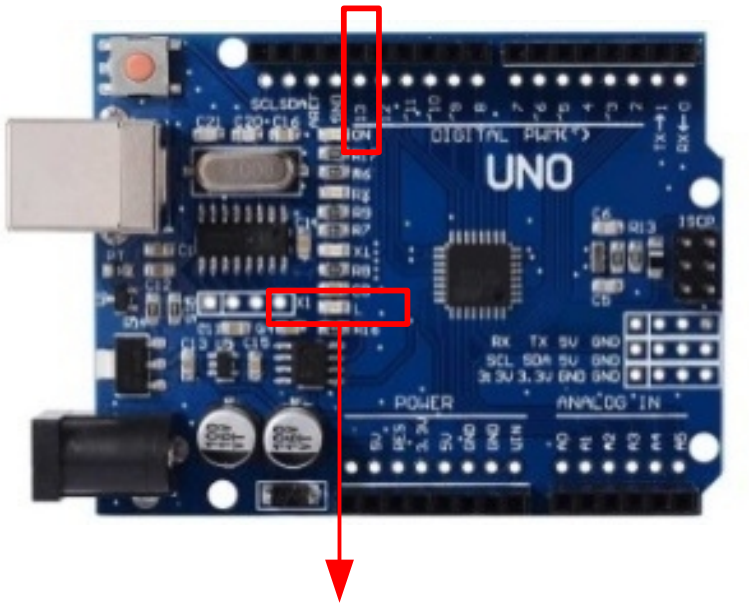
The are many ways to write PROGRAM for the ATMEGA328P micro-controller. We will use the Arduino IDE Software.

The Arduino IDE Software allows us to code our PROGRAM in three different styles (Programming Language)

1. AVR Assembly Language
2. C-Language with AVR Library
3. C-Language with Arduino Library

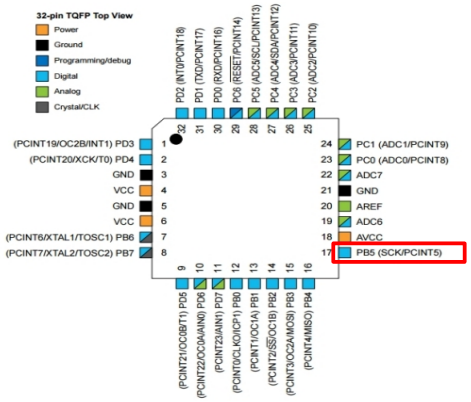
We will make two small PROGRAM to demonstrate the three different Programming styles

1. A PROGRAM to "Turn ON" the Arduino Uno Board "test LED"
2. A PROGRAM to "Blink" the Arduino Uno Board "test LED"



"test LED"
- it can be located somewhere else on the Arduino Uno Board, depending on the manufacturer of the Board

- **"test LED"** is also connected to **Arduno Uno Pin 13** and **ATMEGA328P micro-controller Pin PB5**



1. A PROGRAM to "Turn ON" the Arduino Uno Board "test LED" in three different Styles (1/2)

AVR Assembly Language
Download Source, use the Arduino IDE Software, Open **p_intro_asm** and Upload

```
.global main
main:
  SBI  0x04, 5   ; DDRB(0x04)  Arduino Pin 13(PB5) OUTPUT Pin
  SBI  0x05, 5   ; PORTB(0x05) Arduino Pin 13(PB5) to HIGH
  CALL forever
  RET
forever:
  RJMP forever
```

The Assembly Language uses INSTRUCTION from the AVR INSTRUCTION SET in almost every lines of code, "SBI","CALL","RET","RJMP". It also manipulates the CPU Memory DDRB(0x04) and PORTB(0x05) directly

Note: to turn off LED, change "SBI 0x05, 5" to "CBI 0x05,5"

C-Language with AVR Library
Download Source, use the Arduino IDE Software, Open **p_intro_avr** and Upload

```
#include "avr/io.h"
int main() {
  DDRB  = DDRB  | (1<<DDB5); // DDRB(0x04)  Arduino Pin 13(PB5) OUTPUT Pin
  PORTB = PORTB | (1<<PB5);  // PORTB(0x05) Arduino Pin 13(PB5) to HIGH
  forever();
  return 0;
}
void forever() {
  while(1){};
}
```

We do not see INSTRUCTION SET codes here anymore. They are hidden from us by the C-Language. However, we are still manipulating the CPU Memory DDRB, PORTB directly

Note: to turn off LED, change
"PORTB = PORTB | (1<<PB5);" to "PORTB = PORTB & ~(1<<PB5);"

C-Language with Arduino Library
Download Source, use the Arduino IDE Software, Open **p_intro_arduino** and Upload

```
void setup() {
  pinMode(13, OUTPUT);    // DDRB(0x04)  Arduino Pin 13(PB5) OUTPUT Pin
  digitalWrite(13, HIGH); // PORTB(0x05) Arduino Pin 13(PB5) to HIGH
}
void loop() {}
```

We do not see the INSTRUCTION SET codes and the CPU Memory DDRB and PORTB anymore. They are all hidden and replaced by the functions from the Arduino Library. The Program has become so much easier now

Note: to turn off LED, change
"digitalWrite(13,HIGH);" to "digitalWrite(13,LOW);"

1. A PROGRAM to "Turn ON" the Arduino Uno Board "test LED" in three different Styles (2/2)

AVR Assembly Language (p_intro_asm)

```
Done uploading.

Sketch uses 144 bytes (0%) of program storage space. Maximum is 32256 bytes.
Global variables use 0 bytes (0%) of dynamic memory, leaving 2048 bytes for local variables.
```

Pros:
We can access to the entire INSTRUCTION SET and CPU Memory directly. We are coding one INSTRUCTION SET at a time, manipulating CPU Memory directly gives us alot of flexibility. Our program will also be very small as we only use code what is required. In this example PROGRAM, it is only 144 bytes in size

Cons:
To code Assembly Language we need to know how the INSTRUCTION SET and micro-controller CPU Memory works. Since each line of Assembly code is one line of INSTRUCTION SET, our PROGRAM will contain alot of lines. It will become very hard to manage when the PROGRAM becomes bigger and more complicated

C-Language with AVR Library (p_intro_avr)

```
Done uploading.

Sketch uses 138 bytes (0%) of program storage space. Maximum is 32256 bytes.
Global variables use 0 bytes (0%) of dynamic memory, leaving 2048 bytes for local variables.
```

Pros:
The INSTRUCTION SET are hidden from us, a single line of C-Language Codes can perform multiple INSTRUCTION from the INSTRUCTION SET for us behind the scene. While the INSTRUCTION SET are hidden from us, we can still manipulate CPU Memory directly. We still have a very high degree of flexbility. In this example PROGRAM, it is just 138 bytes in size (even smaller than the Assembly Language)

Cons:
We still have to deal with micro-controller CPU Memory directly. Programming can still be difficult because manipulating CPU Memory requires a deep understanding on how the CPU Memory works

C-Language with Arduino Library (p_intro_arduino)

```
Done uploading.

Sketch uses 712 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables
```

Pros:
The INSTRUCTION SET and CPU Memory are totally hidden from us. We do not need to know anything about the INSTRUCTION SET and the CPU Memory. PROGRAM are now way easier to code. The same PROGRAM that we code can even be used with non AVR based micro-controllers as the PROGRAM codes are not tied to the ATMEGA328 INSTRUCTION SET or ATMEGA CPU Memory.

Cons:
Program size are now bloated. In this example PROGRAM, it is 712 bytes, which is about 5 times larger than the other 2 coding styles. PROGRAM will also run way slower than the other 2 coding styles because it has run alot of codes from Arduino Library. PROGRAM codes are not flexible anymore as we need to compy with rules from the Arduino Library

2. A PROGRAM to "Blink" the Arduino Uno Board "test LED" in three different Styles

The PROGRAM in three different styles for this example will not be posted in this tutorial. However, they are available for download. You can download them and look at them on your own

https://github.com/teaksoon/lmaewapm/blob/main/2021_11_20_program_intro_source.zip

Step 1: Download the file and unzip the source .zip file into the following folders

p_intro_blink_asm – Code in Assembly Language
p_intro_blink_avr – Code in "C-Language with AVR Library"
p_intro_blink_arduino – Code in "C-Language with Arduino Library"

Step 2: Use the Arduino IDE Software, open the file with .ino extension inside each of those folders

Step 3: Upload each of them from the Arduino IDE Software and watch the "test LED" on the Arduino Uno Board. You see the LED, turning ON and OFF every 250ms.

NOTE:
If you look at the Assembly Language Source code (p_intro_blink_asm), just to make a time delay of 250ms, took quite an effort. From that code alone, you will probably understand why we normally avoid using the Assembly Language and use the two C-Language Styles instead.

DO NOT WORRY IF YOU ARE LOST IN THIS TUTORIAL

This tutorial is to show you the three different styles that we can use when we use the Arduino IDE Software

Actually we have another Programming Style. That is, to mix all the three Programming Styles together and get the best of everything or might get confused instead

It is all up to us, we choose what is best for us and what is most suitable for our project

PROGRAM with Assembly Language + C-Language with AVR Library + C-Language with Arduino Library
Download Source, use the Arduino IDE Software, Open **p_intro_mixed_blink** and Upload

```
#include <avr/io.h>

void setup() {
  __asm__("SBI 0x04, 5");  // DDRB(0x04)  Arduino Pin 13(PB5) OUTPUT Pin
}
void loop() {
  PORTB = PORTB^(1<<PB5); // PORTB(0x05) Arduino Pin 13(PB5) toggle HIGH/LOW
  delay(250); // Arduino Library delay 250ms
}
```