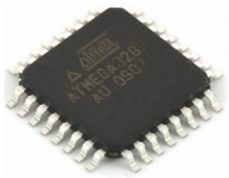
ATMEGA328/Arduino Uno – C-Language_00
https://github.com/teaksoon/lmaewapm



First, we need the ATmega328P Datasheet from the Supplier of the ATMEGA328P Micro-controller (since most of our Arduino Uno has this chip)

I have a copy in my github repository,

https://github.com/teaksoon/lmaewapm/blob/main/ATmega328P_Datasheet.pdf

You should download and keep one copy on your Computer

The ATMEGA328P Datasheet file is very big, there is no need to print it or read everything in it. We may only need to use some of it. For now, take a look at these two topic first (dont worry if they dont make sense to you at all now)

1. **AVR INSTRUCTION SET:** from ATMEGA328 Datasheet: Topic 31. from page 281 "31. Instruction Set Summary"

2. **ATMEGA328 CPU Memory(Register):** from ATMEGA328 Datasheet: Topic 30. from page 271 "30. Register Summary"

Those two topics are the ones that will get the micro-controller CPU do work for us, from those we make our PROGRAM.

There are many Register in the CPU, some **values in the Registers will cause a task being performed.** Sometimes CPU also updates the Registers with certain values for our Program to use (The Datasheet will tell us every single one of them). In order to **update or retrieve the Registers**, we need to use the instructions from the **INSTRUCTION SET**

For example:

```
SBI  DDRB, 5
```

The code above has an instruction "**SBI**" from **AVR INSTRUCTION SET**, will set the **CPU Memory(Register) "DDRB"**, 5th bit, to have a value of '1'

Once the CPU see this instruction "SBI DDRB 5", will set our Arduino Uno I/O Pin 13 to become an OUTPUT Pin.

If we program in **Assembly Language**, we will have to code exclusively with the **INSTRUCTION SET and** the **CPU Memory(Register).** Like the code above

However, when we Program using **C-Language with the Arduino Libraries**, we **seldom see the INSTRUCTION SET or the Registers.** This is because they are all hidden from us by our C-Language and the Arduino Libraries (which makes things alot easier). The **C-Language and Arduino Libraries has it own set of instructions** which will be **converted into AVR INSTRUCTION SET/Registers** for us **by the C-Language Compiler.**

Although most of the time we might not need to use the INSTRUCTION SET and Registers in our C-Language when using the Arduino Libaries, we need to know they exist. As we progress, sooner or later we will be digging into them. **Good thing about C-Language is that, we can access into the INSTRUCTION SET and CPU Memory(Register) if we wish to do so.**

Some of you may be curious, why not just use Assembly Language and make full use of the INSTRUCTION SET and Registers

"*SBI DDRB 5*" looks so simple

In fact seems easier than our familiar C-Language with Arduino Libraries

pinMode(13, 5);

For small PROGRAM, yes... Assembly Language is a great option. It is small and we can access into every single details of the micro-controller

Lets look at another example, this will probably change your minds about coding in Assembly Language

Lets say we want to delay 250ms, in C-Language with Arduino Libraries we will simply code,

delay(250);

but in Assemby Language, there is no delay(250), we have to make our own. The code below is a fixed 250ms delay, if we want a variable delay time, more coding will be needed (we do not want to go there, we just let C-Language do the work for us)

```
delay_250_ms:
  ; One millisecond is 16,000 cycles at 16MHz.
  ; 250ms = ideally 4,000,000 cycles
  ; 601+3998600+800+1 = 4,000,002, closed enough
                      ; Total = 1 cycle
  LDI  r20,200        ; 1-cycle

reset_ctr:
                      ; Total = 4 x 200 = 800 cycle
  NOP                 ; 1-cycle
  NOP                 ; 1-cycle
  LDI  31, 4998>>8    ; 1-cycle
  LDI  30, 4998&255   ; 1-cycle

delay_loop:
                      ; Total = ((4 x 4998) + 1) x 200 = 3998600 cycle
  SBIW r30, 1         ; 2-cycle
  BRNE delay_loop     ; 2-cycle, BRNE=1-cycle when ends

                      ; Total = (3 x 200) + 1 = 601 cycle
  SUBI r20, 1         ; 1-cycle
  BRNE reset_ctr      ; 2-cycle, BRNE=1-cycle when ends

  RET
```

When coding C-Language without the INSTRUCTION SET or the CPU Memory, it does not mean C-Language is unable to do certain things that the Assembly Language can do.

We can still access to the INSTRUCTION SET and the CPU Memory from our C-Language. For example: if we wish to run the "NOP" and access the DDRB CPU Memory from C-Language in Arduino, we can code in C-Language

__asm__("NOP"); // We run the "NOP" from the INSTRUCTION SET

DDRB |= (1<<5); // CPU Memory, DDRB Register, same as "SBI DDRB 5"

So now we make up our minds, it is going to be

C-Language