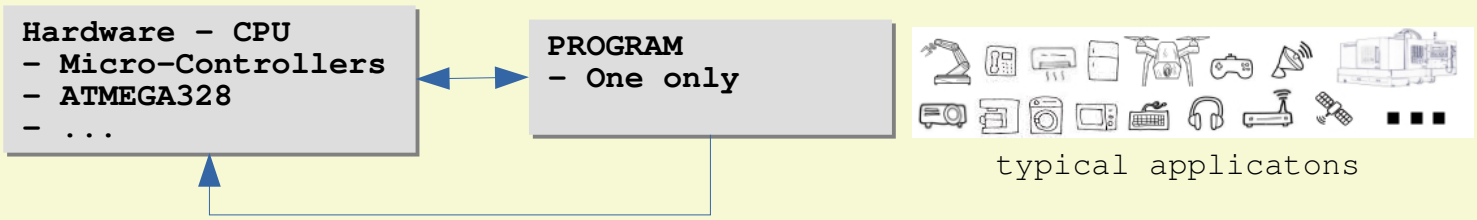
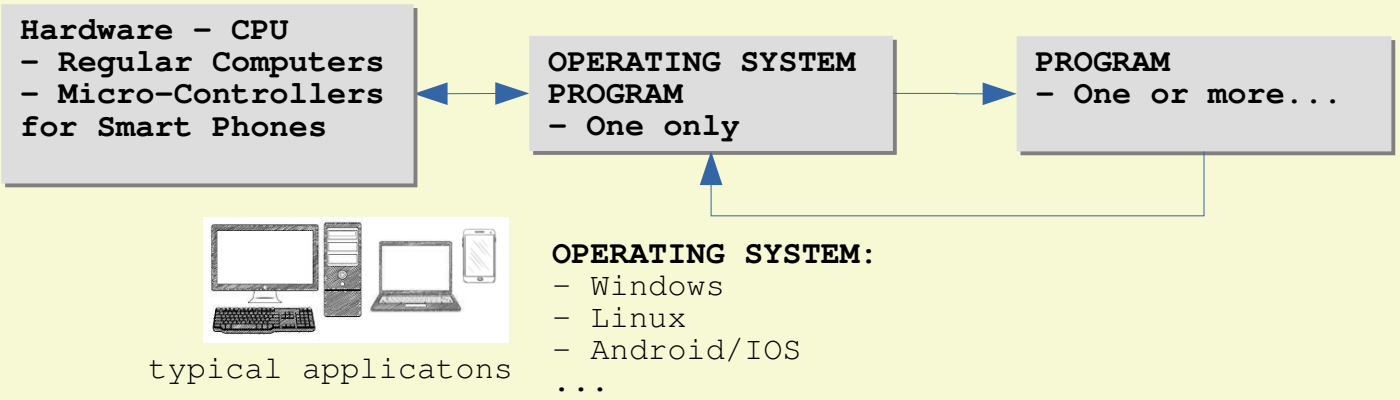
* SETUP 1: PROGRAM and CPU



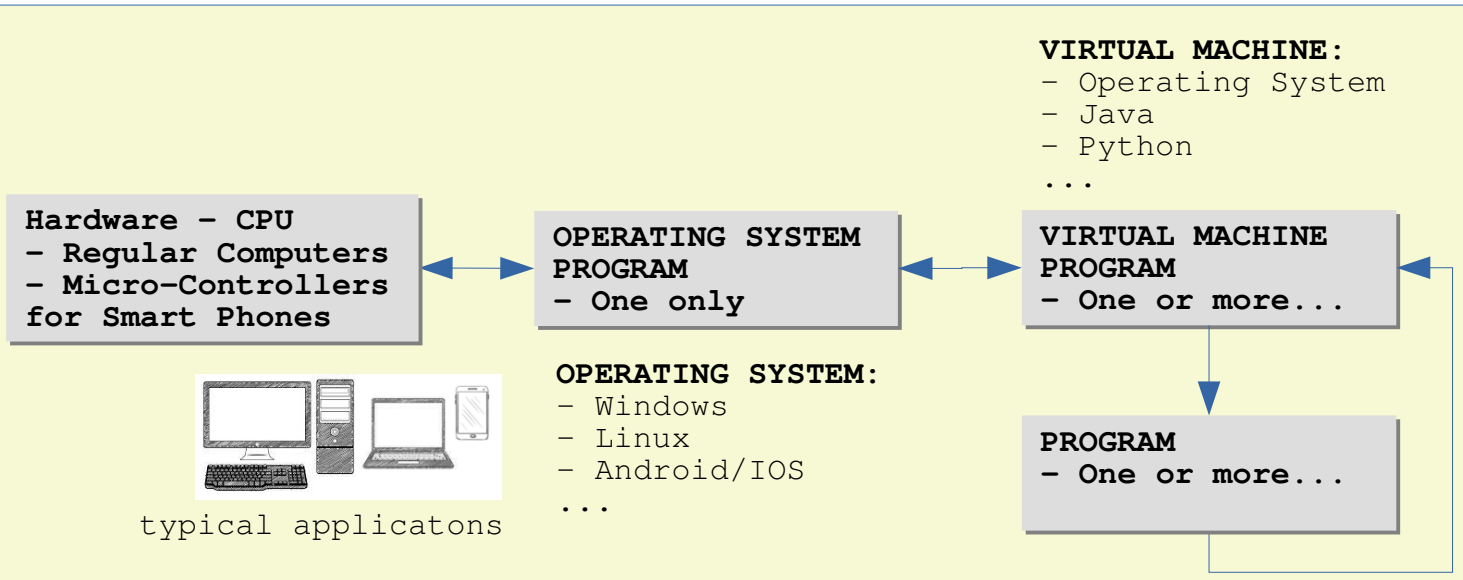
In this type of Setup. The C-Language PROGRAM will interact directly with the CPU (via INSTRUCTION SET and CPU MEMORY)

SETUP 2: PROGRAM and CPU (Requires Operating System)



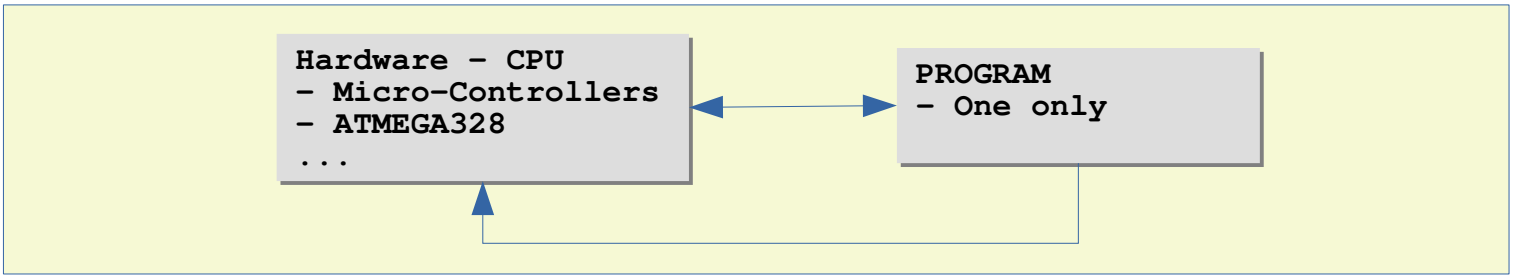
In this type of Setup, a PROGRAM normally interact with the OPERATING SYSTEM, then the OPERATING SYSTEM will deal with the CPU. However, if C-Language is used, the C-Language PROGRAM can interact with both OPERATING SYSTEM and the CPU. This is useful when speed and flexibility is required, direct interaction with the CPU gives the optimum result

SETUP 3: PROGRAM and CPU (Requires Operating System and Virtual Machine)



In this type of Setup. When the PROGRAM is written in Java, Python,... PROGRAM can only interact with a “VIRTUAL MACHINE PROGRAM” which is specific to each of them. The “VIRTUAL MACHINE PROGRAM” will then interact with the OPERATING SYSTEM and the OPERATING SYSTEM will then interact with the CPU. For this kind of system, we will need very powerful CPU

* SETUP 1: PROGRAM and CPU



An **ATMEGA328 micro-controller PROGRAM** is a sequence well arranged "instruction codes" from the **AVR INSTRUCTION SET**. The PROGRAM are mostly consist of "instruction codes" to **manipulate the CPU Memory (REGISTERS)**.

The ATMEGA328 micro-controller will then **perform various task, mostly based on what is in the CPU Memory (REGISTERS)**

From the ATMEGA328P Datasheet

1. **AVR INSTRUCTION SET:** ATMEGA328P Datasheet: Topic 31. from page 281 "31. Instruction Set Summary"
2. **ATMEGA328 CPU Memory(REGISTERS):** ATMEGA328P Datasheet: Topic 30. from page 271 "30. Register Summary"

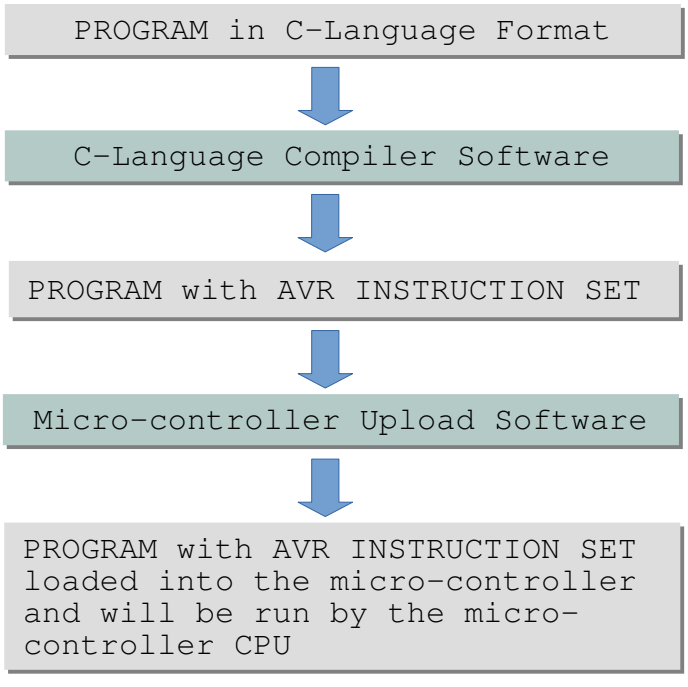
By just looking at "AVR INSTRUCTION SETS" and the "REGISTERS" from the ATMEGA328P Datasheet; we already know that using the AVR INSTRUCTION SET and the CPU Memory(REGISTERS) directly is not easy

Even for a simple task, we will need to do alot of work

We use the easier C-Language instead

Instead of using instructions from the "AVR INSTRUCTION SET", the C-Language PROGRAM uses its own set of "Keywords" and "Symbols" along with some C-Language coding rules.

A C-Language PROGRAM codes can be recognized by the "C-Language Compiler Software". The "C-Language Compiler Software" will convert the C-Language PROGRAM into AVR INSTRUCTION SET PROGRAM for us.



The C-Language has its own simplified “instruction set” in the form of “Keywords” and “Symbols”. C-Language also have some code formatting and arrangement rules that we need to follow.

The 32 C-Language Keywords and Symbols

Keywords

MEMORY	CONTROL
01.void	21.return
02.const	22.if
03.static	23.else
04.signed	24.for
05.unsigned	25.while
06.short	26.do
07.char	27.switch
08.int	28.case
09.long	29.break
10.float	30.default
11.double	31.continue
12.sizeof	32.goto
13.struct	
14.union	
15.enum	
16.typedef	
17.auto	
18.register	
19.extern	
20.volatile	

Symbols

CONTROL	LOGIC	MATH	BIT OP
#	==	*	|
< >	!=	%	&
“	<	/	^
‘	>	+	~
( )	<=	-	<<
{ }	>=		>>
[ ]			
//	&&		
/* */	||		
;	!		
,			
:			
?			
.			
\			
=			
MEMORY			
&			
*			

Below are not part of the C-Language but is often used with the C-Language

The C-Language pre-processor

The C-Language pre-processor give special instructions to the “C-Language Compiler Software” and also acts like a “Search and Replace” tool before the actual “C-Language Compiler Software” process starts.

```
#define  Similar to “Search and Replace”
#include Get extra codes from “.h” Source Code file
#undef   Cancel a #define that was specifed earlier
#ifdef   If a #define already exist
#ifndef  If a #define does not exist
#if      Condition is true.
#else    Condition is false
#elif    Combination of #else and #if
#endif   Ends of the conditional #if
#error   Output error message
#pragma  Special Command to the C-Language Compiler Software
```

C-Language only have 32 keywords and a bunch of Symbols. Out of that, many will not even be used in most of our Programs. This may looks like very little, C-Language is a actually very powerful Programming Language.

Almost all modern computer Operating System is written in C-Language. Almost all “Virtual Machine Software” for other Programming Language, the C-Language itself are also written in C-Language. When comes to micro-controllers, C-Language is used almost everywhere

While making C-Language PROGRAM, we will learn each of the C-Language "Keywords", "Symbols" and its various Rules

C-Language PROGRAM Rules:

- In C-Language PROGRAM the functional "instruction codes" are kept in "containers" known as "functions"

- We must have at least one "function" in a C-Language PROGRAM

- Among the "functions" in a C-Language PROGRAM, we must have one function with "main" as its function name. The C-Langauge PROGRAM will start from the "main" function

WHAT IS A FUNCTION and HOW TO MAKE A FUNCTION ?

A function have 4 Parts,
Part1:output data – if none, code "void" followed by a space
Part2:unique function name
Part3:input data – if none, just leave an empty bracket ( ) pair
Part4:function body – if none, just leave an empty curly bracket { } pair

BELOW is a typical C-Language function

Part1:output data
- specify **datatype**
- followed by a space

Part3:input data
- specify **datatype** and **name**
- Contained within a bracket pair ( )
- can have multiple inputs, separated with comma , also commonly known as Parameters

Part2:unique function name
- cannot have spaces in-between

function header

Part3:function body
- contains "instruction codes"
- contained within a curly bracket pair { }

```
int sum_2_numbers(int a, int b)
{
  int sum;
  sum = a+b;
  return sum;
}
```

"instruction code" to send data inside "sum" to output data. This line of code is only required when there is an output data for this function

Do not worry about the codes and symbols in the example function above, we will learn about them in later lessons.

C-Language PROGRAM Rules:

- In C-Language PROGRAM the functional "instruction codes" are kept in "containers" known as "functions"

- We must have at least one "function" in a C-Language PROGRAM

- Among the "functions" in a C-Language PROGRAM, **we must have one function with "main" as its function name.** The C-Langauge PROGRAM will start from the "main" function

Lets make a bare minimum C-Language PROGRAM
From the "C-Language PROGRAM Rules" above, we only need to make a "main" function in order to make a complete C-Language PROGRAM

Lets make our "main" function step by step, (this step by step way of coding can be used for making any other functions)

Step 1: Make a skeleton function with function name. In this case, it is "main", assuming there are no output and no input required

```
void main(){}
```

Step 2: Does the function produce output ? If yes specify the output "datatype", then followed by one space or more
C-Language Rules(for "main" function only): "main" function must have an "int" datatype output

```
int main(){}
```

Step 3: if the function produce output, we must have an "instruction code" to return an output value, we put the code in the "function body" within the curly bracket { } pair
C-Language rules(for any functions): Each "instruction code" inside the "function body" must end with a semi-colon ;

```
int main(){
  return 0;
}
```

Step 4: Does our function require any input values ? In this case, our "main" function does not need an input value. So, we leave the bracket ( ) pair empty

```
int main(){
  return 0;
}
```

Thats is it, we are done. We make our "main" function in C-Language with all the C-Language rules. We can put that code in our Arduino IDE Software and upload it into our ATMEGA328 micro-controller on our Arduino Uno

In Arduino IDE Software, Save as: "c_language_bare_minimum" and upload

```
int main() {
  return 0;
}
```

This is a bare minimum C-Language PROGRAM. Although it does nothing, it is a LEGAL C-Language PROGRAM. From this small program alone, we used two C-Language Keywords "return" and "int" also a few symbols ( ) { } and ;