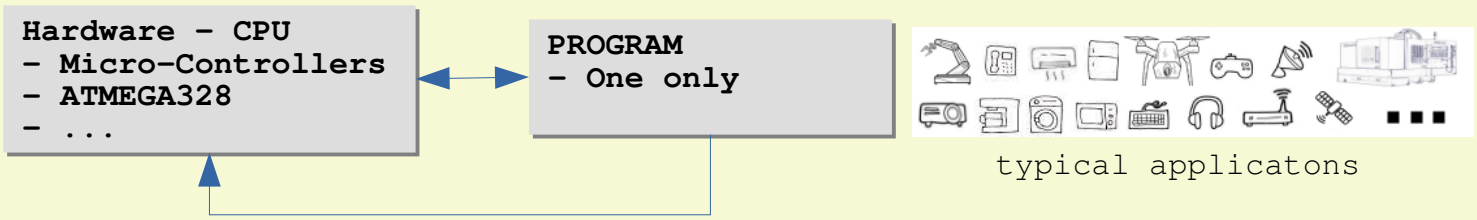
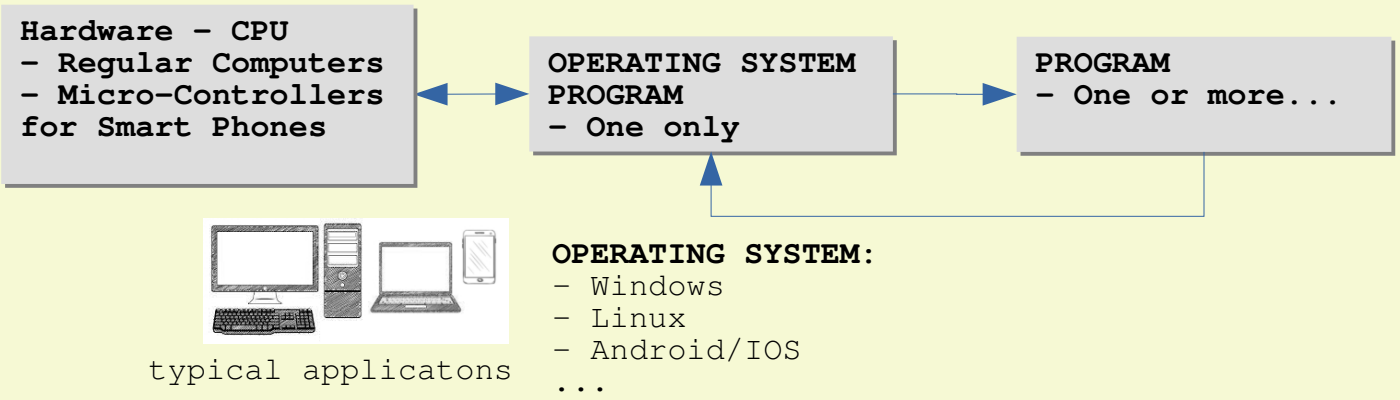
ATMEGA328/Arduino Uno - c_bare_minimum
https://github.com/teaksoon/lmaewapm

* SETUP 1: PROGRAM and CPU



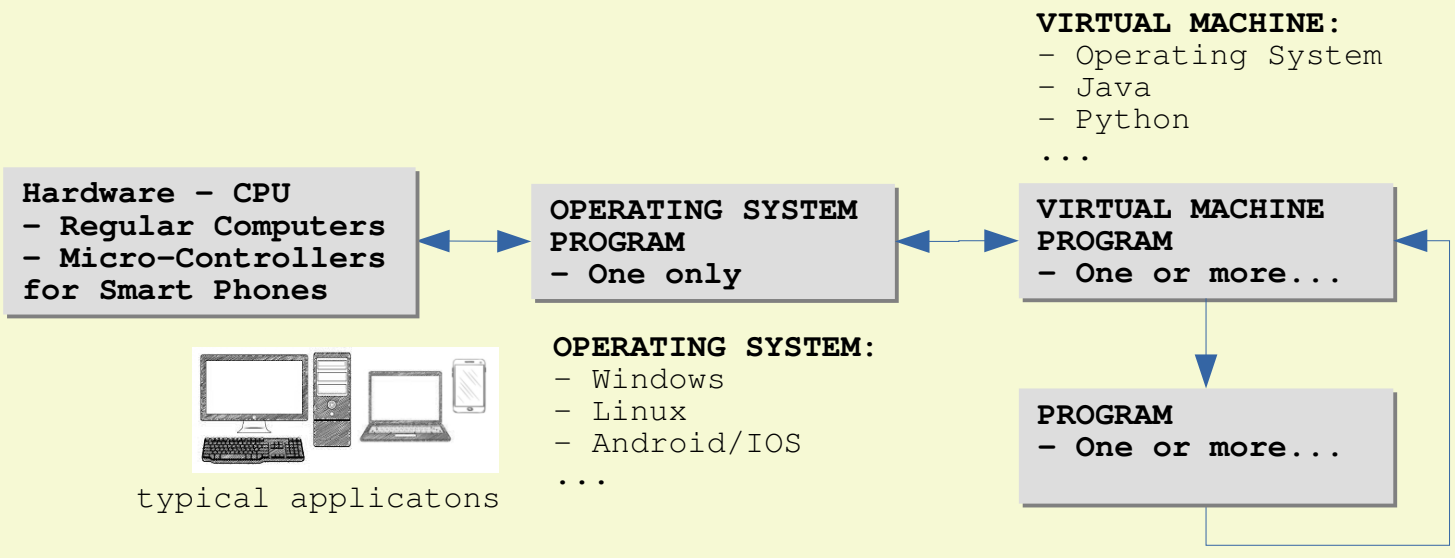
In this type of Setup. The C-Language PROGRAM will interact directly with the CPU (via INSTRUCTION SET and CPU MEMORY)

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

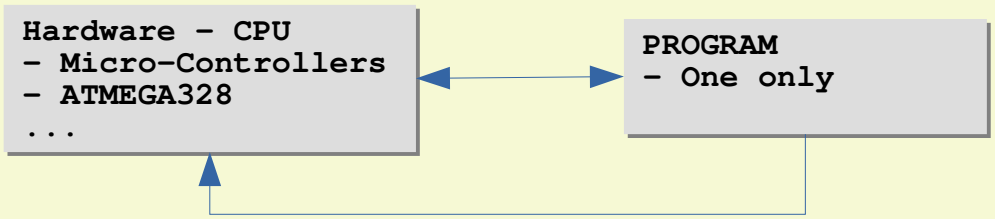
SETUP 3: PROGRAM and CPU (Requires Operating System and Virtual Machine)



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

ATMEGA328/Arduino Uno - c_bare_minimum
https://github.com/teaksoon/lmaewapm

* 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. Instruction Set Summary", from page 281
2. ATMEGA328 CPU Memory(REGISTERS): ATMEGA328P Datasheet:
Topic "30. Register Summary", from page 271

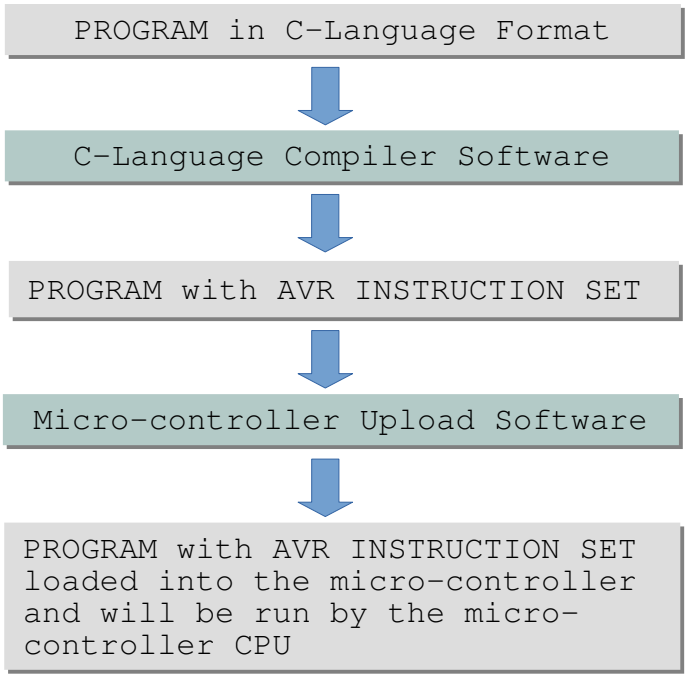
By just looking at "AVR INSTRUCTION SETS" and the "REGISTERS" from the ATMEGA328P Datasheet; we already know that using the AVR INSTRUCTION SET and the CPU Memory(REGISTERS) directly is not easy

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 some Symbols

Keywords

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

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

C-Language: Function Creation and Usage

There are 4 Parts in the "function"

Part1:output data(datatype): if none, use "**void**" as datatype
Part2:function name
Part3:input data(datatype and name): if none, leave empty bracket () pair
Part4:body

Part1: output data
- **datatype** followed by **space**
- single datatype only

Part3:input data
- **datatype** followed by **space** and **name**
- can have multiple inputs, each input data is separated with comma ,

Part2:function_name
- unique and cannot be Keywords
- cannot have spaces in-between the name

function header

Part4: body
- contains "instruction codes"
- within a curly bracket { } pair

```
datatype function_name(datatype name)
{


}
```

function header for "**function declaration**"

If function is kept in a different file, "function declaration" is required. If not declared, the Compiler Software will assume an error in our PROGRAM and will be shown ase errors. "function declaration" are ussually coded before the function is used, normally on top of the PROGRAM file.

Example:

Part1:output data

Part3:input data
- multiple inputs, separated with comma ,

Part2:function name

Creation

function header

Part4: body
- contains "instruction codes"
- 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

Usage

function declaration

Function usage

```
int sum_2_numbers(int a, int b);

...

int result;
result = sum_2_numbers(12,34);
```

ATMEGA328/Arduino Uno – c_bare_minimum
https://github.com/teaksoon/lmaewapm

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

"bare minimum" C-Language PROGRAM.
Based on the "C-Language PROGRAM Rules" above, we only need to make one "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 1/5: Make a skeleton function with its function name. In this case, it is "main", assuming there are no output and no input required

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

Step 2/5: Does the function produce output ? If yes specify the output "datatype", then followed by space
C-Language Rules(for "main" function only): "main" function must have an "int" datatype output

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

Step 3/5: if the function produce output, we must have an "instruction code" to return an output value, we put the "instruction 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 4/5: Does our function require any input values ? "main" function does not need any input values. So, we leave the bracket ( ) pair empty

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

Step 5/5: Does our function require other "instruction codes" ? We have none in this bare minimum PROGRAM, otherwise put them into the "function body" within the curly bracket { }

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

ATMEGA328/Arduino Uno – c_bare_minimum
https://github.com/teaksoon/lmaewapm

Arduino IDE|Save PROGRAM as: **c_bare_minimum**
Enter codes below and upload.

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

This is a bare minimum C-Language PROGRAM with just the bare minimum main() function. It does not do anything that we can see

LETS SEE IN DETAILS WHAT HAPPEN when we click the “Upload” button:

1. The Compiler Software will check for errors and convert this PROGRAM into equivalent “INSTRUCTION SET” and then turned them into a .HEX file

2. The .HEX file is then sent into the micro-controller PROGRAM MEMORY, stored in 1s and 0s (since MEMORY are just “switches” ON/OFF)

3. The micro-controller will “re-start” itself and the micro-controller CPU will run the PROGRAM stored in the PROGRAM MEMORY (this PROGRAM)

4. PROGRAM will start from the main() function (in normal situation, we will have other functions)

5. The main() function will reserve a WORKING MEMORY space (“int” datatype, which is 16-BITS) to store return value

6. Since there are no Input, the main() function proceeds to run the “instruction codes” inside the “function body”. Starting from the “instruction code” after the “function body” opening curly bracket {

7. The “instruction code” is **“return 0;”**
return 0;
This instruction code will put 0 (or any other value) into the WORKING MEMORY that was reserved earlier and then, the function ends. Any codes in the function after the “return” will not run.

8. Once the main() function ends, our PROGRAM ends

Note:
In Regular Computers, the return value from the main() function can be picked up by other PROGRAM. However, in micro-controller, we only have one PROGRAM and do not have another PROGRAM to pick this return value.

We code “return 0;” in our main() function is just for formality sake or for some more advanced micro-controller that can run multiple PROGRAM, like the Smartphone micro-controllers with Operating Systems