Fab Lab Ismailia represent :

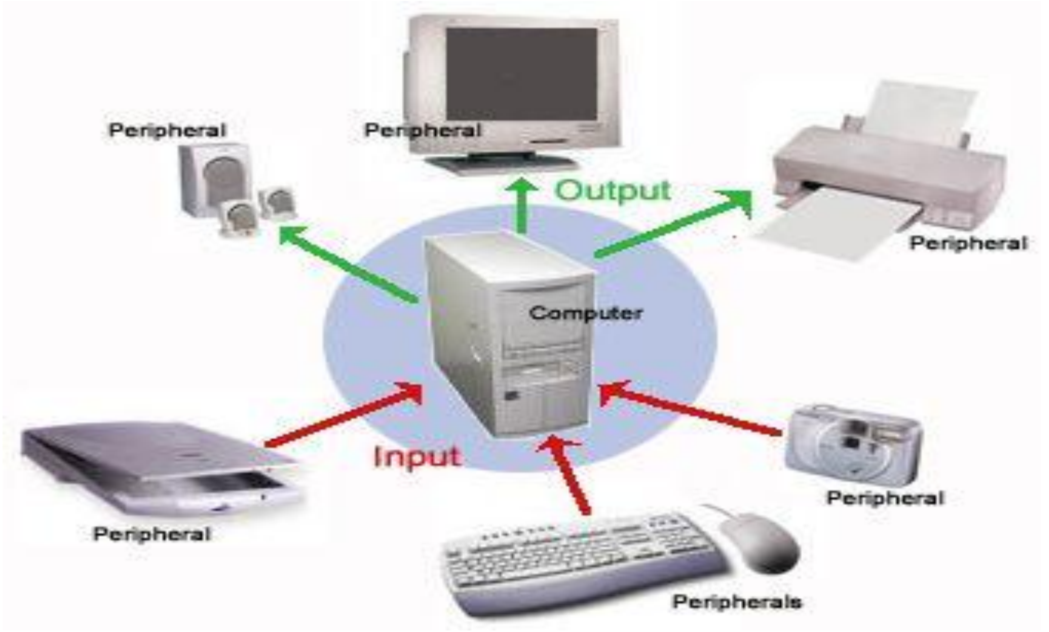
Embedded Systems Workshop

by : Mohammed hemed

What's an Embedded Systems ?

Embedded systems : are combinations of hardware and software that perform a specific function or perform specific functions within a larger system

Embedded System :
is a computer range from performing a single task
which hasn't User Interface (UI) , to more complex systems that use
(GUI) Graphical User Interface like smart phones .
Embedded Systems can be microprocessor
or microcontroller based.

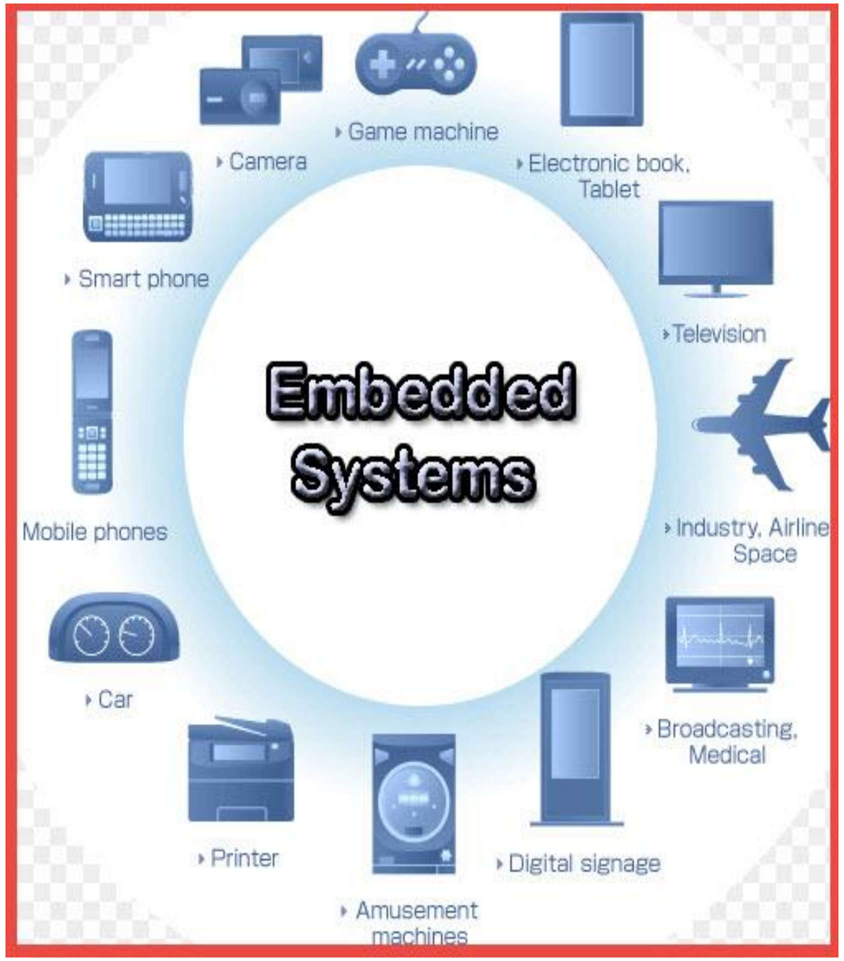


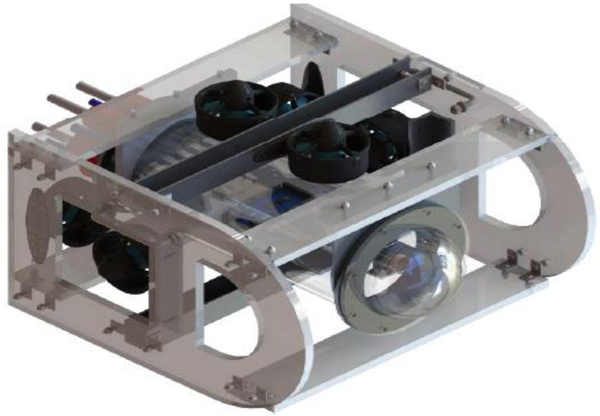
Embedded Systems Applications :

- **Mobile phone systems** (including both customer handsets and base stations).
- **Automotive applications** (including braking systems, traction control, airbag release systems, engine-management units, steer-by-wire systems and cruise control applications).
- **Domestic appliances** (including dishwashers, televisions, washing machines, microwave ovens, video recorders, security systems, garage door controllers).
- **Aerospace applications** (including flight control systems, engine controllers, autopilots and passenger in-flight entertainment systems).
- **Medical equipment**
- **Defense systems** (including radar systems, fighter aircraft flight control systems, radio systems and missile guidance systems).

- **Robotics** : **Minesweepers – ROVs – UAVs – Arm robot , ….. . ….**

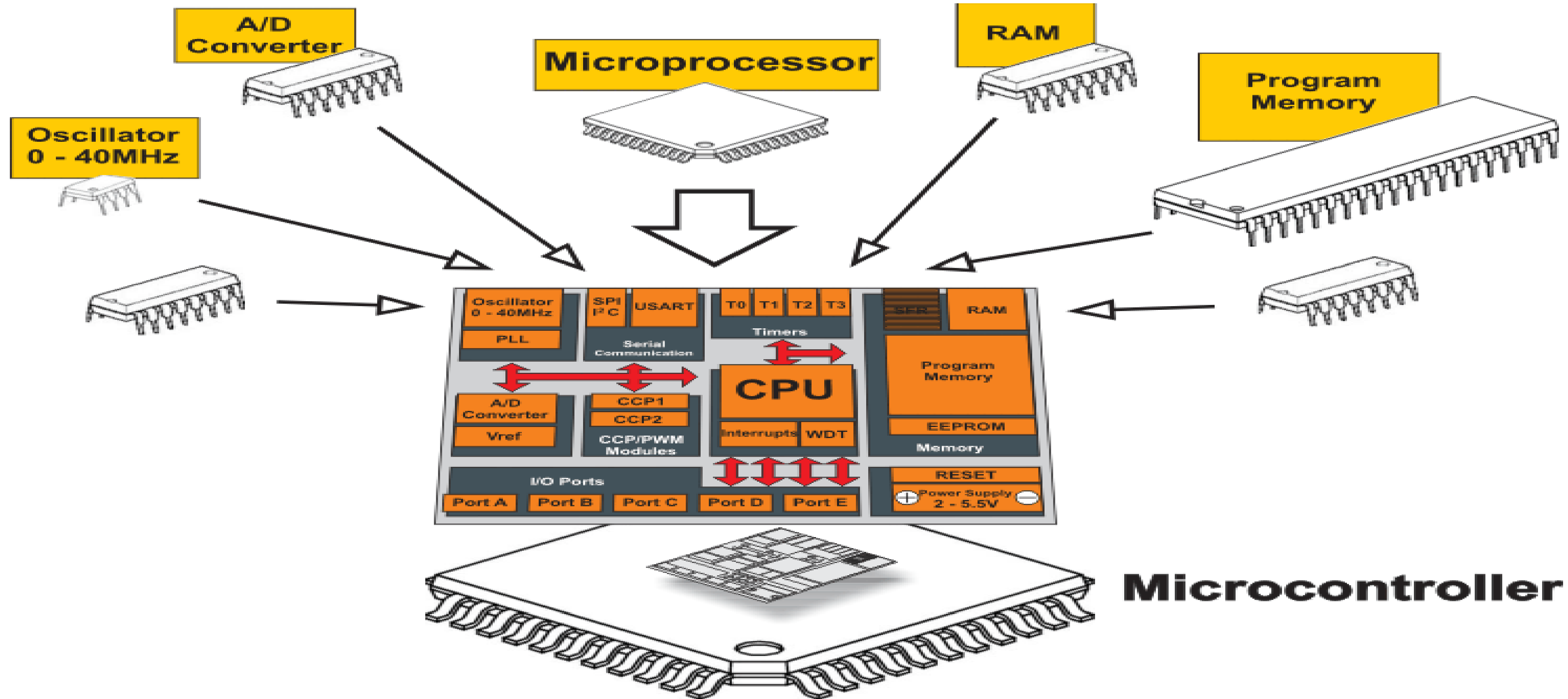
Embedded Systems Apps :





What is an microcontroller ?

- A microcontroller is a single chip, self-contained computer which incorporates all the basic components of a personal computer on a much smaller scale.
- It Contains : Microprocessor + Memory + (Peripherals like I/O Ports – ADC , ...)

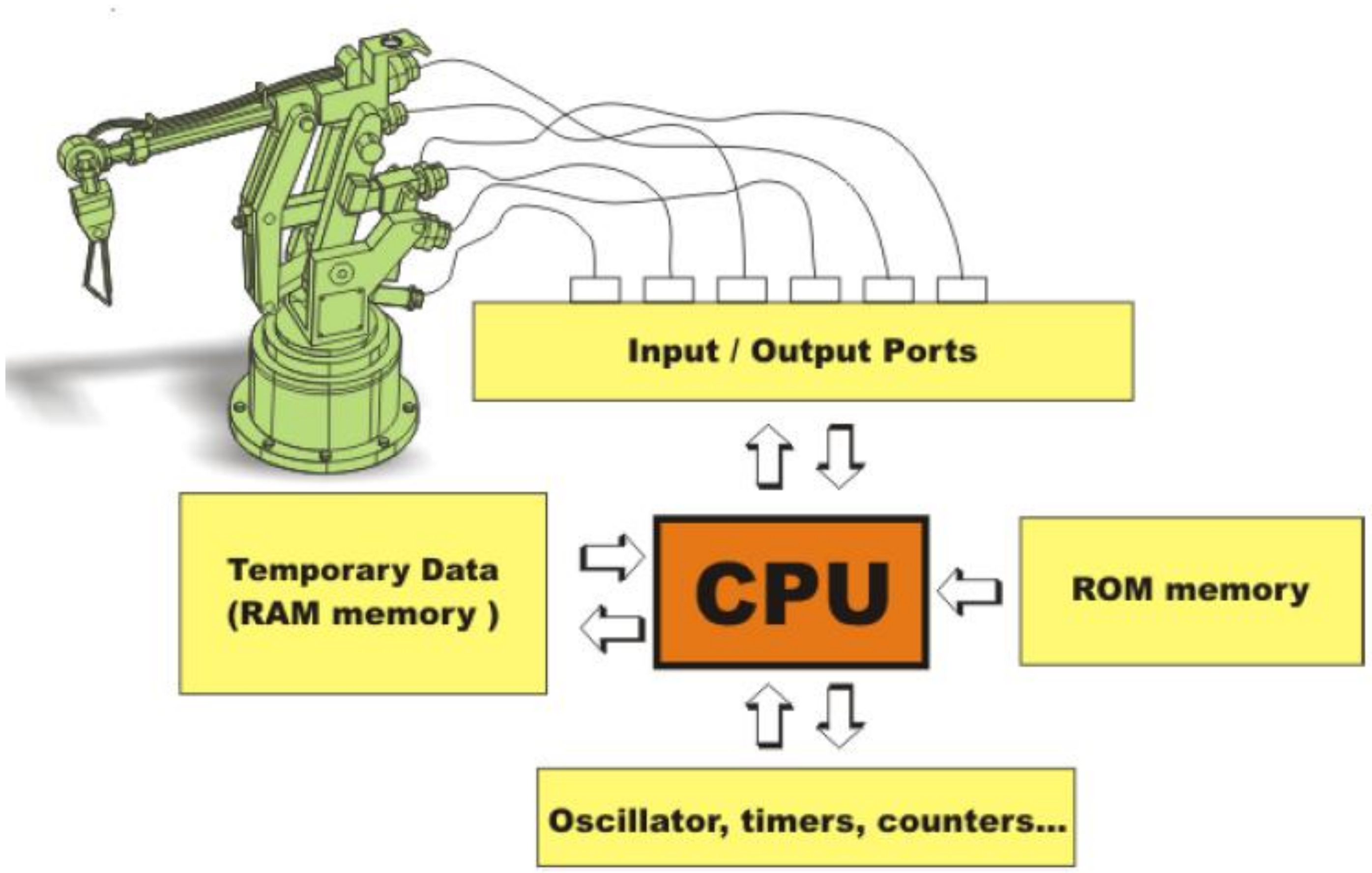


- As Embedded system developer we must know the difference between microprocessor(uP) and microcontroller (uC) .

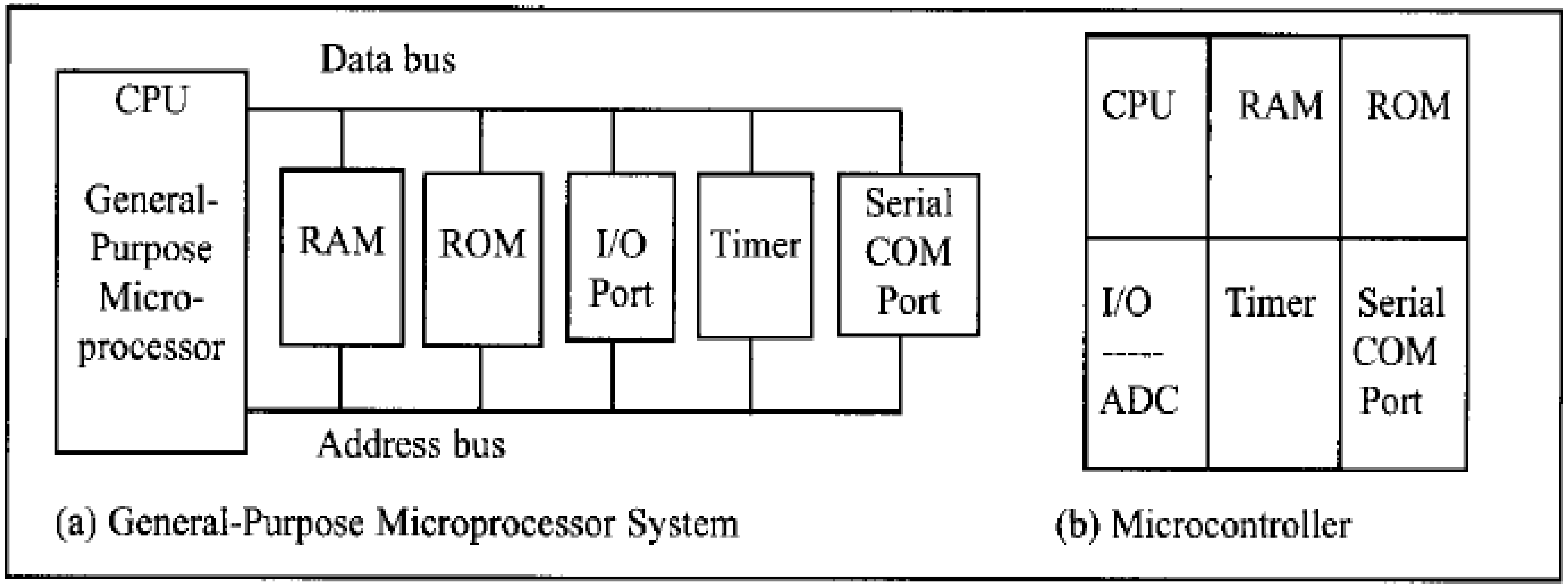
In order to When designing your Embedded system your system maybe Microprocessor or microcontroller based :

microcontroller (uC)	microprocessor(uP)
- Fixed amount of RAM – ROM – I/O peripherals , So user choose what fit his Application , so uC companies make a lot of families of the Same uC , in AVR : Xmega – Atmega – Attiny In PIC : PIC16 – PIC18 – PIC32 …. ,	-The designe decide the amount of RAM – ROM – I/O peripherals , So the system as whole is more expensive than uC .
Not expensive	Expensive
Single purpose	General Purpose

A microprocessor is a part of microcontroller



Microprocessor vs Microcontroller



Criteria of choosing uC

- **We must take care when we choose uC , in :**

1- Space : Maximum size of program memory .

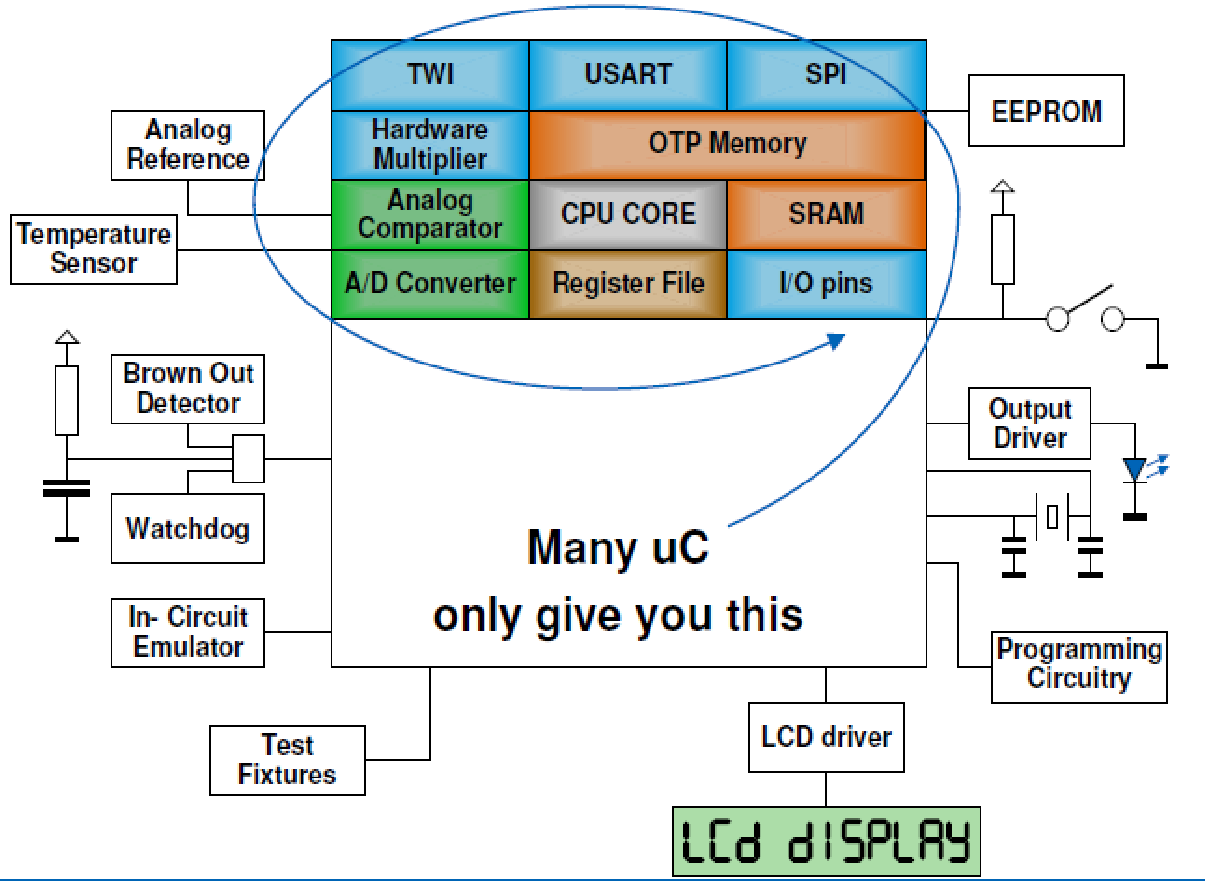
2- Power : The Power consumption is one of the most important issues as in according to this issue we then we choose the appropriate battery which fit our system

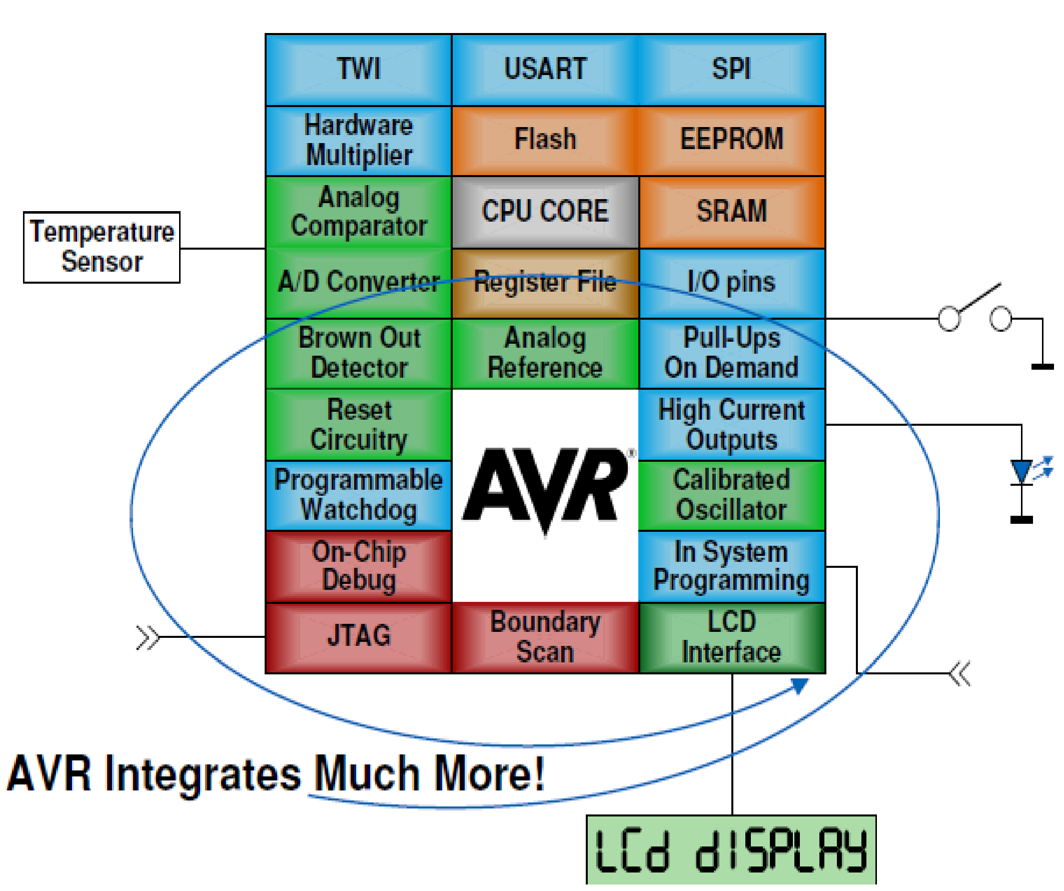
3- Price : if the price of uC is expensive it lead to make the whole system expensive as well .

So as Embedded system Developer we must be careful of these Constrains .

- **In some Application we don't need to a lot of features , we just want to do I/O operations like read signals – turn on/off certain bits , for example TV remote control , So they made something Called : Itty Bitty Processors (IBP) .**

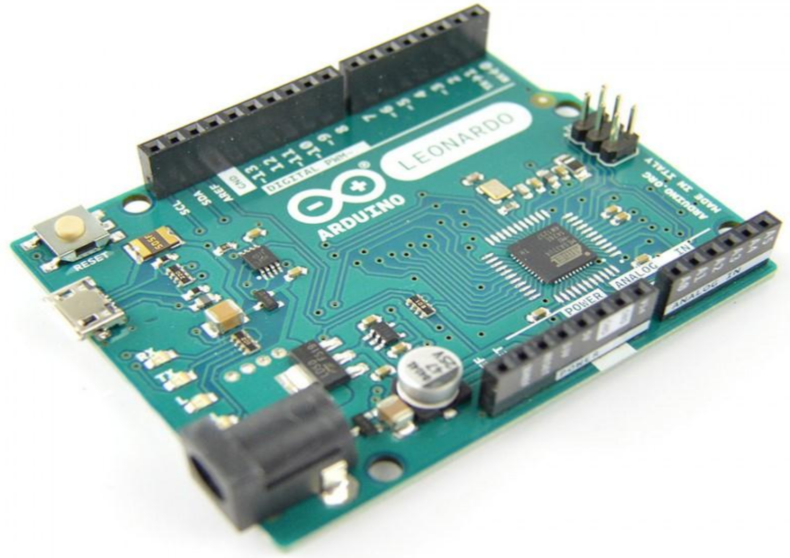
Microcontrollers War

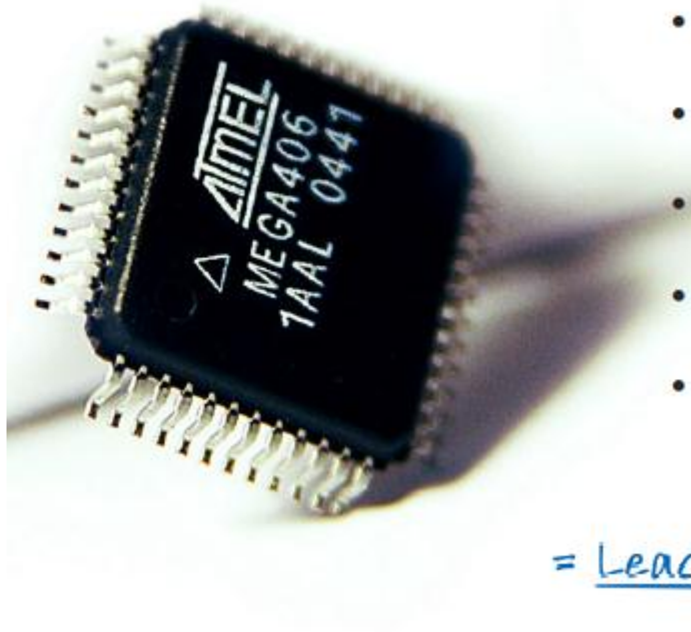




AVR differs from PIC in :

- Speed
- The big community
- Open Source Compilers like : AVR-GCC which support ANCI-C ,
  this lead the designers of Arduino to
  choose AVR uC .
- The architecture of AVR is more Efficient than P
  as you have 32 General Purpose Registers to
  to store and process temporary data but the
  PIC enforce you to use only one Register
  called Accumulator





- High performance
- Low power consumption
- High code density
- Advanced memory technology
- High integration

= Leading 8-bit microcontroller

What is inside a microprocessor ?

1- **General Purpose Registers** (GPRS) : the CPU uses these registers to store information temporarily , (may be two values to be processed)

or the address of the values needed to be fetched from the memory .

these registers may be 8bit – 16bit – 32bit depending on the CPU , the bigger registers the better the CPU (but it will be more expensive)

2- **ALU (Arithmetic & Logic Unit)** : it's responsible for performing arithmetic functions : ADD – SUB – MUL

Logic functions : AND – OR – NOT

3- **Program Counter** : It's responsible for pointing to the address of the next instruction to be executed , then its content are placed on the address bus to find and fetch the desired instruction .

4- **Instruction Decoder** : It's look like a dictionary storing the meaning

Of each instruction and what steps the CPU should take when receiving instruction .

What is inside a microprocessor ?

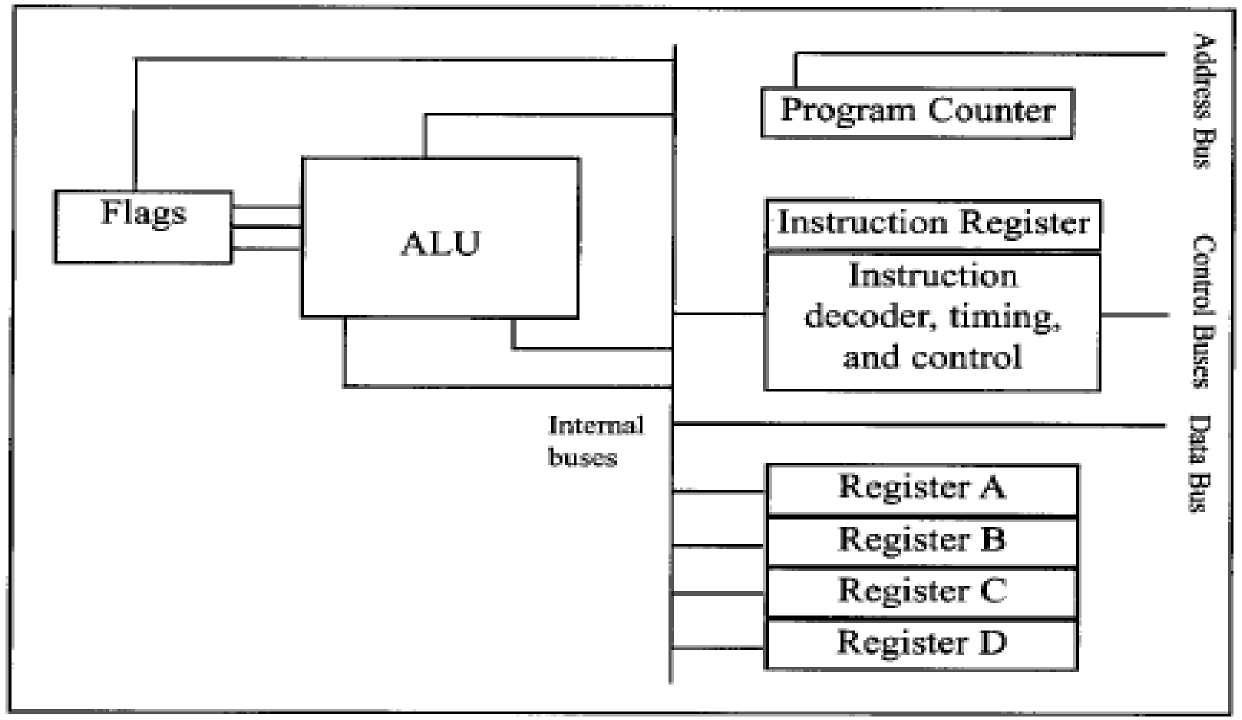
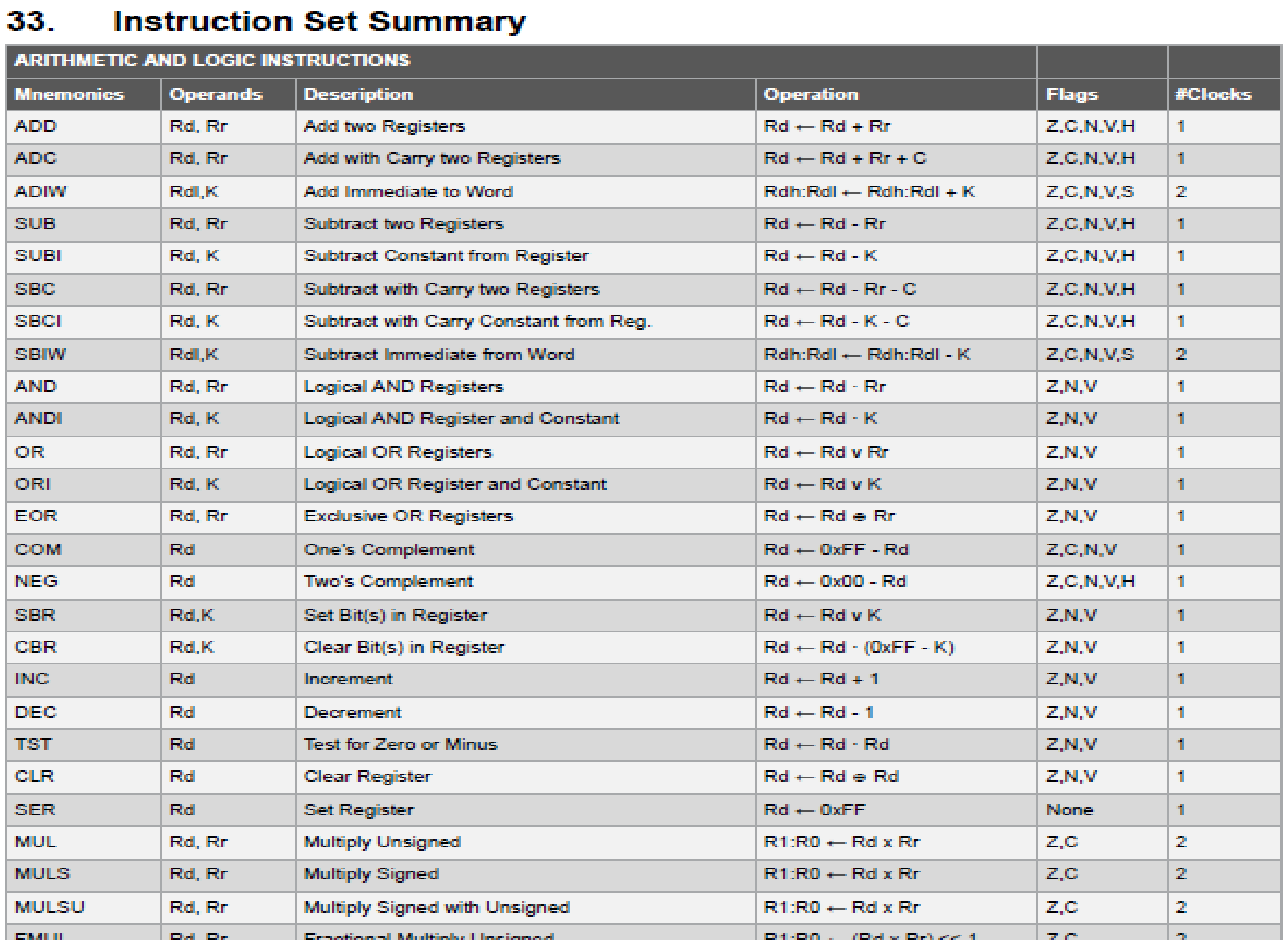


Figure 0-19. Internal Block Diagram of a CPU

Instruction Set Example from Atmega32 Datasheet

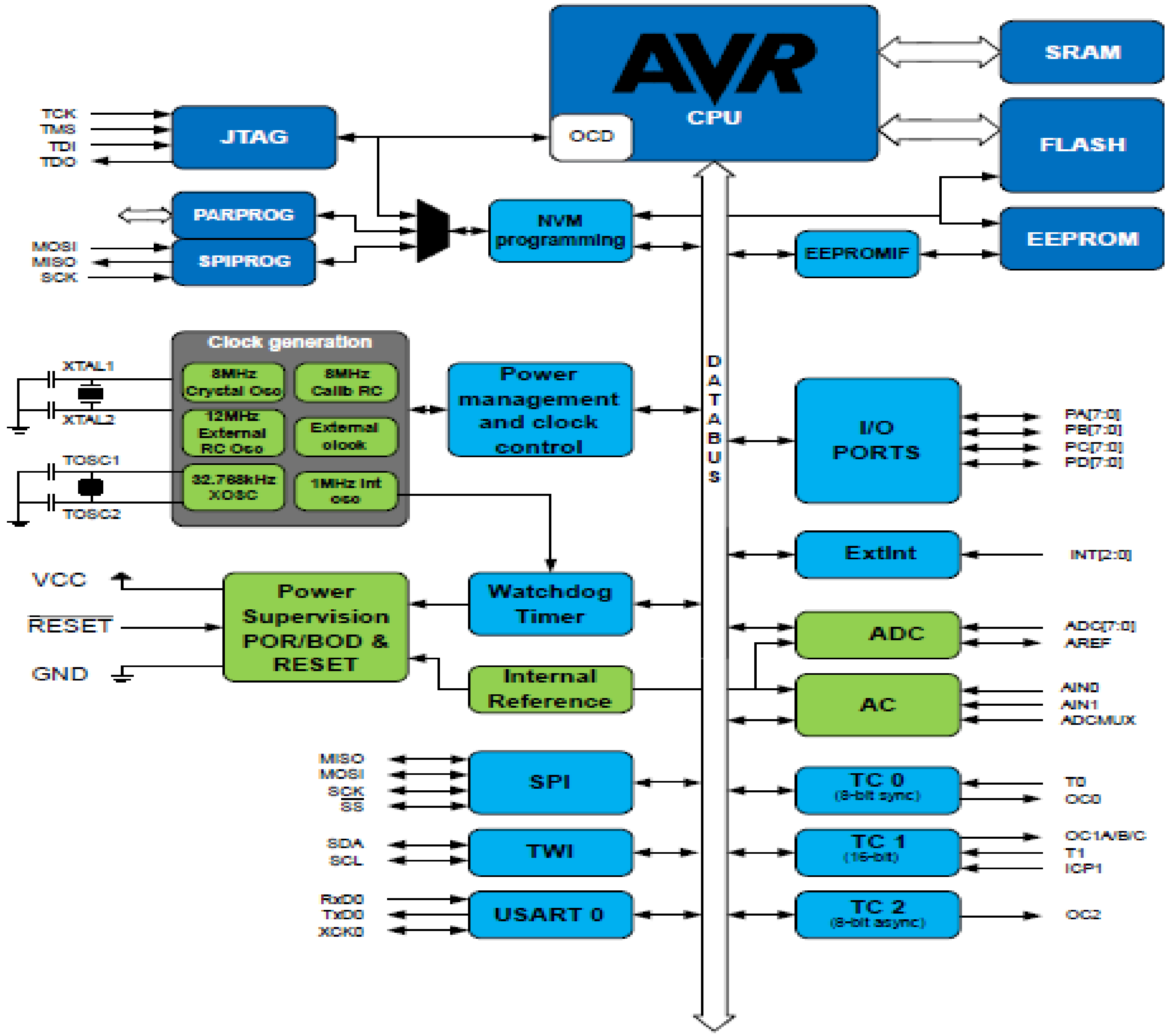


33. Instruction Set Summary

ARITHMETIC AND LOGIC INSTRUCTIONS					
Mnemonics	Operands	Description	Operation	Flags	#Clocks
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	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
AND	Rd, Rr	Logical AND Registers	Rd ← Rd · Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd · K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF - Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 - Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd · (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd - 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd · Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2

AVR Architecture :

Figure 4-1. Block Diagram



AVR Architecture :

AVR CPU Core:

- **Consist of 32 GPRs to store information to be processed**

+ Special Purpose Registers :

- **Status register : contains information about the result of**

the most recently executed arithmetic instruction.

- program counter : contain the address of the next instruction

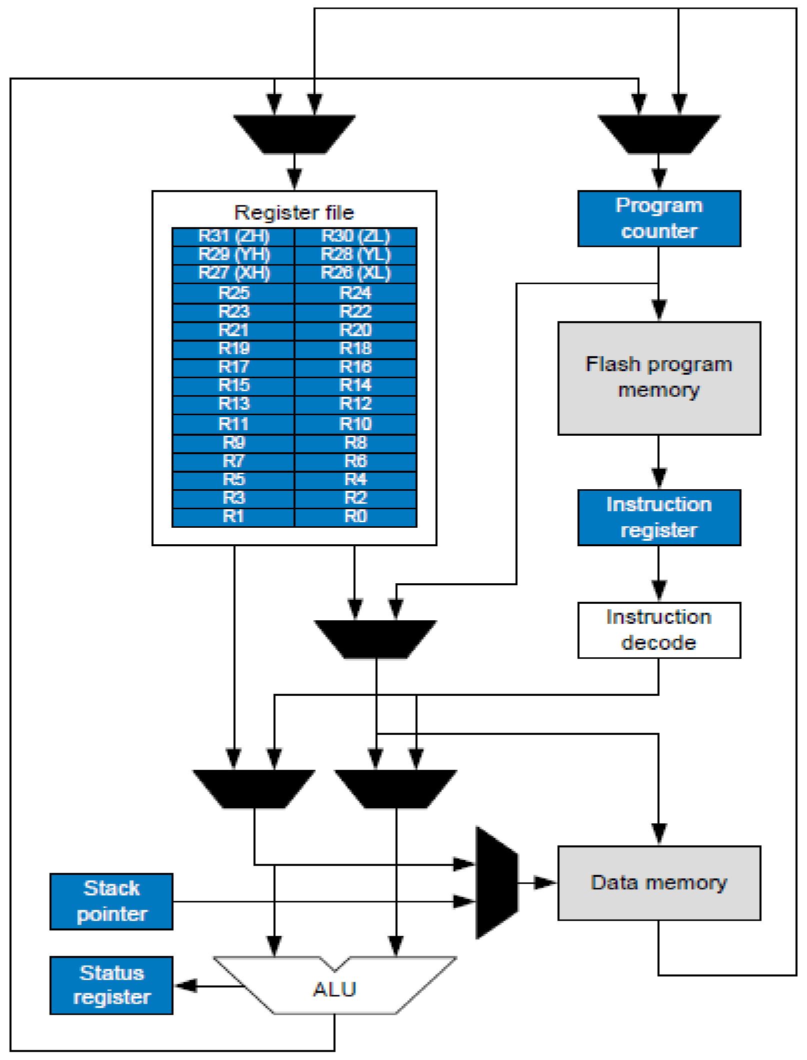
- Instruction Register : contain the instruction itself ,

which its address stored in program counter

- **- Stack pointer : The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls , Stack Pointer must be set to point above start of the SRAM, refer to figure *Data Memory Map* in *SRAM***
- ***Data Memory*.**

+ ALU (Arithmetic & logic unit) .

AVR CPU Core:


Register file
R31 (ZH)
R30 (ZL)
R29 (YH)
R28 (YL)
R27 (XH)
R26 (XL)
R25
R24
R23
R22
R21
R20
R19
R18
R17
R16
R15
R14
R13
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
Program counter
Flash program memory
Instruction register
Instruction decode
Data memory
Stack pointer
Status register
ALU

AVR memories :

- In AVR there are three main memory spaces :

Data memory + Program Memory space + EEPROM Memory for data storage.

1- In-System Reprogrammable Flash Program Memory :

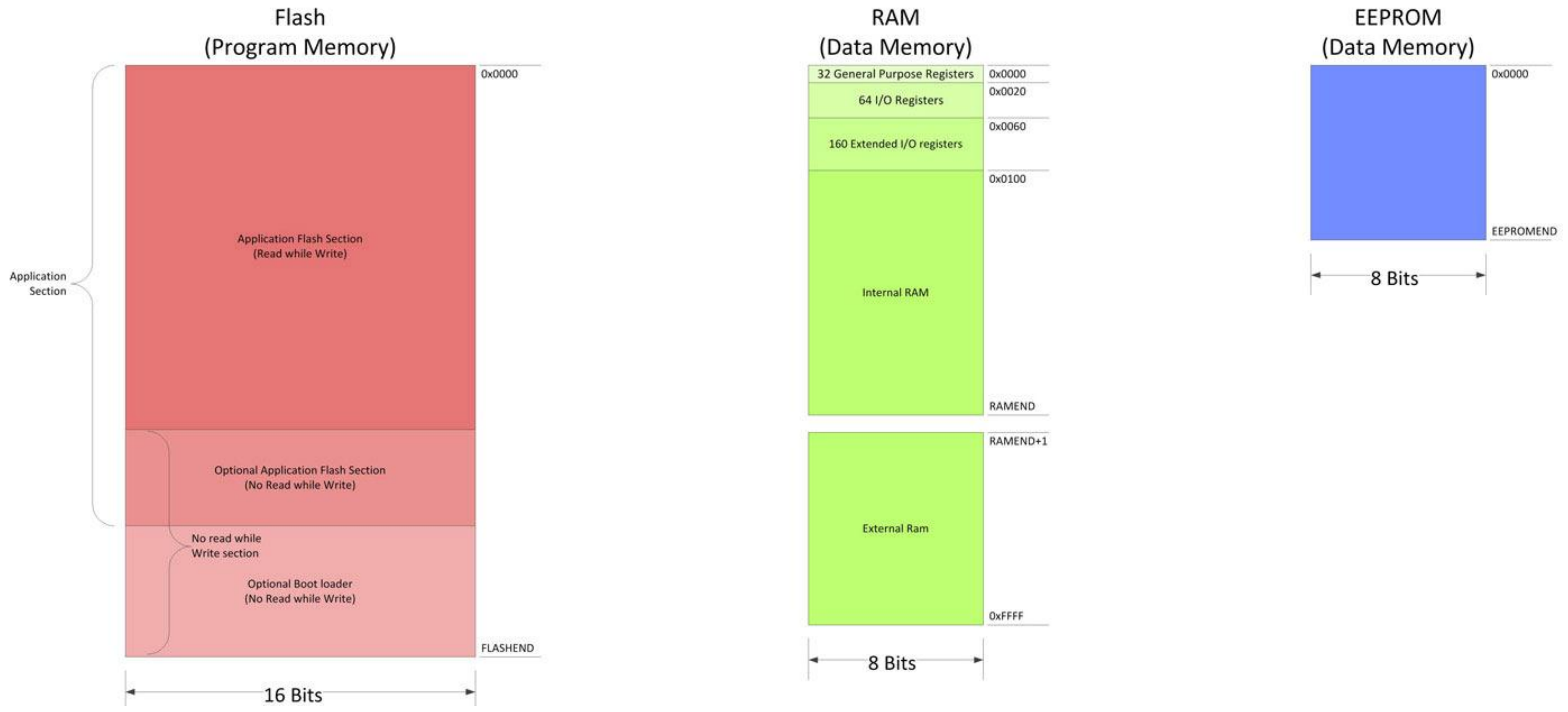
The Flash memory has an endurance of at least 10,000 write/erase cycles.

2- SRAM Data Memory : consist of

- I/O memory to access peripherals
- 32 GPRs
- General purpose Ram

3- EEPROM : To store data permanently , it doesn't lose its data when power is off .

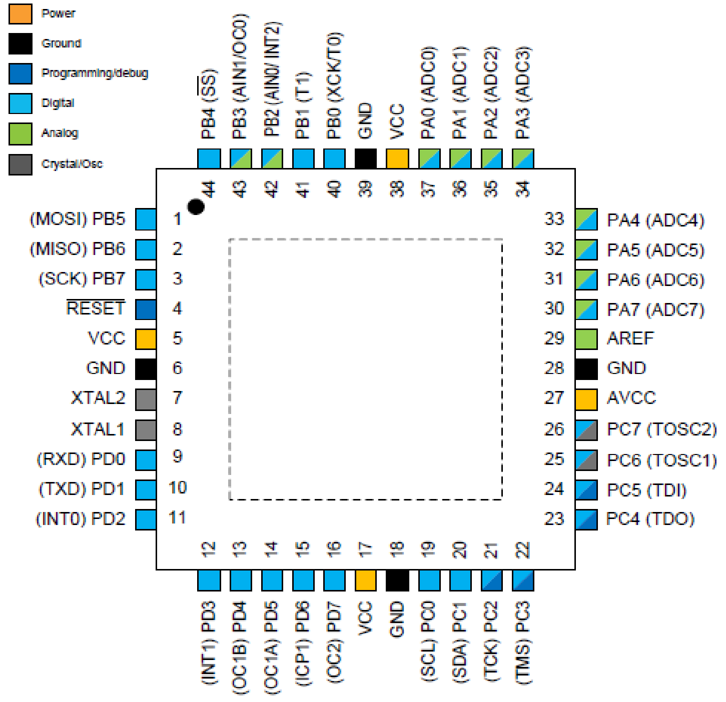
AVR memories :

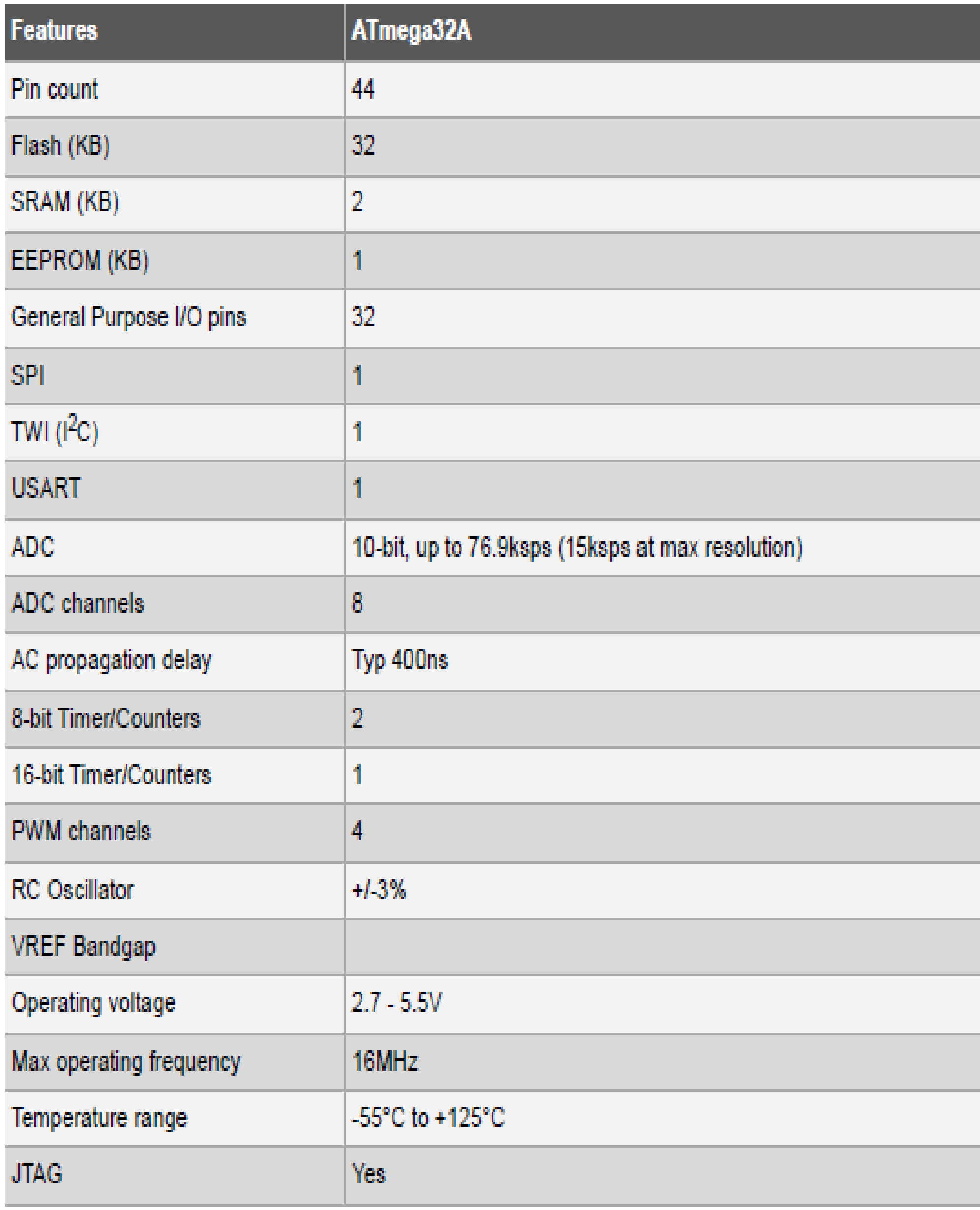


OUR microcontroller Specs :

- **Atmega32 uc :**

- 44 pins
- 32 kbyte flash memory
- 2 kbyte SRAM
- 1024 bytes EEPROM
- Communication protocols :
- UART – I2C – SPI
- ADC (8 channels)
- 2 Timer/counter 8-bit
- Timer/Counter 16-bit
- GPIO 32 pins
- Operating volt = 2.7 : 5.5 v
- Max operating frequency : 16Mhz
- RC internal oscillator : +/- 3%





Features	ATmega32A
Pin count	44
Flash (KB)	32
SRAM (KB)	2
EEPROM (KB)	1
General Purpose I/O pins	32
SPI	1
TWI (I^2C)	1
USART	1
ADC	10-bit, up to 76.9ksps (15ksps at max resolution)
ADC channels	8
AC propagation delay	Typ 400ns
8-bit Timer/Counters	2
16-bit Timer/Counters	1
PWM channels	4
RC Oscillator	+/-3%
VREF Bandgap	
Operating voltage	2.7 - 5.5V
Max operating frequency	16MHz
Temperature range	-55°C to +125°C
JTAG	Yes

How to program an Embedded System

- Computer languages are divided into two types :

1- machine dependent language : like (0,1) language , then to make the code easier they made hexadecimal language to make the code shorter , After that they made mnemonics instructions to every processor or family of processors , you can find it in the datasheet in the address of :

Instruction Set , it called Assembly language .

$$\begin{tikzcd}
\text{Source Code in Assembly} \arrow[r] & \text{Assembler} \arrow[r] & \text{Object Code } 011101010
\end{tikzcd}$$

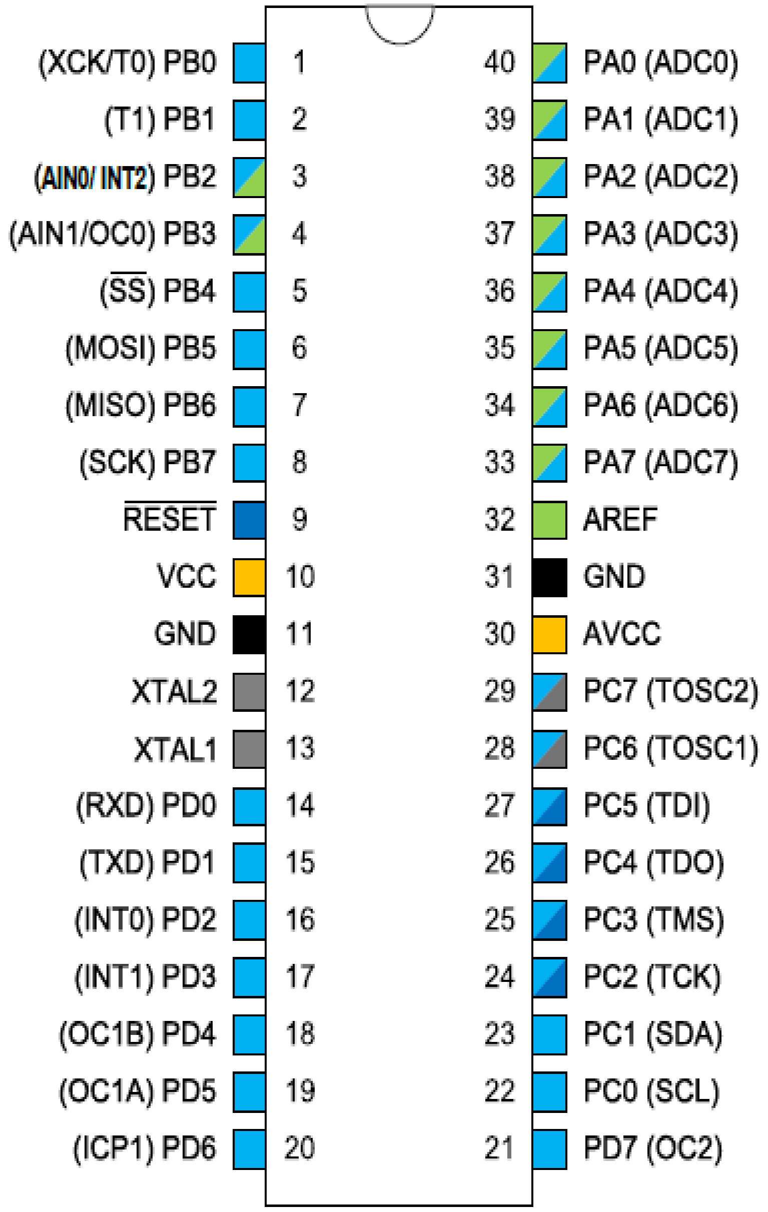
2- High level language like C , we write the code in c language in an IDE like atmel studio then it generate hex file we talk it and load to flash memory , we will talk about C for Embedded Systems next lecture .

Our microcontroller :
- Atmega 32 pinout :

- **Reset** : **This pin is used to reset uC , they make all registers = 0 reset the code which in uc memory .**
- **Vcc** : **Digital Supply Volt 2.7 : 5.5 v**
- **Gnd**
- **XTAL1 – XTAL2** : **To connect with External crystal which used in providing clock to uC**
- **Avcc** : **used to supplay ADC , must connect with Vcc**
- **Aref** : **we will talk about it in ADC lecture .**
- **The other pins are :**

Port (A – B – C – D) used to control electronic components read data (analog – digital) + alternative functions :

- Interrupts – communication protocols , timers , ...



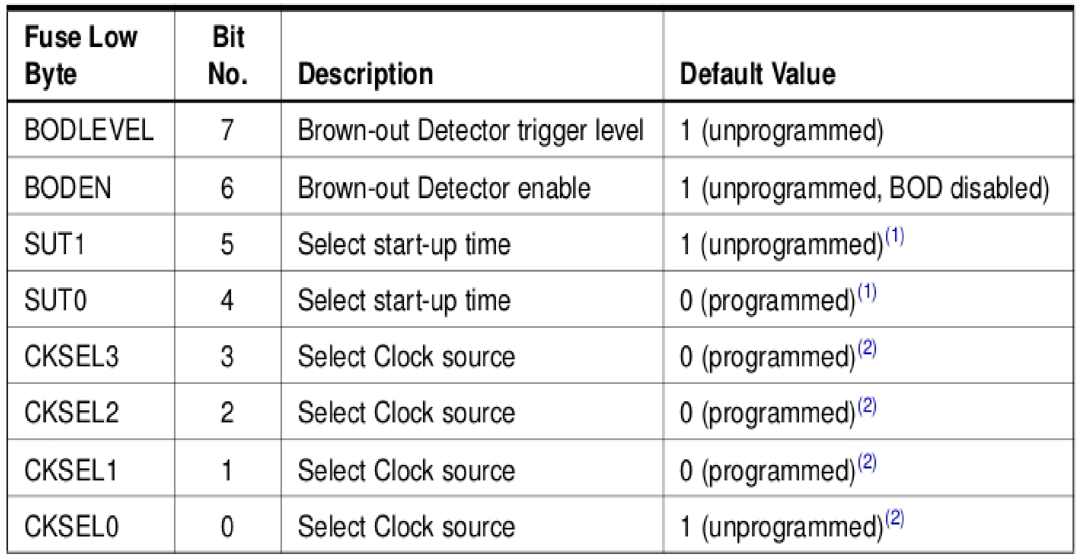
configuration bits

- In Every uCs , you must have a way to control the configuration even if we disconnect the power of uC .
- AVR uC have a default configuration for ex: default speed (1MHz) , you could change this speed to fit your application.
- Fuses : memory units programmed , its contents doesn't change by disconnecting the power divided into : Low fuse byte – High fuse byte - Extended fuse byte .
- If fuse programmed = 0 -   fuse unprogrammed = 1
- Warning :Take care when you deal with fuses .
- If you program the fuse bits in wrong way you can solve this by using high voltage programmer or fuse doctor circuit .
- Lock bits : these bits control the memory configuration :

- protection the flash program memory from copy or read , also you can save some parts of EEPROM Memory from writing to it , and prevent any person to know its contents .
- Customize a fixed part of memory (not accept any modification) after programming it like the bootloader in Arduino which is responsible for receive the hex file from serial port instead of SPI

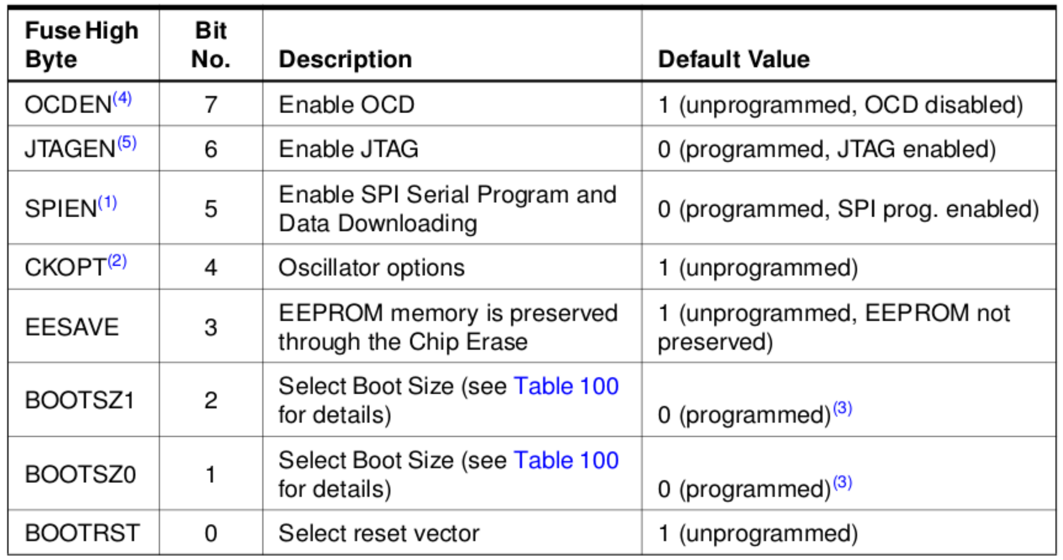
Fuses & Lock bits :

Table 106. Fuse Low Byte



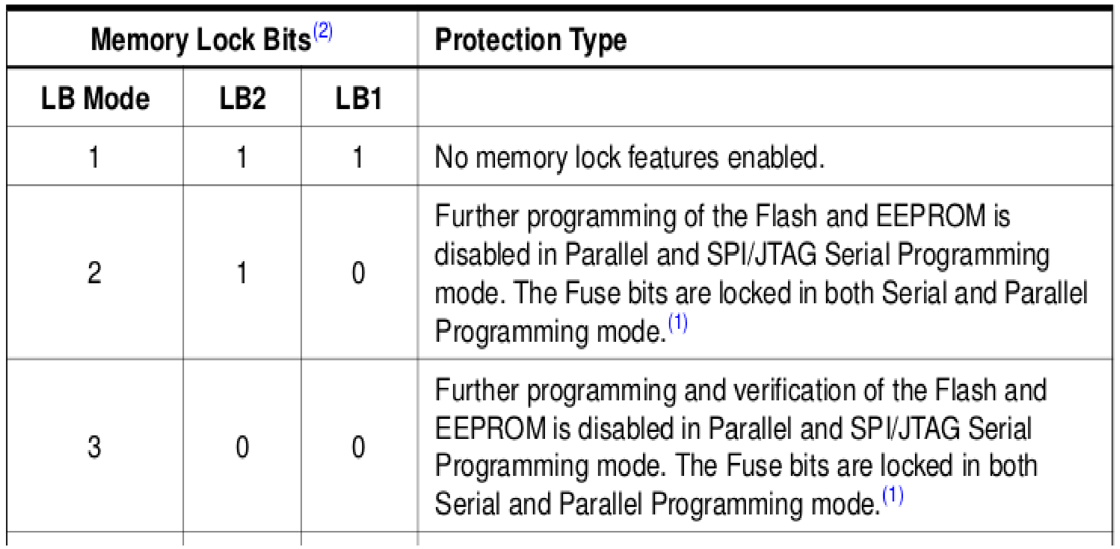
Fuse Low Byte	Bit No.	Description	Default Value
BODLEVEL	7	Brown-out Detector trigger level	1 (unprogrammed)
BODEN	6	Brown-out Detector enable	1 (unprogrammed, BOD disabled)
SUT1	5	Select start-up time	1 (unprogrammed)(1)
SUT0	4	Select start-up time	0 (programmed)(1)
CKSEL3	3	Select Clock source	0 (programmed)(2)
CKSEL2	2	Select Clock source	0 (programmed)(2)
CKSEL1	1	Select Clock source	0 (programmed)(2)
CKSEL0	0	Select Clock source	1 (unprogrammed)(2)

Table 105. Fuse High Byte



Fuse High Byte	Bit No.	Description	Default Value
OCDEN(4)	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN(5)	6	Enable JTAG	0 (programmed, JTAG enabled)
SPIEN(1)	5	Enable SPI Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
CKOPT(2)	4	Oscillator options	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	2	Select Boot Size (see Table 100 for details)	0 (programmed)(3)
BOOTSZ0	1	Select Boot Size (see Table 100 for details)	0 (programmed)(3)
BOOTRST	0	Select reset vector	1 (unprogrammed)

Table 104. Lock Bit Protection Modes



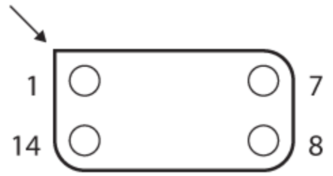
Memory Lock Bits(2)			Protection Type
LB Mode	LB2	LB1	
1	1	1	No memory lock features enabled.
2	1	0	Further programming of the Flash and EEPROM is disabled in Parallel and SPI/JTAG Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode.(1)
3	0	0	Further programming and verification of the Flash and EEPROM is disabled in Parallel and SPI/JTAG Serial Programming mode. The Fuse bits are locked in both Serial and Parallel Programming mode.(1)

Clock Source

- We measure the speed of processing in mega Hz – Giga Hz
- If We know that uC frequency = 1MHz that's mean it can execute 1MIPS , 1 Million Instructions Per Second .
- Our uC have internal RC oscillator (a circuit consist of Resistor + Capacitor) this is the cheapest way to obtain the frequency as you won't connect any additional component , its tolerance = +-3% so it isn't accurate in atomic or critical operations
- External RC Circuit : F = 1 / (R*C) – tolerance = +-5%
- External Crystal (Quartz) : The most common way and the best in most companies . Its tolerance = 0.00001 , it's more accurate than RC oscillator 1000 times .
- External Resonator = Crystal + additional capacitors , it's tolerance = 0.5%
- External Pure Pulse TTL :The most ever accurate oscillators , its tolerance = 0.000005

It's more accurate than Quartz crystal 20 times , but it's expensive .

Clock Sources

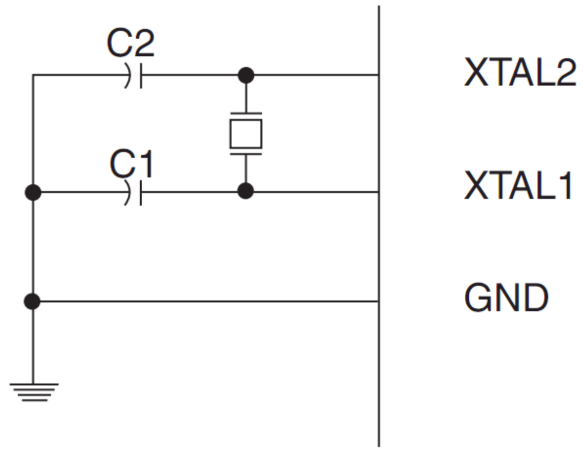


Pin	Function
1	NC
7	GND
8	Output
14	+5VDC

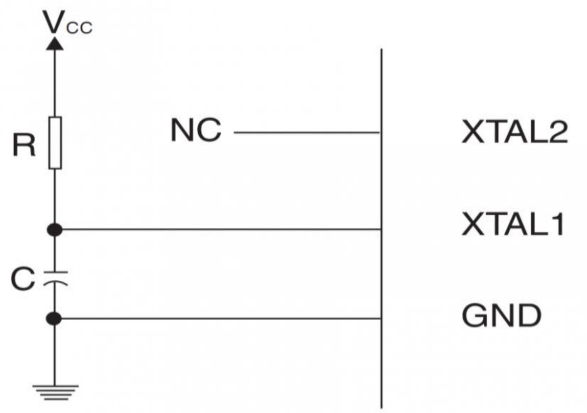
External Pure Pulse TTL

Crystal

Resonator



External quartz crystal



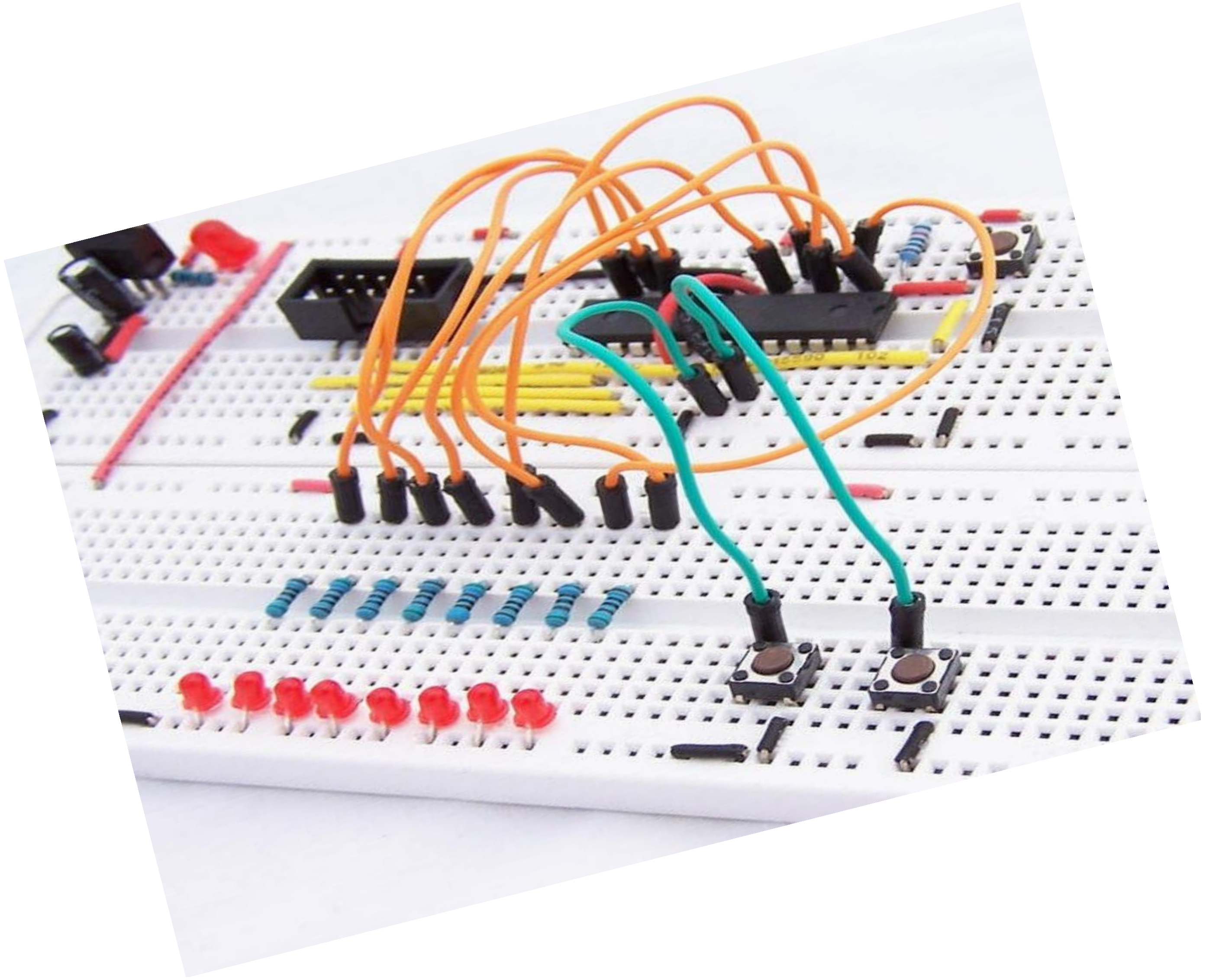
External RC Oscillator

Required Software

- Atmel Studio : to write the code , build it and generate the hex file which we can use in simulation or real project .
- Protues : to simulate the programs without real components
- Extreme burner : to load the hex file in the flash memory of uC , you can also edit fuses configurations
- Editors : notepad++ or sublime to write our libraries .
- Usbasp driver

Required Hardware

- Atmega32 uC
- 8 MHz crystal
- Led bar - Leds – push buttons
- Usbasp programmer
- Breadboard - wires - AVO
- Seven Segments
- BJT transistors : 2n2222
- DC motor
- Servo Motor
- Bluetooth Module
- Light Sensor (LDR)
- Temperature Sensor



References :

- Books :
- Simply AVR - > Abdallah Ali
- The AVR microcontroller & Embedded Systems using Assembly & c -- > Mazidi
- ATMEGA 32A Datasheet
- PIC microcontroller -- > Milan Verle
- Websites :
- https://www.sparkfun.com
- http://maxembedded.com
- https://www.tutorialspoint.com/cprogramming
- https://stackoverflow.com
- https://www.quora.com
- https://www.lucidchart.com

Any questions ?

- Instructor : Mohammed Hemed

- Embedded Systems developer at fab lab Ismailia

Repository link of Embedded workshop Material:

https://github.com/FabLab-Ismailia/Embedded-Systems-Workshop

Contact me :

- Gmail :

mohammedhemed23@gmail.com

- Linkedin :

https://www.linkedin.com/in/mohammedhemed

See you ....,