Getting Started with Micro-processors and Micro-controllers

Ajith Kumar B P

bpajith@gmail.com

4 Microcontrollers and Programming :

Microcontroller vs microprocessor, microcontrollers in embedded systems. Overview of AVR family of microcontrollers, simplified block diagram of AVR microcontroller, General idea of ROM, RAM, EEPROM, I/O pins and peripherals in microcontroller.

AVR architecture and Assembly level programming – General purpose registers, Data memory and instructions, status register and instructions, branch instructions, call and time delay loops; Assembler directives, sample programs.

Text : (Relevant sections from chapters 1,2 and 3: Textbook 4)

Arithmetic and logical instructions – sample programs. (16 hrs)

Text : (Relevant sections from chapters 5: The Book 4)

5. AVR Programming :

I/O programming, I/O port pins and functions, features of ports A, B, C and D, dual role of Ports, sample programs. I/O ports and bit addressability.

Text : (Relevant sections from chapter 4: Book 4)

AVR programming in C:

C language data types for AVR, C programs for arithmetic, logic time delay and I/O operations. (18 hrs)

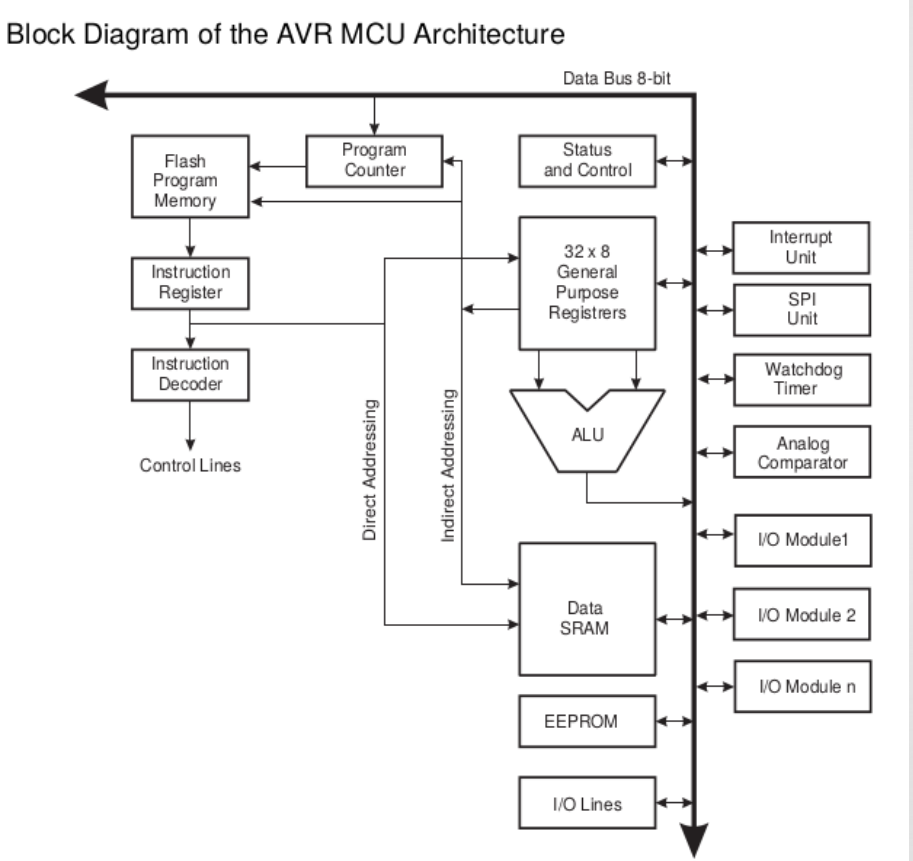
Text : (Relevant sections from chapter 7: Book 4)

Explaining the block diagram on the right covers the syllabus but it will just remain as words.

You need to write some code using Assembly language and execute it to really understand it.

How to achive that is our concern.

This class should be considered as an introduction only. We will do the follow up (including practicals) for those who are interested (well, others seems to lose the registration fee)



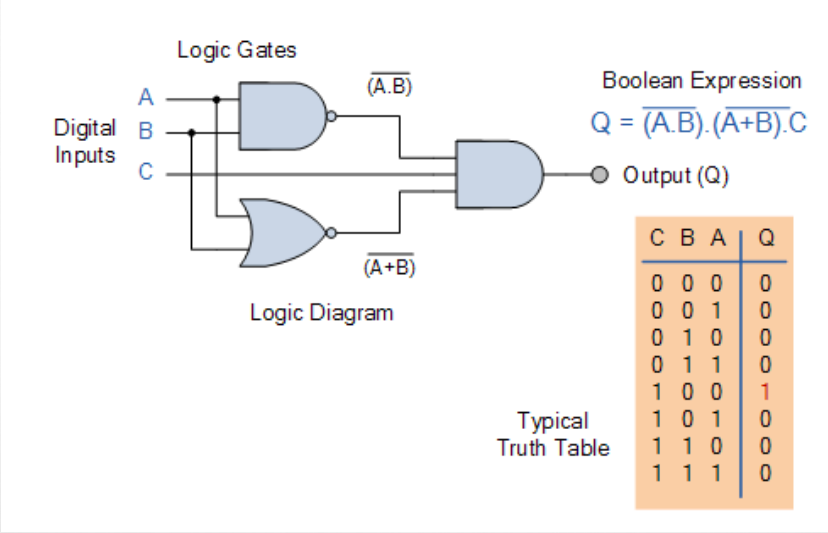
Contents

- Basic building blocks
  - Start with gates and flipflops to reach concepts like Register, Memory, Bus, CPU etc.
- Components of a Microprocessor
- How to prepare code (machine/assembly language)
- How to make it available for the processor
- Practical sessions

Coding in Assembly is a good way to learn the working of a Microcontroller.

Combinatorial Logic Circuits

- No “memory”, “timing” or “feedback loops”
- Operation is instantaneous.
- Performs an operation assigned logically by a Boolean expression or truth table.



Sequential Logic circuits

- Sequential circuit produces an output based on current input and previous input variables. SR flip-flop is an Asynchronous Sequential Circuit.

https://www.geeksforgeeks.org/introduction-of-sequential-circuits/

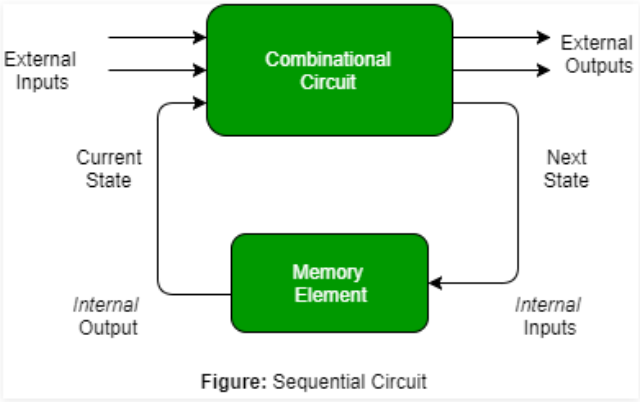
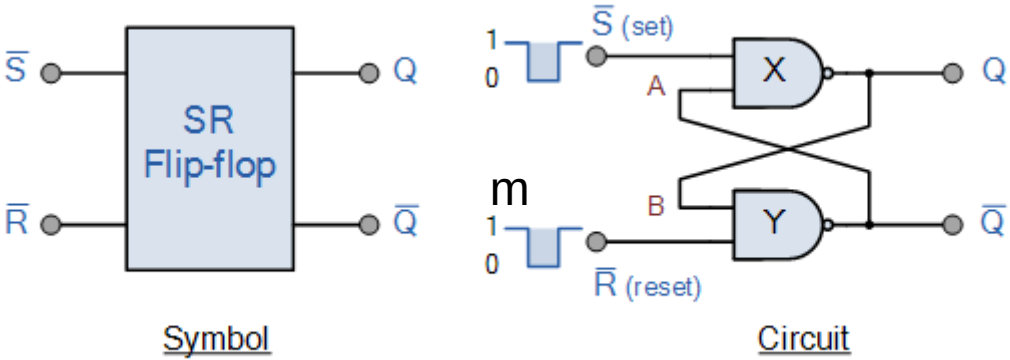
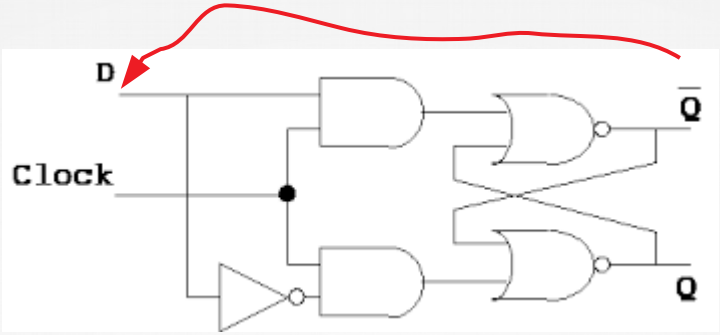


Figure: Sequential Circuit



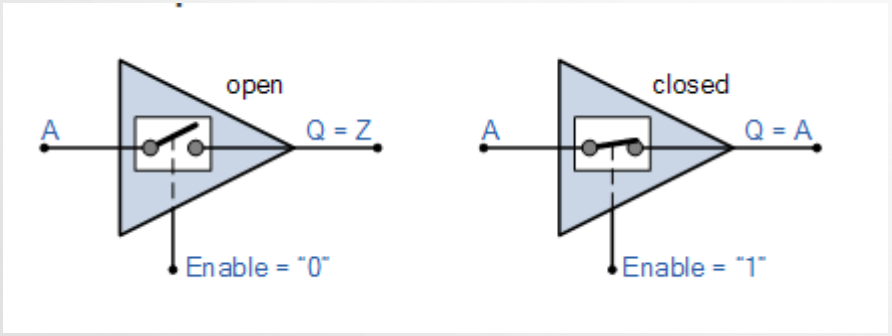
Synchronous Sequential Circuit.

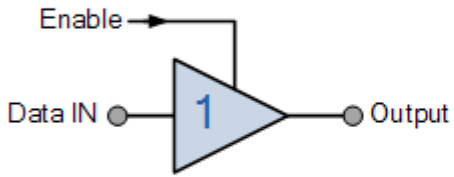


The input ‘D’ is the output of some other circuit. Any change in that may take some time to reach at this input. The clock allows waiting for stabilyzing the new level before accepting it. Imagine a number of such circuits in parallel representing a binary number.

Clocking allows us to build computers, which are state machines: they have a current state, and calculate their next state based on the current state and some inputs.

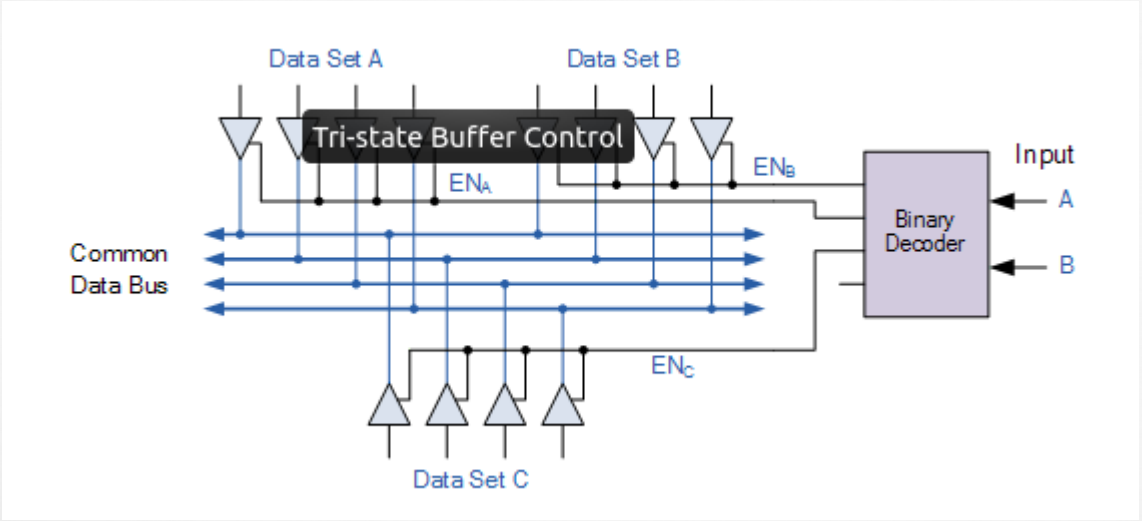
Tristate logic



Symbol	Truth Table		
	Enable	IN	OUT
	0	0	Hi-Z
	0	1	Hi-Z
	1	0	0
	1	1	1
Read as Output = Input if Enable is equal to "1"			

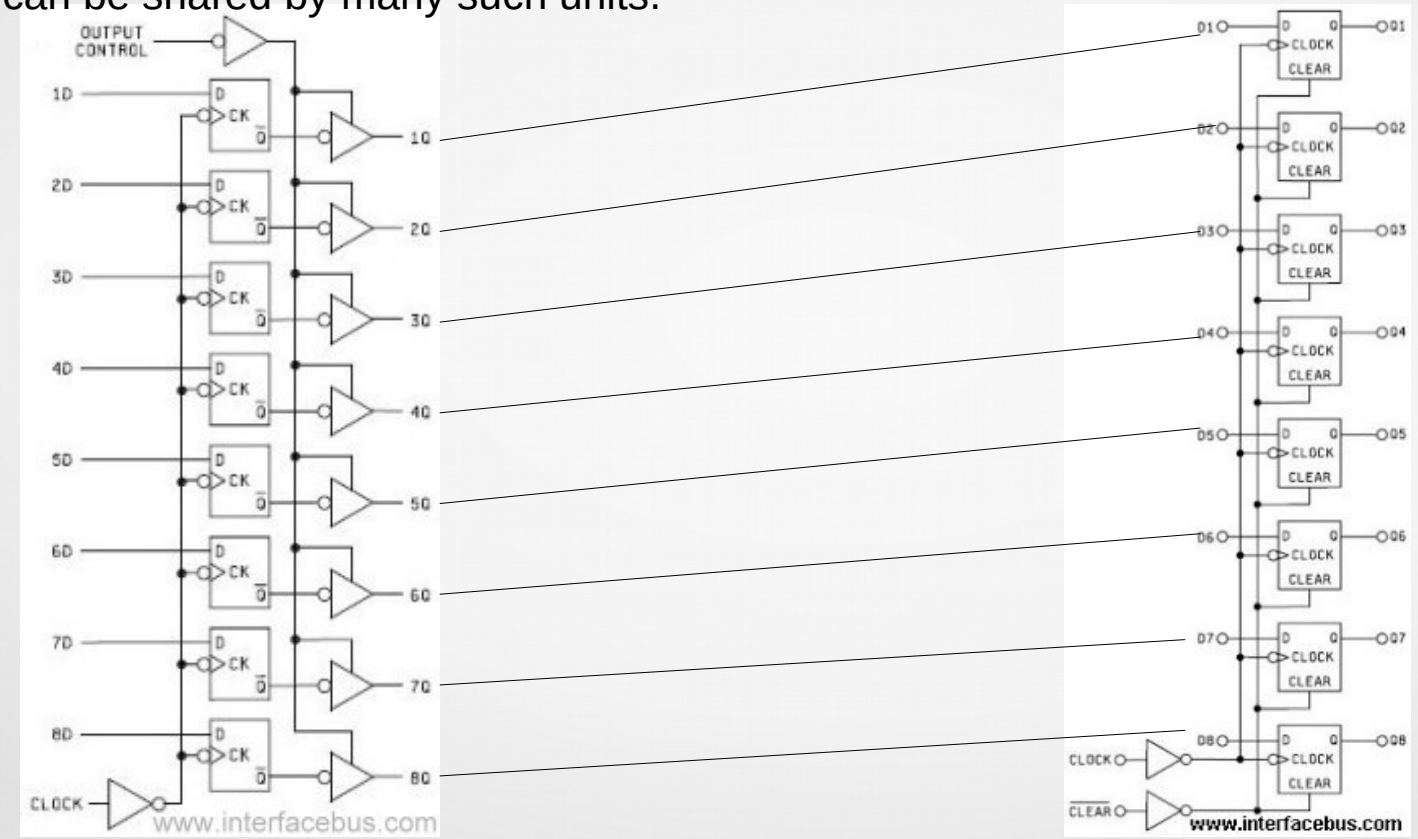
Tri-state Buffer

BUS Architecture



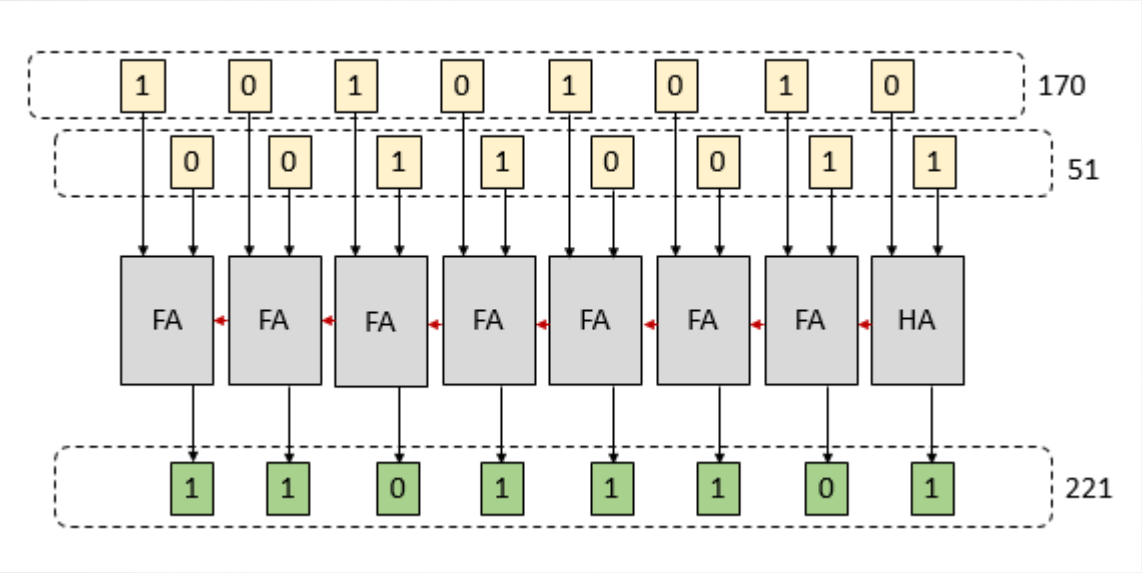
Data transfer

Enable Source OC ... wait .... apply Destination Clock edge
Lines can be shared by many such units.



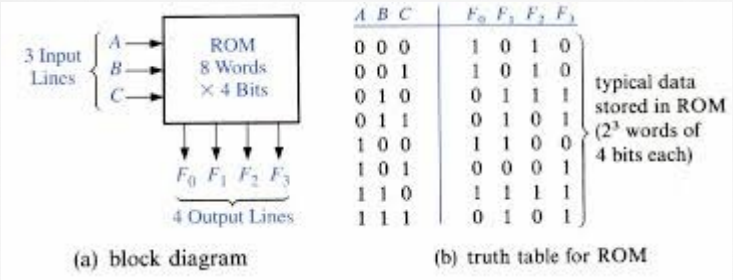
Register

8 bit Adder circuit



A CPU incorporates many components like this in an ordered manner. The clock signal controls timing of the sequence of operations.

Lookup table



In a lookup table a specific number generates another predefined number, where the numbers are a specific combination of bits.

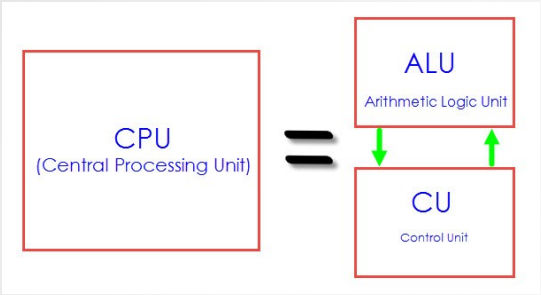
What is happening inside a CPU is somewhat similar to this. A particular combination of bits (called the Instruction) trigger a sequence of actions.

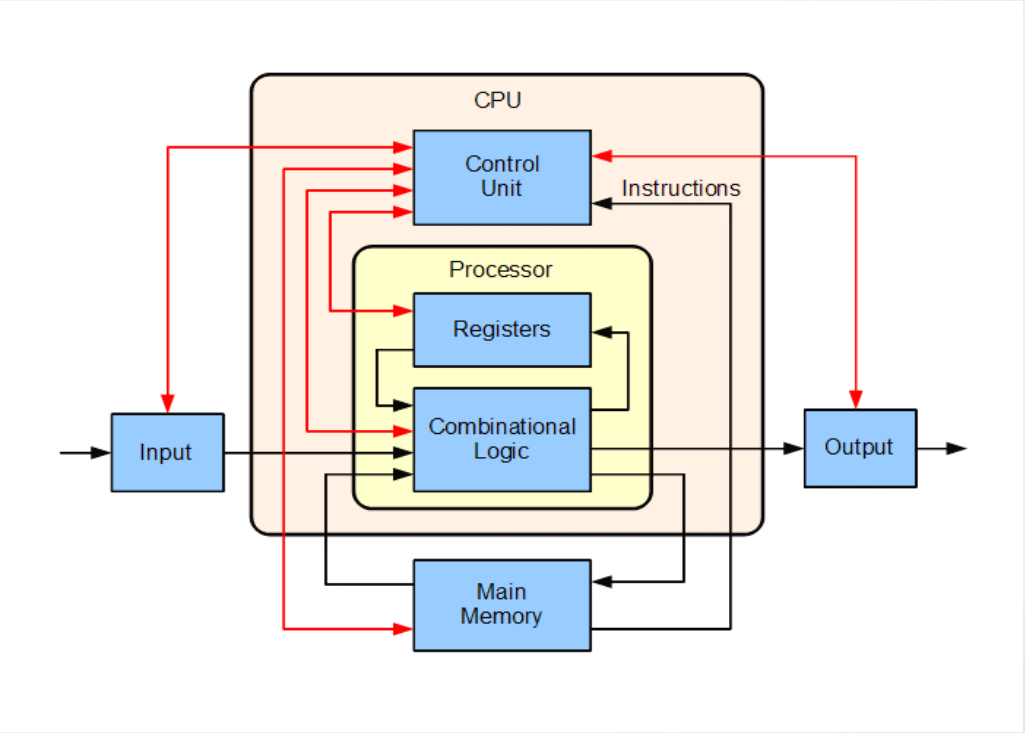
Von Neumann Architecture

Architecture upon which nearly all digital computers have been based

- A single, centralized control, housed in the central processing unit
- A separate storage area, primary memory, which can contain both instructions and data.
- The Instructions are executed by the CPU, and so they must be brought into the CPU from the primary memory.
- The CPU also houses the unit that performs operations on operands, the arithmetic and logic unit (ALU), and so data must be fetched from primary memory and brought into the CPU in order to be acted upon.
- The primary memory has a built-in addressing mechanism, so that the CPU can refer to the addresses of instructions and operands.
- The CPU contains a register bank that constitutes a kind of “scratch pad” where intermediate results can be stored and consulted with greater speed than could primary memory.

Inside the CPU





8085 Microprocessor

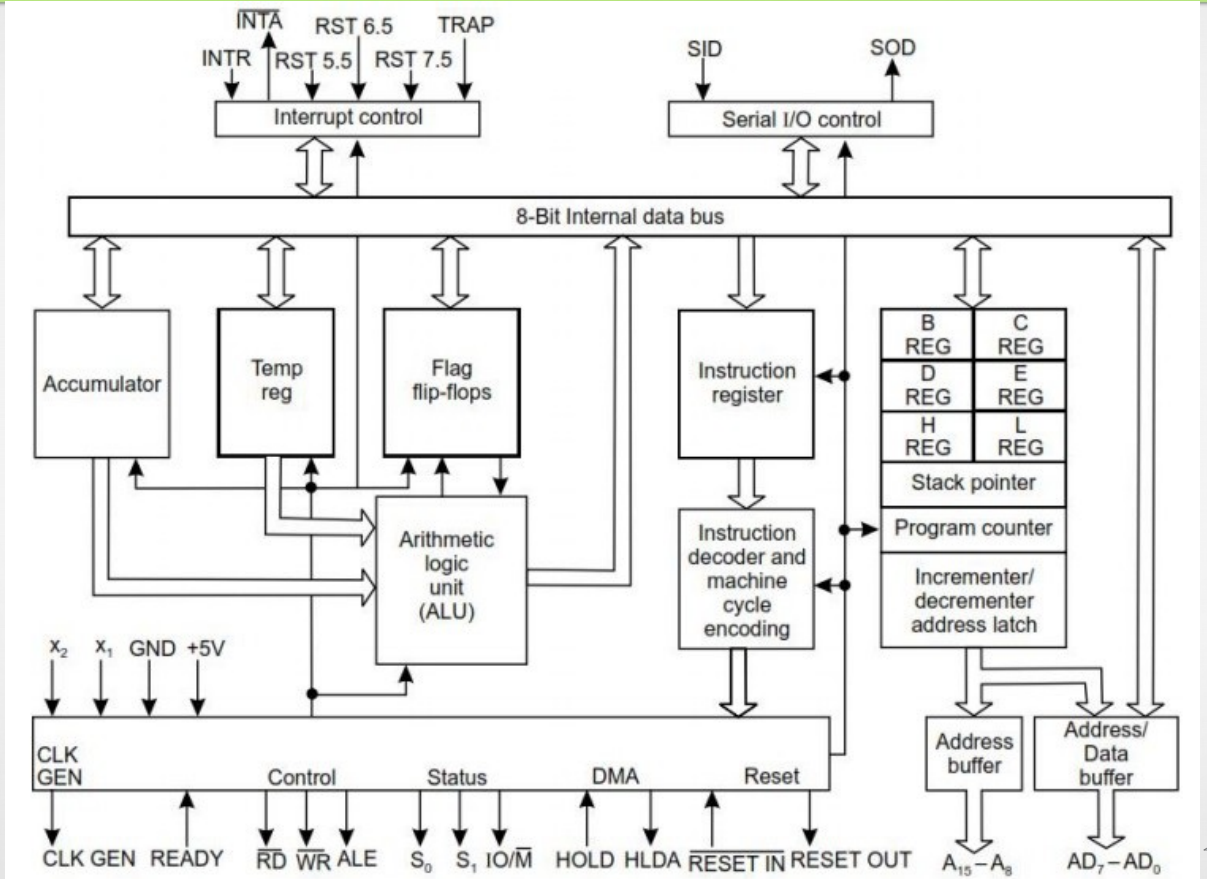
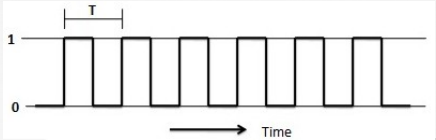


Fig 1.1 Hardware Architecture of 8085

External Memory

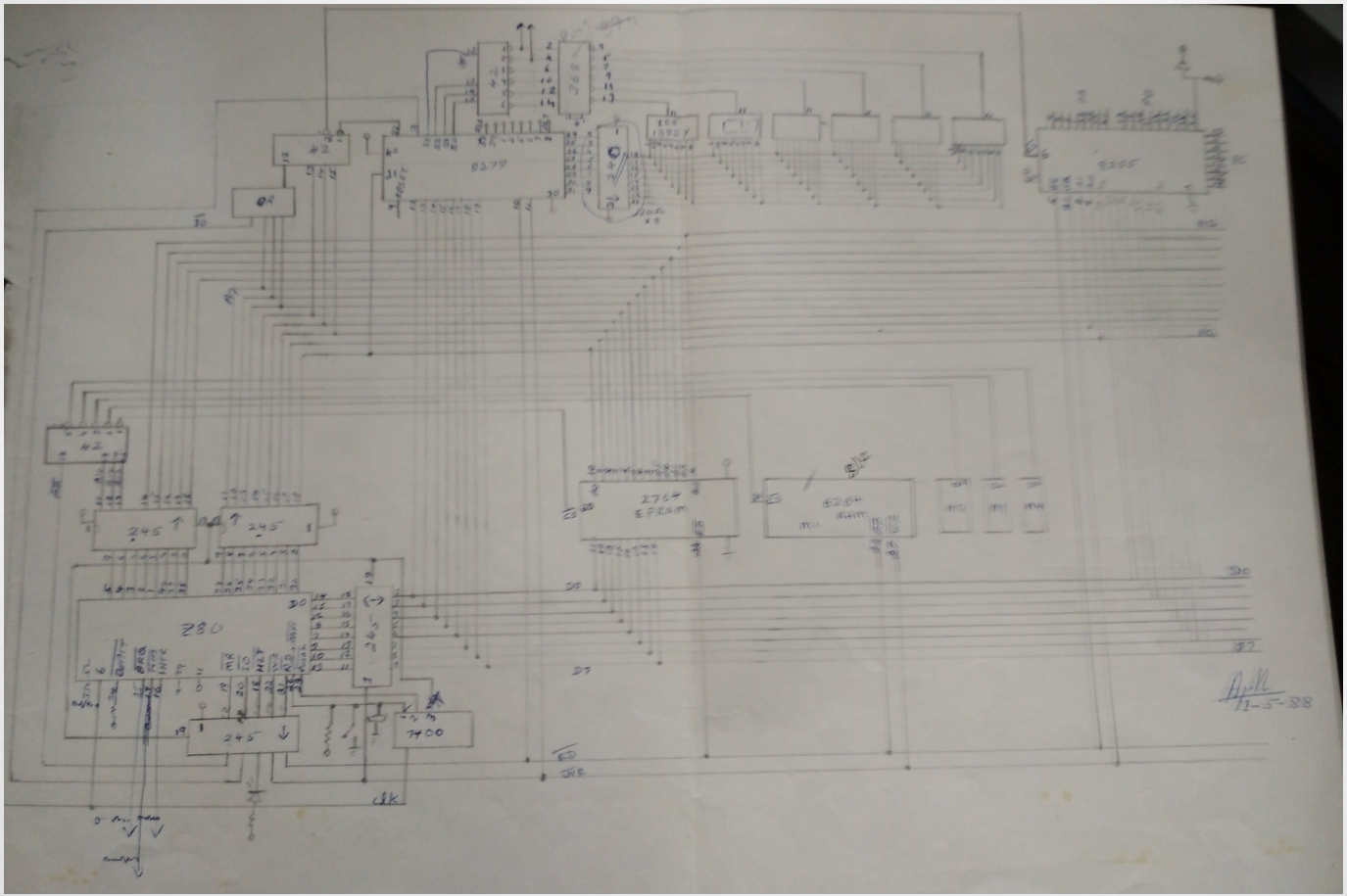
When power is applied or CPU is reset



- The clock starts running.
- Fetch the instruction from the starting address of Main Memory (it need not be zero)
- If needed required data also from memory
- Carry out the Instruction.
- Store the result to memory if specified.
- Proceed to the next instruction

In infinite loop is generally implemented.

Z80 Microprocessor based system



Load
INC
OUT
Jump

Implemented
8 LED Counter

UVEPROM programming tools

EPROM
Programmer

UV Lamp

How to Program a Microprocessor ?

To prepare the code, one need to know about the following:

- Instructions (in Machine, Assembly or some high level language)
- Registers (Need to know the details for machine/assembly coding)
- External Memory (The address range of external memory)
- I/O peripherals

Write the Main memory with the Machine Language Instructions (Opcodes). Using EPROM for a very elementary system. Write the EPROM using a Programmer, insert it in the circuit and power it.

- A more complex system can have a monitor program that will allow the user to enter the instructions at some memory location and then transfer control to that location (Development kits follow this method).

Machine language instructions...

Opcodes Table of 8085 Microprocessor

Here is the complete list of the **instruction set of 8085 microprocessor**.

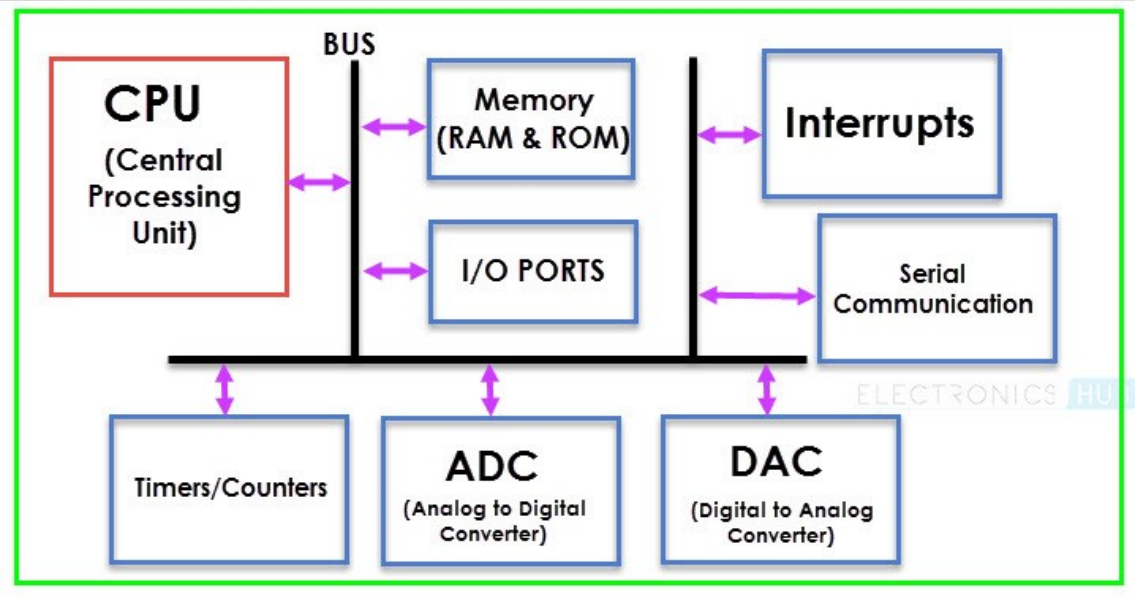
S.No.	Mnemonics, Operand	Opcode	Instruction Word Size
1.	ACI Data	CE	2 Bytes
2.	ADC A	8F	1 Byte
3.	ADC B	88	1 Byte
4.	ADC C	89	1 Byte
5.	ADC D	8A	1 Byte
6.	ADC E	8B	1 Byte
7.	ADC H	8C	1 Byte
8.	ADC L	8D	1 Byte
9.	ADC M	8E	1 Byte
10.	ADD A	87	1 Byte
11.	ADD B	80	1 Byte

Assemblers & Compilers

- Use easy to remember mnemonics
- Assemble translates them into machine language
- Code is written for a specific processor.

High level languages provides a more abstract layer. Concepts like variables, datatypes, control statements etc. are introduced. The same source code can be translated into the machine language of different type of processors, using different compilers.

Microcontrollers (CPU + Memory + Peripherals)

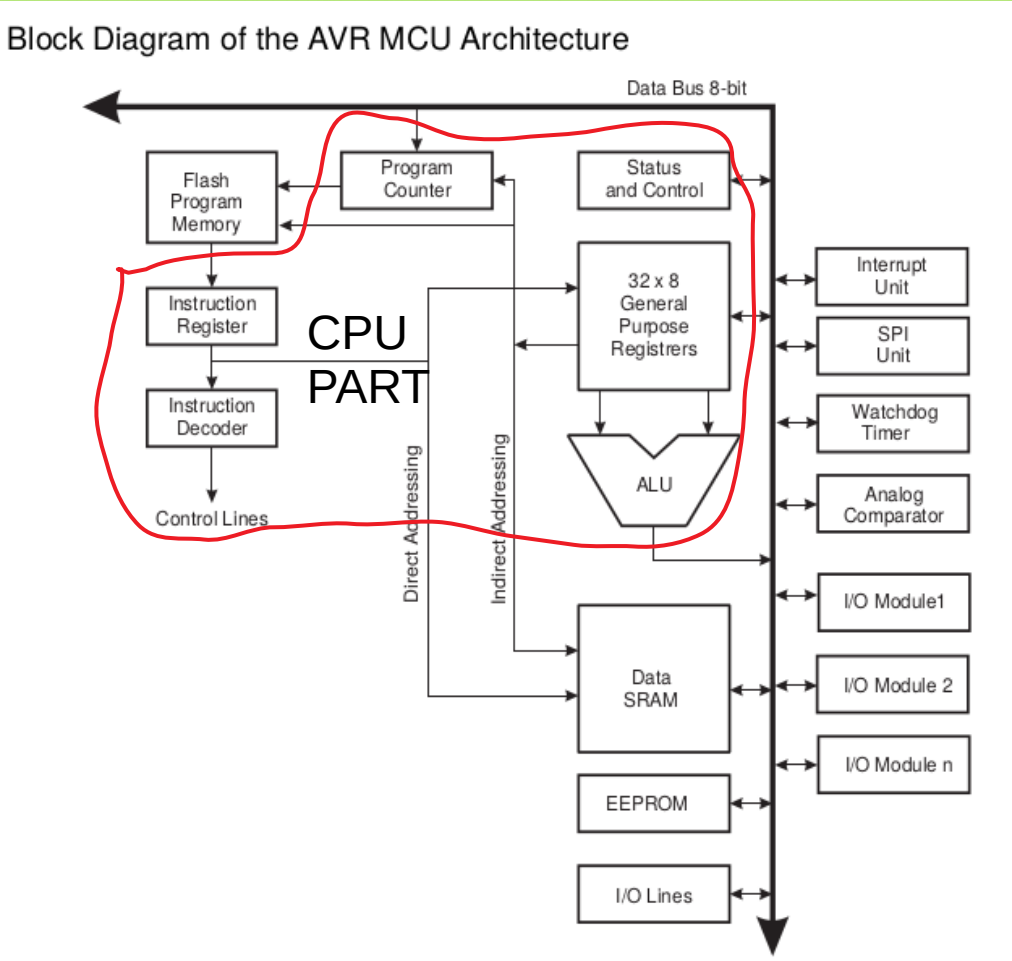


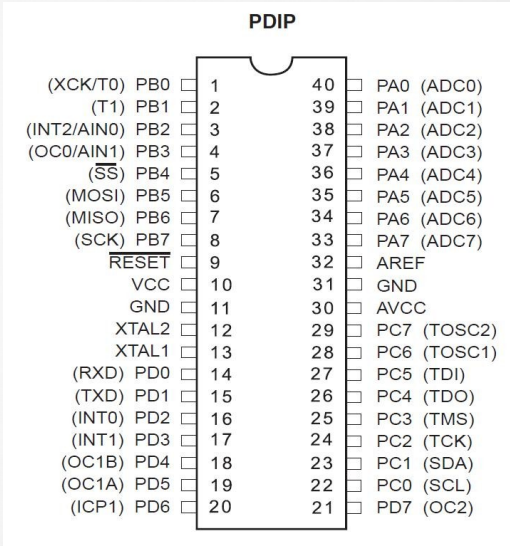
The earliest known Microprocessors are the Intel's 4004 and the Texas Instruments' TMS1000.

The important ones produced by Intel are the 8048 and the 8051.

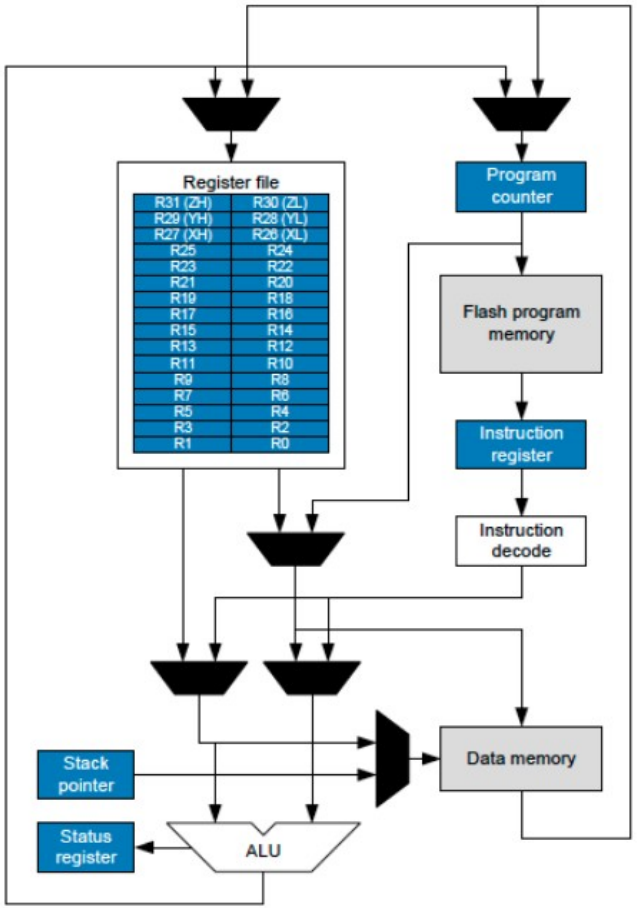
AVR and PIC are the modern ones

AVR Series microcontrollers (ATMega32)





How it works ?



- Fetch the Instruction from the Program Memory (PC location)
- Data may be brought from the Data Memory
- Execute it. The General Purpose Registers may be involved.
- Jump Instructions modify the PC
- Data may be pushed to some location is Data Memory using the Stack Pointer (SP)

CPU Registers

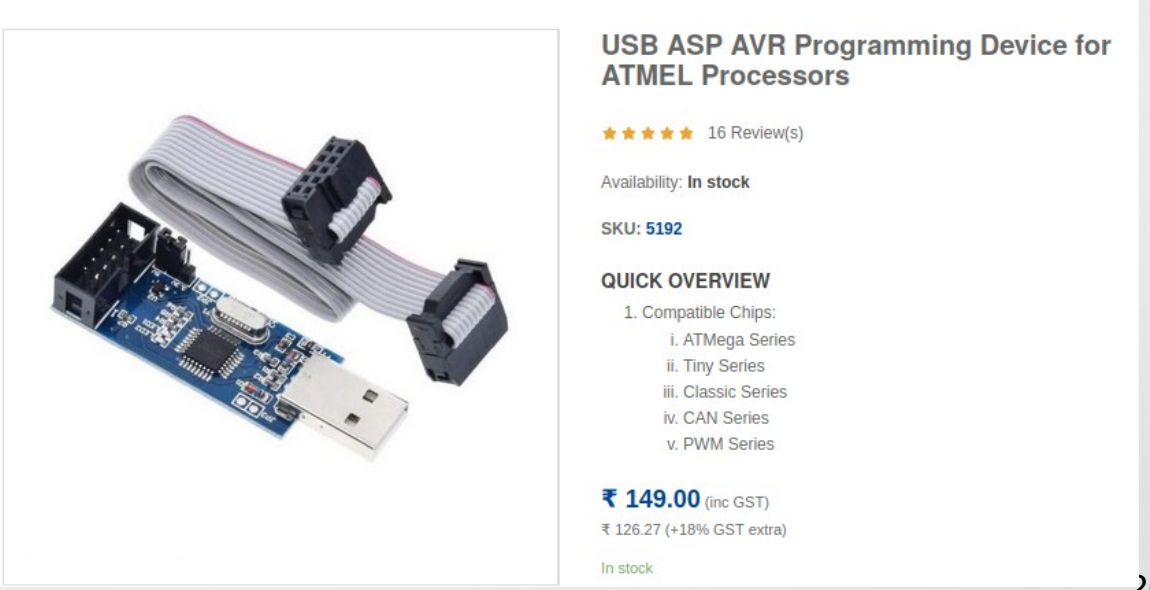
- Can be used directly in assembler commands,
- Operations with their content require only a single command word,
- Are connected directly to the CPU, and are source and target for calculations.
- There are 32 registers in an AVR. They are originally named R0 to R31
-
- Only the registers from R16 to R31 load a constant immediately with the LDI command, R0 to R15 don't do that

Howto put Code into the Program Memory ?

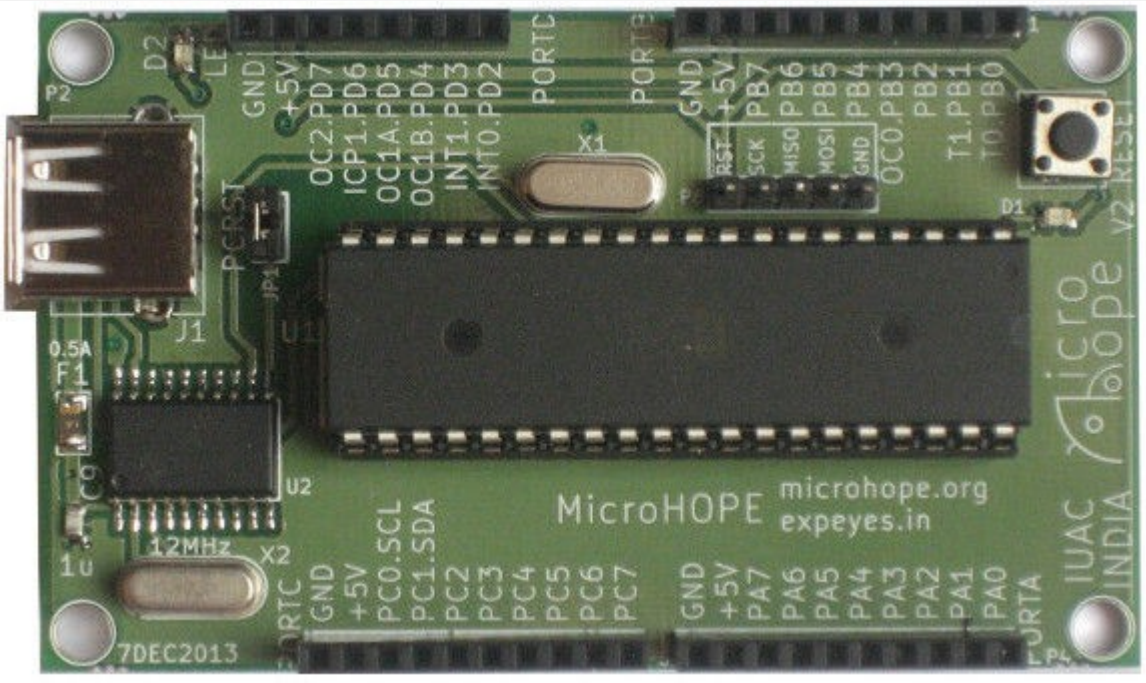
- All modern micro-controllers uses Flash Memory as Program Memory
- This Memory can be accessed using SPI Interface after holding the CPU in the Reset mode.
- SPI has three pins
  - MISO master in slave out
  - MOSI master out slave in
  - SCK serial clock

Use an SPI Programmer that can
Transfer code from a PC to
An AVR processor using
The SPI Interface.

https://robu.in



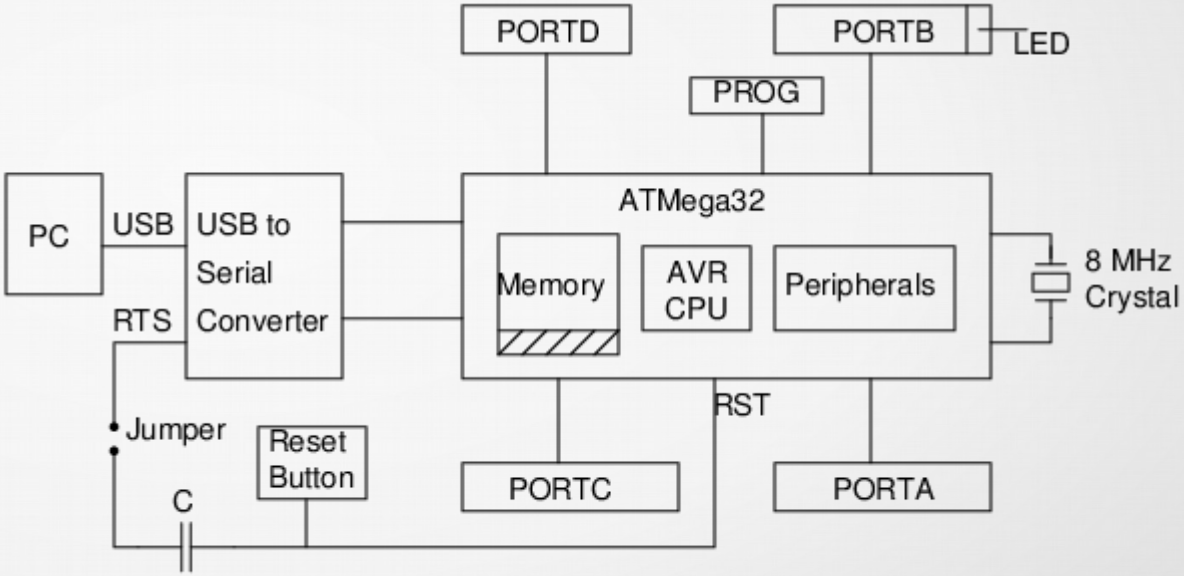
Atmega32 Microcontroller



Bootloader Programs

https://expeyes.in/MicroHOPE/mh-hardware.html

- On a Reset Bootloader Runs
- Waits for New Code via the serial port
- If received, loads it to the Program Memory and transfers control to it.
- Otherwise runs the existing code in the Program Memory.



Simple Examples in Assembler

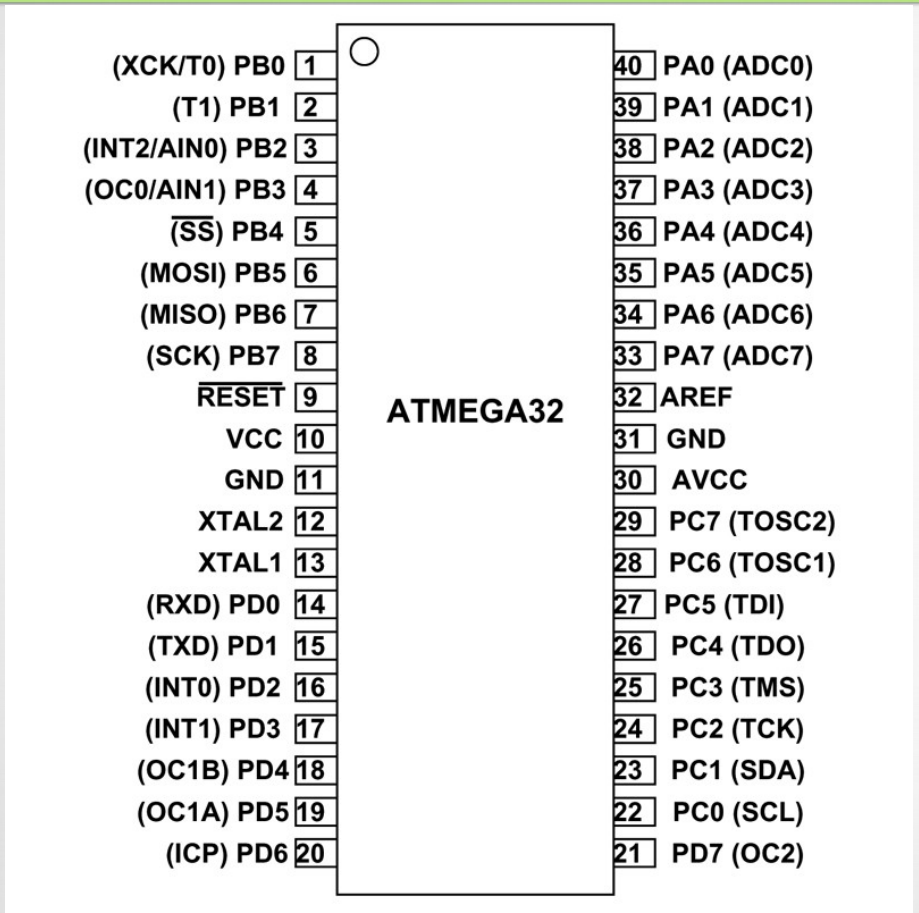
```
; program add.S
; Load 2 registers with some values and add them

.section .text    ; denotes code section
.global main

main:
    LDI    R16, 2        ; load R16 with 2
    LDI    R17, 4        ; load R17 with 4
    ADD    R16, r17      ; R16 <- R16 + R17
    .END
```

Where is my Output ???

I/O ports of Atmega32



To view the program output, connect LEDs to the I/O Ports and send results to them...

Special Function Registers of Atmega32

Configuring the I/O Digital Pins

Each port consists of three registers:

- **DDRx** – Data Direction Register
- **PORTx** – Pin Output Register
- **PINx** – Pin Input Register

where x = Port Name (A, B, C or D)

Name: DDRB
Offset: 0x24
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x04

Bit	7	6	5	4	3	2	1	0
	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – DDRBn: Port B Data Direction [n = 7:0]

Modified Program for Adding two numbers

```
; program add.S
; Load 2 registers with some values and add them. Display result on port B

#include <avr/io.h>
      .section .text    ; denotes code section
      .global main
main:
     LDI    R16, 255    ; load R16 with 255
     STS    DDRB, R16    ; set all bits of port B as output
     LDI    R16, 2       ; load R16 with 2
     LDI    R17, 4       ; load R17 with 4
     ADD    R16, r17    ; R16 <- R16 + R17
     STS    PORTB, R16    ; result to port B
     .END
```

KuttyPy schematic