CH – 5
AVR Microcontroller Internal Hardware & Programming

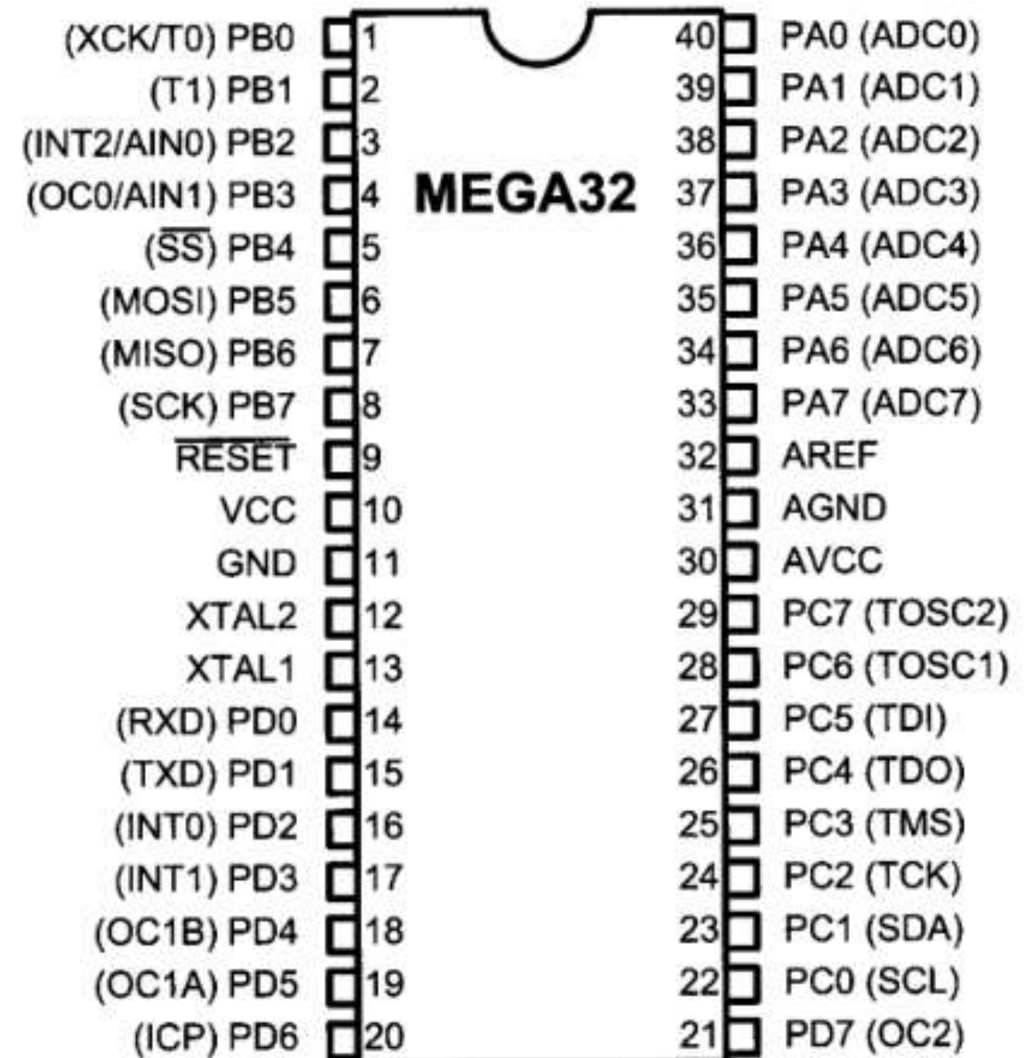
AVR I/O Port Programming

- In AVR microcontroller family, there are many ports available for I/O operations, depending on which family microcontroller you choose.
- For the ATmega32 40-pin chip 32 Pins are available for I/O operation. The four ports PORTA, PORTB, PORTC, and PORTD are programmed for performing desired operation.

The number of ports in AVR family varies depending on number of pins available on chip.

Pins	8-pin	28-pin	40-pin	64-pin	100-pin
Chip	ATtiny25/45/85	ATmega8/48/88	ATmega32/16	ATmega64/128	ATmega1280
Port A			X	X	X
Port B	6 bits	X	X	X	X
Port C		7 bits	X	X	X
Port D		X	X	X	X
Port E				X	X
Port F				X	X
Port G				5 bits	6 bits
Port H					X
Port J					X
Port K					X
Port L					X

Note: X indicates that the port is available.



- The 40-pin AVR has four ports for using any of the ports as an input or output port, it must be accordingly programmed.
- The Registers Addresses for ATmega32 Ports is given below:
  PORTx,
  DDRx
  PINx

- Each port in AVR microcontroller has three I/O registers associated with it. They are designated as.

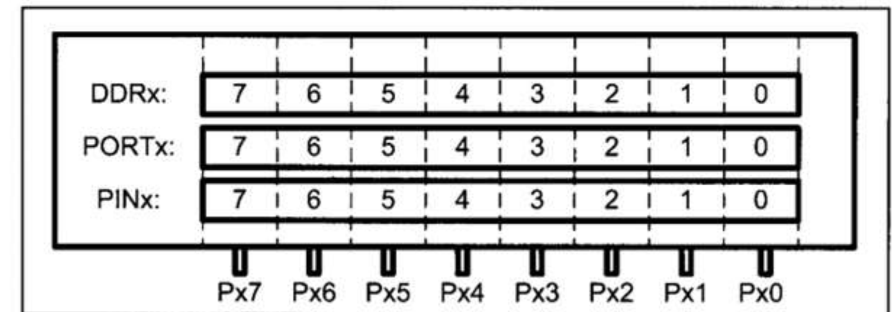
- Each of I/O registers is 8 bits wide, and each port has a maximum of 8 pins, therefore each bit of I/O registers affects one of the pins.

- For accessing I/O registers associated with the ports the common relationship between the registers and the pins of AVR microcontroller

Table 4-2: Register Addresses for ATmega32 Ports

Port	Address	Usage
PORTA	$3B	output
DDRA	$3A	direction
PINA	$39	input
PORTB	$38	output
DDRB	$37	direction
PINB	$36	input
PORTC	$35	output
DDRC	$34	direction
PINC	$33	input
PORTD	$32	output
DDRD	$31	direction
PIND	$30	input

The relation between the Registers and the Pins of AVR is shown below:



DDRx: Data Direction Register

- Before reading or writing the data from the ports, their direction needs to be set. Unless the PORT is configured as output, the data from the registers will not go to controller pins.
- This register is used to configure the PORT pins as Input or Output.
- Writing 1's to DDRx will make the corresponding PORTx pins as output.
- Similarly writing 0's to DDRx will make the corresponding PORTx pins as Input.

Note: DDRx (X – any port from (PORT A ,PORT B PORT C, PORT D)

DDRx	Port Type
1's	O/P
0's	I/P

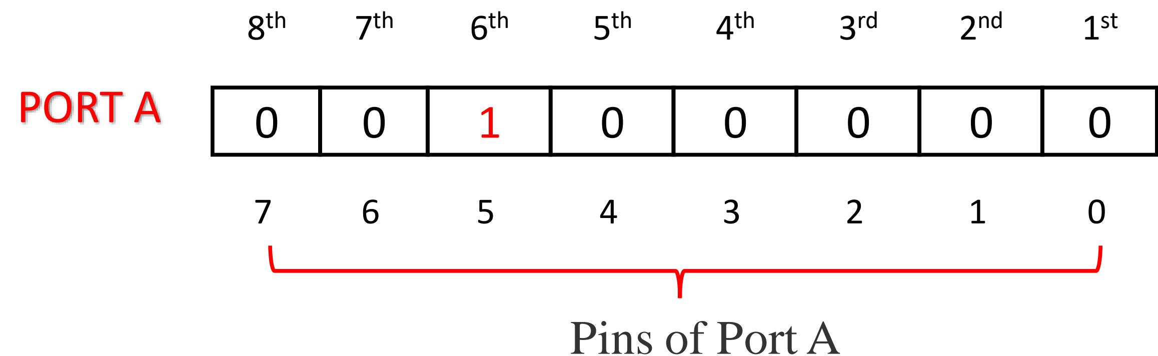
Example :
1. if we want to make port B as Output : DDRB = 0xFF;
2. If we want to make 6th pin of port A is 0 : DDRA.5 = 0x40;

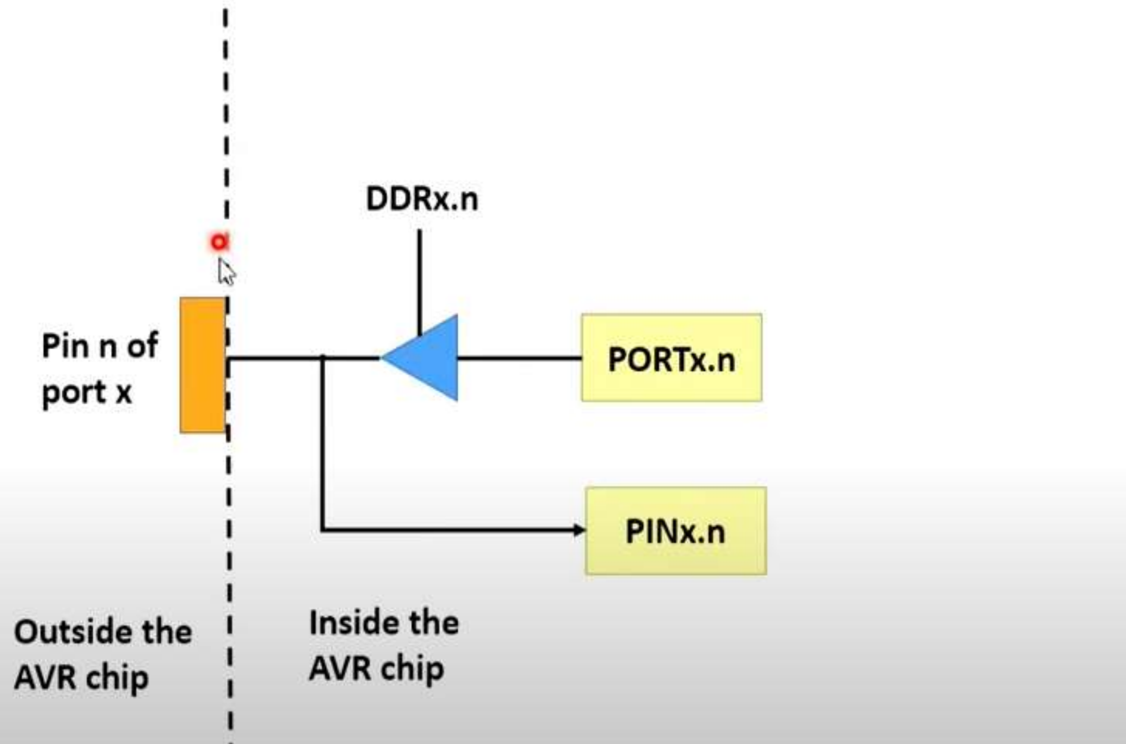
Example :

```
1.DDRB = 0xff; // Configure PORTB as Output.

2.DDRC = 0x00; // Configure PORTC as Input.

3.DDRD = 0x0F; // Configure lower nibble of PORTD as
                  Output and higher nibble as Input
```





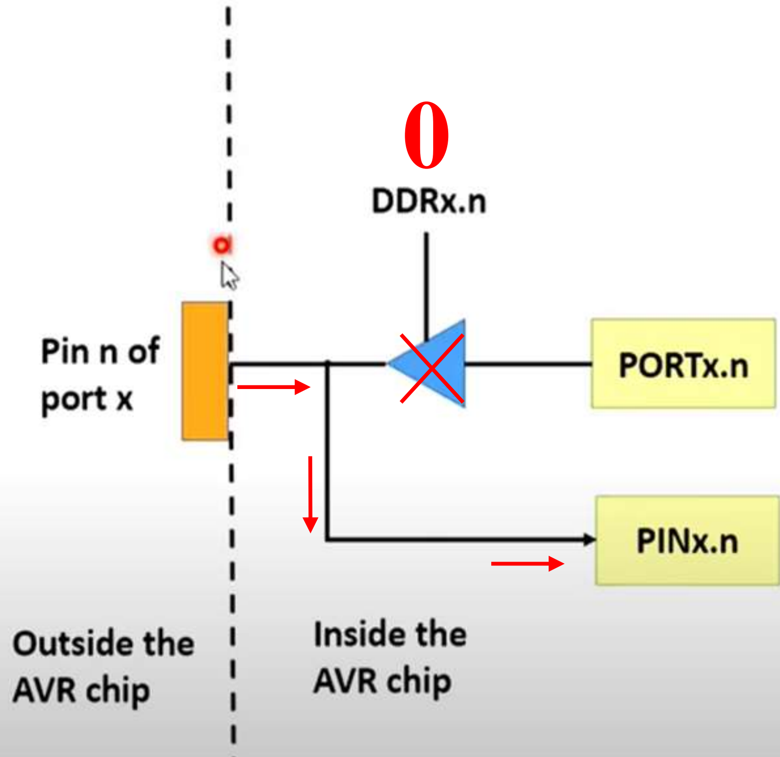
- DDRx register is 8 bit internal register of AVR. The role of DDRx register is to set physical pin of port input or output.
- 1 = O/P Pin
- 0 = I/P Pin

- PORT register is 8 bit internal register of AVR. The role of PORT register is to send the data on pin n of port x.

- Any output device i.e. LED or LCD is connected with physical pin will turn ON or OFF by sending data through PORTx register.

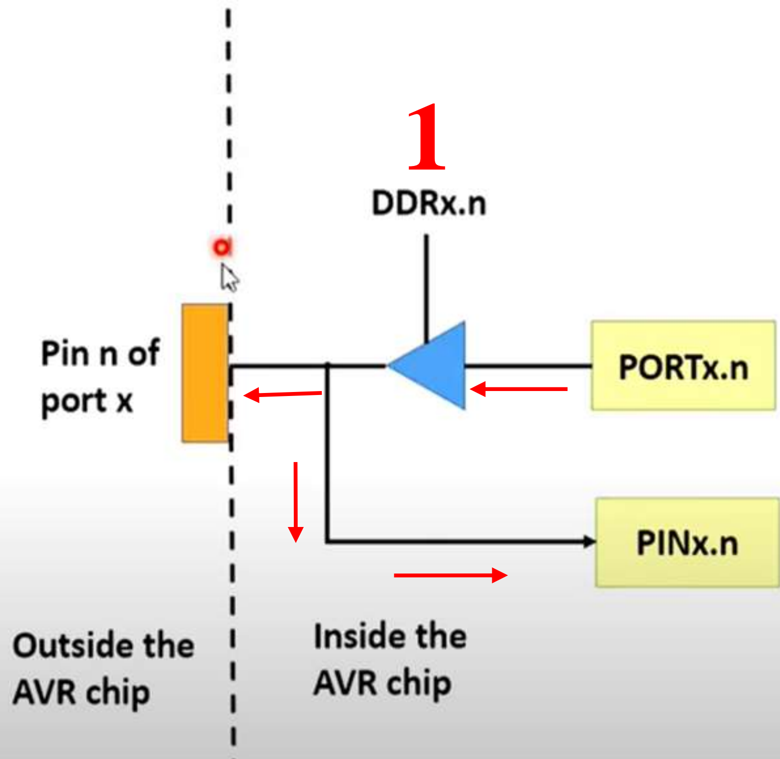
- PINx is also 8 bit internal register. The role of this register is to read data from input device like switch PUSH button which is connected with physical pin of AVR.

I/P port working :



- The role of DDRx register is to set physical pin of port input or output. 1 = O/P Pin 0 = I/P Pin
- When I/P device is connected with physical pin of AVR, put 0 in DDRx register to make that port as an input port.
- Tristate buffer which is connected with DDRx register will get disable as soon as it received 0.
- Once it get disable the data from input device will come in to the PINx register.
- Hence we can read the status of an input device.

O/P port working :



- The role of DDRx register is to set physical pin of port input or output. 1 = O/P Pin 0 = I/P Pin
- When O/P device is connected with physical pin of AVR, put 1 in DDRx register to make that port as an output port.
- Tristate buffer which is connected with DDRx register will get enable as soon as it received 1.
- Once it get enable the data from an output device will transfer on the device through PINx register.
- Hence we can send the data to an output device.

PORTx: PORTx register can be used for two purposes:

1. To output data:

- when port is configured as output then PORTx register is used.
- When we set bits in DDRx to 1, corresponding pins becomes output pins.
- Now we can write the data into respective bits in PORTx register.
- This will immediately change the output state of pins according to data we have written on the ports.

For example:

1. To output data in variable x on port A

```
1.DDRA = 0xFF;        //make port A as outputs
2.PORTA = x;          //output variable on port
```

2. To output 0xFF data on port B

```
1.DDRB = 0b11111111;    //set all the pins of port B as outputs
2.PORTB = 0xFF;         //write the data on port
```

3. To output data on only 0th bit of port C

```
1.DDRC.0 = 1;      //set only 0th pin of port C as an output
2.PORTC.0 = 1;     //make it high signal.
```

PINx register:

- PINx register used to read the data from port pins. In order to read the data from port pin, first we have to change the portxs data direction to input.

- This is done by setting bits in DDRx to zero. If port is made output, then reading PINx register will give a data that has been output on port pins.

- There are two input modes.

- Either we can use port pins as internal pull up or as tri stated inputs. It will be explained as shown below:

- For reading the data from port A.

```
1.DDRA = 0x00;    //Set port A as input
2.x = PINA;       //Read contents of port a
```

Data types and directives

AVR Data types

AVR data types

- ❖ **It has only 8 bit data type as all registers are of 8 bits.**

AVR data format representation

- Byte in AVR can be represented in 4 ways

1) Hex numbers

Ex:-

0X15

$15

2) Binary Numbers

0b10100011

0B00010011

3) Decimal numbers

Ex:-

15

2

4) ASCII Numbers

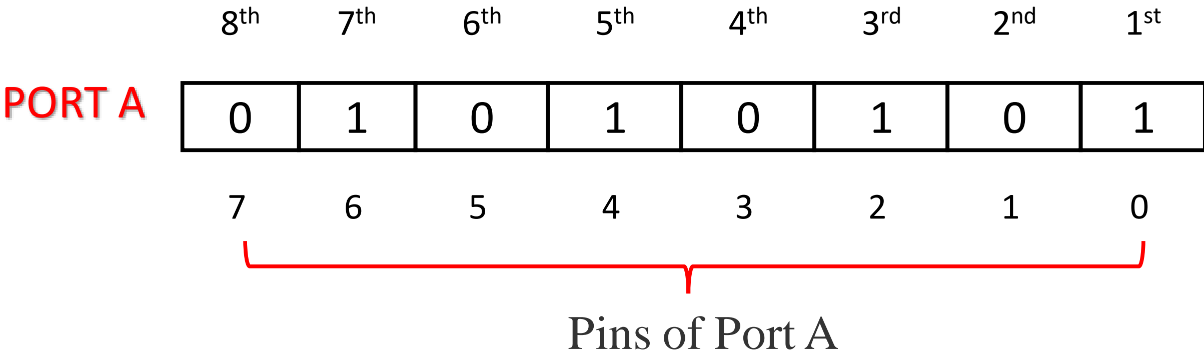
'9' = 0x39

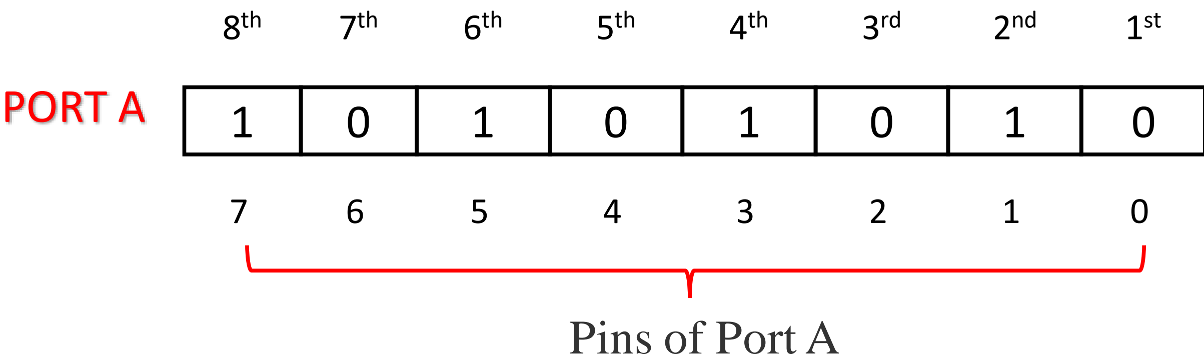
'1' =0x31

'A' =0x41

PORT A as an output port Sending 55 and AA continuously.

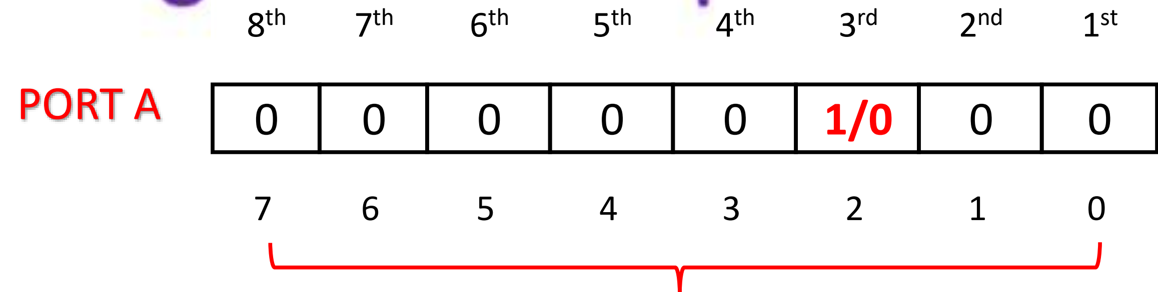
.Include “M32DEF.INC”
LDI R15, 0xFF
OUT DDRA, R15
REPEAT:LDI R16, 0x55
OUT PORTA, R15
CALL DELAY
COM R15
OUT PORTA, R15
CALL DELAY
RJMP REPEAT



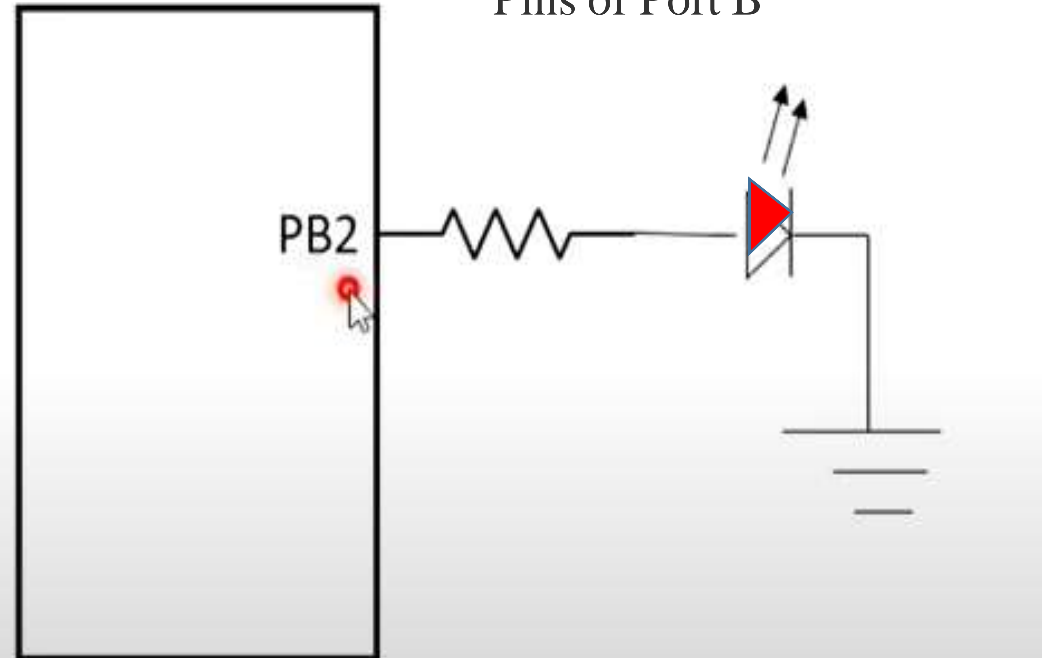


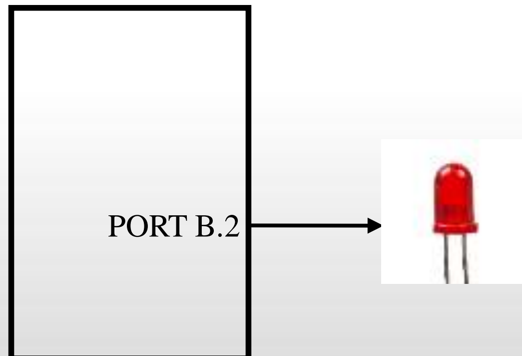
1. Make Port A as an output port
2. Load 0xff into DDRA register through any general purpose register
3. Send 55 on port through OUT instruction
4. Call delay
5. Send AA on the same port through OUT instruction
6. Call delay
7. Repeat all steps

Turn led connected on PB2 pin on and off with some random delay without disturbing rest of the pins.



SBI DDRB, 2
AGAIN: SBI PORTB, 2
CALL DELAY
CBI PORTB, 2
CALL DELAY
RJMP AGAIN





SBI (SET Bit in i/o register)

EX

SBI PORTB, 5 // set the pin 5 of port B

SBI a, b | 1001 | 1010 | aaaa | abbb |

$0 \leq a \leq 31$

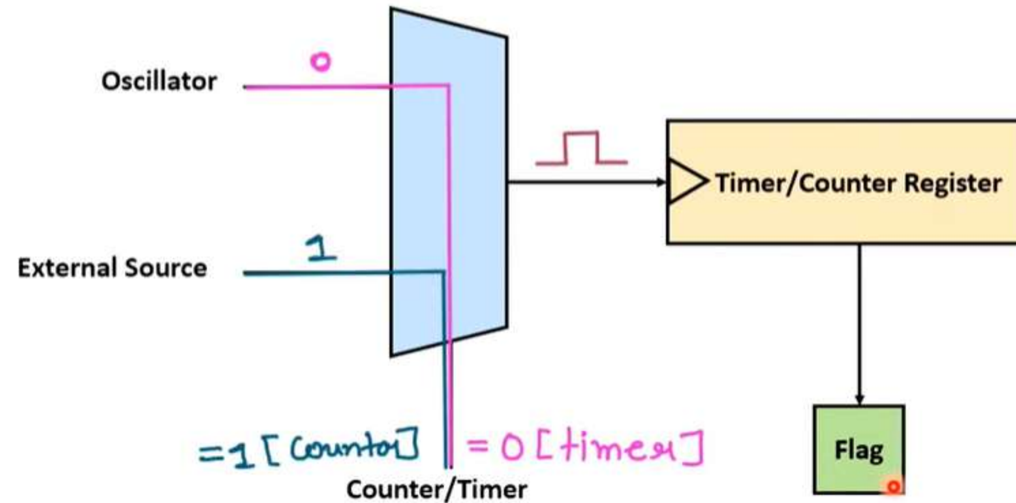
$0 \leq b \leq 7$

Instruction format of SBI

```c
#include <avr/io.h>
#define F_CPU 16000000UL  // 16 MHz
#include <util/delay.h>

int main(void)
{
DDRB = 0xFF;
    while (1)
    {
        PORTB = 0b00010000; //Port D bit4 On
        _delay_ms(1000);
        PORTB = 0b00000000; //Port D bit4 Off
        _delay_ms(1000);
    }
}
```

AVR Timer



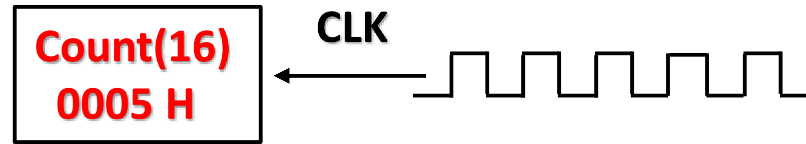
- Timers come in handy when you want to set some time interval like your alarm. This can be very precise to a few microseconds.
- Timers/Counters are essential part of any modern MCU. Remember it is the same hardware unit inside the MCU that is used either as Timers or Counter. Timers/counters are an independent unit inside a micro-controller. They basically run independently of what task CPU is performing. Hence they come in very handy, and are primarily used for the following:
  1. **Internal Timer**: As an internal timer the unit, ticks on the oscillator frequency. The oscillator frequency can be directly feed to the timer or it can be pre-scaled. In this mode it used generate precise delays. Or as precise time counting machine.
  2. **External Counter**: In this mode the unit is used to count events on a specific external pin on a MCU.
  3. **Pulse width Modulation(PWM) Generator**: PWM is used in speed control of motors and various other applications.
- Atmega32 has 3 timer units, timer 0, timer 1 and timer 2 respectively.

Difference between counter and timer :

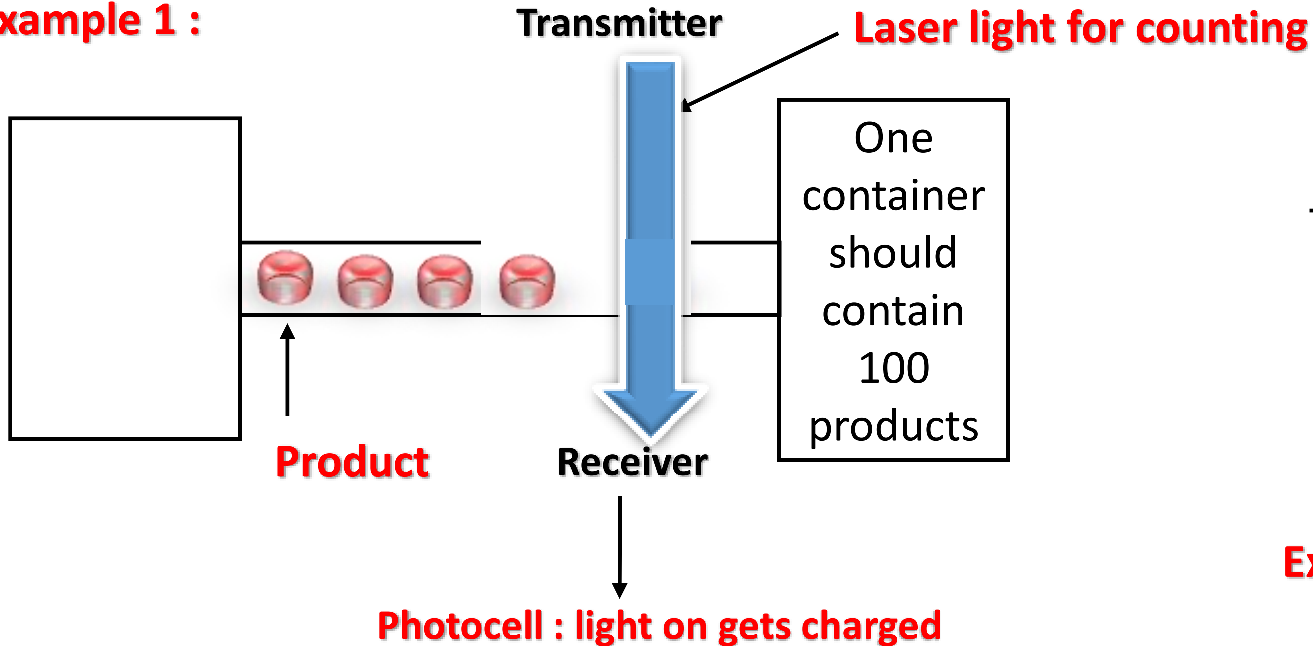
1. Timer and counter are **not exact same**
2. They are **almost same**
3. Its totally depend upon the **intension of programmer** whether device is used as a counter or timer

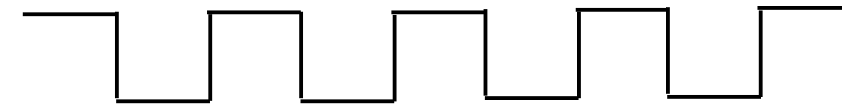
Timer : 0 1 2 3 4 5 Space between digits are fixed

Counter : 0 1 2 3 4 5 Space between digits are not fixed

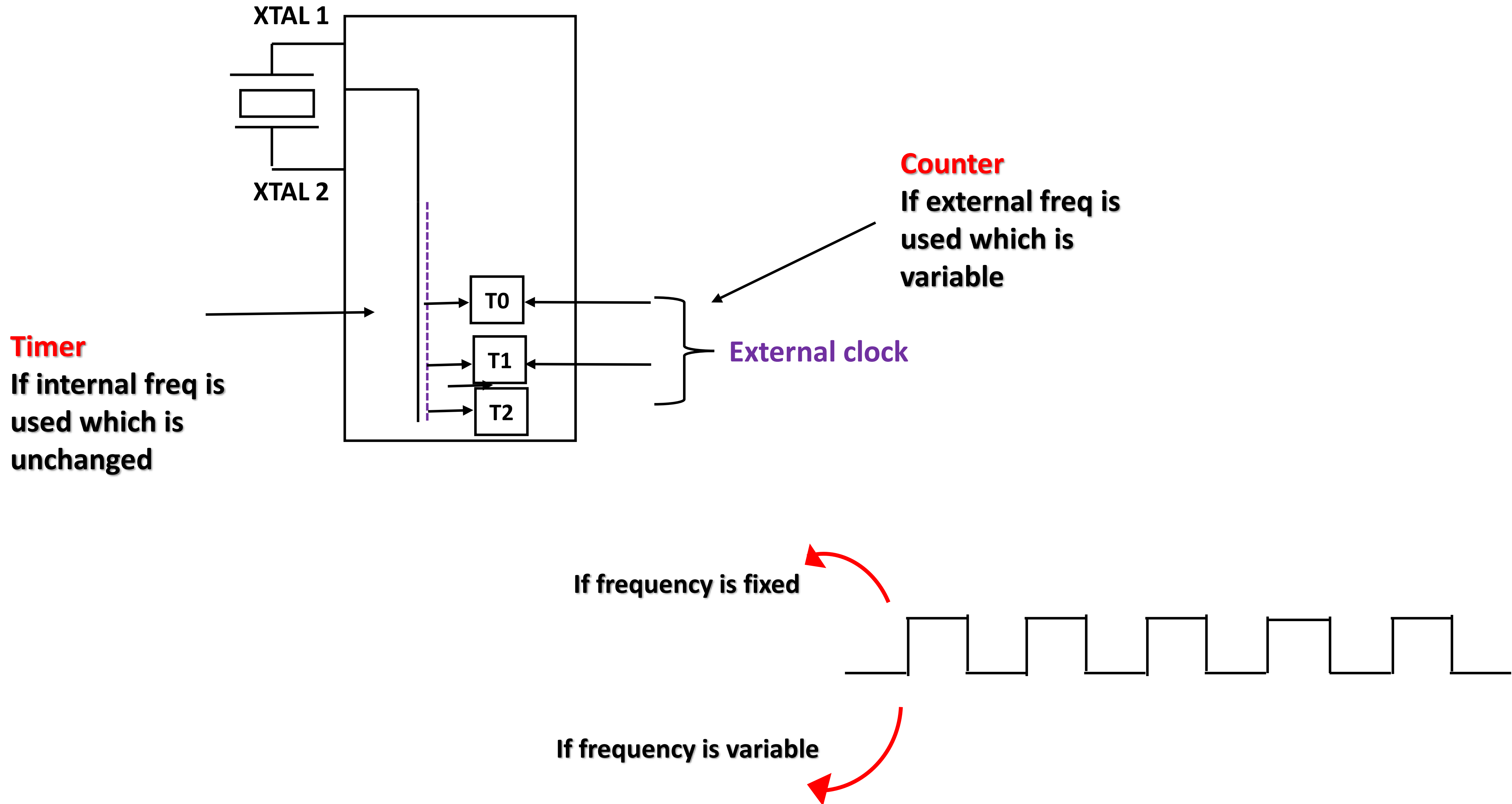


Example 1 :



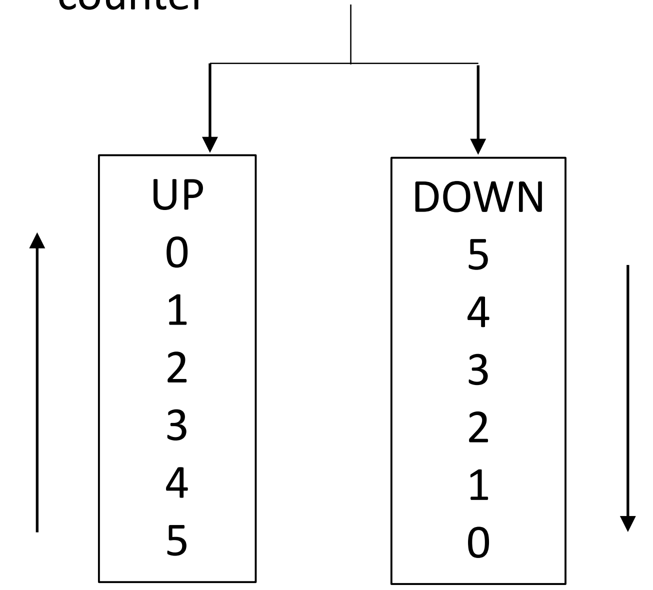


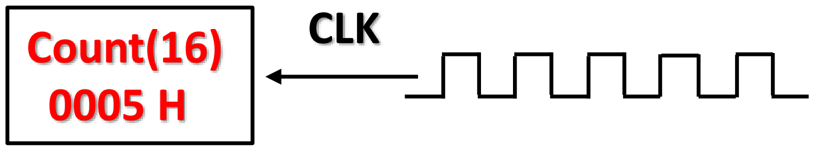
Example 2 : In bank counting notes


XTAL 1
XTAL 2
T0
T1
T2
Counter
If external freq is used which is variable
External clock
Timer
If internal freq is used which is unchanged
If frequency is fixed
If frequency is variable

1. To generate delay first we need to **load count** value.
2. In case of 8051, 16 bit counters are used.
3. There fore count value is 16 bit number i.e. **65,536**
4. Range for counter is **0000** to **FFFF**
5. To hold count value we required special registers i.e. **SFR**
6. But SFR is 8 bit , so we required 2 , 8 bit SFR

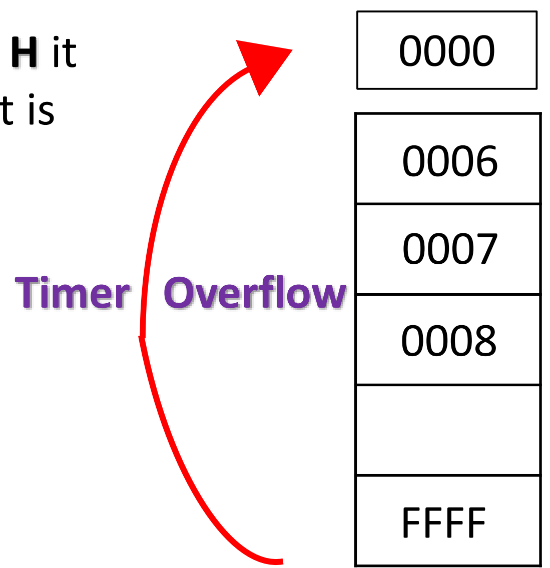
There are two types of counter





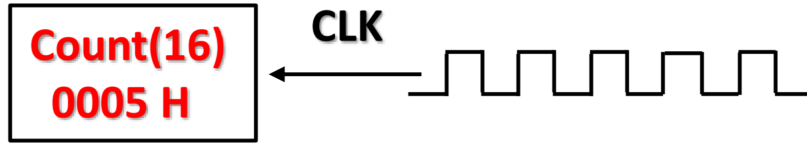
Example count = 0005 H

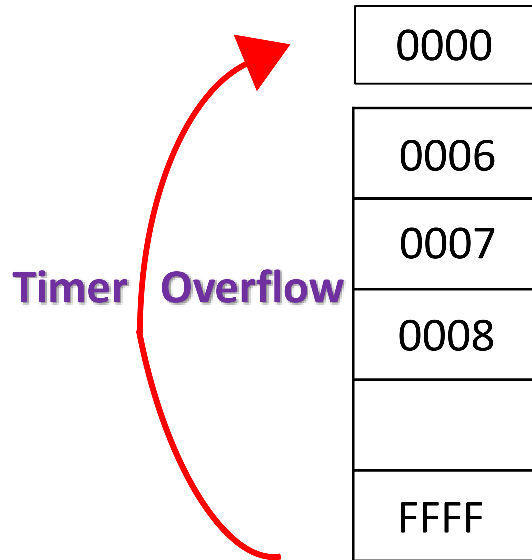
When counter is incremented after **FFFF H** it will **rollback** to the **0000 H** and this event is called as **Timer Overflow**



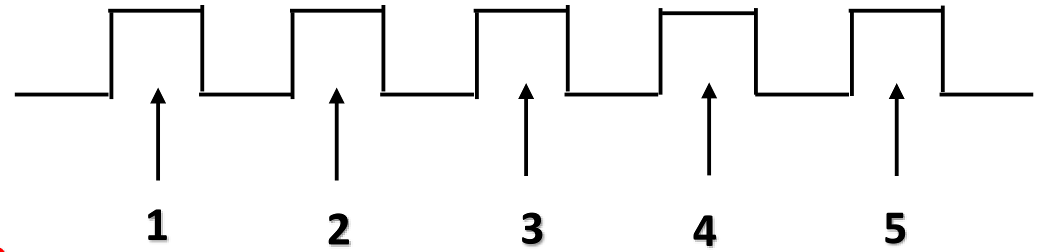
When counting is over timer will send an interrupt to processor through **flag**

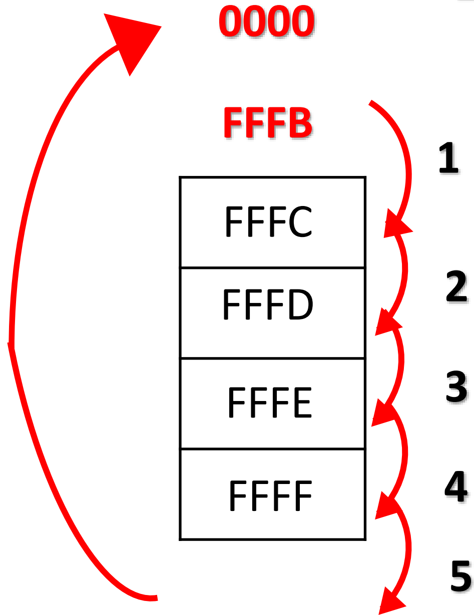
How to calculate count value :





0000
0006
0007
0008
FFFF





Counting over means overflow of timer

There fore +1 is used

Count value = FFFF – 5 + 1
= 65535 -5
= 65530 + 1
= 65531 (FFFB H)

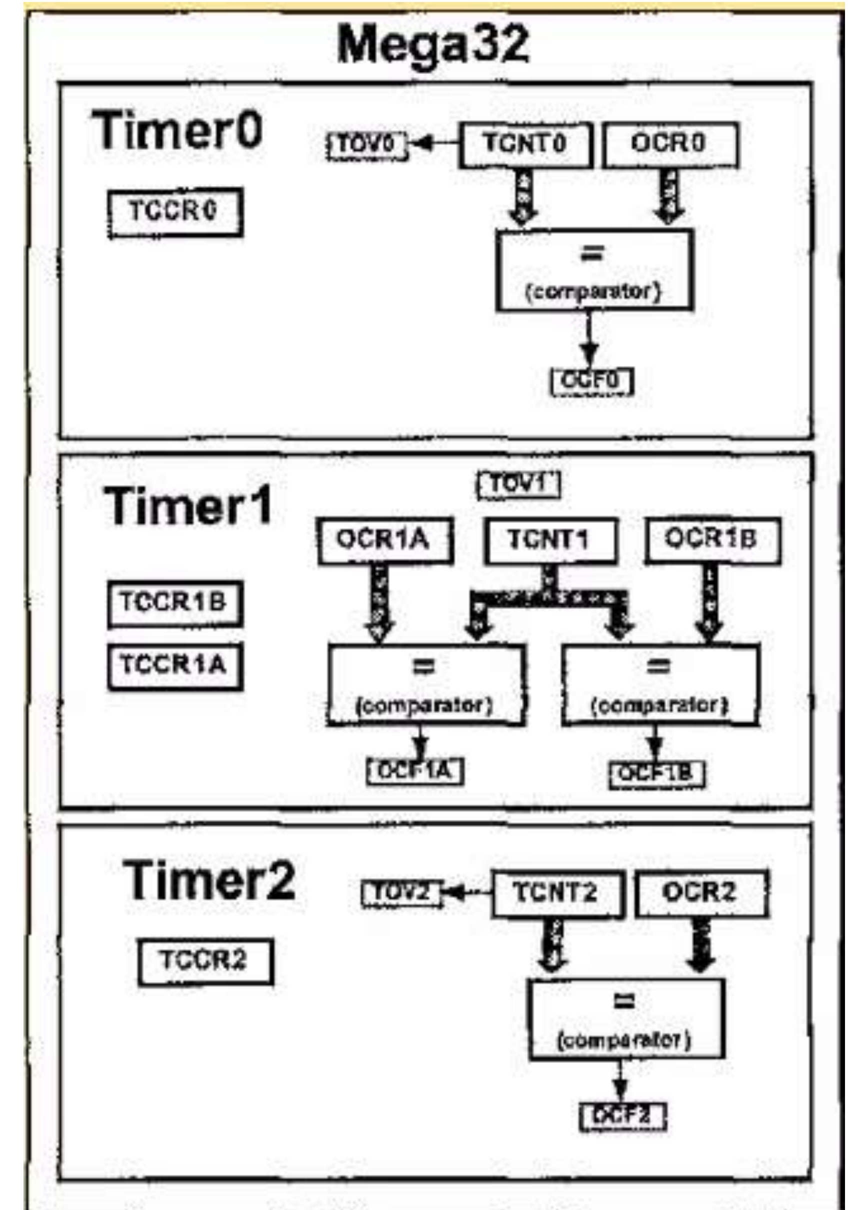
Formula for count value = Max count – Desired count + 1

Basic Registers of Timers

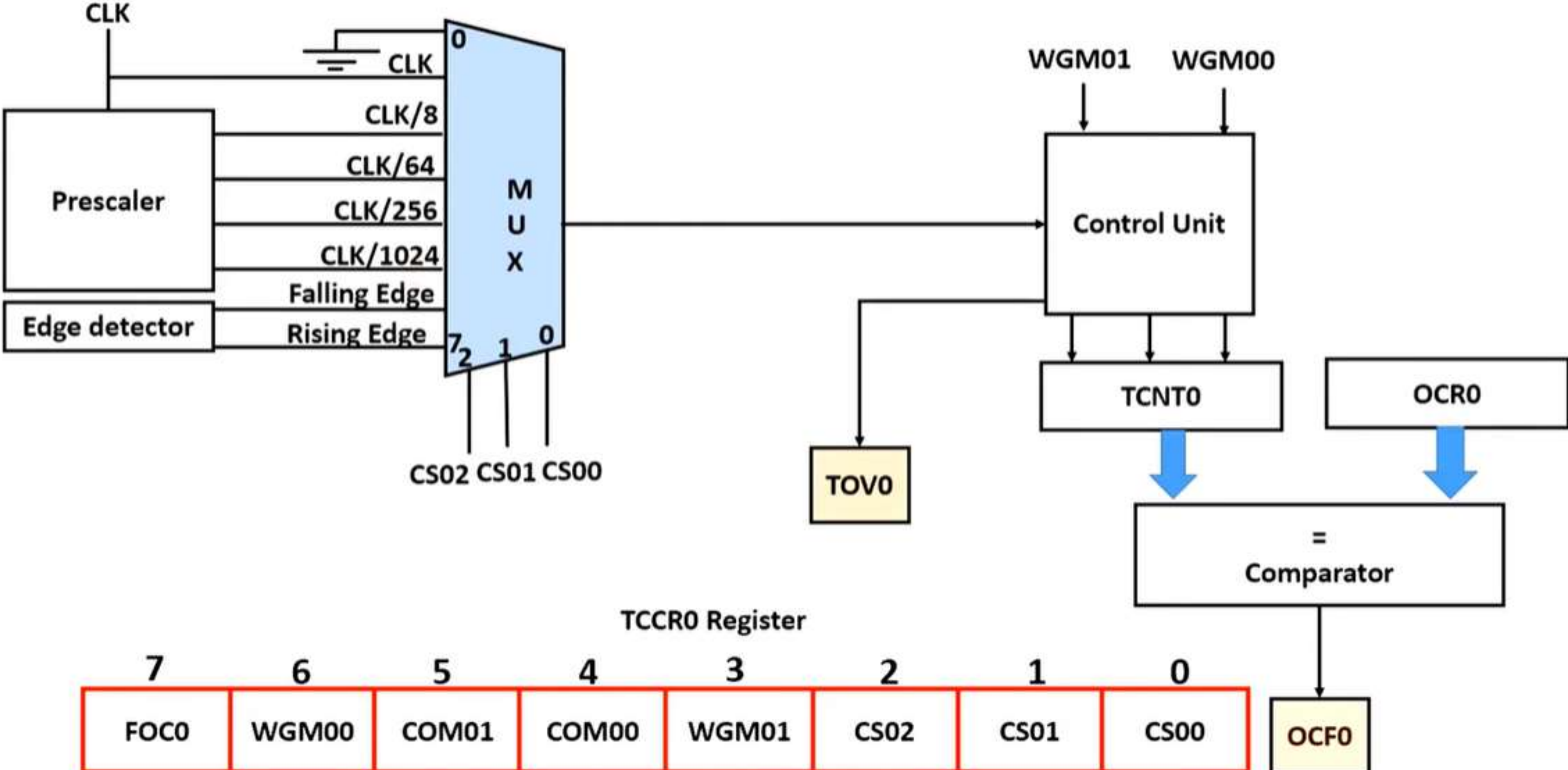
In AVR ATmega16 / ATmega32, there are three timers:
- **Timer0**: 8-bit timer
- **Timer1**: 16-bit timer
- **Timer2**: 8-bit timer

Basic registers and flags of the Timers :

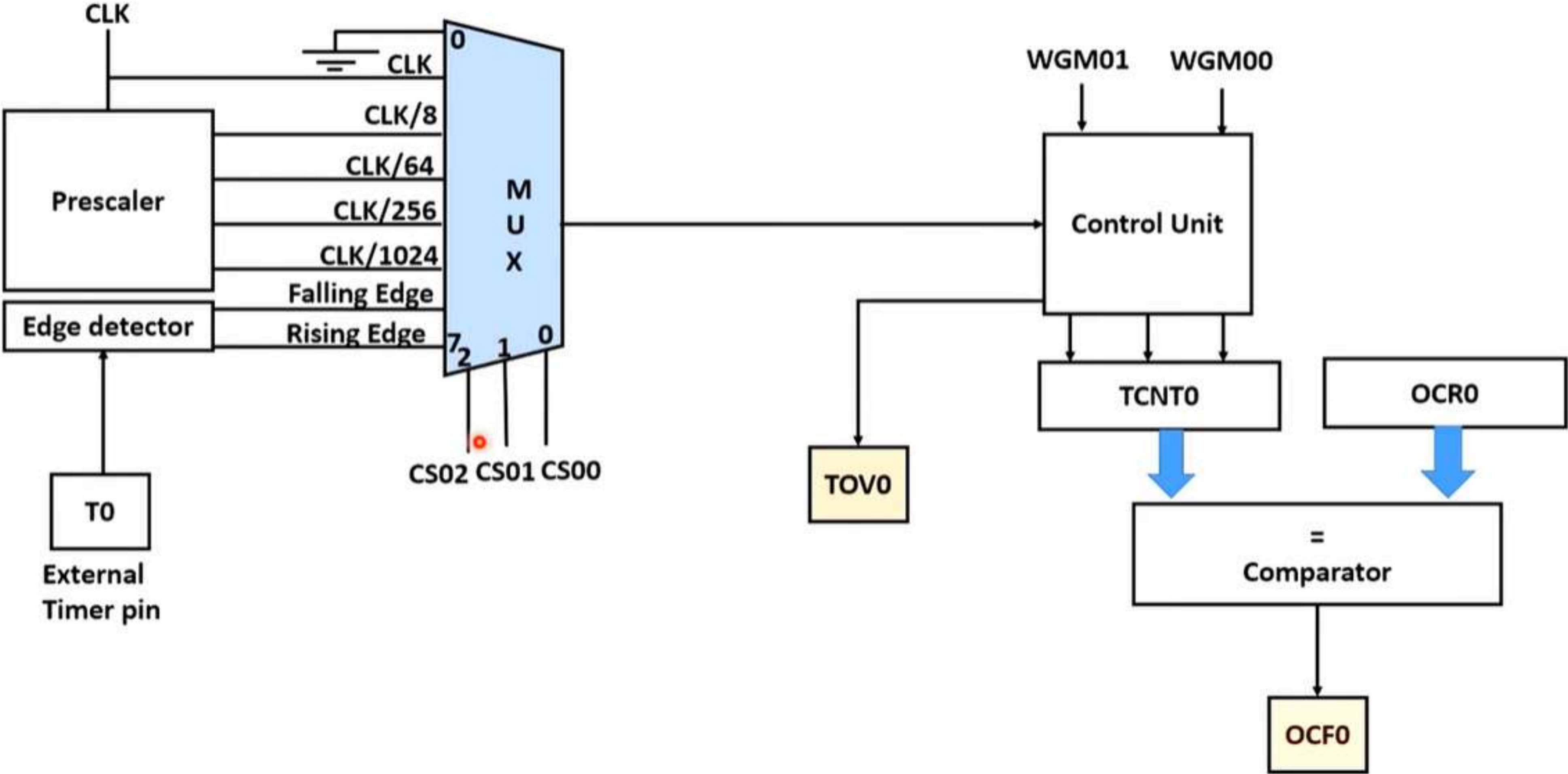
TCNTn: Timer / Counter Register :Every timer has a timer/counter register. It is zero upon reset. We can access value or write a value to this register. It counts up with each clock pulse.
TOVn: Timer Overflow Flag : Each timer has a Timer Overflow flag. When the timer overflows, this flag will get set.
TCCRn: Timer Counter Control Register: This register is used for setting the modes of timer/counter.
OCRn: Output Compare Register: The value in this register is compared with the content of the TCNTn register. When they are equal, the OCFn flag will get set.



Timer 0 as a Timer



Timer 0 as a Counter



TCNT0 Register

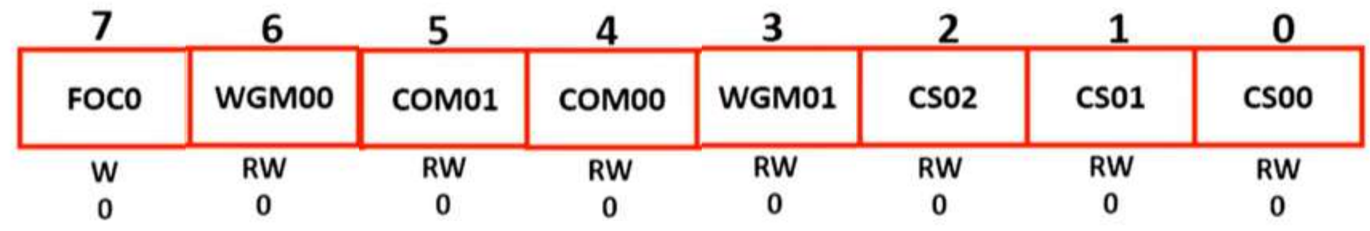
- Every timer has a timer/counter register.
- It is zero upon reset.
- We can access value or write a value to this register.
- It counts up with each clock pulse.
- Maximum count value is FF i.e. 255

OCR0 Register

- This register is used when we want to use timer in CTC mode.
- The value in this register is compared with the content of the TCNTn register.
- When they are equal, the OCFn flag will get set.

TCCR0 Register (Timer/Counter 0 Control Register)

- This is very important register of timer because it controls entire operation of timer.
- This register is used for setting the modes of timer/counter.



	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

FOC0 D7 Force Compare Match

WGM00, WGM01 D6	D3	Timer0 Mode selector bits
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

COM01:00 D5	D4	Compare Output Mode
0	0	Normal Port Op. OC0 disconnected
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

CS02:00 D2	D1	D0	Timer 0 Clock selector
0	0	0	No clock Source (T0 Stopped)
0	0	1	clk (No Prescaling)
0	1	0	clk/8
0	1	1	clk/64
1	0	0	clk/256
1	0	1	clk/1024
1	1	0	External clk source on T0 pin ↓ Edge
1	1	1	External clk source on T0 pin ↑ Edge

Finding the value to be loaded in TCNT0 to generate the desired delay

1. $T_{clock} = 1/F_{timer}$ (If we use prescaler than we need to divide base frequency with that and finally inverse it to have Tclock)

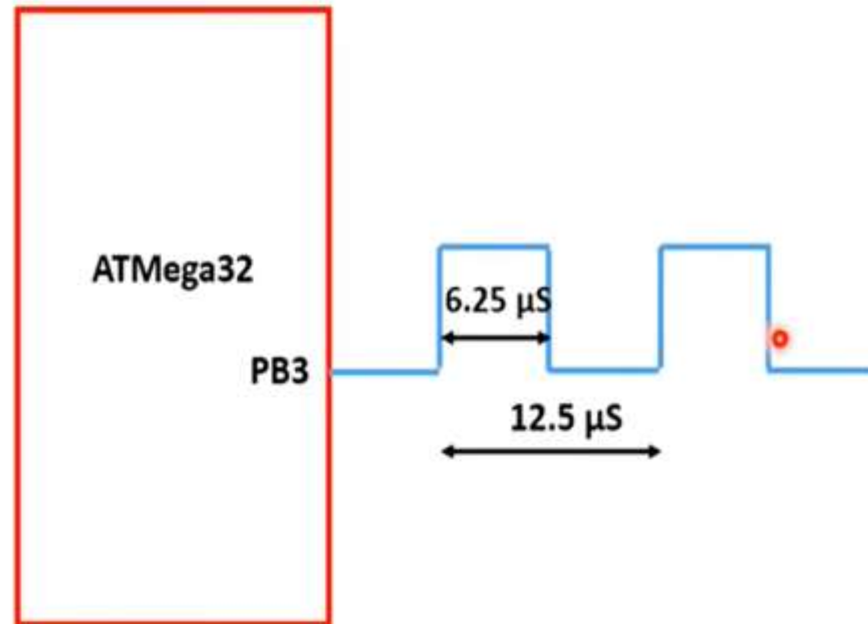
2. Divide the desired time delay by T_{clock}. This says how many clocks we need.

3. Perform 256 – n, where n is the decimal value we got in step 2

4. Convert result of step 3 in to hex

5. Set TCNT0 = Hex number of step 4

Assuming that XTAL = 8MHz, write a program to generate a square wave with a period of 12.5µs on pin PB3.(Here we use no prescaler clock option)



TCNT0 Value Calculation

- T = 12.5 μS

So, T/2 = 6.25 μS   we need to generate this delay

XTAL f = 8MHz

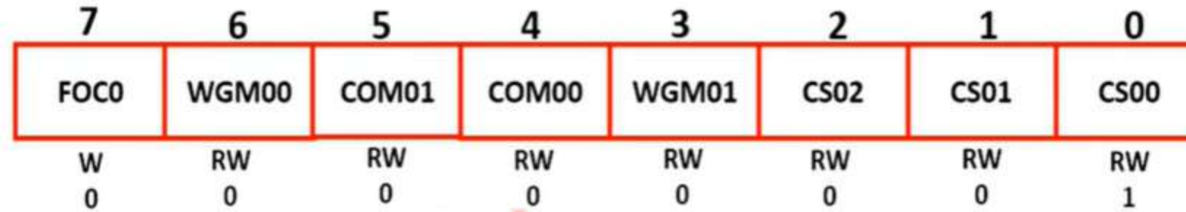
T = 1/f = 0.125 μS

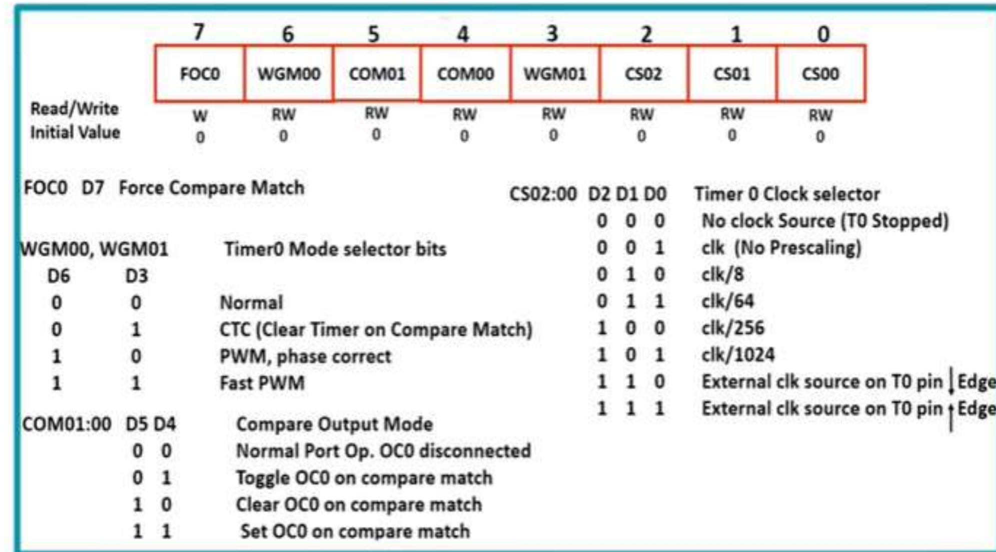
No of clocks to be needed = 6.25 μS/0.125 μS = 50 Clocks

So, 256 – 50 = 206 = 0xCE

TCNT0 = 0XCE

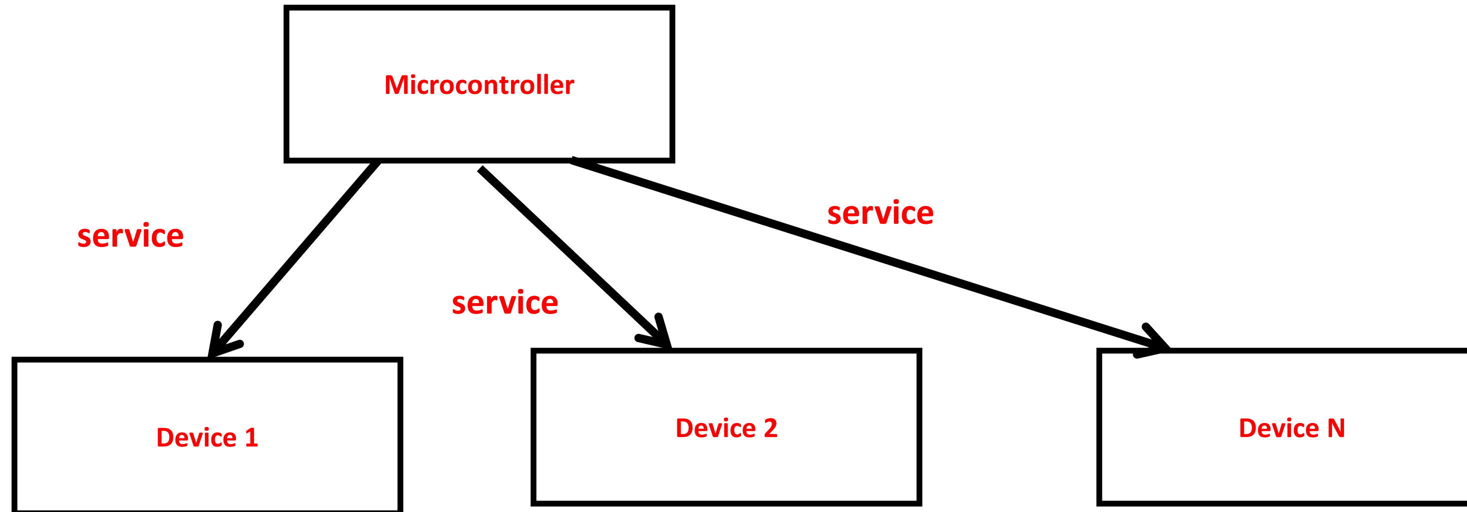
TCCR0 Value calculation





TCCR0 = 0x01 ; Selects Timer0, Normal mode, int clk, no Prescaler

AVR Interrupts



A single microcontroller can serve several devices. There are two methods to provide services are :

- **Polling**
- **Interrupts**

Polling :

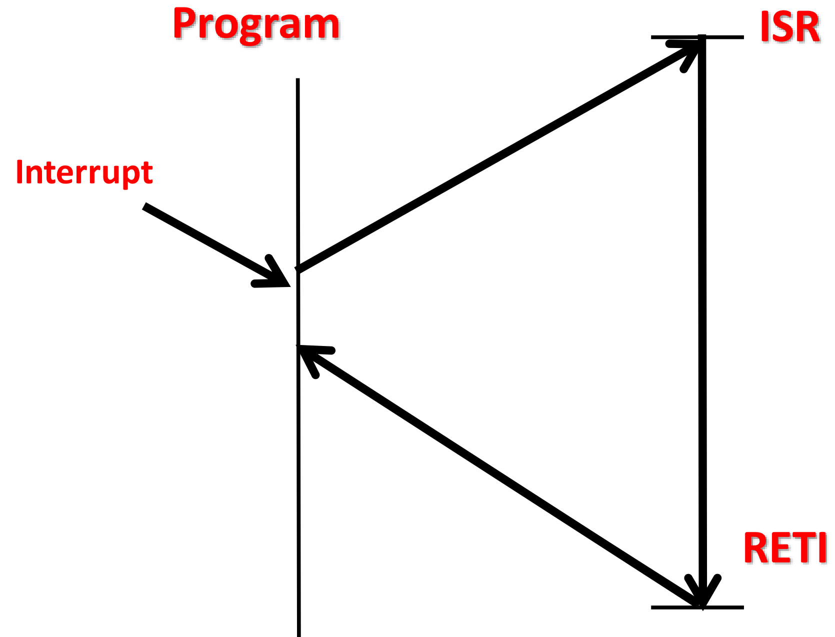
1. In polling microcontroller continuously monitors the status of given device.
2. When status condition met, it performs the service.
3. After that it moves on to monitor next device until each one is serviced.
4. Although polling can monitor the status of several devices and serve each of them as certain conditions are met, it is not an efficient use of microcontroller.
5. The polling method cannot assign priority since it checks all devices in Round-robin method.
6. In this method microcontroller can not ignore the service request of device

Interrupt :

1. In interrupt method, whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal
2. Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device.
3. Program associated with interrupt is called as ISR (interrupt service routine) .
4. Advantage of interrupt is that the microcontroller can serve many device as per priority.
5. In this method microcontroller can ignore the interrupt request of device

ISR (Interrupt Service Routine :

1. For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler.
2. When an interrupt invoked, the microcontroller runs the interrupt service routine.



IVT(Interrupt Vector Table):

1. For every interrupt there is a fixed location in memory that holds the **address of its ISR** .
2. The group of memory locations set aside to hold the addresses of ISRs is called the interrupt vector table.

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

Steps in Executing Interrupt :

1. It finishes the current instruction and saves the address of next instruction (PC) on stack.

2. It also saves the current status of all the interrupts internally (Not on the stack)

3. It jumps to a fixed location in memory called the interrupt vector table (IVT) that holds the address of the interrupt service routine (ISR).

4. The microcontroller gets the address of the ISR from the IVT and jumps to it. It starts to execute the ISR until it reaches the last instruction of subroutine i.e. RETI (return from interrupt).

5. Upon RETI instruction it returns, first controller gets the PC address from stack and then start execution from given address.

Sources of interrupts in the AVR :

- There are many sources of interrupts in the AVR, depending on which peripheral is incorporated intro the chip.
- The following are some of the most widely used sources of interrupts in the AVR :

1. There are at least two interrupts set aside for each of the timer, one for over flow and another for compare match.

2. Three interrupts are set aside for external hardware interrupts:
   - PORTD.2 – INT0
   - PORTD.3 – INT1
   - PORTB.2 - INT2

3. Serial communication’s USART has three interrupts, one for receive and two for transmit.

4. The SPI interrupts.

5. The ADC interrupts.

Serial Port of AVR

Communication

- The word communication specifies, data transfer between two points.
- The data may be digital or analog in nature.
- We will consider only digital data transfer because processor is digital circuit.
- There are two types of data transfer

  - **Parallel data transfer(no. of bits at a time) – parallel communication**

    Faster but costly because required many lines

  - **Serial data transfer(1 bit at a time) – serial communication**

    Slower but cheaper because required single line

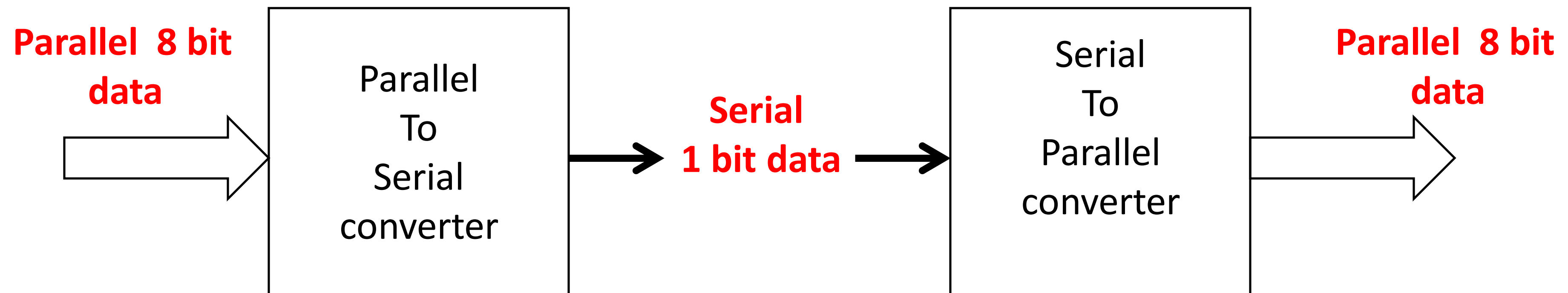
- For long distance mostly serial communication preferred.

Advantage from serial communication : It is cheap

Advantage from parallel communication : Fast

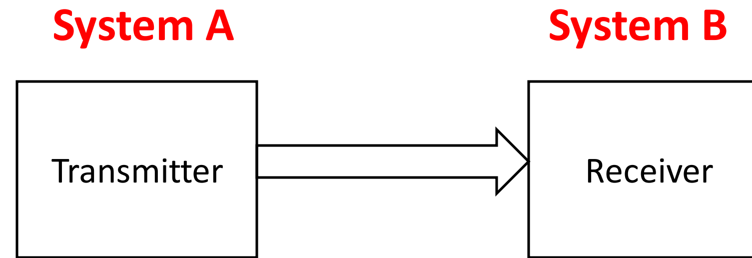
8051 : Cheap + Fast

Serial Data Transfer



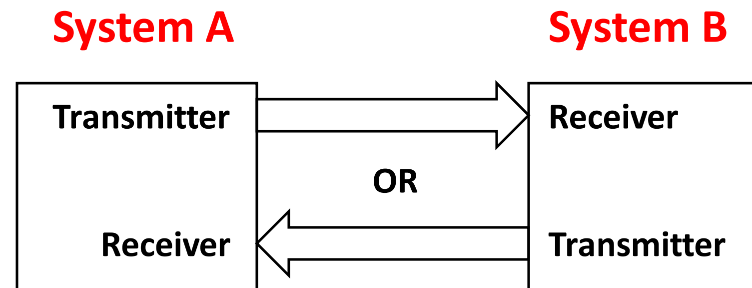
Types of communication system :

1. **Simplex :**
   The simplex is **one way** transmission. The connection exists such that data transfer takes place only in one direction. System A is transmitter and B is receiver

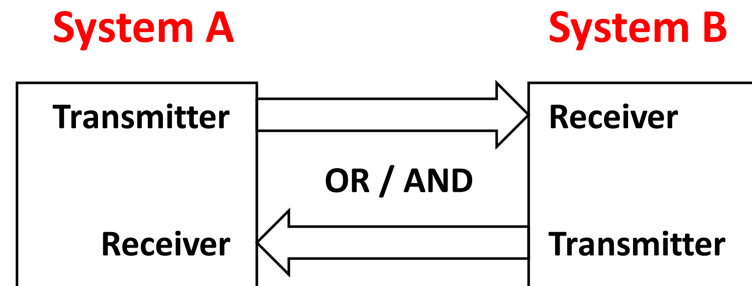


2. **Duplex :**
   The simplex is **two way** transmission. There are two groups.
   a) Half duplex b) full duplex

a) **Half Duplex :**
it is a connection between two terminals such that, data may travel in both the directions, but transmission activated at a time.



b) **Full Duplex :**
it is a connection between two terminals such that, data may travel in both the directions simultaneously. So it will contain one way transmission at a time.



Serial Transmission Formats :

- Asynchronous Data Transfer

- Synchronous Data Transfer

Synchronous

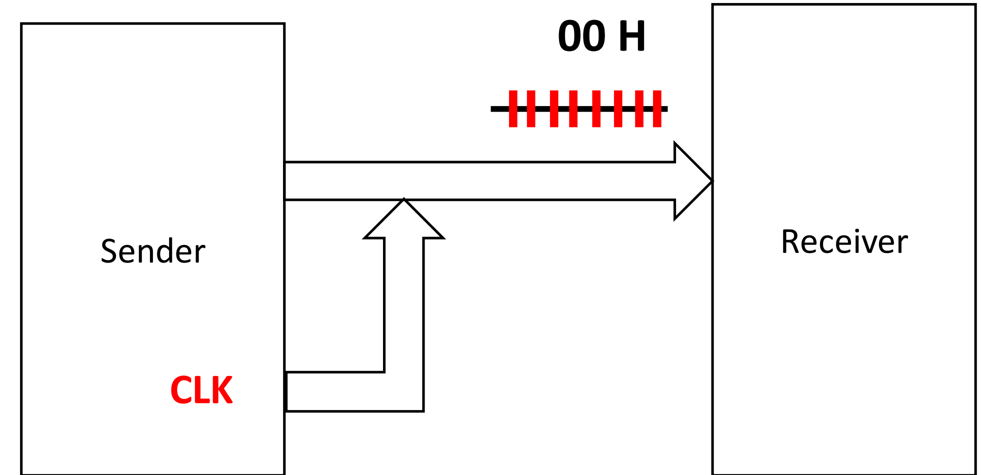
1. **Sender and receiver both are operating on common clock.**
2. **Sender can send data and CLK**

Asynchronous

1. **Sender and receiver do not operate on common clock.**
2. **Sender can send only data not CLK**

Both are always at same frequency

Sender and receiver always on same frequency



1. Sender wants to send 00 H data. i.e. 00000000 (total 8 0's)
2. Sender goes sending 0's.
3. In this case can't say how many 0's are there. May be 2 0's , 3 0's or 1000 0's

How to know the number of 0's

Synchronous : on same frequency and shared common CLK

4. If receiver samples this line (read this line) exactly at the same frequency at which sender is sending.

5. If sender's interval = receiver's interval then only

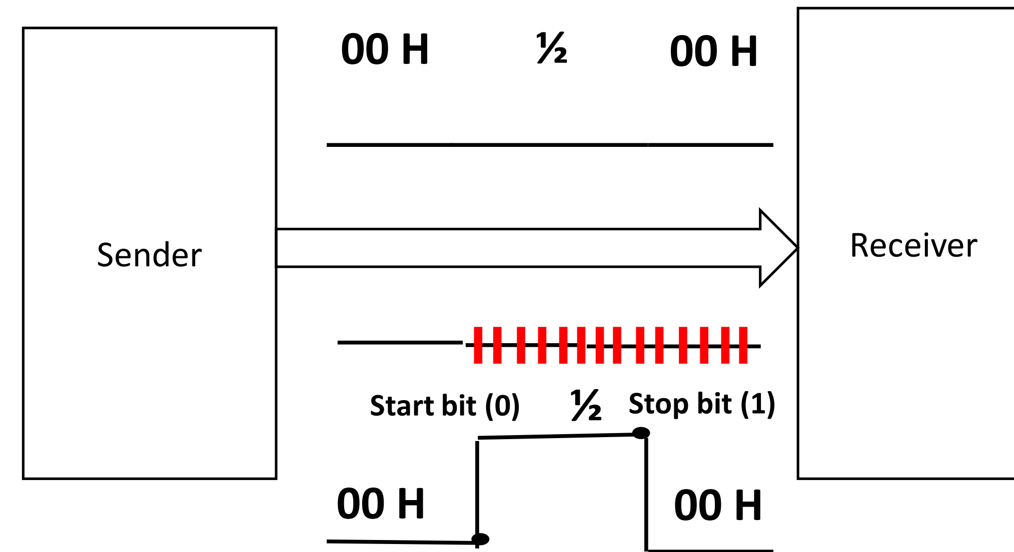
8- 0's look like 8- 0's

6. If sender sends 10 bits / sec and receiver receives data 8 bits / sec then receiver loss 2 bits.

Example : **Asynchronous : on same frequency but not shared common CLK**

Sender :

1. Sender sends **00 H** to the receiver and take **break of ½ hour** means no communication for ½ hour.
2. After **½ hour** sender again sends **00 H** data.
3. When sender sends first 00 H data , line was _______
4. For next ½ hour there is no signal pass on therefore line was in the state of last i.e. 0
5. After ½ hour sender sends next data i.e. again 00 H , therefore the state of line was ________

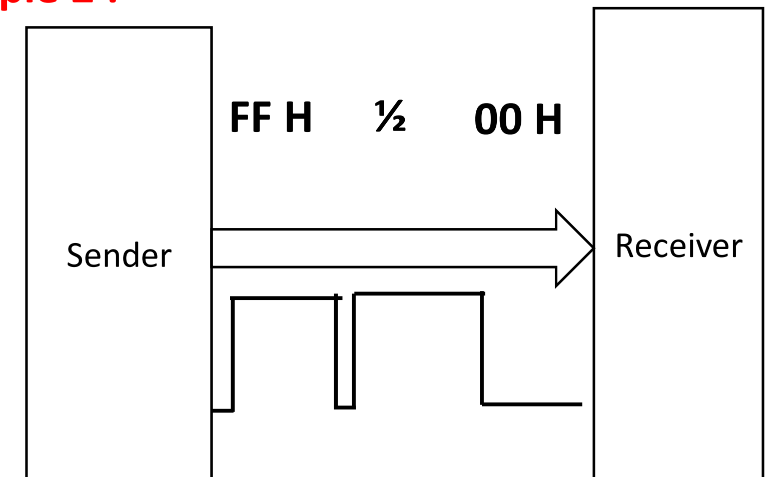


Both are always at same frequency

Receiver :

1. Both are working on same frequency receiver knows the sample rate, so receiver samples first 8 bits.
2. Receiver don't have idea about the next signals of line whether is next data or garbage value, hence sender will accept the garbage value as a data with sampling.
3. Hence we required stop bit (1) to inform the receiver that data bits are over.
4. Now some how receiver understand that garbage value received hence until any changes occurs on line it will not accept next data.
5. But next data is again 00 H
6. Receiver will not accept the actual data.
7. Hence we required start bit (0) to inform the receiver that has to be start

Example 2 :



Deadline in UART is always 1, for starting of comm required 0 on line

Speed (Baud rate)

It is a rate at which data is transferred or received in serial communication.

- As we know the bit rate is “Number of bits per second (bps)”, also known as Baud rate in Binary system.
- Normally this defines how fast the serial line is.
- There are some standard baud rates defined e.g. 1200, 2400, 4800, 19200, 115200 bps, etc. Normally 9600 bps is used where speed is not a critical issue.

Example : Radio channels

- If we want to listen 93.5 we have to set 93.5 channel.
- If it is change to 93.7 then will get disturbance.
- There fore frequency or baud rate is important.