به نام خدا

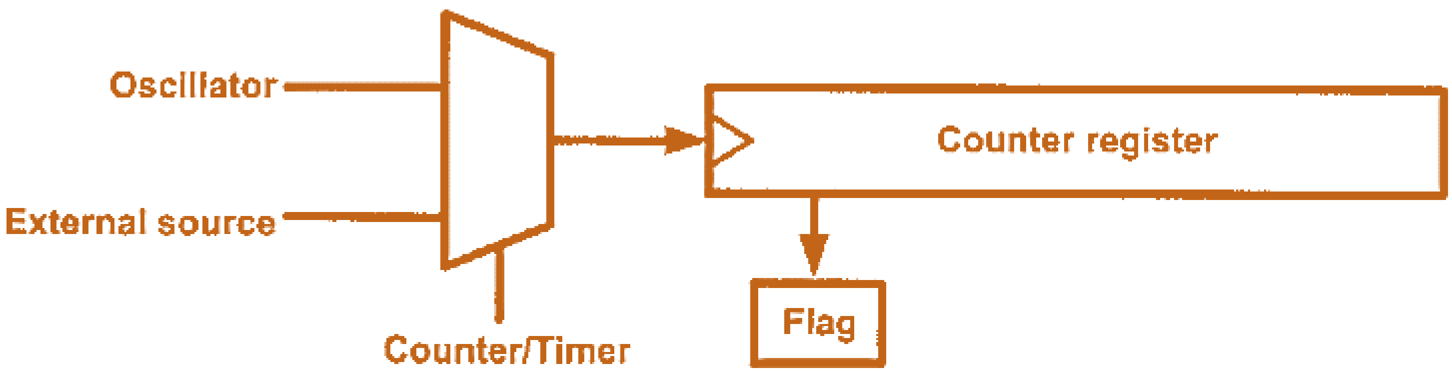
آشنایی با تایمرها

Timer in ATmega32

Dr. Aref Karimiafshar
A.karimiafshar@ec.iut.ac.ir

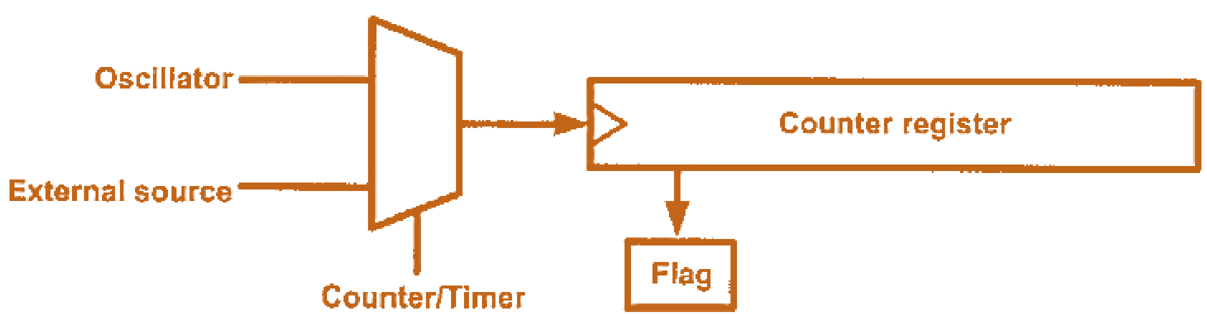
AVR Timer

- Counter registers in µC
  - To count an event
  - Generate time delay



AVR Timer

When we want to count an event, we connect the external event source to the clock pin of the counter register. Then, when an event occurs externally, the content of the counter is incremented; in this way, the content of the counter represents how many times an event has occurred. When we want to generate time delays, we connect the oscillator to the clock pin of the counter. So, when the oscillator ticks, the content of the counter is incremented. As a result, the content of the counter register represents how many ticks have occurred from the time we have cleared the counter. Since the speed of the oscillator in a microcontroller is known, we can calculate the tick period, and from the content of the counter register we will know how much time has elapsed.



Time Delay

So, one way to generate a time delay is to clear the counter at the start time and wait until the counter reaches a certain number. For example, consider a microcontroller with an oscillator with frequency of 1 MHz; in the microcontroller, the content of the counter register increments once per microsecond. So, if we want a time delay of 100 microseconds, we should clear the counter and wait until it becomes equal to 100.

In the microcontrollers, there is a flag for each of the counters. The flag is set when the counter overflows, and it is cleared by software. The second method to generate a time delay is to load the counter register and wait until the counter overflows and the flag is set. For example, in a microcontroller with a frequency of 1 MHz, with an 8-bit counter register, if we want a time delay of 3 microseconds, we can load the counter register with $FD and wait until the flag is set after 3 ticks. After the first tick, the content of the register increments to $FE; after the second tick, it becomes $FF; and after the third tick, it overflows (the content of the register becomes $00) and the flag is set.

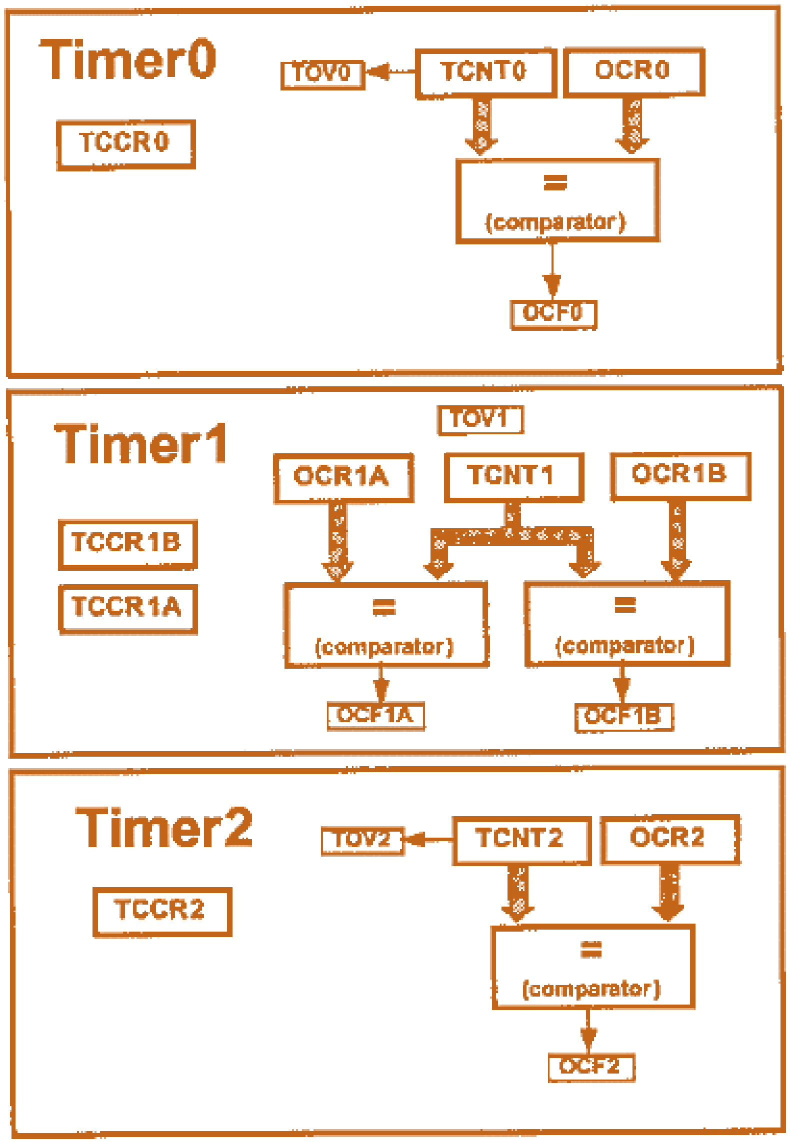
AVR Timers

- AVR has one to six timers
  - Depending on family member
  - Timers 0, 1, 2, 3, 4, and 5
  - Can be used as:
    - Timers: to generate time delay
    - Counters: count events happening outside the μC
  - Some T/C are 8-bit and some are 16-bit
  - For example: ATmega32
    - Three timers
      - Timer0: 8-bit
      - Timer1: 16-bit
      - Timer2: 8-bit

Programming the Timers

- Every timer needs a clock pulse to tick
  - The clock source can be internal or external
- Internal clock source
  - The frequency of the crystal oscillator is fed into the timer
  - Called timer
- External clock source
  - Feed pulses through one of the AVR's pin
  - Called counter

Timers ATmega32



Basic Registers of Timers

- TCNTn $\xrightarrow{\text{ATmega32}}$ TCNT0, TCNT1 and TCNT2
- TCNT is counter register (Timer/CouNTer)
  - Counts up with each pulse
  - Upon reset, it contains zero
  - You can load a value into TCNT or read from it
- TOVn $\xrightarrow{\text{ATmega32}}$ TOV0, TOV1 and TOV2
- TOV is a flag (Timer OVerflow)
  - When a timer overflows, its TOV flag will be set

Basic Registers of Timers

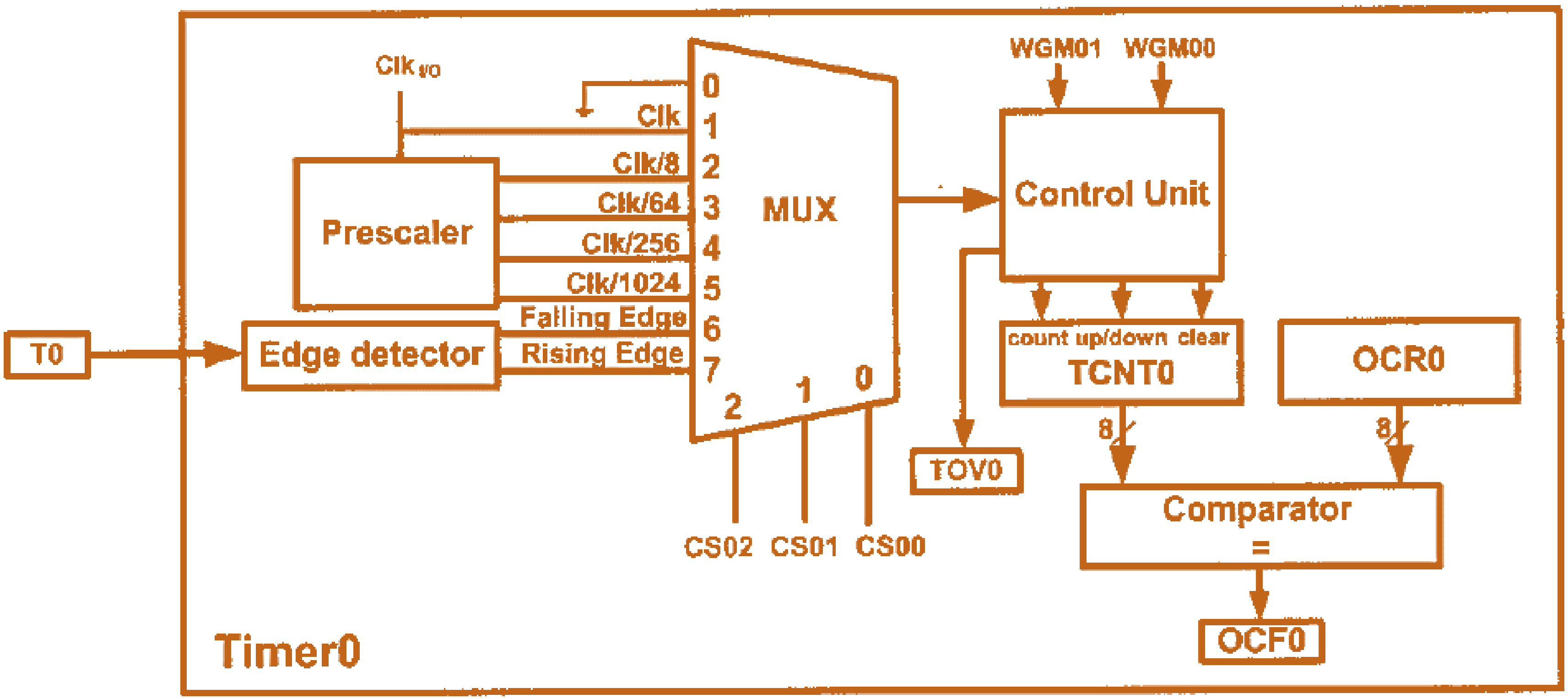
- TCCRn $\xrightarrow{\text{ATmega32}}$ TCCR0, TCCR1$_{A,B}$ and TCCR2
- TCCR is register (Timer/Counter Control Register)
  - For setting modes of operation
    - Specify a timer to work as a timer or counter
- OCRn $\xrightarrow{\text{ATmega32}}$ OCR0, OCR1$_{A,B}$ and OCR2
- OCR is a register (Output Compare Register)
  - The content of OCR is compared with the content of TCNT
  - When they are equal the OCFn flag will be set
- OCFn (Output Compare Flag)

Basic Registers of Timers

- Timer registers are located in I/O register memory
- You can read or write from timer registers using IN and OUT instructions, like other I/O registers
- For example, load TCNT0 with 25

```
LDI R20,25          ;R20 = 25
OUT TCNT0,R20       ;TCNT0 = R20
```

Timer 0 ATmega32



Timer0 ATmega32

- Timer0 is 8-bit → TCNT0 is 8-bit

TCNT0	D7	D6	D5	D4	D3	D2	D1	D0

- TCCR0 (Timer/Counter Control Register)
  - An 8-bit register used for control of timer0

Bit	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

TCCR0 (Timer/Counter Control Register)

Bit	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: This is a write-only bit, which can be used while generating a wave. Writing 1 to it causes the wave generator to act as if a compare match had occurred.

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:
These bits control the waveform generator

CS02:00

D2	D1	D0	Timer0 clock selector
0	0	0	No clock source (Timer/Counter 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 clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

TCCR0 (Timer/Counter Control Register)

CS02:CS00 (Timer0 clock source)

These bits in the TCCR0 register are used to choose the clock source. If CS02:CS00 = 000, then the counter is stopped. If CS02–CS00 have values between 001 and 101, the oscillator is used as clock source and the timer/counter acts as a timer. In this case, the timers are often used for time delay generation.

If CS02–CS00 are 110 or 111, the external clock source is used and it acts as a counter.

WGM01:00

Timer0 can work in four different modes: Normal, phase correct PWM, CTC, and Fast PWM. The WGM01 and WGM00 bits are used to choose one of them.

TOV0 (Timer0 Overflow)

The flag is set when the counter overflows, going from $FF to $00. As we will see soon, when the timer rolls over from $FF to 00, the TOV0 flag is set to 1 and it remains set until the software clears it.

```
LDI   R20,0x01
OUT   TIFR,R20     ;TIFR = 0b00000001
```

TIFR

TIFR (Timer/counter Interrupt Flag Register) register

The TIFR register contains the flags of different timers,

Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

TOV0 D0 Timer0 overflow flag bit
0 = Timer0 did not overflow.
1 = Timer0 has overflowed (going from $FF to $00).

OCF0 D1 Timer0 output compare flag bit
0 = compare match did not occur.
1 = compare match occurred.

TOV1 D2 Timer1 overflow flag bit

OCF1B D3 Timer1 output compare B match flag

OCF1A D4 Timer1 output compare A match flag

ICF1 D5 Input Capture flag

TOV2 D6 Timer2 overflow flag

OCF2 D7 Timer2 output compare match flag

Example

Find the value for TCCR0 if we want to program Timer0 in Normal mode, no prescaler. Use AVR's crystal oscillator for the clock source.

	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
TCCR0 =	0	0	0	0	0	0	0	1

Find the timer's clock frequency and its period for various AVR-based systems, with the following crystal frequencies. Assume that no prescaler is used.

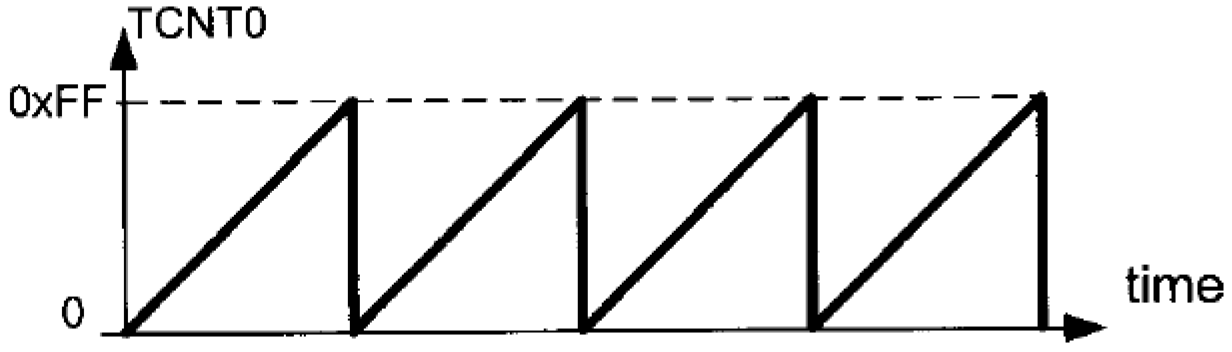
(a) 10 MHz (b) 8 MHz (c) 1 MHz

(a) F = 10 MHz and T = 1/10 MHz = 0.1 µs
(b) F = 8 MHz and T = 1/8 MHz = 0.125 µs
(c) F = 1 MHz and T = 1/1 MHz = 1 µs

Mode of Operation

Normal mode

In this mode, the content of the timer/counter increments with each clock. It counts up until it reaches its max of 0xFF. When it rolls over from 0xFF to 0x00, it sets high a flag bit called TOV0 (Timer Overflow). This timer flag can be monitored.



Steps to program Timer0 in Normal Mode

1. Load the TCNT0 register with the initial count value.
2. Load the value into the TCCR0 register, indicating which mode (8-bit or 16-bit) is to be used and the prescaler option. When you select the clock source, the timer/counter starts to count, and each tick causes the content of the timer/counter to increment by 1.
3. Keep monitoring the timer overflow flag (TOV0) to see if it is raised. Get out of the loop when TOV0 becomes high.
4. Stop the timer by disconnecting the clock source, using the following instructions:

```
LDI   R20,0x00
OUT   TCCR0,R20   ;timer stopped, mode=Normal
```

5. Clear the TOV0 flag for the next round.
6. Go back to Step 1 to load TCNT0 again.

Example

In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the PORTB.5 bit. Timer0 is used to generate the time delay.

```
.INCLUDE "M32DEF.INC"
.MACRO       INITSTACK          ;set up stack
      LDI    R20,HIGH(RAMEND)
      OUT    SPH,R20
      LDI    R20,LOW(RAMEND)
      OUT    SPL,R20
.ENDMACRO
      INITSTACK
      LDI    R16,1<<5     ;R16 = 0x20 (0010 0000 for PB5)
      SBI    DDRB,5       ;PB5 as an output
      LDI    R17,0
      OUT    PORTB,R17    ;clear PORTB
BEGIN:RCALL DELAY         ;call timer delay
      EOR    R17,R16      ;toggle D5 of R17 by Ex-Oring with 1
      OUT    PORTB,R17    ;toggle PB5
      RJMP BEGIN
;-------------------Time0 delay
DELAY:LDI    R20,0xF2     ;R20 = 0xF2
      OUT    TCNT0,R20    ;load timer0
      LDI    R20,0x01
      OUT    TCCR0,R20    ;Timer0, Normal mode, int clk, no prescaler
AGAIN:IN     R20,TIFR     ;read TIFR
      SBRS   R20,TOV0     ;if TOV0 is set skip next instruction
      RJMP   AGAIN
      LDI    R20,0x0
      OUT    TCCR0,R20    ;stop Timer0
      LDI    R20,(1<<TOV0)
      OUT    TIFR,R20     ;clear TOV0 flag by writing a 1 to TIFR
      RET
```

Example cnt.

In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the PORTB.5 bit. Timer0 is used to generate the time delay.

```
.INCLUDE "M32DEF.INC"
.MACRO      INITSTACK        ;set up stack
      LDI   R20,HIGH(RAMEND)
      OUT   SPH,R20
      LDI   R20,LOW(RAMEND)
      OUT   SPL,R20
.ENDMACRO
      INITSTACK
      LDI   R16,1<<5    ;R16 = 0x20 (0010 0000 for PB5)
      SBI   DDRB,5      ;PB5 as an output
      LDI   R17,0
      OUT   PORTB,R17   ;clear PORTB
BEGIN:RCALL DELAY       ;call timer delay
      EOR   R17,R16     ;toggle D5 of R17 by Ex-Oring with 1
      OUT   PORTB,R17   ;toggle PB5
      RJMP BEGIN
;-------------------Time0 delay
DELAY:LDI   R20,0xF2    ;R20 = 0xF2
      OUT   TCNT0,R20   ;load timer0
      LDI   R20,0x01
      OUT   TCCR0,R20   ;Timer0, Normal mode, int clk, no prescaler
AGAIN:IN    R20,TIFR    ;read TIFR
      SBRS  R20,TOV0    ;if TOV0 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x0
      OUT   TCCR0,R20   ;stop Timer0
      LDI   R20,(1<<TOV0)
      OUT   TIFR,R20    ;clear TOV0 flag by writing a 1 to TIFR
      RET
```

In the above program notice the following steps:

1. 0xF2 is loaded into TCNT0.
2. TCCR0 is loaded and Timer0 is started.
3. Timer0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of F3, F4, F5, F6, F7, F8, F9, FA, FB, and so on until it reaches 0xFF. One more clock rolls it to 0, raising the Timer0 flag (TOV0 = 1). At that point, the "`SBRS R20,TOV0`" instruction bypasses the "`RJMP AGAIN`" instruction.
4. Timer0 is stopped.
5. The TOV0 flag is cleared.

Example

In the previous **Example**, calculate the amount of time delay generated by the timer. Assume that XTAL = 8 MHz.

We have 8 MHz as the timer frequency. As a result, each clock has a period of T = 1 /8 MHz = 0.125 µs. In other words, Timer0 counts up each 0.125 µs resulting in delay = number of counts × 0.125 µs.
The number of counts for the rollover is 0xFF – 0xF2 = 0x0D (13 decimal). However, we add one to 13 because of the extra clock needed when it rolls over from FF to 0 and raises the TOV0 flag. This gives 14 × 0.125 µs = 1.75 µs for half the pulse.

Example

In the last Example, calculate the frequency of the square wave generated on pin PORTB.5. Assume that XTAL = 8 MHz.

To get a more accurate timing, we need to add clock cycles due to the instructions.

```
                                        Cycles
      LDI    R16,0x20
      SBI    DDRB,5
      LDI    R17,0
      OUT    PORTB,R17
BEGIN:RCALL  DELAY                          3
      EOR    R17,R16                        1
      OUT    PORTB,R17                      1
      RJMP   BEGIN                          2
DELAY:LDI    R20,0xF2                       1
      OUT    TCNT0,R20                      1
      LDI    R20,0x01                       1
      OUT    TCCR0,R20                      1
AGAIN:IN     R20,TIFR                       1
      SBRS   R20,0                      1 / 2
      RJMP   AGAIN                          2
      LDI    R20,0x0                        1
      OUT    TCCR0,R20                      1
      LDI    R20,0x01                       1
      OUT    TIFR,R20                       1
      RET                                   4
                                           24
```

$T = 2 \times (14 + 24) \times 0.125\ \mu s = 9.5\ \mu s$ and $F = 1 / T = 105.263$ kHz.

Finding the value to be load into timer

Assuming that we know the amount of timer delay we need, the question is how to find the values needed for the TCNT0 register. To calculate the values to be loaded into the TCNT0 registers, we can use the following steps:

1. Calculate the period of the timer clock using the following formula:
   $T_{clock} = 1/F_{Timer}$
   where F_{Timer} is the frequency of the clock used for the timer. For example, in no prescaler mode, $F_{Timer} = F_{oscillator}$. T_{clock} gives the period at which the timer increments.
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 the result of Step 3 to hex, where *xx* is the initial hex value to be loaded into the timer's register.
5. Set TCNT0 = *xx*.

Example

Assuming that XTAL = 8 MHz, write a program to generate a square wave with a period of 12.5 μs on pin PORTB.3.

For a square wave with T = 12.5 μs we must have a time delay of 6.25 μs. Because XTAL = 8 MHz, the counter counts up every 0.125 μs. This means that we need 6.25 μs / 0.125 μs = 50 clocks. 256 – 50 = 206 = 0xCE. Therefore, we have TCNT0 = 0xCE.

```
.INCLUDE "M32DEF.INC"
      INITSTACK
      LDI   R16,0x08
      SBI   DDRB,3        ;PB3 as an output
      LDI   R17,0
      OUT   PORTB,R17
BEGIN:RCALL DELAY
      EOR   R17,R16       ;toggle D3 of R17
      OUT   PORTB,R17     ;toggle PB3
      RJMP BEGIN
;---------------- Timer0 Delay
DELAY:LDI   R20,0xCE
      OUT   TCNT0,R20     ;load Timer0
      LDI   R20,0x01
      OUT   TCCR0,R20     ;Timer0, Normal mode, int clk, no prescaler
AGAIN:IN    R20,TIFR      ;read TIFR
      SBRS  R20,TOV0      ;if TOV0 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x00
      OUT   TCCR0,R20     ;stop Timer0
      LDI   R20,(1<<TOV0)
      OUT   TIFR,R20      ;clear TOV0 flag
      RET
```

Example

Modify TCNT0 to get the largest time delay possible. Find the delay in ms. In your calculation, exclude the overhead due to the instructions in the loop.

To get the largest delay we make TCNT0 zero. This will count up from 00 to 0xFF and then roll over to zero.

```
.INCLUDE "M32DEF.INC"
      INITSTACK
      LDI   R16,0x08
      SBI   DDRB,3       ;PB3 as an output
      LDI   R17,0
      OUT   PORTB,R17
BEGIN:RCALL DELAY
      EOR   R17,R16      ;toggle D3 of R17
      OUT   PORTB,R17    ;toggle PB3
      RJMP  BEGIN
;---------------- Timer0 Delay
DELAY:LDI   R20,0x00
      OUT   TCNT0,R20    ;load Timer0 with zero
      LDI   R20,0x01
      OUT   TCCR0,R20    ;Timer0, Normal mode, int clk, no prescaler
AGAIN:IN    R20,TIFR     ;read TIFR
      SBRS  R20,TOV0     ;if TOV0 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x00
      OUT   TCCR0,R20    ;stop Timer0
      LDI   R20,(1<<TOV0)
      OUT   TIFR,R20     ;clear TOV0 flag
      RET
```

Example cnt.

Modify TCNT0 to get the largest time delay possible. Find the delay in ms. In your calculation, exclude the overhead due to the instructions in the loop.

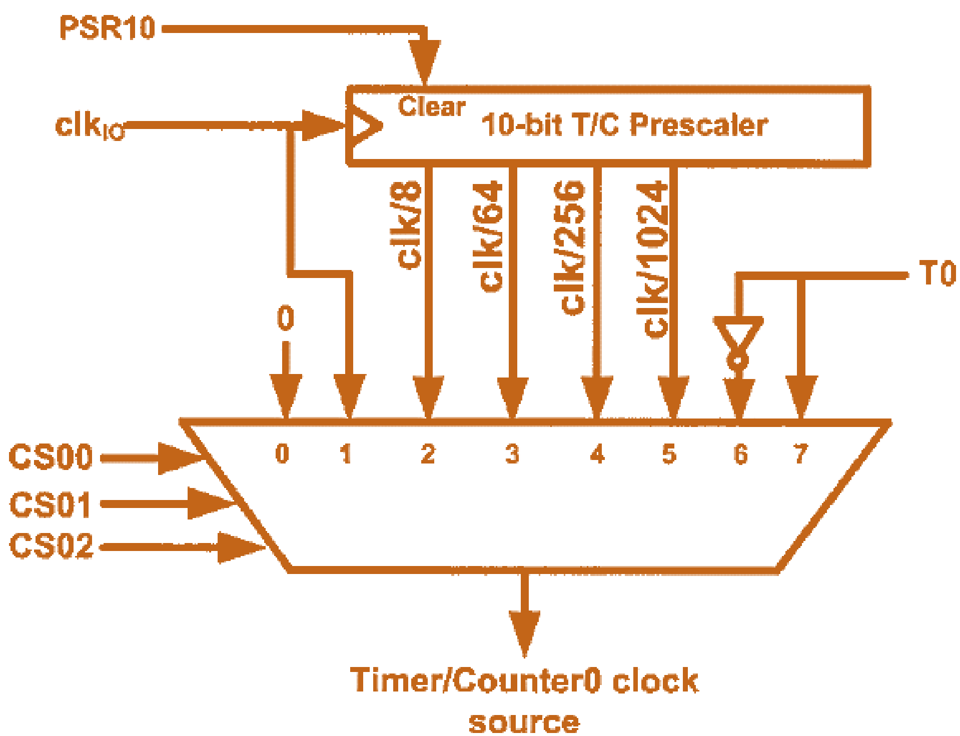
Making TCNT0 zero means that the timer will count from 00 to 0xFF, and then will roll over to raise the TCNT0 flag. As a result, it goes through a total of 256 states. Therefore, we have delay = $(256 - 0) \times 0.125\ \mu s = 32\ \mu s$. That gives us the smallest frequency of $1 / (2 \times 32\ \mu s) = 1 / (64\ \mu s) = 15.625$ kHz.

Prescaler and generating a large time delay

- The size of the time of delay depends on
  - Crystal frequency
  - Timer's register
- These factors are beyond the control of AVR programmer
- The largest time delay achieved by making TCNT0 zero
- What if that is not enough?
  - We can use the prescaler option
    - Increase delay by reducing the period

Prescaler Option

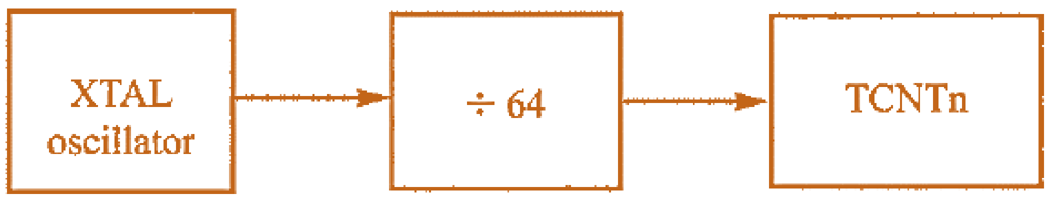
- Prescaler option of TCCR0 allows us to divide the instruction clock by a factor of 8 to 1024



Example

Find the timer's clock frequency and its period for various AVR-based systems, with the following crystal frequencies. Assume that a prescaler of 1:64 is used.

(a) `8 MHz` (b) `16 MHz` (c) `10 MHz`



(a) 1/64 × 8 MHz = 125 kHz due to 1:64 prescaler and T = 1/125 kHz = 8 µs
(b) 1/64 × 16 MHz = 250 kHz due to prescaler and T = 1/250 kHz = 4 µs
(c) 1/64 × 10 MHz = 156.2 kHz due to prescaler and T = 1/156 kHz = 6.4 µs

Find the value for TCCR0 if we want to program Timer0 in Normal mode with a prescaler of 64 using internal clock for the clock source.

we have TCCR0 = 0000 0011; XTAL clock source, prescaler of 64.

TCCR0 =

0	0	0	0	0	0	1	1
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

Example

Examine the following program and find the time delay in seconds. Exclude the overhead due to the instructions in the loop. Assume XTAL = 8 MHz.

```
.INCLUDE "M32DEF.INC"
       INITSTACK
       LDI    R16,0x08
       SBI    DDRB,3       ;PB3 as an output
       LDI    R17,0
       OUT    PORTB,R17
BEGIN:RCALL DELAY
       EOR    R17,R16      ;toggle D3 of R17
       OUT    PORTB,R17    ;toggle PB3
       RJMP   BEGIN
;---------------- Timer0 Delay
DELAY:LDI    R20,0x10
       OUT    TCNT0,R20    ;load Timer0
       LDI    R20,0x03
       OUT    TCCR0,R20    ;Timer0, Normal mode, int clk, prescaler 64
AGAIN:IN     R20,TIFR     ;read TIFR
       SBRS   R20,TOV0     ;if TOV0 is set skip next instruction
       RJMP   AGAIN
       LDI    R20,0x0
       OUT    TCCR0,R20   ;stop Timer0
       LDI    R20,1<<TOV0
       OUT    TIFR,R20    ;clear TOV0 flag
       RET
```

TCNT0 = 0x10 = 16 in decimal and 256 – 16 = 240. Now 240 × 64 × 0.125 μs = 1920 μs, or we have 240 × 8 μs = 1920 μs.

Example

Assume XTAL = 8 MHz. (a) Find the clock period fed into Timer0 if a prescaler option of 1024 is chosen. (b) Show what is the largest time delay we can get using this prescaler option and Timer0.

(a) 8 MHz × 1/1024 = 7812.5 Hz due to 1:1024 prescaler and T = 1/7812.5 Hz = 128 ms = 0.128 ms

(b) To get the largest delay, we make TCNT0 zero. Making TCNT0 zero means that the timer will count from 00 to 0xFF, and then roll over to raise the TOV0 flag. As a result, it goes through a total of 256 states. Therefore, we have delay = (256 – 0) × 128 μs = 32,768 μs = 0.032768 seconds.

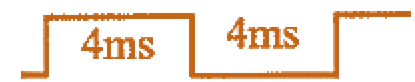
Example

Assuming XTAL = 8 MHz, write a program to generate a square wave of 125 Hz frequency on pin PORTB.3. Use Timer0, Normal mode, with prescaler = 256.

Look at the following steps:

(a) T = 1 / 125 Hz = 8 ms, the period of the square wave.

(b) 1/2 of it for the high and low portions of the pulse = 4 ms



(c) (4 ms / 0.125 μs) / 256 = 125 and 256 – 125 = 131 in decimal, and in hex it is 0x83.

(d) TCNT0 = 83 (hex)

Example cnt.

Assuming XTAL = 8 MHz, write a program to generate a square wave of 125 Hz frequency on pin PORTB.3. Use Timer0, Normal mode, with prescaler = 256.

```
.INCLUDE "M32DEF.INC"
      .MACRO  INITSTACK          ;set up stack
      LDI     R20,HIGH(RAMEND)
      OUT     SPH,R20
      LDI     R20,LOW(RAMEND)
      OUT     SPL,R20
.ENDMACRO
```

```
      INITSTACK
      LDI    R16,0x08
      SBI    DDRB,3        ;PB3 as an output
      LDI    R17,0
BEGIN:OUT    PORTB,R17     ;PORTB = R17
      CALL   DELAY
      EOR    R17,R16       ;toggle D3 of R17
      RJMP   BEGIN

;----------------- Timer0 Delay
DELAY:LDI    R20,0x83
      OUT    TCNT0,R20     ;load Timer0
      LDI    R20,0x04
      OUT    TCCR0,R20     ;Timer0, Normal mode, int clk, prescaler 256

AGAIN:IN     R20,TIFR      ;read TIFR
      SBRS   R20,TOV0      ;if TOV0 is set skip next instruction
      RJMP   AGAIN

      LDI    R20,0x0
      OUT    TCCR0,R20     ;stop Timer0
      LDI    R20,1<<TOV0
      OUT    TIFR,R20      ;clear TOV0 flag
      RET
```

Example

Find (a) the frequency of the square wave generated in the following code, and (b) the duty cycle of this wave. Assume XTAL = 8 MHz.

```
.INCLUDE "M32DEF.INC"
      LDI   R16,HIGH(RAMEND)
      OUT   SPH,R16
      LDI   R16,LOW(RAMEND)
      OUT   SPL,R16           ;initialize stack pointer
      LDI   R16,0x20
      SBI   DDRB,5            ;PB5 as an output
      LDI   R18,-150
BEGIN:SBI   PORTB,5           ;PB5 = 1
      OUT   TCNT0,R18         ;load Timer0 byte
      CALL  DELAY
      OUT   TCNT0,R18         ;reload Timer0 byte
      CALL  DELAY
      CBI   PORTB,5           ;PB5 = 0
      OUT   TCNT0,R18         ;reload Timer0 byte
      CALL  DELAY
      RJMP BEGIN

;----- Delay using Timer0
DELAY:LDI   R20,0x01
      OUT   TCCR0,R20   ;start Timer0, Normal mode, int clk, no prescaler
AGAIN:IN    R20,TIFR    ;read TIFR
      SBRS  R20,TOV0    ;monitor TOV0 flag and skip if high
      RJMP  AGAIN
      LDI   R20,0x0
      OUT   TCCR0,R20   ;stop Timer0
      LDI   R20,1<<TOV0
      OUT   TIFR,R20    ;clear TOV0 flag bit
      RET
```

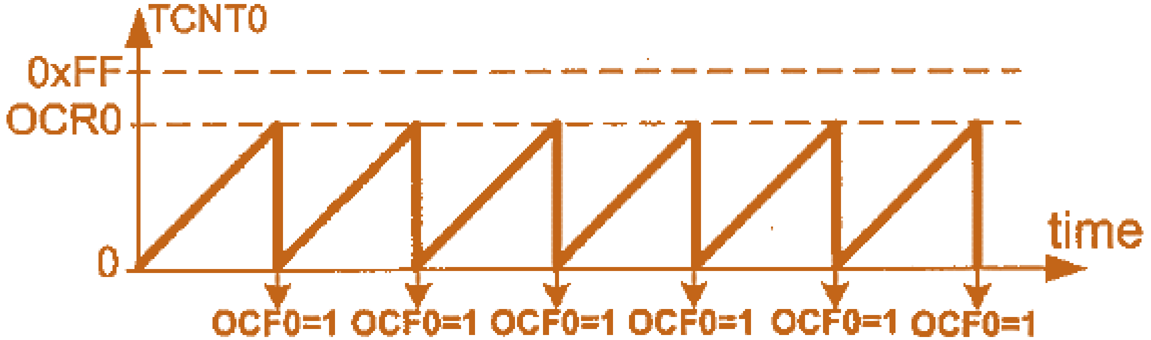
Example cnt.

Find (a) the frequency of the square wave generated in the following code, and (b) the duty cycle of this wave. Assume XTAL = 8 MHz.

For the TCNT0 value in 8-bit mode, the conversion is done by the assembler as long as we enter a negative number. This also makes the calculation easy. Because we are using 150 clocks, we have time for the DELAY subroutine = 150 × 0.125 μs = 18.75 μs. The high portion of the pulse is twice the size of the low portion (66% duty cycle). Therefore, we have: T = high portion + low portion = 2 × 18.75 μs + 18.75 μs = 56.25 μs and frequency = 1 / 56.25 μs = 17.777 kHz.

Clear Timer0 on Compare Match (CTC)

- OCR0 register is used with CTC mode
- In the CTC mode
  - As with the normal mode, the timer is incremented with a clock
  - But it counts up until the content of TCNT0 becomes equal to the content of OCR0
  - Then the timer will be cleared and the OCF0 flag will be set with the next clock



Example

In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the PORTB.5 bit. Timer0 is used to generate the time delay. Analyze the program.

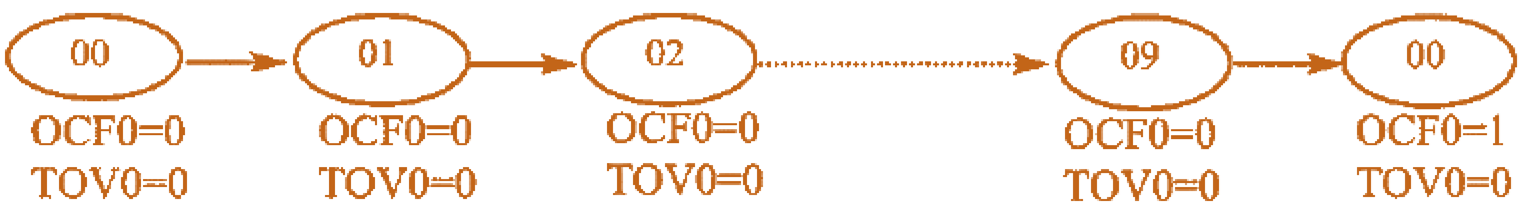
```
.INCLUDE "M32DEF.INC"
       INITSTACK
       LDI    R16,0x08
       SBI    DDRB,3               ;PB3 as an output
       LDI    R17,0
BEGIN:OUT    PORTB,R17            ;PORTB = R17
       RCALL DELAY
       EOR    R17,R16              ;toggle D3 of R17
       RJMP   BEGIN
;---------------- Timer0 Delay
DELAY:LDI    R20,0
       OUT    TCNT0,R20
       LDI    R20,9
       OUT    OCR0,R20             ;load OCR0
       LDI    R20,0x09
       OUT    TCCR0,R20            ;Timer0, CTC mode, int clk
AGAIN:IN     R20,TIFR             ;read TIFR
       SBRS   R20,OCF0             ;if OCF0 is set skip next inst.
       RJMP   AGAIN
       LDI    R20,0x0
       OUT    TCCR0,R20            ;stop Timer0
       LDI    R20,1<<OCF0
       OUT    TIFR,R20             ;clear OCF0 flag
       RET
```

Example cnt.

In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the PORTB.5 bit. Timer0 is used to generate the time delay. Analyze the program.

In the above program notice the following steps:

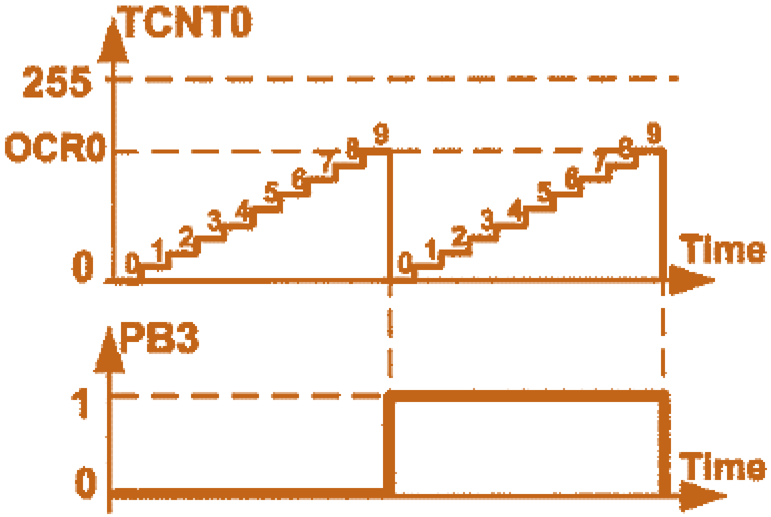
1. 9 is loaded into OCR0.
2. TCCR0 is loaded and Timer0 is started.
3. Timer0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of 00, 01, 02, 03, and so on until it reaches 9. One more clock rolls it to 0, raising the Timer0 compare match flag (OCF0 = 1). At that point, the "`SBRS R20,OCF0`" instruction bypasses the "`RJMP AGAIN`" instruction.
4. Timer0 is stopped.
5. The OCF0 flag is cleared.



Example

Find the delay generated by Timer0 in the last Example. Do not include the overhead due to instructions. (XTAL = 8 MHz)

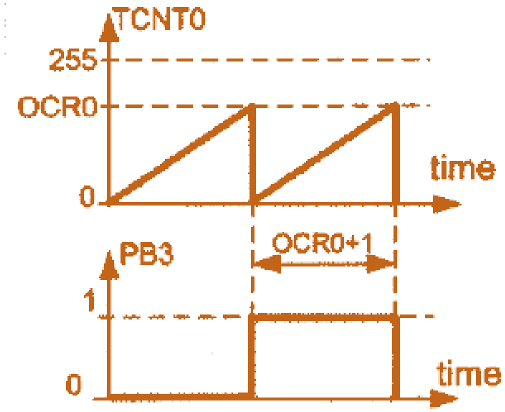
OCR0 is loaded with 9 and TCNT0 is cleared; Thus, after 9 clocks TCNT0 becomes equal to OCR0. On the next clock, the OCF0 flag is set and the reset occurs. That means the TCNT0 is cleared after 9 + 1 = 10 clocks. Because XTAL = 8 MHz, the counter counts up every 0.125 µs. Therefore, we have 10 × 0.125 µs = 1.25 µs.



Example

Find the delay generated by Timer0 in the following program. Do not include the overhead due to instructions. (XTAL = 8 MHz)

```
.INCLUDE "M32DEF.INC"
        LDI   R16,0x08
        SBI   DDRB,3        ;PB3 as an output
        LDI   R17,0
        OUT   PORTB,R17
        LDI   R20,89
        OUT   OCR0,R20      ;load Timer0
BEGIN:LDI   R20,0x0B
        OUT   TCCR0,R20     ;Timer0, CTC mode, prescaler = 64
AGAIN:IN    R20,TIFR      ;read TIFR
        SBRS  R20,OCF0      ;if OCF0 flag is set skip next instruction
        RJMP  AGAIN
        LDI   R20,0x0
        OUT   TCCR0,R20     ;stop Timer0 (This line can be omitted)
        LDI   R20,1<<OCF0
        OUT   TIFR,R20      ;clear OCF0 flag
        EOR   R17,R16       ;toggle D3 of R17
        OUT   PORTB,R17     ;toggle PB3
        RJMP  BEGIN
```



Due to prescaler = 64 each timer clock lasts 64 × 0.125 µs = 8 µs. OCR0 is loaded with 89; thus, after 90 clocks OCF0 is set. Therefore we have 90 × 8 µs = 720 µs.

Example

Assuming XTAL = 8 MHz, write a program to generate a delay of 25.6 ms. Use Timer0, CTC mode, with prescaler = 1024.

Due to prescaler = 1024 each timer clock lasts 1024 × 0.125 µs = 128 µs. Thus, in order to generate a delay of 25.6 ms we should wait 25.6 ms / 128 µs = 200 clocks. Therefore the OCR0 register should be loaded with 200 – 1 = 199.

```
DELAY:LDI   R20,0
      OUT   TCNT0,R20
      LDI   R20,199
      OUT   OCR0,R20          ;load OCR0
      LDI   R20,0x0D
      OUT   TCCR0,R20         ;Timer0, CTC mode, prescaler = 1024
AGAIN:IN    R20,TIFR          ;read TIFR
      SBRS  R20,OCF0          ;if OCF0 is set skip next inst.
      RJMP  AGAIN
      LDI   R20,0x0
      OUT   TCCR0,R20         ;stop Timer0
      LDI   R20,1<<OCF0
      OUT   TIFR,R20          ;clear OCF0 flag
      RET
```

Example

Assuming XTAL = 8 MHz, write a program to generate a delay of 1 ms.

As XTAL = 8 MHz, the different outputs of the prescaler are as follows:

Prescaler	Timer Clock	Timer Period	Timer Value
None	8 MHz	1/8 MHz = 0.125 µs	1 ms/0.125 µs = 8000
8	8 MHz/8 = 1 MHz	1/1 MHz = 1 µs	1 ms/1 µs = 1000
64	8 MHz/64 = 125 kHz	1/125 kHz = 8 µs	1 ms/8 µs = **125**
256	8 MHz/256 = 31.25 kHz	1/31.25 kHz = 32 µs	1 ms/32 µs = **31.25**
1024	8 MHz/1024 = 7.8125 kHz	1/7.8125 kHz= 128 µs	1 ms/128 µs = **7.8125**

From the above calculation we can only use the options Prescaler = 64, Prescaler = 256, or Prescaler = 1024. We should use the option Prescaler = 64 since we cannot use a decimal point. To wait 125 clocks we should load OCR0 with 125 – 1 = 124.

```
DELAY:LDI    R20,0
      OUT    TCNT0,R20          ;TCNT0 = 0
      LDI    R20,124
      OUT    OCR0,R20           ;OCR0 = 124
      LDI    R20,0x0B
      OUT    TCCR0,R20          ;Timer0, CTC mode, prescaler = 64
AGAIN:IN     R20,TIFR           ;read TIFR
      SBRS   R20,OCF0           ;if OCF0 is set skip next instruction
      RJMP   AGAIN
      LDI    R20,0x0
      OUT    TCCR0,R20          ;stop Timer0
      LDI    R20,1<<OCF0
      OUT    TIFR,R20           ;clear OCF0 flag
      RET
```

Notice!

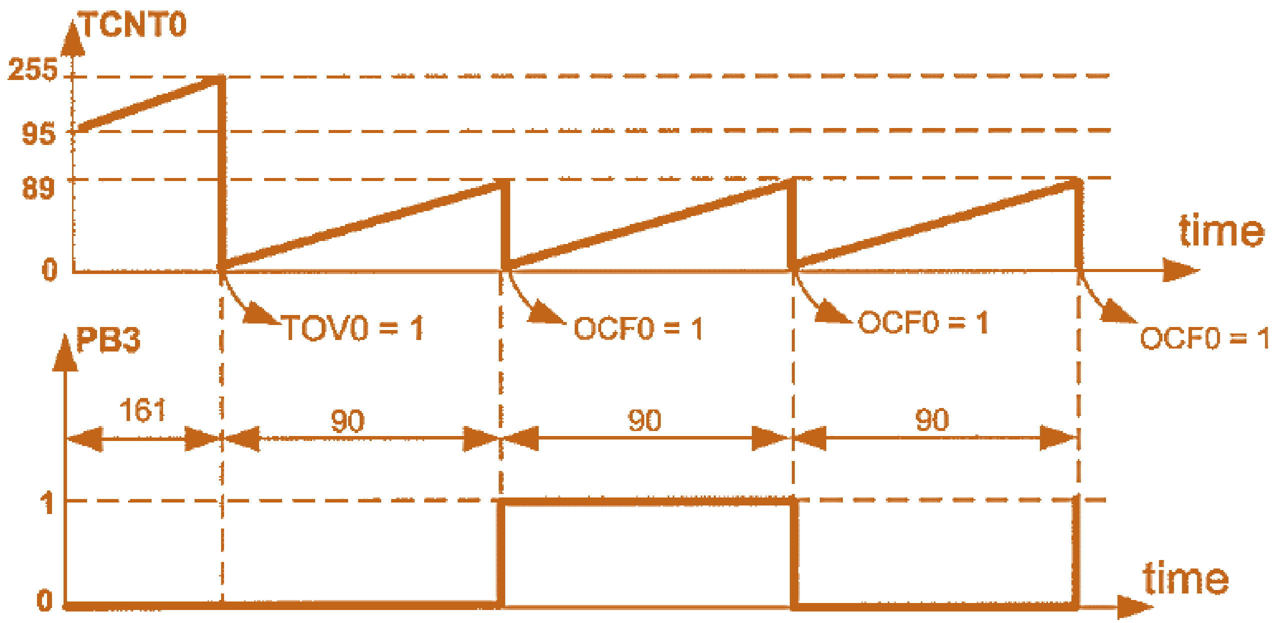
Notice that the comparator checks for equality; thus, if we load the OCR0 register with a value that is smaller than TCNT0's value, the counter will miss the compare match and will count up until it reaches the maximum value of $FF and rolls over. This causes a big delay and is not desirable in many cases.

Example:

In the following program, how long does it take for the PB3 to become one? Do not include the overhead due to instructions. (XTAL = 8 MHz)

```
.INCLUDE "M32DEF.INC"
        SBI     DDRB,3          ;PB3 as an output
        CBI     PORTB,3         ;PB3 = 0
        LDI     R20,89
        OUT     OCR0,R20        ;OCR0 = 89
        LDI     R20,95
        OUT     TCNT0,R20       ;TCNT0 = 95
BEGIN:LDI       R20,0x09
        OUT     TCCR0,R20       ;Timer0, CTC mode, prescaler = 1
AGAIN:IN        R20,TIFR        ;read TIFR
        SBRS    R20,OCF0        ;if OCF0 flag is set skip next inst.
        RJMP    AGAIN
        LDI     R20,0x0
        OUT     TCCR0,R20       ;stop Timer0 (This line can be omitted)
        LDI     R20,1<<OCF0
        OUT     TIFR,R20    ;clear OCF0 flag
        EOR     R17,R16     ;toggle D3 of R17
        OUT     PORTB,R17   ;toggle PB3
        RJMP    BEGIN
```

Example cnt.



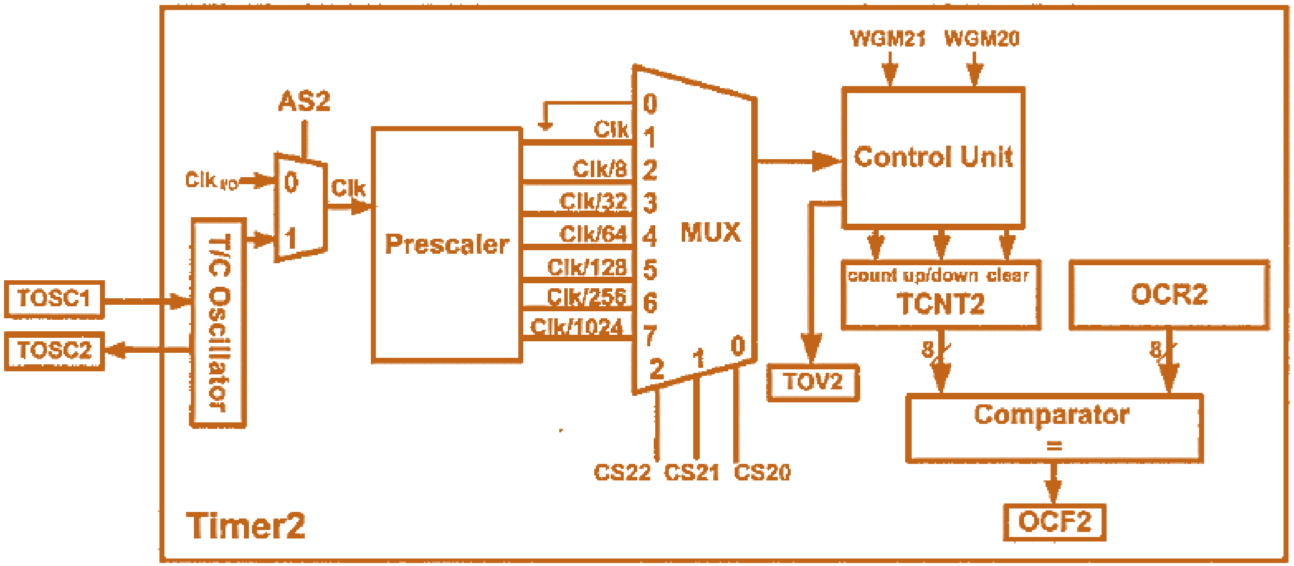
Since the value of TCNT0 (95) is bigger than the content of OCR0 (89), the timer counts up until it gets to \$FF and rolls over to zero. The TOV0 flag will be set as a result of the overflow. Then, the timer counts up until it becomes equal to 89 and compare match occurs. Thus, the first compare match occurs after $161 + 90 = 251$ clocks, which means after $251 \times 0.125\ \mu s = 31.375\ \mu s$. The next compare matches occur after 90 clocks, which means after $90 \times 0.125\ \mu s = 11.25\ \mu s$.

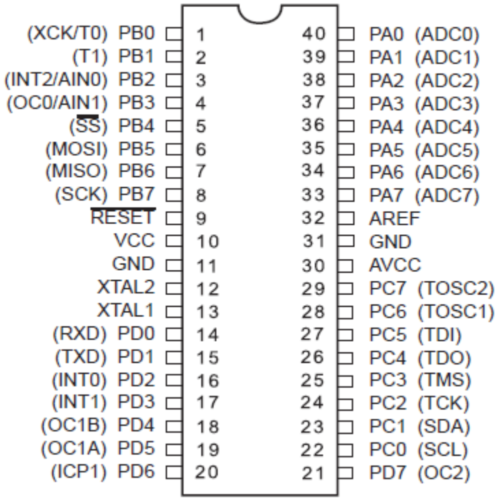
Timer2 ATmega32

Timer2 is an 8-bit timer. Therefore it works the same way as Timer0.
But there are two differences between Timer0 and Timer2:

1. Timer2 can be used as a real time counter. To do so, we should connect a crystal of 32.768 kHz to the TOSC1 and TOSC2 pins of AVR and set the AS2 bit.

2. In Timer0, when CS02–CS00 have values 110 or 111, Timer0 counts the external events. But in Timer2, the multiplexer selects between the different scales of the clock. In other words, the same values of the CS bits can have different meanings for Timer0 and Timer2.





Timer2 ATmega32

Bit	7	6	5	4	3	2	1	0
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

FOC2 D7 Force compare match: a write-only bit, which can be used while generating a wave. Writing 1 to it causes the wave generator to act as if a compare match had occurred.

WGM20, WGM21

D6	D3	Timer2 mode selector bits
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

COM21:20 D5 D4 Compare Output Mode:
These bits control the waveform generator (see Chapter 15).

CS22:20

D2	D1	D0	Timer2 clock selector
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 32
1	0	0	clk / 64
1	0	1	clk / 128
1	1	0	clk / 256
1	1	1	clk / 1024

Asynchronous status register (ASSR)

Bit	7	6	5	4	3	2	1	0
					AS2	TCN2UB	OCR2UB	TCR2UB

AS2 When it is zero, Timer2 is clocked from $clk_{I/O}$. When it is set, Timer2 works as RTC.

Example (Timer2 Programming)

Find the value for TCCR2 if we want to program Timer2 in normal mode with a prescaler of 64 using internal clock for the clock source.

we have TCCR2 = 0000 0100; XTAL clock source, prescaler of 64.

	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
TCCR2 =	0	0	0	0	0	1	0	0

Example

Using a prescaler of 64, write a program to generate a delay of 1920 μs. Assume XTAL = 8 MHz.

Timer clock = 8 MHz/64 = 125 kHz → Timer Period = 1 / 125 kHz = 8 μs → Timer Value = 1920 μs / 8 μs = 240

```
;----------------- Timer2 Delay
DELAY:LDI    R20,-240     ;R20 = 0x10
      OUT    TCNT2,R20    ;load Timer2
      LDI    R20,0x04
      OUT    TCCR2,R20    ;Timer2, Normal mode, int clk, prescaler 64
AGAIN:IN     R20,TIFR     ;read TIFR
      SBRS   R20,TOV2     ;if TOV2 is set skip next instruction
      RJMP   AGAIN
      LDI    R20,0x0
      OUT    TCCR2,R20    ;stop Timer2
      LDI    R20,1<<TOV2
      OUT    TIFR,R20     ;clear TOV2 flag
      RET
```

Example

Using CTC mode, write a program to generate a delay of 8 ms. Assume XTAL = 8 MHz.

As XTAL = 8 MHz, the different outputs of the prescaler are as follows:

Prescaler	Timer Clock	Timer Period	Timer Value
None	8 MHz	1/8 MHz = 0.125 μs	8 ms / 0.125 μs = 64 k
8	8 MHz/8 = 1 MHz	1/1 MHz = 1 μs	8 ms / 1 μs = 8000
32	8 MHz/32 = 250 kHz	1/250 kHz = 4 μs	8 ms / 4 μs = 2000
64	8 MHz/64 = 125 kHz	1/125 kHz = 8 μs	8 ms / 8 μs = 1000
128	8 MHz/128 = 62.5 kHz	1/62.5 kHz = 16 μs	8 ms / 16 μs = 500
256	8 MHz/256 = 31.25 kHz	1/31.25 kHz = 32 μs	8 ms / 32 μs = **250**
1024	8 MHz/1024 = 7.8125 kHz	1/7.8125 kHz= 128 μs	8 ms / 128 μs = **62.5**

From the above calculation we can only use options Prescaler = 256 or Prescaler = 1024. We should use the option Prescaler = 256 since we cannot use a decimal point. To wait 250 clocks we should load OCR2 with 250 – 1 = 249.

Example cnt.

Using CTC mode, write a program to generate a delay of 8 ms. Assume XTAL = 8 MHz.

TCCR2 =	0	0	0	0	1	1	1	0
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20

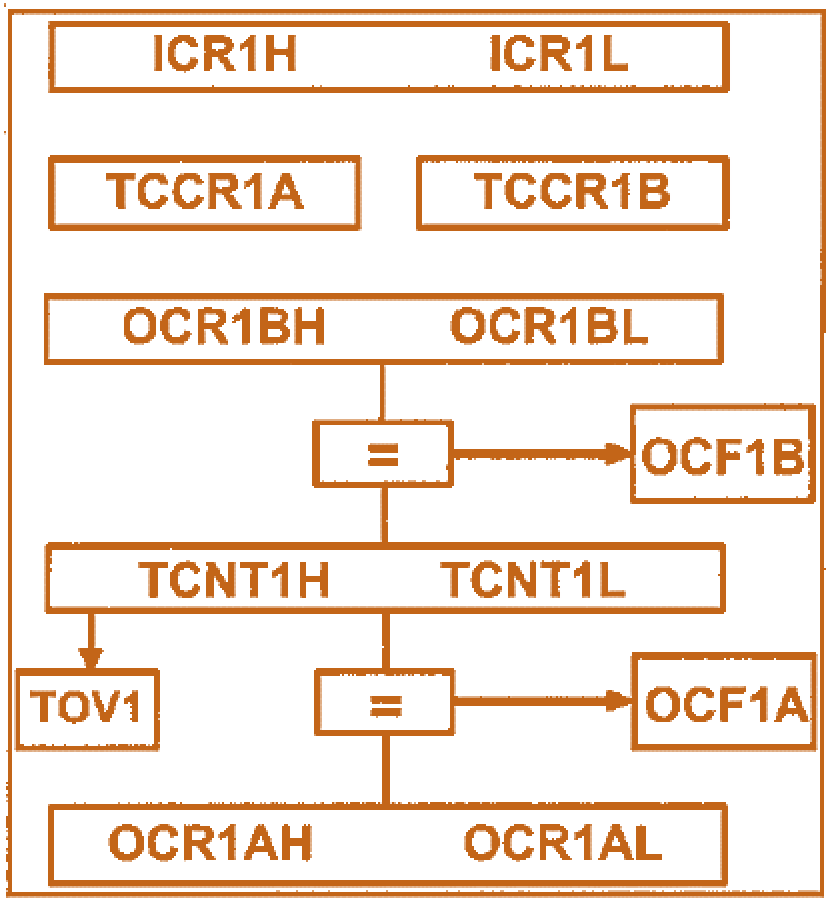
```
;---------------- Timer2 Delay
DELAY:LDI   R20,0
      OUT   TCNT2,R20         ;TCNT2 = 0
      LDI   R20,249
      OUT   OCR2,R20          ;OCR2 = 249
      LDI   R20,0x0E
      OUT   TCCR2,R20         ;Timer2,CTC mode,prescaler = 256
AGAIN:IN    R20,TIFR          ;read TIFR
      SBRS  R20,OCF2          ;if OCF2 is set skip next inst.
      RJMP  AGAIN
      LDI   R20,0x0
      OUT   TCCR2,R20         ;stop Timer2
      LDI   R20,1<<OCF2
      OUT   TIFR,R20          ;clear OCF2 flag
      RET
```

Timer1 ATmega32

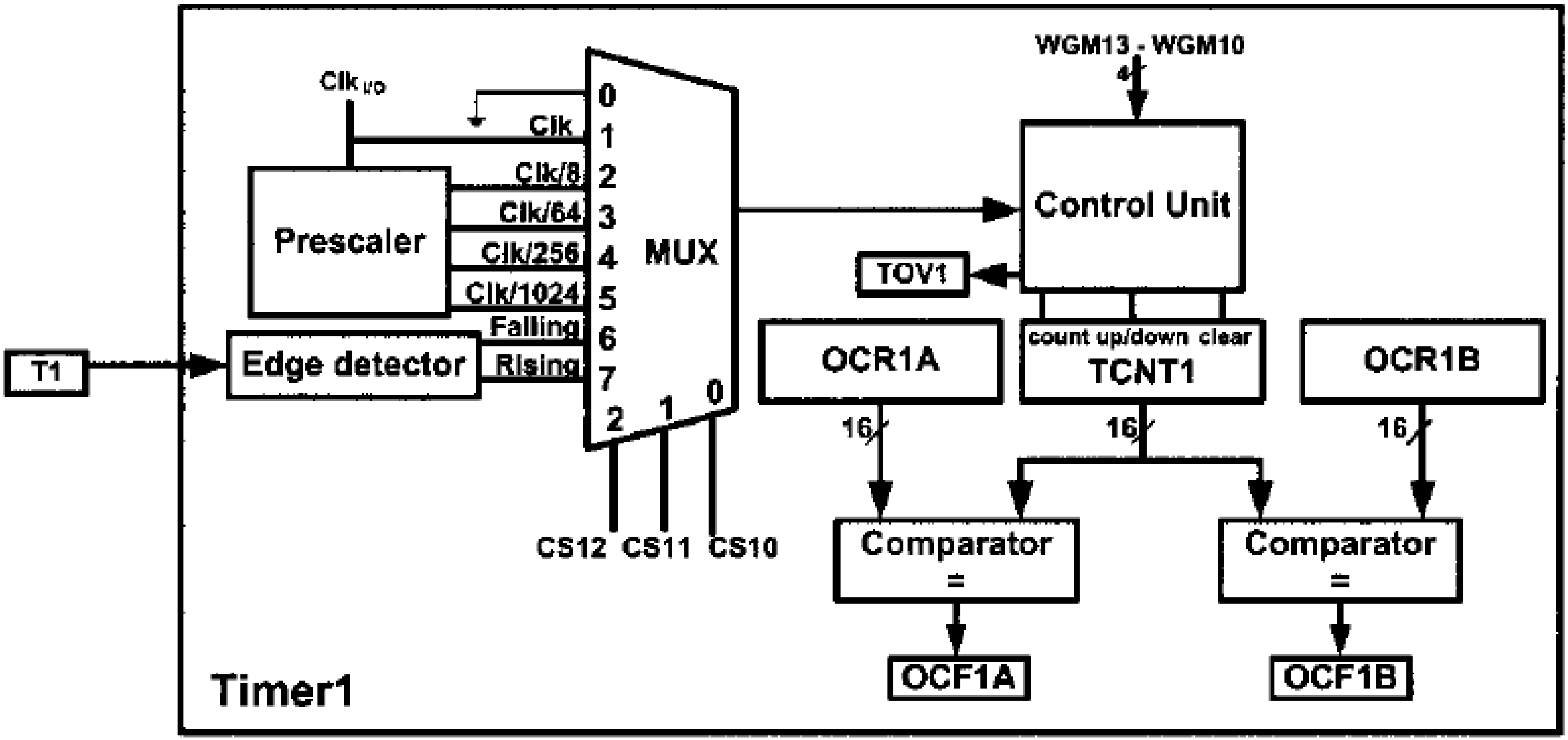
Since Timer1 is a 16-bit timer its 16-bit register is split into two bytes. These are referred to as TCNT1L (Timer1 low byte) and TCNT1H (Timer1 high byte).

Timer1 also has two control registers named TCCR1A (Timer/counter 1 control register) and TCCR1B. The TOV1 (timer overflow) flag bit goes HIGH when overflow occurs. Timer1 also has the prescaler options of 1:1, 1:8, 1:64, 1:256, and 1:1024.

There are two OCR registers in Timer1: OCR1A and OCR1B. There are two separate flags for each of the OCR registers, which act independently of each other.



Timer1 ATmega32



Timer1 ATmega32

The TIFR register contains the TOV1, OCF1A, and OCF1B flags.

Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

TOV0	D0	Timer0 overflow flag bit
	0 = Timer0 did not overflow.	
	1 = Timer0 has overflowed (going from \$FF to \$00).	
OCF0	D1	Timer0 output compare flag bit
	0 = compare match did not occur.	
	1 = compare match occurred.	
TOV1	D2	Timer1 overflow flag bit
OCF1B	D3	Timer1 output compare B match flag
OCF1A	D4	Timer1 output compare A match flag
ICF1	D5	Input Capture flag
TOV2	D6	Timer2 overflow flag
OCF2	D7	Timer2 output compare match flag

TCCR

Bit	7	6	5	4	3	2	1	0
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

COM1A1:COM1A0	D7 D6	Compare Output Mode for Channel A (discussed in Section 9-3)
COM1B1:COM1B0	D5 D4	Compare Output Mode for Channel B (discussed in Section 9-3)
FOC1A	D3	Force Output Compare for Channel A (discussed in Section 9-3)
FOC1B	D2	Force Output Compare for Channel B
WGM11:10	D1 D0	Timer1 mode

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	**TCCR1B**
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ICNC1	D7	Input Capture Noise Canceler 0 = Input Capture is disabled. 1 = Input Capture is enabled.
ICES1	D6	Input Capture Edge Select 0 = Capture on the falling (negative) edge 1 = Capture on the rising (positive) edge
	D5	Not used
WGM13:WGM12	D4 D3	Timer1 mode

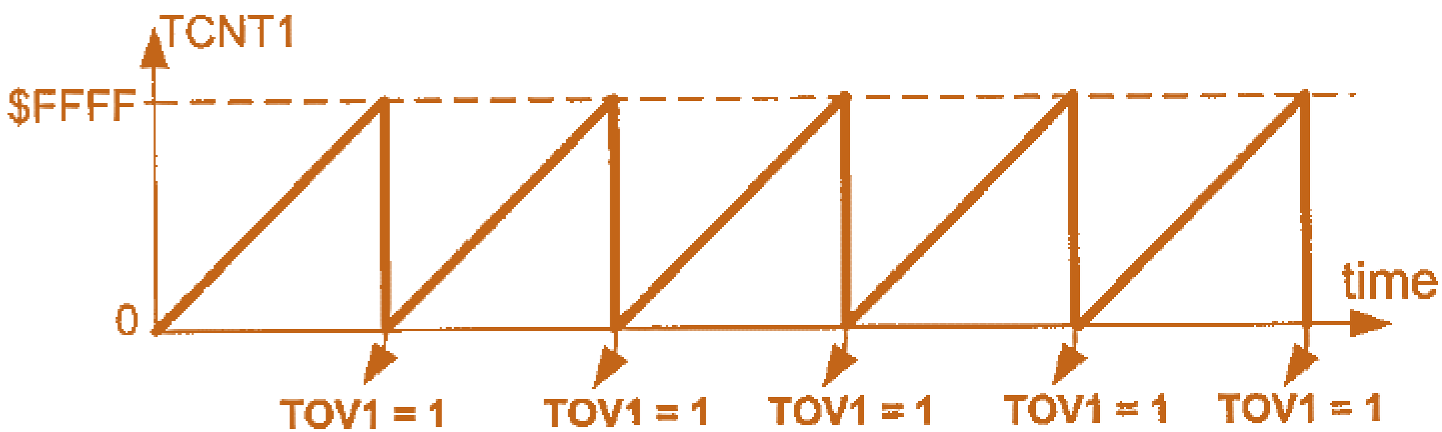
TCCR

Mode	WGM13	WGM12	WGM11	WGM10	Timer/Counter Mode of Operation	Top	Update of OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

CS12:CS10	D2D1D0	Timer1 clock selector
	0 0 0	No clock source (Timer/Counter 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 clock source on T1 pin. Clock on falling edge.
	1 1 1	External clock source on T1 pin. Clock on rising edge.

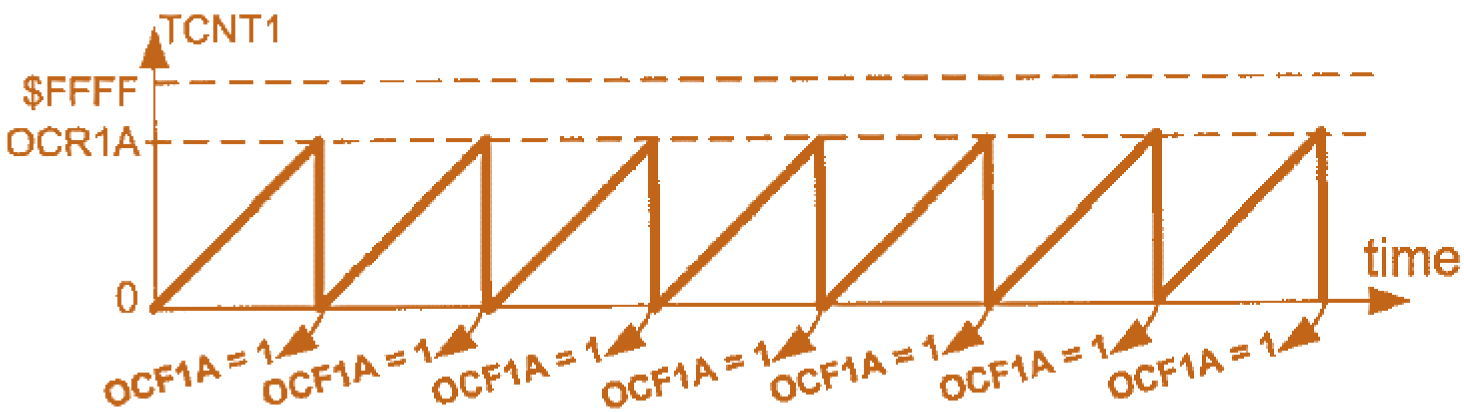
Normal Mode (WGM13:10=0000)

In this mode, the timer counts up until it reaches $FFFF (which is the maximum value) and then it rolls over from $FFFF to 0000. When the timer rolls over from $FFFF to 0000, the TOV1 flag will be set.



CTC Mode (WGM13:10=0100)

In mode 4, the timer counts up until the content of the TCNT1 register becomes equal to the content of OCR1A (compare match occurs); then, the timer will be cleared when the next clock occurs. The OCF1A flag will be set as a result of the compare match as well.



Example

Find the values for TCCR1A and TCCR1B if we want to program Timer1 in mode 0 (Normal), with no prescaler. Use AVR's crystal oscillator for the clock source.

TCCR1A = 0000 0000 WGM11 = 0, WGM10 = 0
TCCR1B = 0000 0001 WGM13 = 0, WGM12 = 0, oscillator clock source, no prescaler

Find the values for TCCR1A and TCCR1B if we want to program Timer1 in mode 4 (CTC, Top = OCR1A), no prescaler. Use AVR's crystal oscillator for the clock source.

TCCR1A = 0000 0000 WGM11 = 0, WGM10 = 0
TCCR1B = 0000 1001 WGM13 = 0, WGM12 = 1, oscillator clock source, no prescaler

Example

Find the frequency of the square wave generated by the following program if XTAL = 8 MHz. In your calculation do not include the overhead due to instructions in the loop.

```
.INCLUDE "M32DEF.INC"
      INITSTACK
      LDI   R16,0x20
      SBI   DDRB,5       ;PB5 as an output
      LDI   R17,0
      OUT   PORTB,R17    ;PB5 = 0
BEGIN:RCALL DELAY
      EOR   R17,R16      ;toggle D5 of R17
      OUT   PORTB,R17    ;toggle PB5
      RJMP  BEGIN
;--------------- Timer1 delay
DELAY:LDI   R20,0xD8
      OUT   TCNT1H,R20   ;TCNT1H = 0xD8
      LDI   R20,0xF0
      OUT   TCNT1L,R20   ;TCNT1L = 0xF0
      LDI   R20,0x00
      OUT   TCCR1A,R20   ;WGM11:10 = 00
      LDI   R20,0x01
      OUT   TCCR1B,R20   ;WGM13:12 = 00, Normal mode, prescaler = 1
AGAIN:IN    R20,TIFR     ;read TIFR
      SBRS  R20,TOV1     ;if TOV1 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x00
      OUT   TCCR1B,R20   ;stop Timer1
      LDI   R20,0x04
      OUT   TIFR,R20     ;clear TOV1 flag
      RET
```

Example cnt.

Find the frequency of the square wave generated by the following program if XTAL = 8 MHz. In your calculation do not include the overhead due to instructions in the loop.

WGM13:10 = 0000 = 0x00, so Timer1 is working in mode 0, which is Normal mode, and the top is 0xFFFF.
FFFF + 1 – D8F0 = 0x2710 = 10,000 clocks, which means that it takes 10,000 clocks.
As XTAL = 8 MHz each clock lasts 1/(8M) = 0.125 μs and delay = 10,000 × 0.125 μs = 1250 μs = 1.25 ms and frequency = 1 / (1.25 ms × 2) = 400 Hz.
In this calculation, the overhead due to all the instructions in the loop is not included.

Notice that instead of using hex numbers we can use HIGH and LOW directives, as shown below:

```
LDI    R20,HIGH (65536-10000)         ;load Timer1 high byte
OUT    TCNT1H,R20   ;TCNT1H = 0xD8
LDI    R20,LOW (65536-10000)          ;load Timer1 low byte
OUT    TCNT1L,R20   ;TCNT1L = 0xF0
```

or we can simply write it as follows:

```
LDI    R20,HIGH (-10000)              ;load Timer1 high byte
OUT    TCNT1H,R20   ;TCNT1H = 0xD8
LDI    R20,LOW (-10000)               ;load Timer1 low byte
OUT    TCNT1L,R20   ;TCNT1L = 0xF0
```

Example

Find the frequency of the square wave generated by the following program if XTAL = 8 MHz. In your calculation do not include the overhead due to instructions in the loop.

```
.INCLUDE "M32DEF.INC"
      SBI   DDRB,5              ;PB5 as an output
BEGIN:SBI   PORTB,5             ;PB5 = 1
      RCALL DELAY
      CBI   PORTB,5             ;PB5 = 0
      RCALL DELAY
      RJMP  BEGIN
;--------------- Timer1 delay
DELAY:LDI   R20,0x00
      OUT   TCNT1H,R20
      OUT   TCNT1L,R20          ;TCNT1 = 0
      LDI   R20,0
      OUT   OCR1AH,R20
      LDI   R20,159
      OUT   OCR1AL,R20          ;OCR1A = 159 = 0x9F
      LDI   R20,0x0
      OUT   TCCR1A,R20          ;WGM11:10 = 00
      LDI   R20,0x09
      OUT   TCCR1B,R20          ;WGM13:12 = 01,CTC mode, prescaler = 1
AGAIN:IN    R20,TIFR            ;read TIFR
      SBRS  R20,OCF1A           ;if OCF1A is set skip next instruction
      RJMP  AGAIN
      LDI   R20,1<<OCF1A
      OUT   TIFR,R20            ;clear OCF1A flag
      LDI   R19,0
      OUT   TCCR1B,R19          ;stop timer
      OUT   TCCR1A,R19
      RET
```

Example

Find the frequency of the square wave generated by the following program if XTAL = 8 MHz. In your calculation do not include the overhead due to instructions in the loop.

WGM13:10 = 0100 = 0x04 therefore, Timer1 is working in mode 4, which is a CTC mode, and max is defined by OCR1A.

159 + 1 = 160 clocks

XTAL = 8 MHz, so each clock lasts 1/(8M) = 0.125 μs.

Delay = 160 × 0.125 μs = 20 μs and frequency = 1 / (20 μs × 2) = 25 kHz.

In this calculation, the overhead due to all the instructions in the loop is not included.

Accessing 16-bit Registers

The AVR is an 8-bit microcontroller, which means it can manipulate data 8 bits at a time, only. But some Timer1 registers, such as TCNT1, OCR1A, ICR1, and so on, are 16-bit; in this case, the registers are split into two 8-bit registers, and each one is accessed individually. This is fine for most cases. For example, when we want to load the content of SP (stack pointer), we first load one half and then the other half, as shown below:

```
LDI   R16, 0x12
OUT   SPL, R16
LDI   R16, 0x34
OUT   SPH, R16   ;SP = 0x3412
```

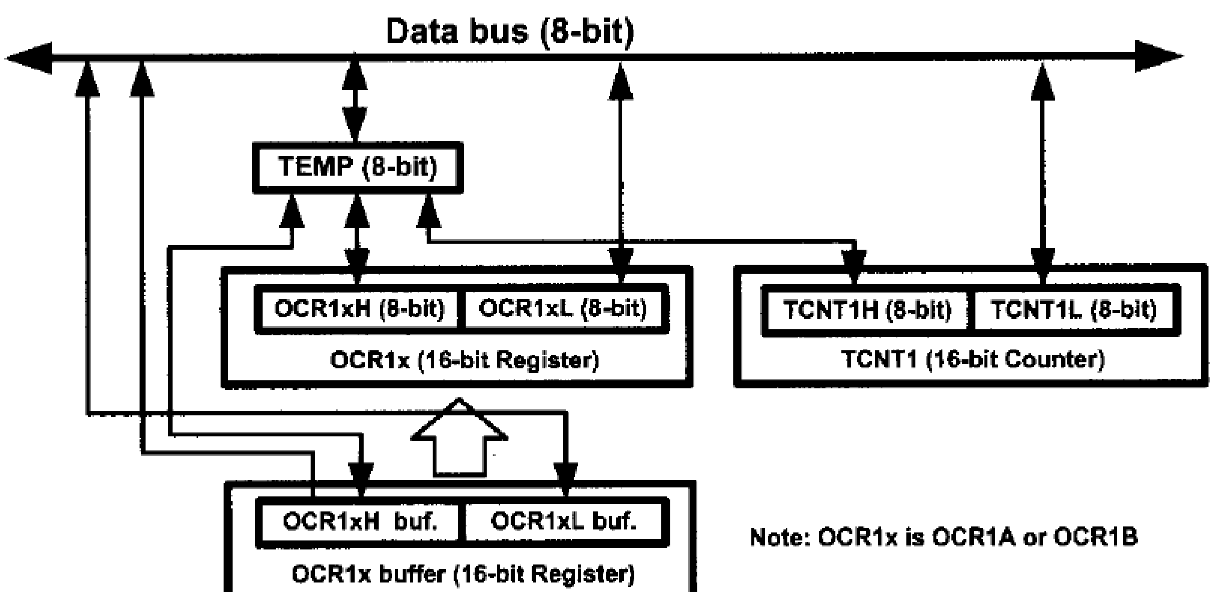
In 16-bit timers, however, we should read/write the entire content of a register at once, otherwise we might have problems. For example, imagine the following scenario:

The TCNT1 register contains 0x15FF. We read the low byte of TCNT1, which is 0xFF, and store it in R20. At the same time a timer clock occurs, and the content of TCNT1 becomes 0x1600; now we read the high byte of TCNT1, which is now 0x16, and store it in R21. If we look at the value we have read, R21:R20 = 0x16FF. So, we believe that TCNT1 contains 0x16FF, although it actually contains 0x15FF.

Accessing 16-bit Registers

This problem exists in many 8-bit microcontrollers. But the AVR designers have resolved this issue with an 8-bit register called TEMP, which is used as a buffer. When we write or read the high byte of a 16-bit register, such as TCNT1, the value will be written into the TEMP register. When we write into the low byte of a 16-bit register, the content of TEMP will be written into the high byte of the 16-bit register as well. For example, consider the following program:

```
LDI   R16, 0x15
OUT   TCNT1H, R16      ;store 0x15 in TEMP of Timer1
LDI   R16, 0xFF
OUT   TCNT1L, R16      ;TCNT1L = R16, TCNT1H = TEMP
```



Accessing 16-bit Registers cnt.

After the execution of "OUT TCNT1H, R16", the content of R16, 0x15, will be stored in the TEMP register. When the instruction "OUT TCNT1L, R16" is executed, the content of R16, 0xFF, is loaded into TCNT1L, and the content of the TEMP register, 0x15, is loaded into TCNT1H. So, 0x15FF will be loaded into the TCNT1 register at once.

Notice that according to the internal circuitry of the AVR, we should first write into the high byte of the 16-bit registers and then write into the lower byte. Otherwise, the program does not work properly. For example, the following code:

```
LDI   R16, 0xFF
OUT   TCNT1L, R16      ;TCNT1L = R16, TCNT1H = TEMP
LDI   R16, 0x15
OUT   TCNT1H, R16      ;store 0x15 in TEMP of Timer1
```

does not work properly. This is because, when the TCNT1L is loaded, the content of TEMP will be loaded into TCNT1H. But when the TCNT1L register is loaded, TEMP contains garbage (improper data), and this is not what we want.

Counter Programming

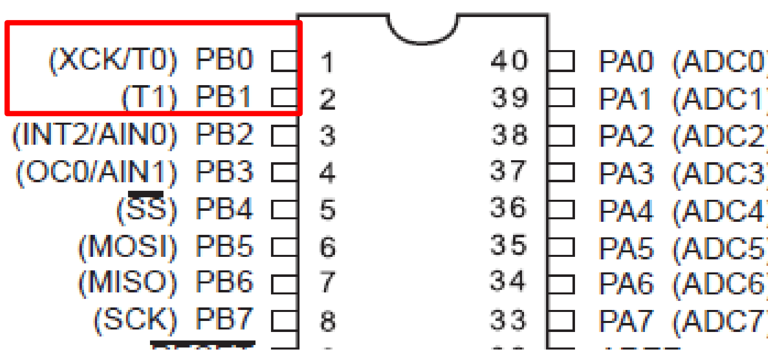
- Use of timer as an event counter
- Source of frequency
  - timer→ AVR's crystal
  - Counter→ pulse outside the AVR
- In counter mode
  - Registers such as TCNT, TCCR and OCR0 are the same as for the timer

Counter Programming

TCCR0 register decide the source of the clock for the timer. If CS02:00 is between 1 and 5, the timer gets pulses from the crystal oscillator. In contrast, when CS02:00 is 6 or 7, the timer is used as a counter and gets its pulses from a source outside the AVR chip.

Therefore, when CS02:00 is 6 or 7, the TCNT0 counter counts up as pulses are fed from pin T0 (Timer/Counter 0 External Clock input). In ATmega32/ATmega16, T0 is the alternative function of PORTB.0.

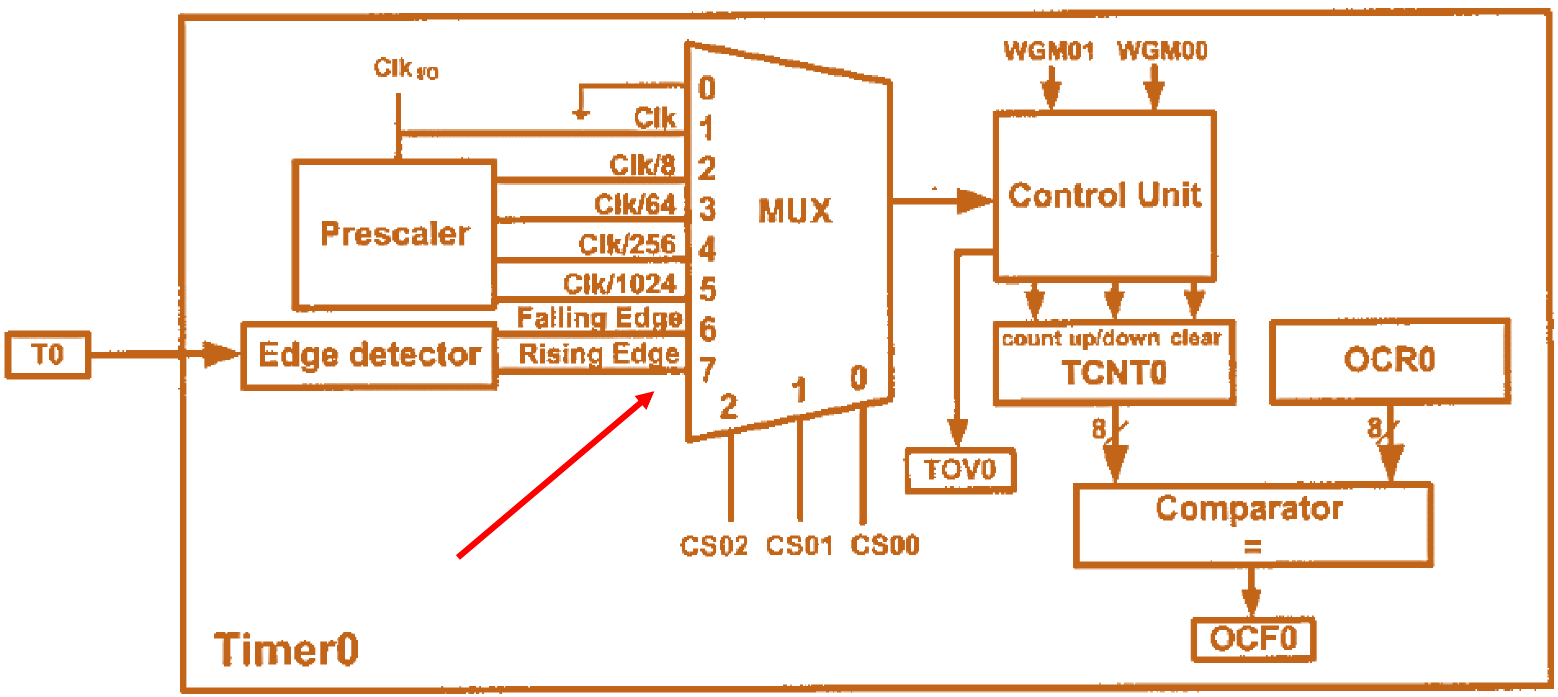
In the Timer0, when CS02:00 is 6 or 7, pin T0 provides the clock pulse and the counter counts up after each clock pulse coming from that pin. Similarly, for Timer1, when CS12:10 is 6 or 7, the clock pulse coming in from pin T1 (Timer/Counter 1 External Clock input) makes the TCNT1 counter count up. When CS12:10 is 6, the counter counts up on the negative (falling) edge. When CS12:10 is 7, the counter counts up on the positive (rising) edge. In ATmega32/ATmega16, T1 is the alternative function of PORTB.1.



Example

Find the value for TCCR0 if we want to program Timer0 as a Normal mode counter. Use an external clock for the clock source and increment on the positive edge.

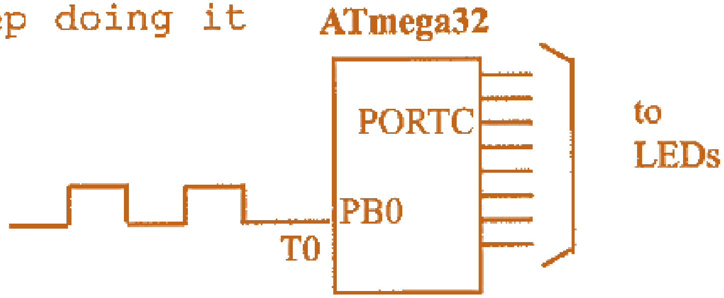
TCCR0 = 0000 0111 Normal, external clock source, no prescaler



Example

Assuming that a 1 Hz clock pulse is fed into pin T0 (PB0), write a program for Counter0 in normal mode to count the pulses on falling edge and display the state of the TCNT0 count on PORTC.

```
.INCLUDE "M32DEF.INC"
      CBI   DDRB,0          ;make T0 (PB0) input
      LDI   R20,0xFF
      OUT   DDRC,R20        ;make PORTC output
      LDI   R20,0x06
      OUT   TCCR0,R20       ;counter, falling edge
AGAIN:
      IN    R20,TCNT0
      OUT   PORTC,R20       ;PORTC = TCNT0
      IN    R16,TIFR
      SBRS  R16,TOV0        ;monitor TOV0 flag
      RJMP  AGAIN           ;keep doing if Timer0 flag is low
      LDI   R16,1<<TOV0
      OUT   TIFR, R16       ;clear TOV0 flag
      RJMP  AGAIN           ;keep doing it
```

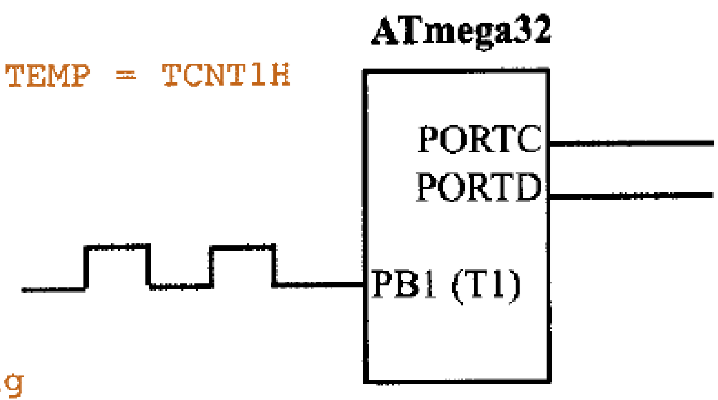


PORTC is connected to 8 LEDs and input T0 (PB0) to 1 Hz pulse.

Example

Assuming that clock pulses are fed into pin T1 (PB1), write a program for Counter1 in Normal mode to count the pulses on falling edge and display the state of the TCNT1 count on PORTC and PORTD.

```
.INCLUDE "M32DEF.INC"

        CBI     DDRB,1          ;make T1 (PB1) input
        LDI     R20,0xFF
        OUT     DDRC,R20        ;make PORTC output
        OUT     DDRD,R20        ;make PORTD output
        LDI     R20,0x0
        OUT     TCCR1A,R20
        LDI     R20,0x06
        OUT     TCCR1B,R20      ;counter, falling edge

AGAIN:
        IN      R20,TCNT1L      ;R20 = TCNT1L, TEMP = TCNT1H
        OUT     PORTC,R20       ;PORTC = TCNT0
        IN      R20,TCNT1H      ;R20 = TEMP
        OUT     PORTD,R20       ;PORTD = TCNT0
        IN      R16,TIFR
        SBRS    R16,TOV1
        RJMP    AGAIN           ;keep doing it
        LDI     R16,1<<TOV1     ;clear TOV1 flag
        OUT     TIFR, R16
        RJMP    AGAIN           ;keep doing it
```



Notice!

might think monitoring the TOV and OCR flags is a waste of the microcontroller's time. You are right. There is a solution to this: the use of interrupts. Using interrupts enables us to do other things with the microcontroller. When a timer Interrupt flag such as TOV0 is raised it will inform us. This important and powerful feature of the AVR is discussed next .

Programming Timers in C

As we saw , the general-purpose registers of the AVR are under the control of the C compiler and are not accessed directly by C statements. All of the SFRs (Special Function Registers), however, are accessible directly using C statements. As an example of accessing the SFRs directly, we saw how to access ports PORTB–PORTD .

In C we can access timer registers such as TCNT0, OCR0, and TCCR0 directly using their names.

```
PORTB = 0x01;
DDRC = 0xFF;
DDRD = 0xFF;

TCCR1A = 0x00;
TCCR1B = 0x06;

TCNT1H = 0x00;
TCNT1L = 0x00;
```

Example

Write a C program to toggle all the bits of PORTB continuously with some delay. Use Timer0, Normal mode, and no prescaler options to generate the delay.

```
#include "avr/io.h"
void T0Delay ( );
int main ( )
{
      DDRB = 0xFF;        //PORTB output port

      while (1)
      {
            PORTB = 0x55;     //repeat forever
            T0Delay ( );      //delay size unknown
            PORTB = 0xAA;     //repeat forever
            T0Delay ( );
      }
}

void T0Delay ( )
{
      TCNT0 = 0x20;              //load TCNT0
      TCCR0 = 0x01;              //Timer0, Normal mode, no prescaler
      while ((TIFR&0x1)==0);     //wait for TF0 to roll over
      TCCR0 = 0;
      TIFR = 0x1;                //clear TF0
}
```

Example

Write a C program to toggle only the PORTB.4 bit continuously every 70 μs. Use Timer0, Normal mode, and 1:8 prescaler to create the delay. Assume XTAL = 8 MHz.

XTAL = 8MHz ➔ $T_{machine\ cycle}$ = 1/8 MHz

Prescaler = 1:8 ➔ T_{clock} = 8 × 1/8 MHz = 1 μs

70 μs/1 μs = 70 clocks ➔ 1 + 0xFF – 70 = 0x100 – 0x46 = 0xBA = **186**

```
#include "avr/io.h"

void T0Delay ( );

int main ( )
{
        DDRB = 0xFF;        //PORTB output port

        while (1)
        {
                T0Delay ( );            //Timer0, Normal mode
                PORTB = PORTB ^ 0x10;   //toggle PORTB.4
        }
}

void T0Delay ( )
{
        TCNT0 = 186;        //load TCNT0
        TCCR0 = 0x02;       //Timer0, Normal mode, 1:8 prescaler
        while ((TIFR&(1<<TOV0))==0);  //wait for TOV0 to roll over

        TCCR0 = 0;          //turn off Timer0
        TIFR = 0x1;         //clear TOV0
}
```

Example

Write a C program to toggle only the PORTB.4 bit continuously every 2 ms. Use Timer1, Normal mode, and no prescaler to create the delay. Assume XTAL = 8 MHz.

XTAL = 8 MHz ➔ $T_{machine\ cycle}$ = 1/8 MHz = 0.125 µs

Prescaler = 1:1 ➔ T_{clock}= 0.125 µs

2 ms/0.125 µs = 16,000 clocks = 0x3E80 clocks   1 + 0xFFFF – 0x3E80 = 0xC180

```
#include "avr/io.h"

void T1Delay ( );

int main ( )
{
	DDRB = 0xFF;            //PORTB output port

	while (1)
	{
		PORTB = PORTB ^ (1<<PB4); //toggle PB4
		T1Delay ( );        //delay size unknown
	}
}


void T1Delay ( )
{
	TCNT1H = 0xC1;     //TEMP = 0xC1
	TCNT1L = 0x80;

	TCCR1A = 0x00;     //Normal mode
	TCCR1B = 0x01;     //Normal mode, no prescaler

	while ((TIFR&(0x1<<TOV1))==0);      //wait for TOV1 to roll over

	TCCR1B = 0;
	TIFR = 0x1<<TOV1;        //clear TOV1
}
```

Counter Programming in C

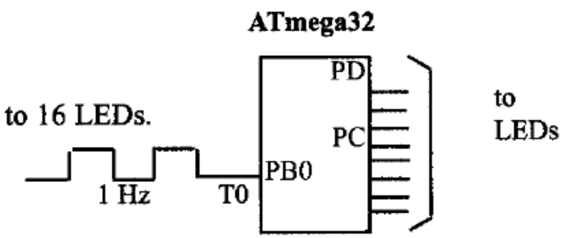
Timers can be used as counters if we provide pulses from outside the chip instead of using the frequency of the crystal oscillator as the clock source. By feeding pulses to the T0 (PB0) and T1 (PB1) pins, we use Timer0 and Timer1 as Counter 0 and Counter 1, respectively. Study the next Examples to see how Timers 0 and 1 are programmed as counters using C language.

Example

Assuming that a 1 Hz clock pulse is fed into pin T0, use the TOV0 flag to extend Timer0 to a 16-bit counter and display the counter on PORTC and PORTD.

```c
#include "avr/io.h"

int main ( )
{
      PORTB = 0x01;           //activate pull-up of PB0
      DDRC = 0xFF;            //PORTC as output
      DDRD = 0xFF;            //PORTD as output

      TCCR0 = 0x06;           //output clock source
      TCNT0 = 0x00;

      while (1)
      {
            do
            {
                  PORTC = TCNT0;
            }while((TIFR&(0x1<<TOV0))==0);//wait for TOV0 to roll over

            TIFR = 0x1<<TOV0;         //clear TOV0
            PORTD ++;                 //increment PORTD
      }
}
```

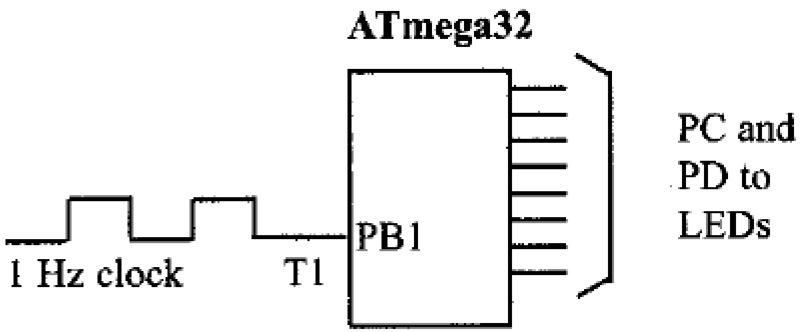
PORTC and PORTD are connected to 16 LEDs.
T0 (PB0) is connected to a
1-Hz external clock.



Example

Assume that a 1-Hz external clock is being fed into pin T1 (PB1). Write a C program for Counter1 in rising edge mode to count the pulses and display the TCNT1H and TCNT1L registers on PORTD and PORTC, respectively.

```
#include "avr/io.h"

int main ( )
{
	PORTB = 0x01;		//activate pull-up of PB0
	DDRC = 0xFF;		//PORTC as output
	DDRD = 0xFF;		//PORTD as output

	TCCR1A = 0x00;		//output clock source
	TCCR1B = 0x07;		//output clock source

	TCNT1H = 0x00;		//set count to 0
	TCNT1L = 0x00;		//set count to 0

	while (1)		//repeat forever
	{
		do
		{
			PORTC = TCNT1L;
			PORTD = TCNT1H;		//place value on pins
		}while((TIFR&(0x1<<TOV1))==0);//wait for TOV1

		TIFR = 0x1<<TOV1;		//clear TOV1
	}
}
```



پایان

موفق و پیروز باشید