AVR Microcontroller

Microprocessor Course

Chapter 9

AVR TIMER PROGRAMMING IN

ASSEMBLY AND C

Azar1401 (ver 1.3)

AVR TIMER PROGRAMMING IN ASSEMBLY

Many applications need to count an event or generate time delays. So, there are counter registers in microcontrollers for this purpose.

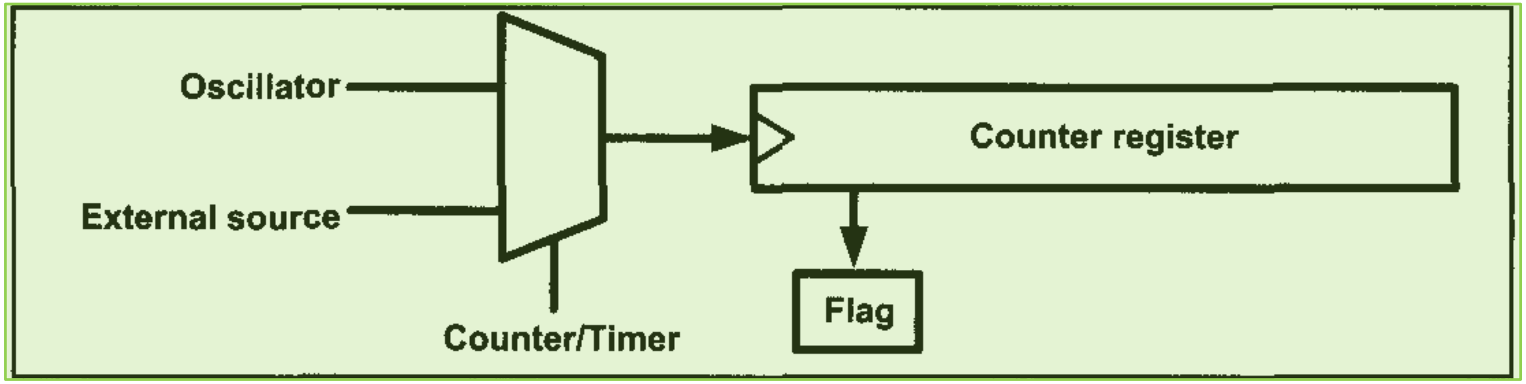


Figure 9-1. A General View of Counters and Timers in Microcontrollers

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.

AVR TIMER PROGRAMMING IN ASSEMBLY

HOW TO GENERATE DELAYS

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.

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.

The AVR has one to six timers depending on the family member. They are referred to as Timers 0, 1, 2, 3, 4 and 5. They can be used as timers to generate a time delay or as counters to count events happening outside the microcontroller.

Counter/Timer

AVR TIMER PROGRAMMING IN ASSEMBLY

In the AVR some of the timers/counters are 8-bit and some are 16-bit.

In ATmega32, there are three timers: Timer0, Timer1, and Timer2.

Timer0 and Timer2 are 8-bit, while Timer1 is 16-bit.

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Internal Clock

Timer: If we use the internal clock source, then the frequency of the crystal oscillator is fed into the timer. Therefore, it is used for time delay generation and consequently is called a timer.

External Clock

Counter: By choosing the external clock option, we feed pulses through one of the AVR's pins. This is called a counter.

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Basic registers of timers

For each of the timers, there is a TCNTn (timer/counter) register. The TCNTn register is a counter. Upon reset, the TCNTn contains zero.

Each timer has a TOVn (Timer Overflow) flag, as well. When a timer overflows, its TOVn flag will be set.

Each timer also has the TCCRn (timer/counter control register) register - for setting modes of operation.

Each timer also has an OCRn (Output Compare Register) register. The content of the OCRn is compared with the content of the TCNTn. When they are equal the OCFn (Output Compare Flag) flag will be set.

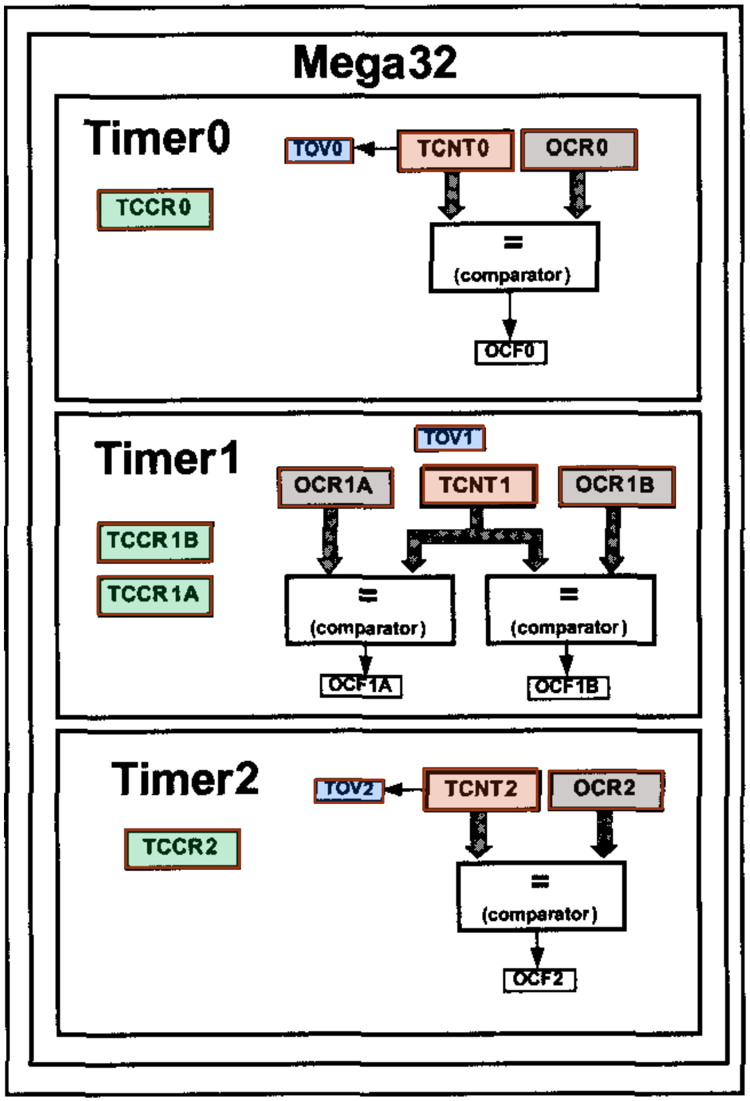


Figure 9-2. Timers in ATmega32

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

The timer registers are located in the I/O register memory.

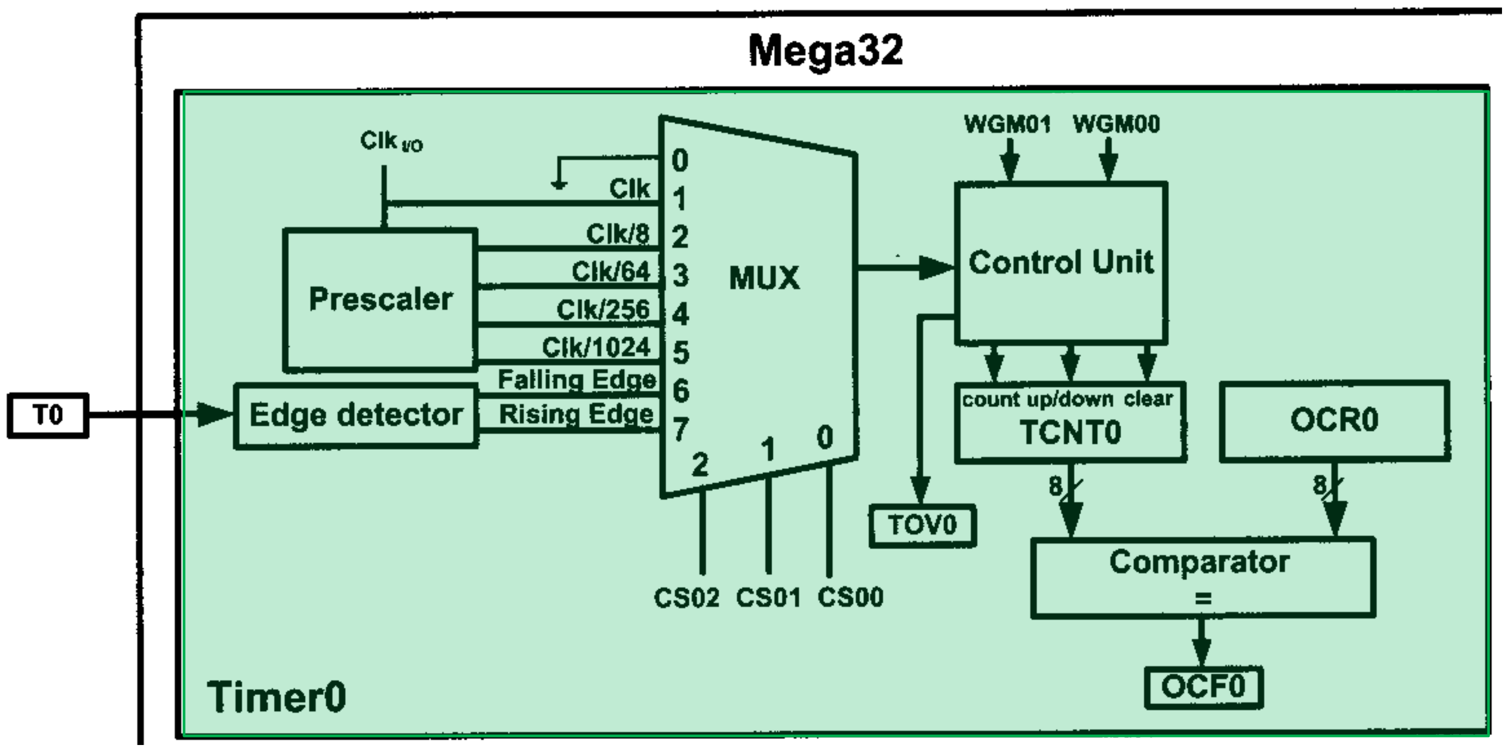
Therefore, you can read or write from timer registers using IN and OUT instructions, like the other I/O registers. For example, the following instructions load TCNT0 with 25:

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


mashhoun@iust.ac.ir

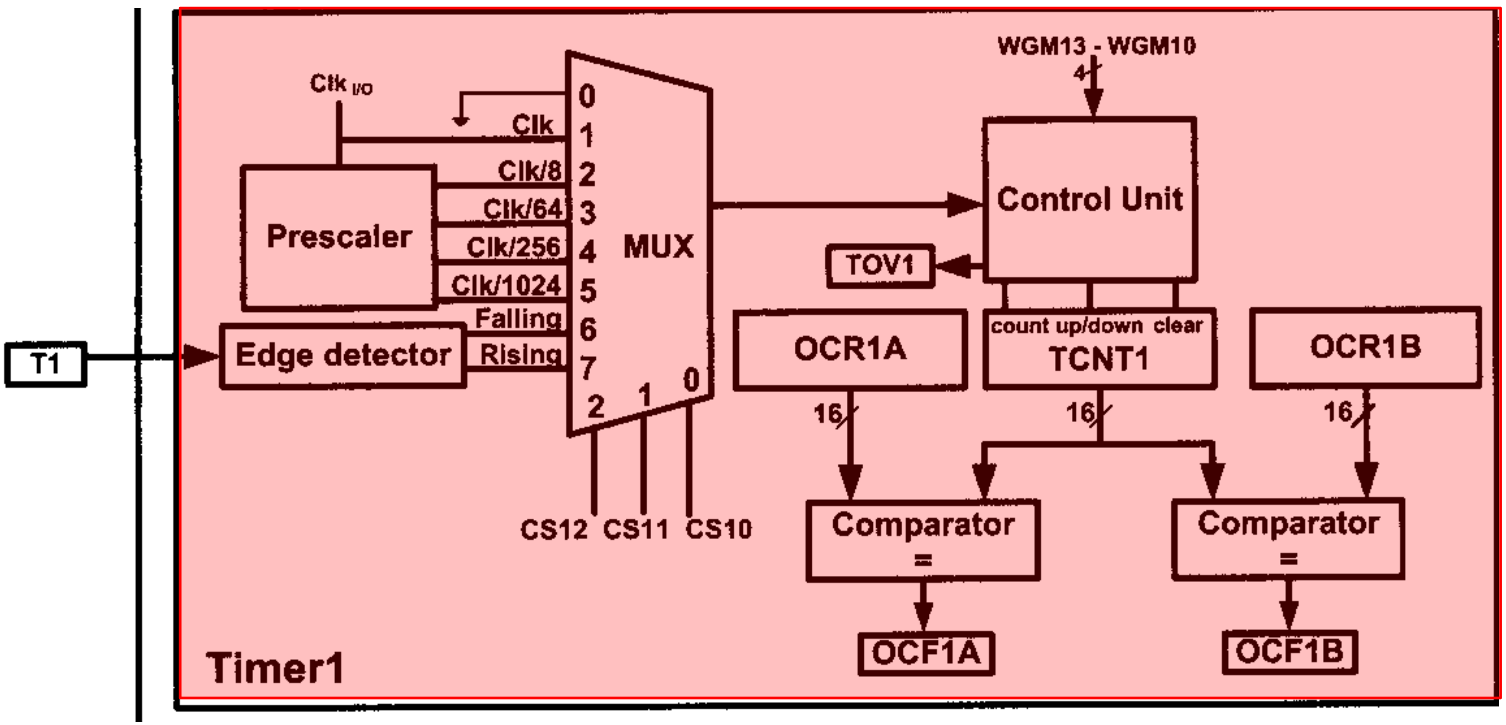
AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

The internal structure of the ATmega32 timer0 is shown in followingFigure.



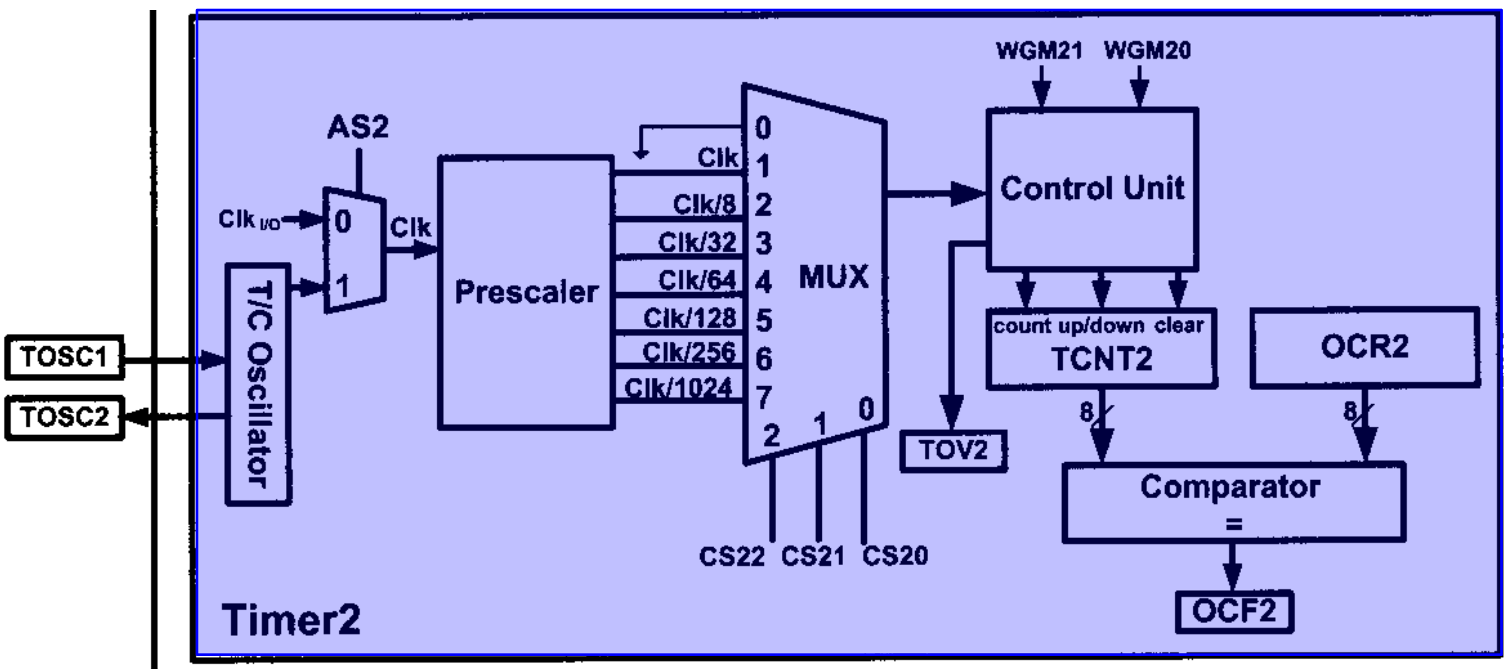
AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

The internal structure of the ATmega32 timer1 is shown in Figure 9-3.



AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

The internal structure of the ATmega32 timer2 is shown in Figure 9-3.



AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

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 (see Chapter 15).

mashhoun@iust.ac.ir

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

TCCR0 (Timer/Counter Control Register) register

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.

Figure 9-5. TCCR0 (Timer/Counter Control Register) Register

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Table 40 shows the COM01:0 bit functionality when the WGM01:0 bits are set to fast PWM mode.

Table 40. Compare Output Mode, Fast PWM Mode[1]

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at BOTTOM, (nin-inverting mode)
1	1	Set OC0 on compare match, clear OC0 at BOTTOM, (inverting mode)

Table 41 shows the COM01:0 bit functionality when the WGM01:0 bits are set to phase correct PWM mode.

Table 41. Compare Output Mode, Phase Correct PWM Mode[1]

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

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

Figure 9-6. TIFR (Timer/Counter Interrupt Flag Register)

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

TOV0 (Timer0 Overflow)

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.

The strange thing about this flag is that in order to clear it we need to write 1 to it.

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


mashhoun@iust.ac.ir
Iran Univ of Science & Tech

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

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.

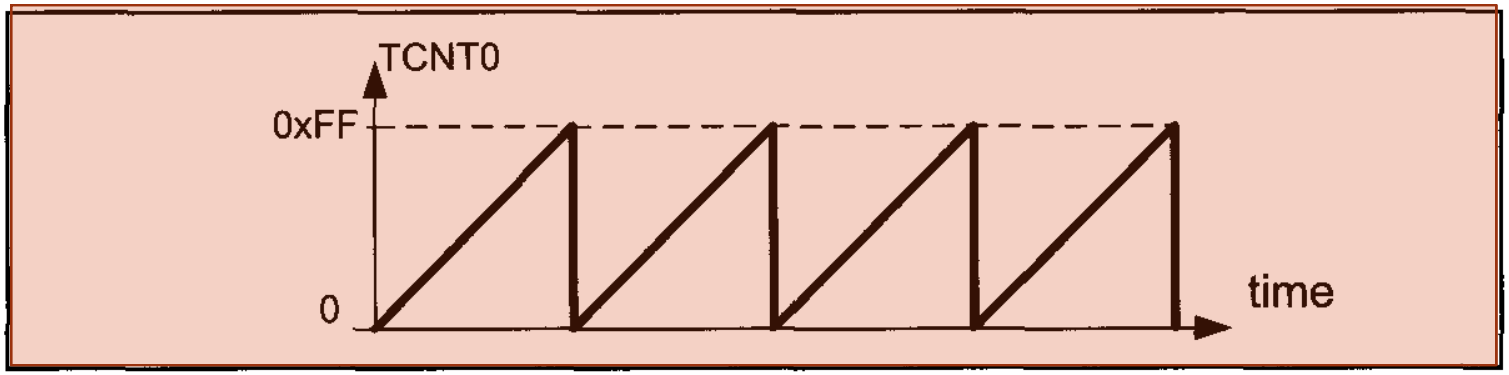


Figure 9-7. Timer/Counter 0 Normal Mode

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Steps to program Timer0 in Normal mode

To generate a time delay using Timer0 in Normal mode, the following steps are taken:

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

mashhoun@iust.ac.ir Iran Univ of Science & Tech

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-3

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"
.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
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

```
;----------------------------Time0 delay
DELAY:  LDI     R20,0xF2        ;R20 = 0xF2
        OUT     TCNT0,R20       ;load timer
        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 Timer()
        LDI     R20,(1<<TOV0)
        OUT     TIFR,R20        ;clear TOV0 flag by writing a 1 to TIER
        RET
```

Solution:

In the above program notice the following steps:

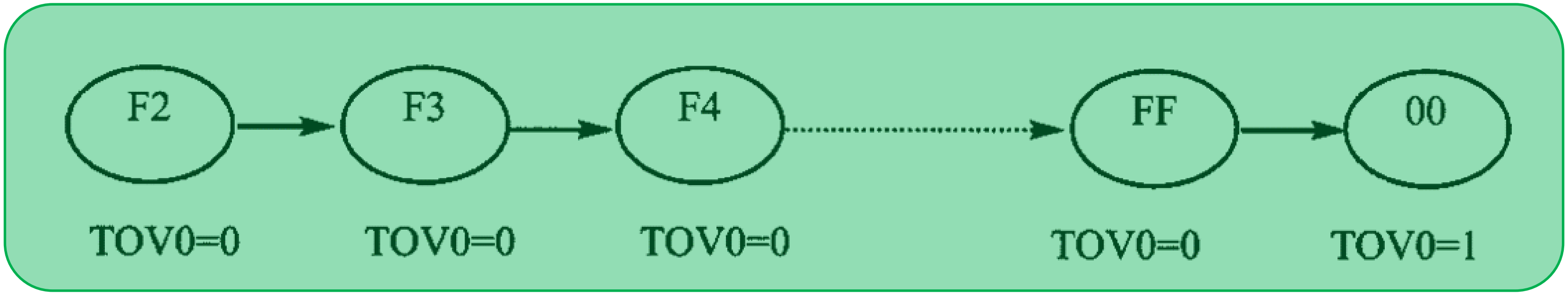
1. 0xF2 is loaded into TCNT0.
2. TCCR0 is loaded and Timer0 is started.

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Solution:

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.



mashhoun@iust.ac.ir Iran Univ of Science & Tech

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-4

In Example 9-3, calculate the amount of time delay generated by the timer. Assume that XTAL = 8 MHz.

Solution:

We have 8 MHz as the timer frequency. As a result, each clock has a period of

$$T = 1/8 \text{ MHz} = 0.125 \text{ µs}.$$

In other words, Timer0 counts up each 0.125 its resulting in delay = number of counts x 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 x 0.125 = 1.75 µs for half the pulse.

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-5

In Example 9-3, calculate the frequency of the square wave generated on pin PORTB.5. Assume that XTAL — 8 MHz.

Solution:

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
        SERS    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
```

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

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

(a) in hex	**(b) in decimal**
(FF − XX + 1) × 0.125 µs where XX is the TCNT0, initial value. Notice that XX value is in hex.	Convert XX value of the TCNT0 register to decimal to get a NNN decimal number, then (256 − NNN) × 0.125 µs

Figure 9-8. Timer Delay Calculation for XTAL = 8 MHz with No Prescaler

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-6

Find the delay generated by Timer0 in the following code, using both of the methods of Figure 9-8. Do not include the overhead due to instructions. (XTAL = 8 MHz)

```
.INCLUDE "M32DEF.INC"
        INITSTACK                   ;add its definition from Example 9-3
        LDI     R16,0x20
        SBI     DDRB,5              ;PB5 as an output
        LDI     R17,0
        OUT     PORTB,R17
BEGIN:  RCALL   DELAY
        EOR     R17,R16             ;toggle D5 of R17
        OUT     PORTS, R17          ;toggle PB5
        RJMP    BEGIN

DELAY:  LDI     R20,0x003E
        OUT     TCNT0,R20           ;load timer°
        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 Timer°
        LDI     R20,(1<<TOV0)       ;R20 = 0x01
        OUT     TIFR,R20            ;clear TOV0 flag
        RET
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Solution:

(a) (FF - 3E + 1) = 0xC2 = 194 in decimal and 194 x 0.125 µs = 24.25 µs.

(b) Because TCNT0 = 0x3E = 62 (in decimal) we have 256 - 62 =194. This means that the timer counts from 0x3E to 0xFF. This plus rolling over to 0 goes through a total of 194 clock cycles, where each clock is 0.125 µs in duration. Therefore, we have 194 x 0.125 µs = 24.25 µs as the width of the pulse.

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Finding values to be loaded into the 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. 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. 5. Set TCNT0 = xx.

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-7

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

Solution:

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 us = 50 clocks. 256 - 50 = 206 = 0xCE. Therefore, we have TCNT0 = 0xCE.

mashhoun@iust.ac.ir Iran Univ of Science & Tech

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

```
.INCLUDE "M32DEF.INC"
        INITSTACK                       ;add its definition from Example 9-3
        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
        LDI     R20,0x01
        OUT     TCCR0,R20
AGAIN:  IN      R20,TIFR
        SBRS    R20,TOV0
        RJMP    AGAIN
        LDI     R20,0x00
        OUT     TCCR0,R20
        LDI     R20,(1<<TOV0)
        OUT     TIFR,R20                ;clear
        RET
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-8

Assuming that XTAL = 8 MHz, modify the program in Example 9-7 to generate a square wave of 16 kHz frequency on pin PORTB.3.

Solution:

Look at the following steps.

(a) T = 1 / F = 1 / 16 kHz = 62.5µs the period of the square wave.

(b) 1/2 of it for the high and low portions of the pulse is 31.25 µs.

(c) 31.25 µs / 0.125 µs = 250 and 256 - 250 = 6, which in hex is 0x06.

(d) TCNT0 = 0x06.

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Using the Windows calculator to find TCNT0

The calculator in Windows is a handy tool to find the TCNT0 value. Assume that we would like to find the TCNT0 value for a time delay that uses 135 clocks of 0.125 μs. The following steps show the calculation:

1. Bring up the scientific calculator in MS Windows and select decimal.
2. Enter 135.
3. Select hex. This converts 135 to hex, which is 0x87.
4. Select +/- *to give -135 decimal (0x79).*
5. The lowest two digits (79) of this hex value are for TCNT0. We ignore all the Fs on the left because our number is 8-bit data.

mashhoun@iust.ac.ir Iran Univ of Science & Tech

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Prescaler and generating a large time delay

As we have seen in the examples so far, the size of the time delay depends on two factors,

(a) the crystal frequency, and

(b) the timer's 8-bit register.

Both of these factors are beyond the control of the AVR programmer.

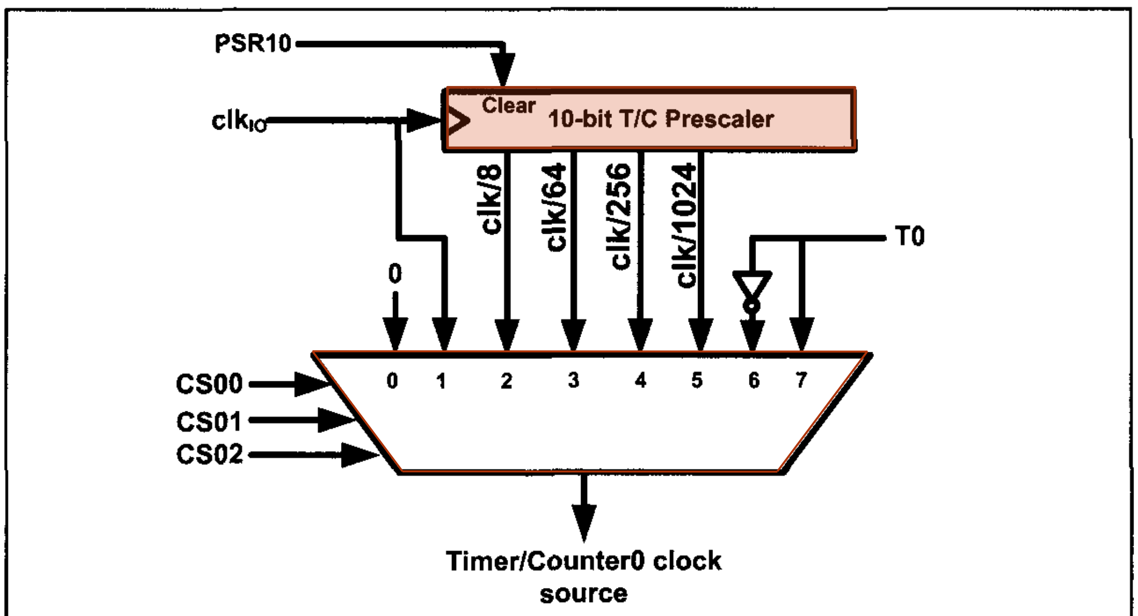


Figure 9-9. Timer/Counter 0 Prescaler

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-9

Modify TCNT0 in Example 9-7 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.

Solution:

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

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) x 0.125 µs = 32 µs . That gives us the smallest frequency of 1 / (2 x 32 µs) - 1 / (64 µs) = 15.625 kHz.

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

```
.INCLUDE "M32DEF.INC"
        INITSTACK               ;add its definition from Example 9-3
        LDI     R16,0x08
        SBI     DDRB,3          ;PB3 as an output
        LDI     R17,0
        OUT     PORTB,RI7
BEGIN:  RCALL   DELAY
        EOR     R17,R16         ;toggle D3 of RI7
        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 cik, no prescaler
AGAIN:  IN      R20,TIFR        ;read TIFR
        SERS    R20,TOV0        ;if TOVO 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
```

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

We saw in Example 9-9 that the largest time delay is achieved by making TCNT0 zero. What if that is not enough?

We can use the prescaler option in the TCCR0 register to increase the delay by reducing the period. The prescaler option of TCCR allows us to divide the instruction clock by a factor of 8 to 1024 as was shown in Figure 9-5. The prescaler of Timer/Counter0 is shown in Figure 9-9.

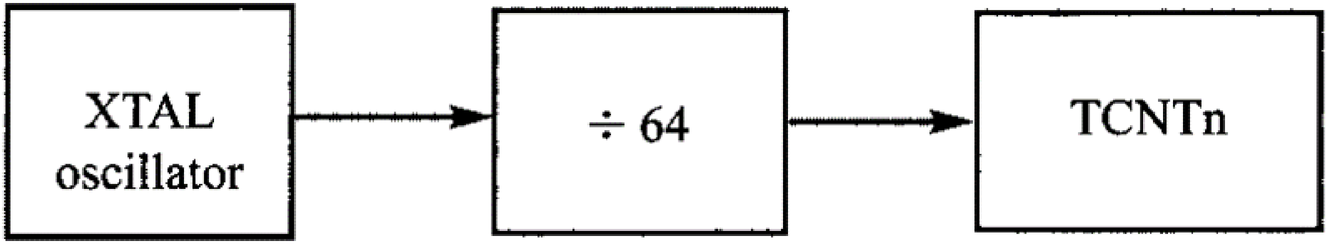
AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-10

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

Solution:



(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

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-11

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.

Solution:

From Figure 9-5 we have TCCR0 = 0000 0011; XTAL clock source, prescaler of 64.

From Figure 9-5 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

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-12

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

```
.INCLUDE "M32DEF.INC"
        INITSTACK                ;add its definition from Example 9-3
        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
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

```
;---------------------------- 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     TCCRO,R20       ;stop Timer0
        LDI     R20,1<<TOV0
        OUT     TIFR,R20        ;clear TOV0 flag RET
        ret
```

Solution:

TCNT0 = 0x10 = 16 in decimal and 256 - 16 = 240.

Now $240 \times 64 \times 0.125\ \mu s = 1920\ \mu s$,

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-13

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.

Solution:

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

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.

mashhoun@iust.ac.ir Iran Univ of Science & Tech

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-14

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.

Solution:

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 gs) / 256 = 125 and 256 -125 = 131 in decimal, and in hex it is 0x83.

(d) TCNT0 = 83 (hex)

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

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

        INITSTACK
        LDI     R16,0x08
        SBI     DDRB,3
        LDI     R17,0
BEGIN:  OUT     PORTB,R17
        CALL    DELAY
        EOR     R17,R16
        RJMP    BEGIN
```

```
;---------------------------------------
DELAY:  LDI     R20,0x83
        OUT     TCNT0,R20
        LDI     R20,0x04
        OUT     TCCR0,R20
AGAIN:  IN      R20,TIFR
        SBRS    R20,TOV0
        RJMP    AGAIN
        LDI     R20,0x0
        OUT     TCCR0,R20
        LDI     R20,1<<TOV0
        OUT     TIFR,R20
        RET
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Assemblers and negative values

Because the timer is in 8-bit mode, we can let the assembler calculate the value for TCNT0. For example, in the "**`LDI R20,-100`**" instruction, the assembler will calculate the -100 = 9C and make R20 = 9C in hex. This makes our job easier. See Examples 9-15 and 9-16.

Example 9-15

Find the value (in hex) loaded into TCNT0 for each of the following cases.

```
(a) LDI R20, -200        (b) LDI R17, -60        (C) LDI R25, -12
    OUT TCNT0,R20            OUT TCNT0,R17            OUT TCNT0,R25
```

Solution:

You can use the Windows scientific calculator to verify the results provided by the assembler. In the Windows calculator, select decimal and enter 200. Then select hex, then +1- to get the negative value. The following is what we get.

Decimal	**2's complement (TCNT0 value)**
-200	0x38
-60	0xC4
-12	0xF4

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-16

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
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

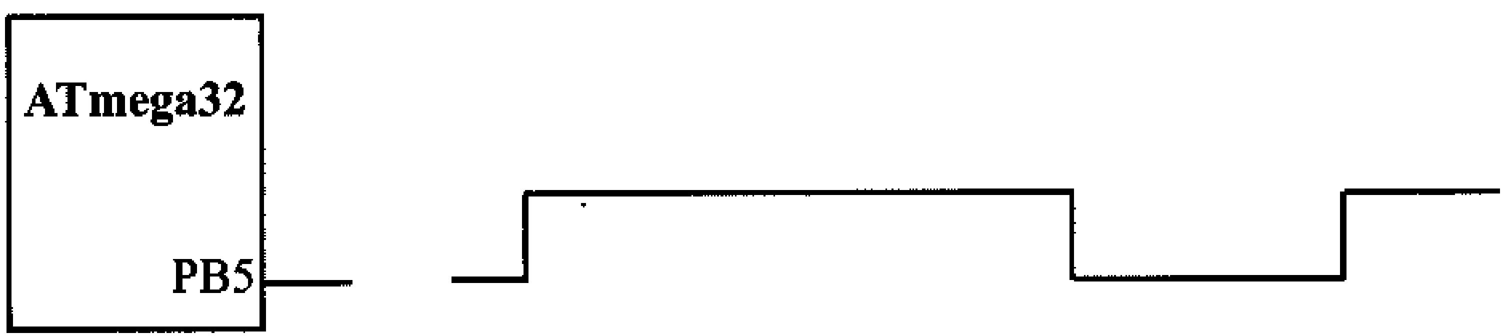
Solution:

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.



AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

CTC mode programming (Clear Timer0 on compare match)

The OCR0 register is used with CTC mode. As with the Normal mode, in the CTC mode, the timer is incremented with a clock. But it counts up until the content of the TCNT0 register becomes equal to the content of OCR0 (compare match occurs); then, the timer will be cleared and the OCF0 flag will be set when the next clock occurs. The OCF0 flag is located in the TIFR register.

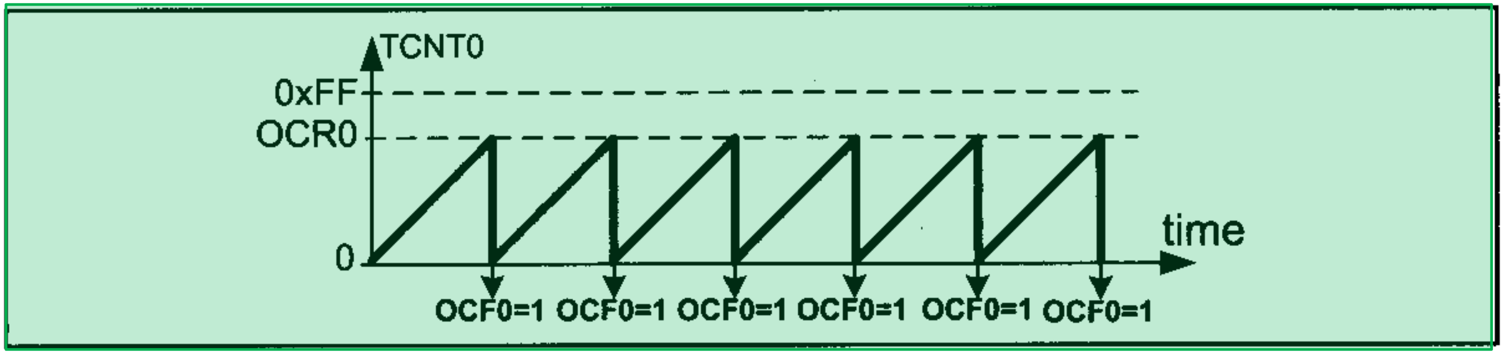


Figure 9-10. Timer/Counter 0 CTC Mode

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-17

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               ;add its definition from Example 9-3
        LDI     R16,0x20
        SBI     DDRB,5          ;PB3 as an output
        LDI     R17,0
BEGIN:  OUT     PORTB,R17       ;PORTB = R17
        RCALL   DELAY
        EOR     R17,R16         ;toggle D3 of R17
        RJMP    BEGIN
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

```
.INCLUDE "M32DEF.INC"
        INITSTACK               ;add its definition from Example 9-3
        LDI     R16,0x20
        SBI     DDRB,5          ;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 OCRO
        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
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Solution:

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.

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

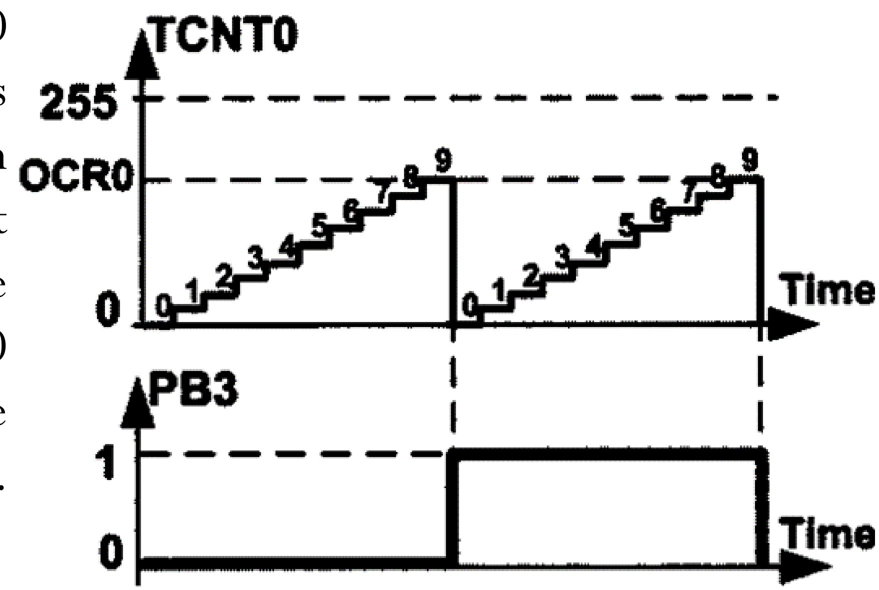
Example 9-18

Find the delay generated by Timer0 in Example 9-17. Do not include the overhead due to instructions. (XTAL = 8 MHz)

Solution:

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

$10 \times 0.125\ \mu s = 1.25\ \mu s.$



AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

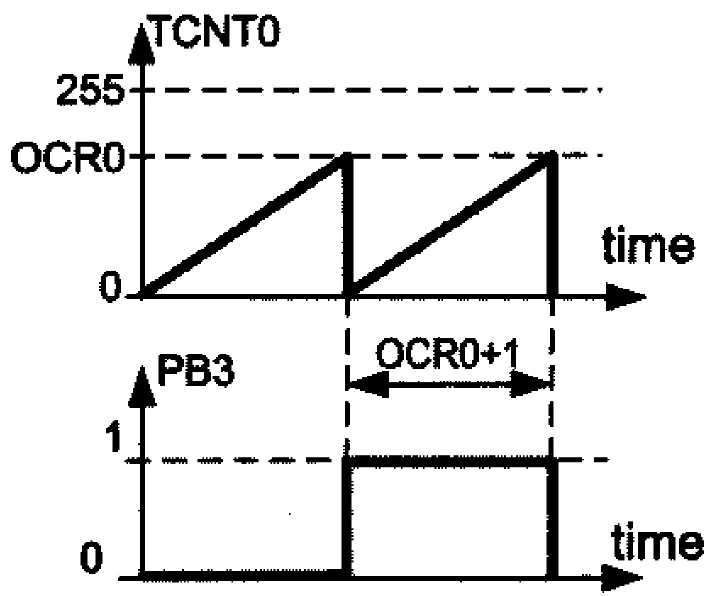
Example 9-19

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

```
.INCLUDE "M32DEF.INCU
        INITSTACK                       ;add its definition from Example 9-3
        LDI     R16,0x08
        SBI     DDRB,3                  ;PB3 as an output
        LDI     R17,0
        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
        RJMP    AGAIN
        LDI     R20,0x0
        OUT     TCCR0,R20               ;stop Timer0
        LDI     R20,1<<OCF0
        OUT     TIFR,R20                ;clear OCF0 flag
        EOR     R17,R16                 ;toggle D3 of R17
        OUT     PORTB,R17               ;toggle PB3
        RJMP    BEGIN
```

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

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



AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-20

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

Solution:

Due to prescaler = 1024 each timer clock lasts 1024 x 0.125 µs = 1280. 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
```

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

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.

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

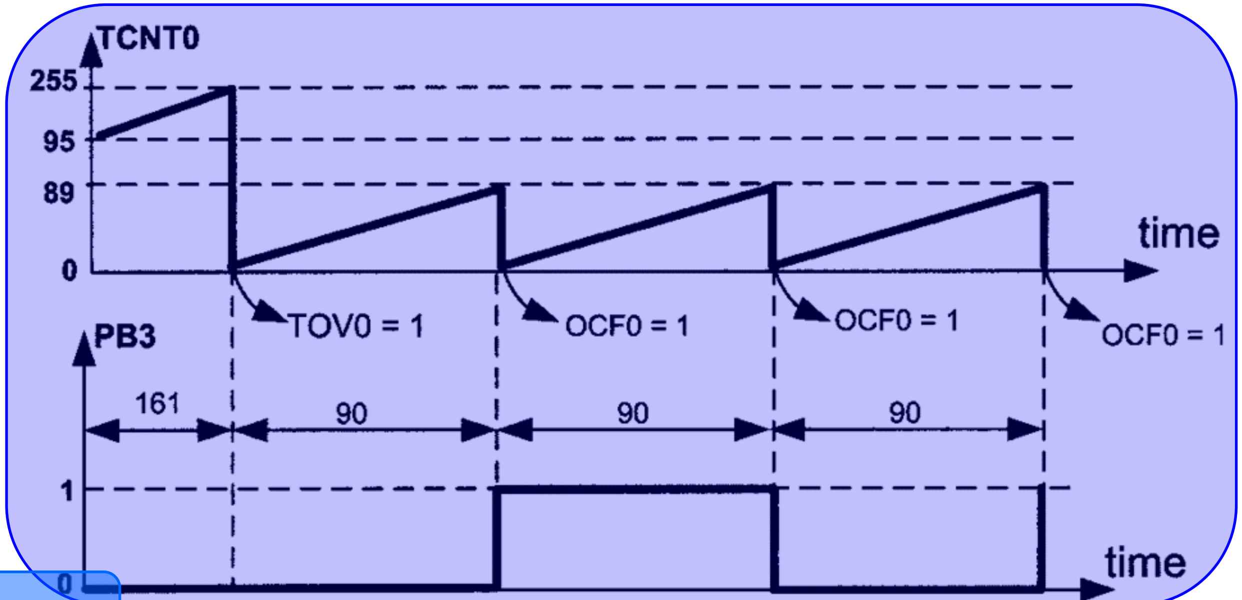
Example 9-22

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.INCU
        SBI     DDRB,3          ;PB3 as an output
        CBI     PORTB,3         ;PB3 = 0
        LDI     R20,89
        OUT     OCR0,R20
        LDI     R20,95
        OUT     TCNT0,R20
BEG1N:  LDI     R20,0x09
        OUT     TCCR0,R20       ;Timer0, CTC mode, prescaler=1
again:  IN      R20,TIFR        ;read TIFR
        SBRS    R20,OCF0
        RJMP    AGAIN
        LDI     R20,0x0
        OUT     TCCR0,R20       ;stop Timer0
        LDI     R20,1<<OCF0
        OUT     TIFR,R20        ;clear OCF0 flag
        EOR     R17,R16         ;toggle D3 of R17
        OUT     PORTB,R17       ;toggle PB3
        RJMP    BEGIN
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2



Solution:

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 × 0.125 µs = 31.375 µs. The next compare matches occur after 90 clocks, which means after 90 × 0.125 µs = 11.25 µs.

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Timer2 programming

Timer2 is an 8-bit timer. Therefore it works the same way as Time0. 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 TOSCl and TOSC2 pins of AVR and set the AS2 bit. See Figure 9-12.

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. Compare Figure 9-11 with Figure 9-5 and examine Examples 9-23 through 9-25.

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Timer2 programming

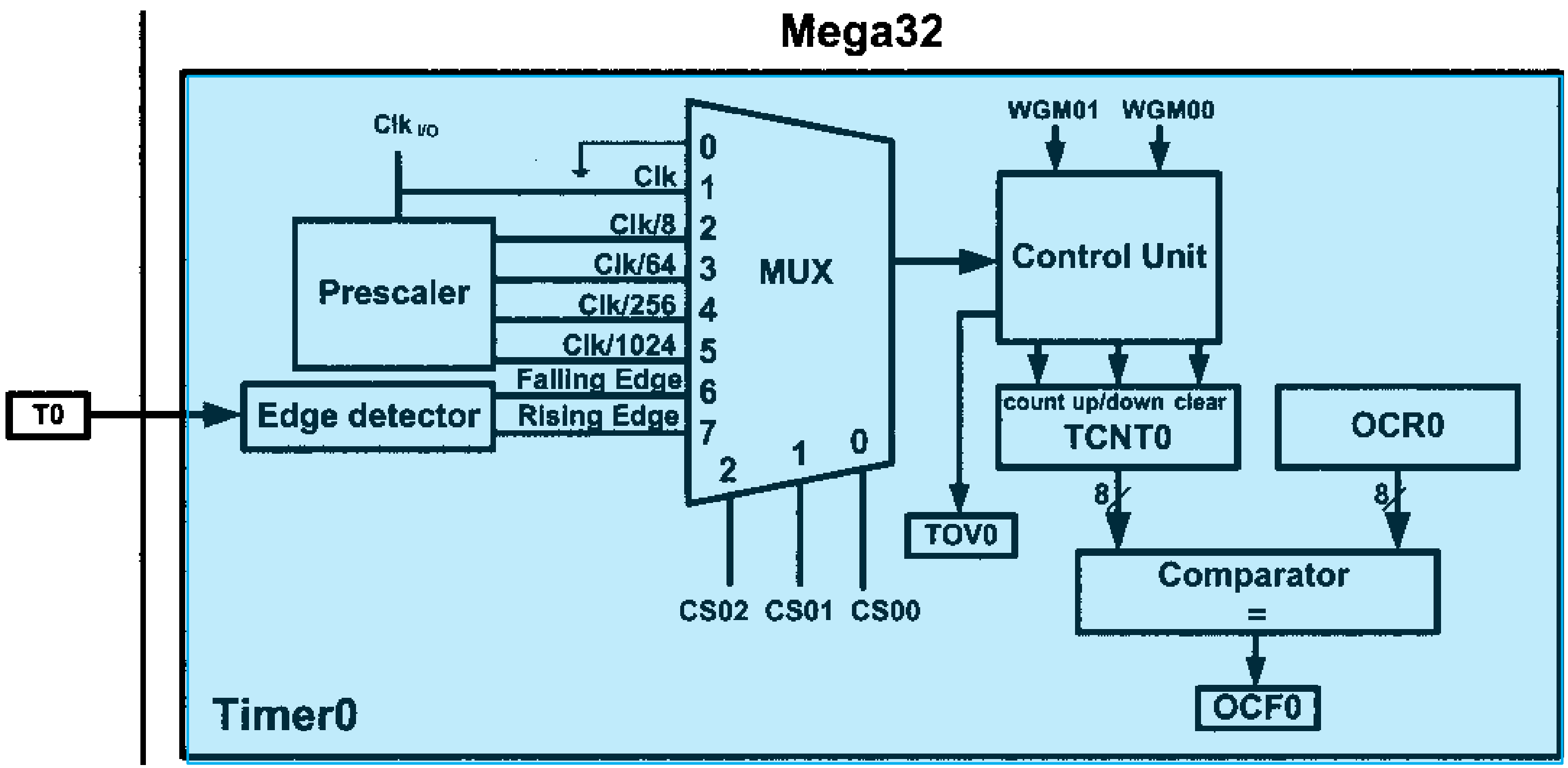


Figure 9-13 Atmega32 Timer

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Timer2 programming

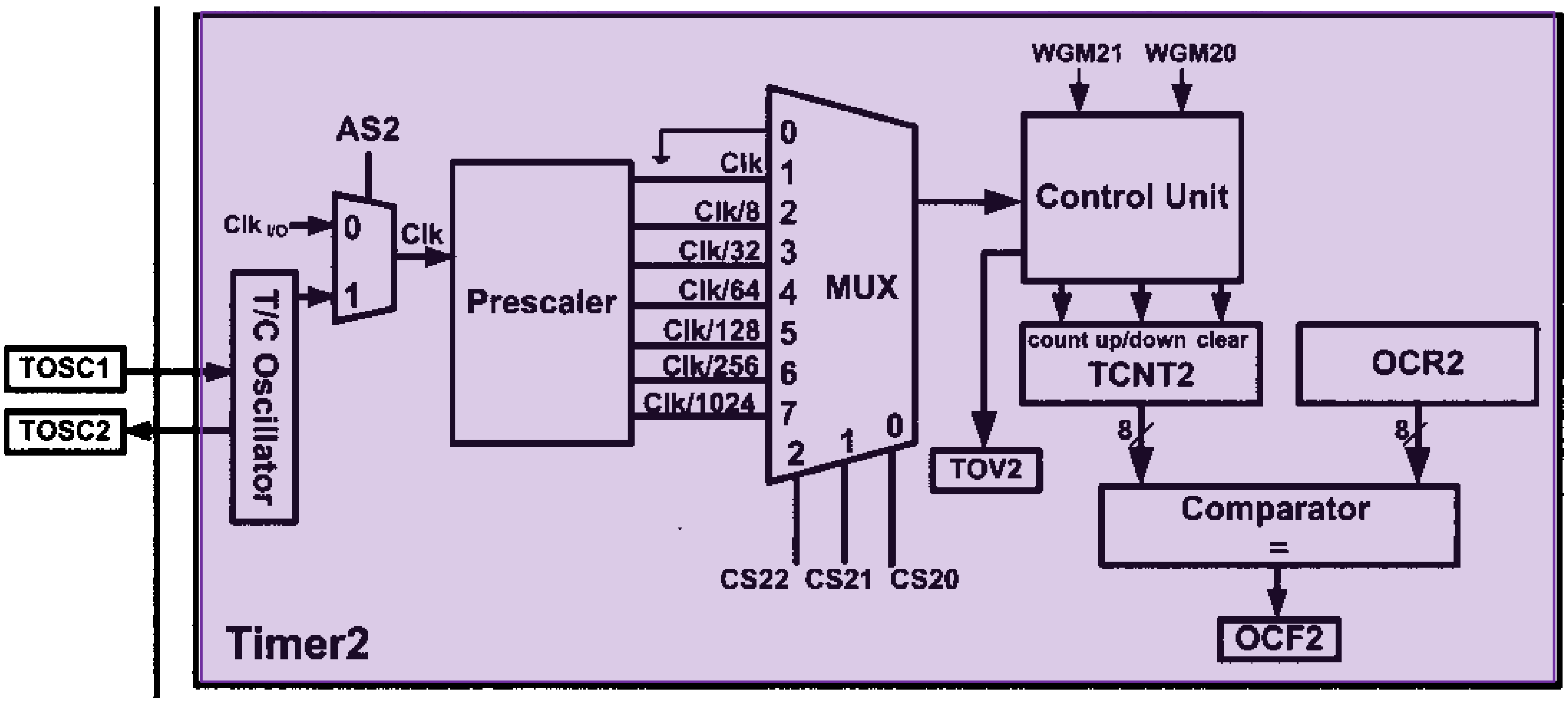


Figure 9-13 Atmega32 Timer

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

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

Figure 9-11. TCCR2 (Timer/Counter Control Register) Register

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-23

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.

Solution:

From Figure 9-11 we have TCCR2 = 0000 0100; XTAL clock source, prescaler of 64.

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

Compare the answer with Example 9-11.

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-24

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

Solution:

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
```

Compare the above program with the DELAY subroutine in Example 9-12.
There are two differences between the two programs:
1- The register names are different. TCNT2 instead of TCNT0
2- The values of TCCRn are different for the same prescaler.

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-25

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

Solution:

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	8ms/0.125µs=64k
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 = 41.1 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.

mashhoun@iust.ac.ir Iran Univ of Science & Tech

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-25 (Cont.)

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
```

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Timer1 programming

Timer1 has the prescaler options of 1:1, 1:8, 1:64, 1:256, and 1:1024.

There are two registers in Timer1: OCR1A and OCR1B.

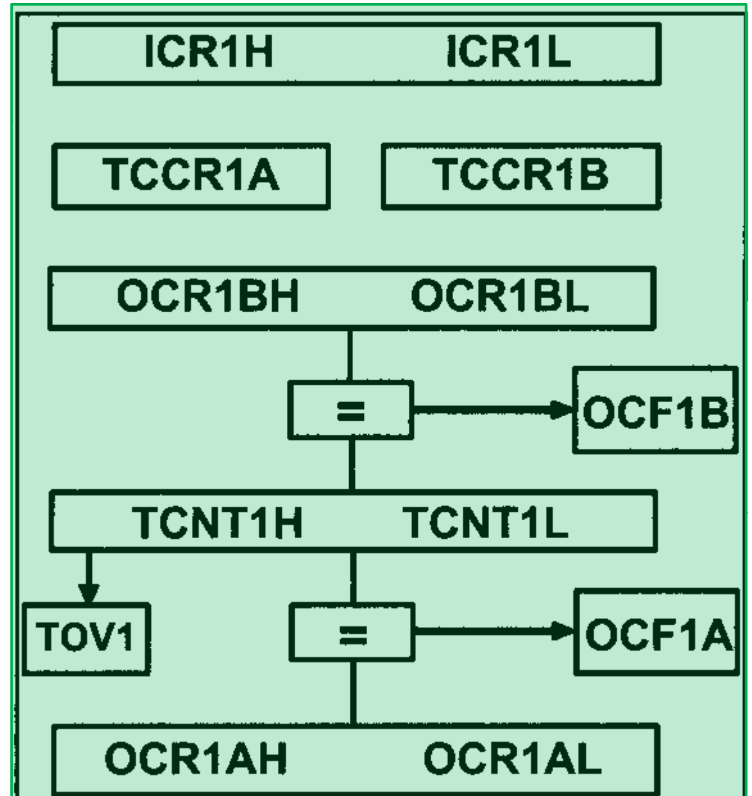


Figure 9-14. Simplified Diagram of Timer1

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Timer/Counter 16-bit

As Timer1 is a 16-bit timer, the OCR registers are 16-bit registers as well and they are made of two 8-bit registers. For example, OCR1A is made of OCR1AH (OCR1A high byte) and OCR1AL (OCR1A low byte).

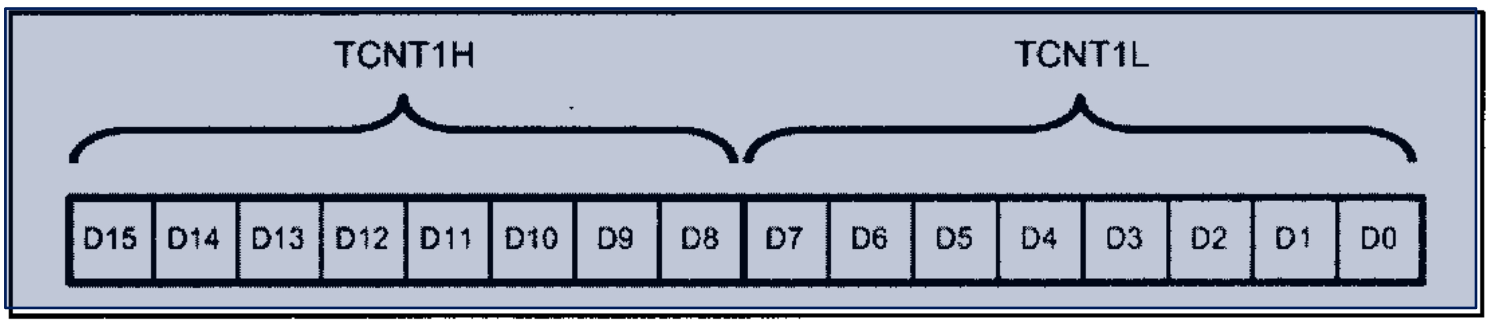


Figure 9-15. Timer1 High and Low Registers

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Input Capture Register

There is also an auxiliary register named ICR1, which is used in operations such as capturing. ICR1 is a 16-bit register made of ICR1H and ICR1L.

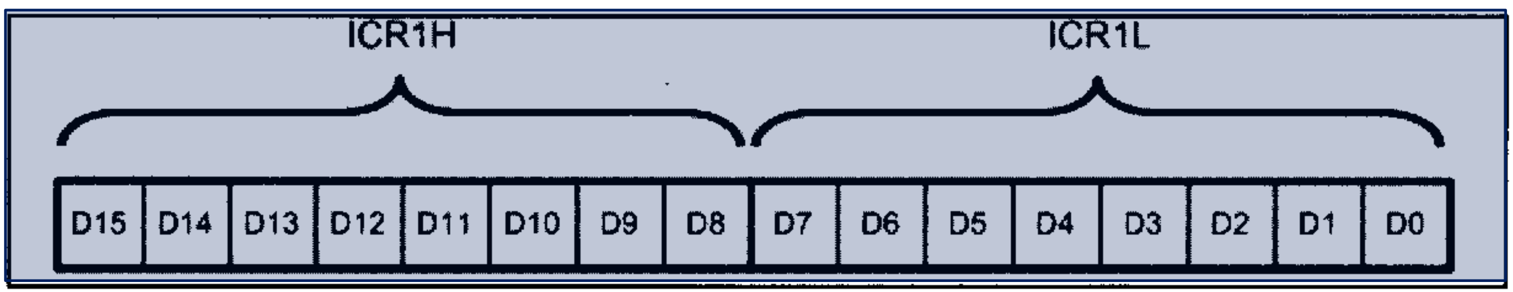


Figure 9-19. Input Capture Register (ICR) for Timer1

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

TCCR1A Control Register

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 (discussed in Section 9-3)
WGM11:10	D1 D0	Timer1 mode (discussed in Figure 9-18)

Figure 9-17. TCCR1A (Timer 1 Control) Register

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

TCCR1B Control Register – Figure 9-18

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

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

TCCR1B Control Register – Figure 9-18

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

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

TCCR1B Control Register – Figure 9-18

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.

Figure 9-18. TCCR1B (Timer 1 Control) Register

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Timer1 operation modes

Normal mode (WGM13:10 = 0000)

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

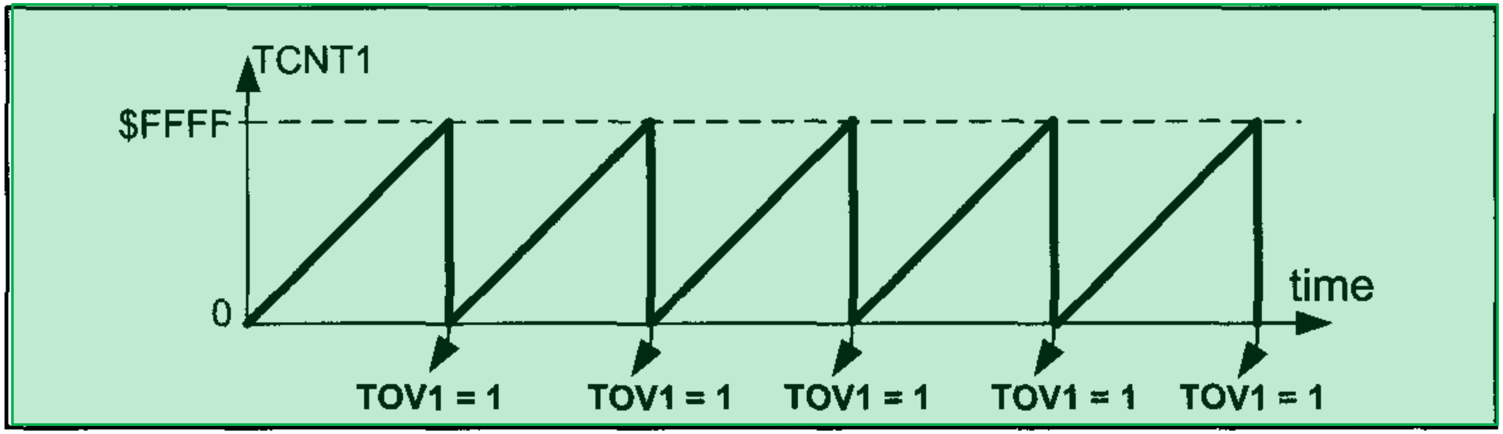


Figure 9-20. TOV in Normal and Fast PWM

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

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.

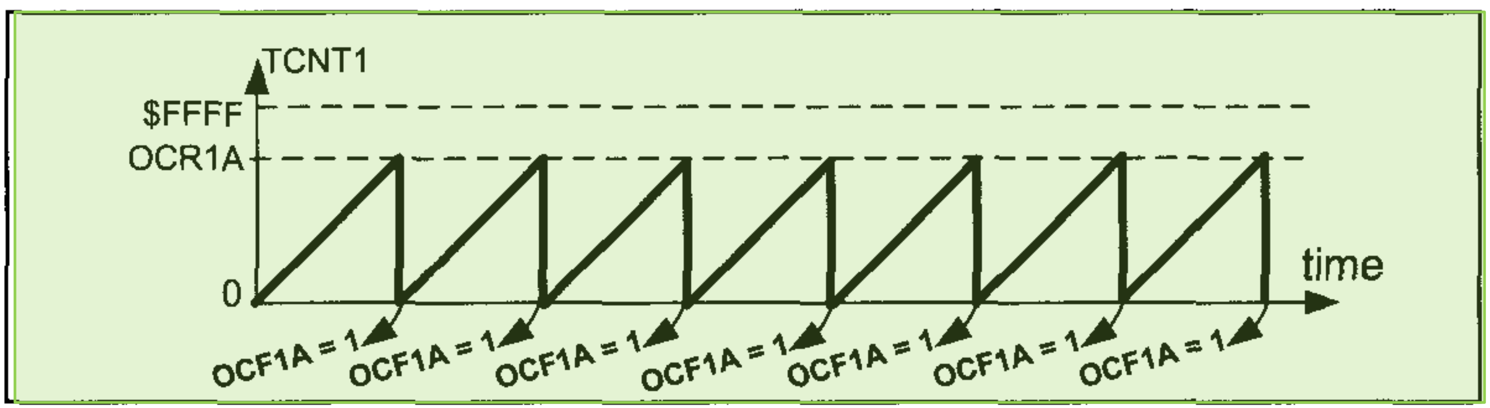


Figure 9-21. OCF1A in CTC Mode

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-26

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.

Solution:

TCCR1A = 0000 0000   WGM11 = 0,   WGM10 = 0

TCCRIB = 0000 0001   WGM13 = 0,   WGM12 = 0, oscillator clock source, no prescaler

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-27

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

```
.INCLUDE "M32DEF.INC"
        INITSTACK                       ;add its definition from Example 9-3
        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 RE
        RET
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Solution:

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 x 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:

```
LDI     R20,HIGH(65535-10000)
OUT     TCNT1H,R20
LDI     R20,LOW(65535-10000)
OUT     TCNT1L,R20
```

Or we can simply write it as follows:

```
LDI     R20,HIGH(-10000)
OUT     TCNT1H,R20
LDI     R20,LOW(-10000)
OUT     TCNT1L,R20
```

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-28

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.

Solution:

TCCR1A = 0000 0000 WGM11 = 0, WGM10 = 0

TCCR1B = 0000 1001 WGM13 = 0, WGMI2 = 1, oscillator clock source, no prescaler

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-29

Find the frequency of the square wave generated by the following program if XTAL = 8 MHz. 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         ;PBS = 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          ;OCRIA = 159 = Ox9F
        LDI     R20,0x0
        OUT     TCCR1A,R20          ;WGMII:10 = 00
        LDI     R20,0x09
        OUT     TCCR1B,R20          ;WGM13:12 = 01,CTC mode, prescaler = 1
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

```
AGAIN:  IN      R20,TIFR            ;read TIFR
        SBRS    R20,OCFIA           ;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
```

Solution:

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 x 0.125 µs = 20 *µs and frequency = 1 / (20 µs x 2) = 25 kHz.*

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

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.

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

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.

The AVR designers have resolved this issue with an 8-bit register called TEMP, which is used as a buffer.

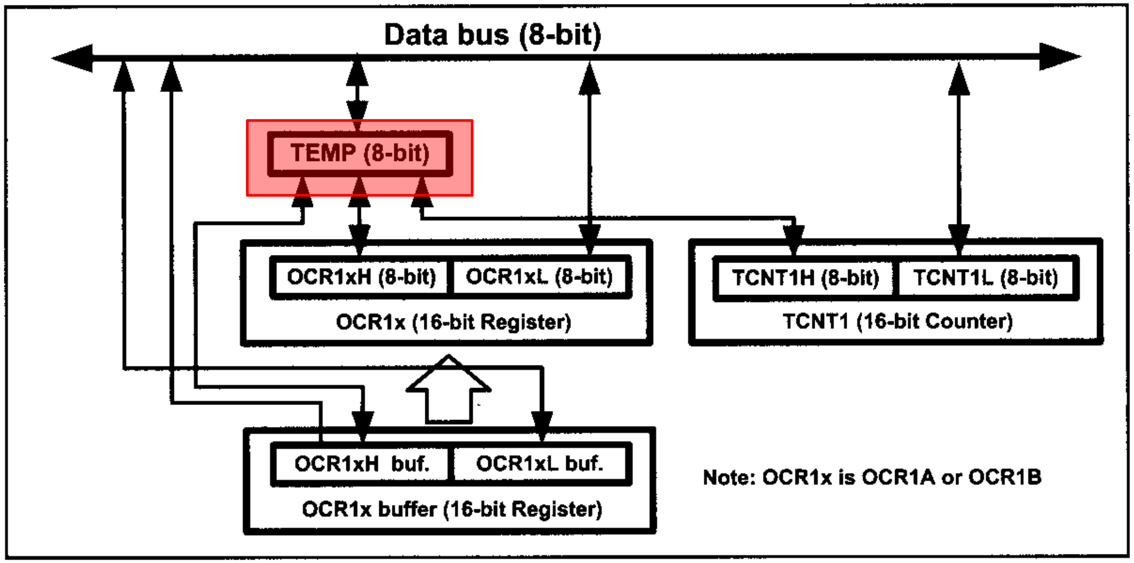


Figure 9-22. Accessing 16-bit Registers through TEMP

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

For example, consider the following program:

```
LDI     R16,0x15
OUT     TCNT1H,R16          ;store 0x1b in TEMP of Timer1
LDI     R16,0xFF
OUT     TCNT1L, R16         ;TCNT1L = R16, TCNT1H = TEMP
```

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.

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.

mashhoun@iust.ac.ir Iran Univ of Science & Tech

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

When we read the low byte of 16-bit registers, the content of the high byte will be copied to the TEMP register. So, the following program reads the content of TCNT1 :

```
IN      R20,TCNT1L          ;R20 = TCNT1L, TEMP = TCNT1H
IN      R21,TCNT1H          ;R21 = TEMP of Timer1
```

We must pay attention to the order of reading the high and low bytes of the 16-bit registers.

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-30

Assuming XTAL = 8 MHz, write a program that toggles PB5 once per millisecond.

Solution:

XTAL = 8 MHz means that each clock takes 0.125 µs. Now for 1ms delay, we need 1ms/0.125µs = 8000 clocks = 0x1F40 clocks. We initialize the timer so that after 8000 clocks the OCF1A flag is raised, and then we will toggle the PB5.

```
.INCLUDE "M32DEF.INC"
        LDI     R16,HIGH(RAMEND)
        OUT     SPH,R16
        LDI     R16,LOW(RAMEND)
        OUT     SPL,R16                     ;initialize the stack
        SBI     DDRB,5                      ;PB5 as an output
BEGIN:  SBI     PORTB,5                     ;PB5 = 1
        RCALL   DELAY_ims
        CBI     PORTE,5                     ;PB5 = 0
        RCALL   DELAY_1ms
        RJMP    BEGIN
```

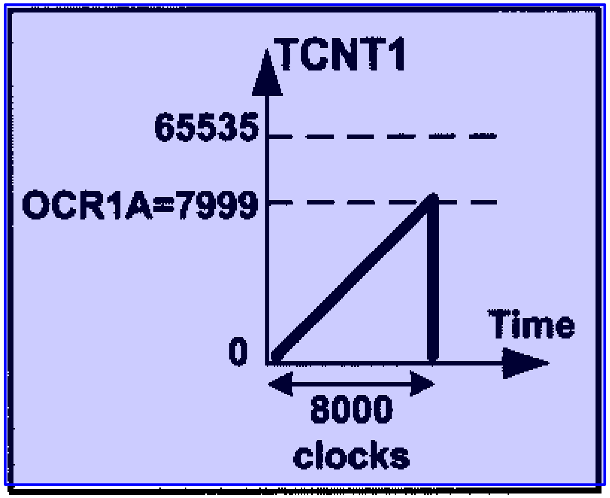
AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

```
;---------------------------------------- Timer1 Delay
DELAY_1ms:
        LDI     R20,0x00
        OUT     TCNT1H,R20              ;TEMP = 0
        OUT     TCNT1L,R20              ;TCNT1L = 0, TCNT1H = TEMP

        LDI     R20,,HIGH(8000-1)
        OUT     OCR1AH,R20              ;TEMP = 0x1F
        LDI     R20,LOW(8000-1)
        OUT     OCR1AL,R20              ;OCRIAL = 0x3F, OCR1AH = TEMP
        LDI     R20,0x0
        OUT     TCCR1A,R20              ;WGM11:10 = 00
        LDI     R20,0x09
        OUT     TCCR1B,R20              ;WGM13:12 = 01, CTC mode, CS = 1
AGAIN:  IN      R20,TIFR                ;read TIFR ;if OCF1A is set skip next instruction
        SBRS    R20,OCF1A
        RJMP    AGAIN
        LDI     R20,1<<OCF1A            ;clear OCF1A flag
        OUT     TIFR,R20
        LDI     R19,0
        OUT     TCCR1B,R19              ;stop timer
        OUT     TCCR1A,R19
        RET
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2



Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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	

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2

Example 9-31

Rewrite Example 9-30 using the TOV1 flag.

Solution:

To wait 1 ms we should load the TCNT1 register so that it rolls over after **8000 = 0x1F40** clocks. In Normal mode the top value is 0xFFFF = 65535. 65535 + 1 – 8000 = 57536 = 0xE0C0. Thus, we should load TCNT1 with 57536, or 0xE0C0 in hex, or we can simply use 65536 – 8000, as shown below:

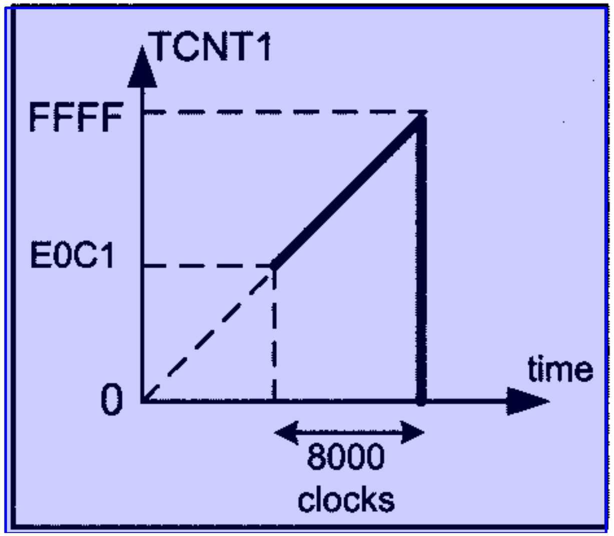
```
.INCLUDE "M32DEF.INC"
        LDI     R16,HIGH(RAMEND)        ;initialize stack pointer
        OUT     SPH,R16
        LDI     R16,LOW(RAMEND)
        OUT     SPL,R16

        SBI     DDRB,5                  ;PB5 as an output
BEGIN:  SBI     PORTB,S                 ;PB5 = 1
        RCALL   DELAY_1ms
        CBI     PORTB,5                 ;PB5 = 0
        RCALL   DELAY_1ms
        RJMP    BEGIN
```

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

```
;---------------------------------------- Timer1 DELAY
DELAY_1ms:
        LDI     R20,HIGH(65536-8000)        ;R20 = high byte of 57536
        OUT     TCNT1H,R20                  ;TEMP = 0xE0
        LDI     R20,LOW(65536-8000)         ;R20 = low byte of 57536
        OUT     TCNTIL,R20                  ;TCNT1L = 0xC1, TCNT1H = TEMP
        LDI     R20,0x0
        OUT     TCCR1A,R20                  ;WGM11:10 = 00
        LDI     R20,0x1
        OUT     TCCR1B,R20                  ;WGM13:12 = 00, Normal mode, CS = 1
AGAIN:  IN      R20,TIFR                    ;read TIFR
        SBRS    R20,TOV1                    ;if OCF1A is set skip next instruction
        RJMP    AGAIN
        LDI     R20,1<<TOV1
        OUT     TIFR,R20                    ;clear TOV1 flag
        LDI     R19,0
        OUT     TCCR1B,R19                  ;stop timer
        OUT     TCCR1A,R19
        RET
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.1 PROGRAMMING TIMERS 0,1 AND 2



Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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	

AVR TIMER PROGRAMMING IN ASSEMBLY
9.1 PROGRAMMING TIMERS 0,1 AND 2

Generating a large time delay using prescaler

Size of time delay depends on
(a) Crystal frequency
(b) The timers's 16-bit register

Both of these factors are beyond the control of the AVR programmer. We can use the prescaler option in the TCCR1B register to increase the delay by reducing the period.

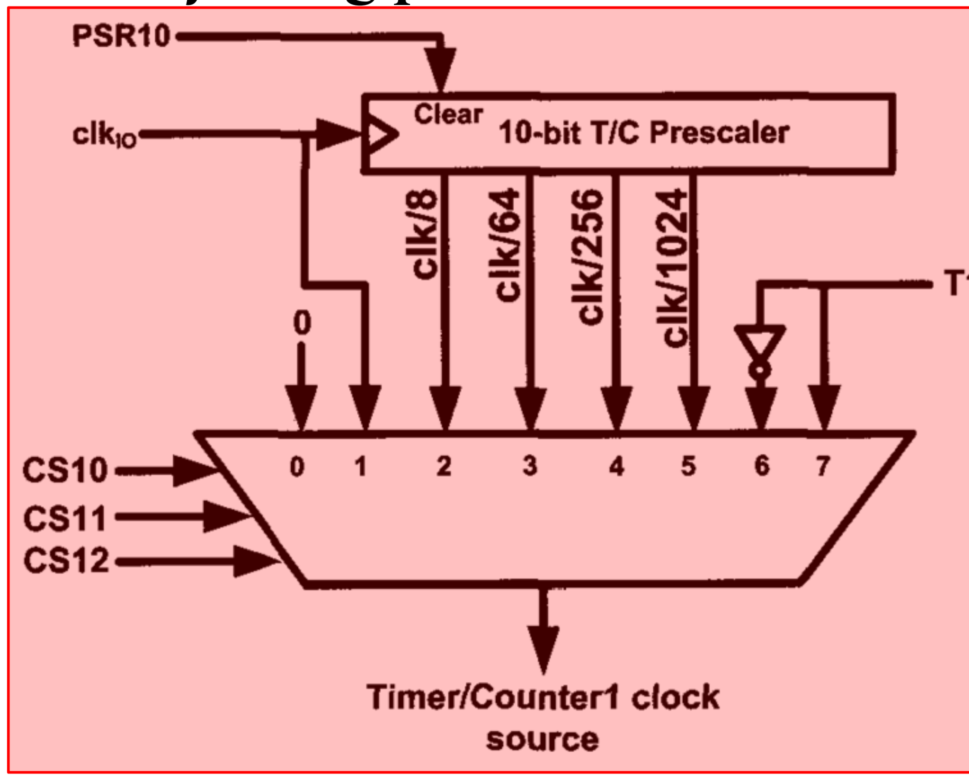


Figure 9-23. Timer/Counter 1 Prescaler

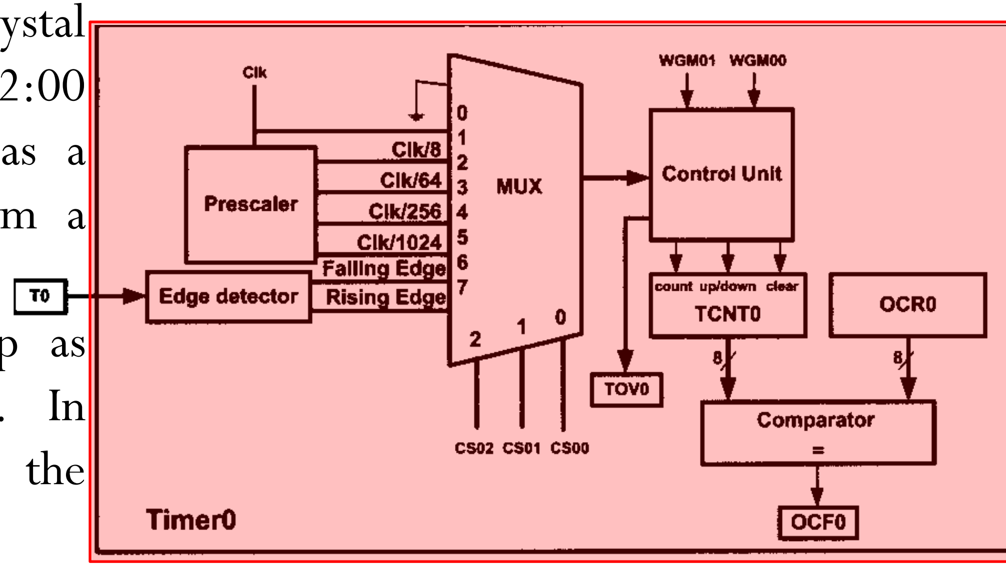
AVR TIMER PROGRAMMING IN ASSEMBLY
9.2 COUNTER PROGRAMMING

CS00, CS01, and CS02 bits in the TCCR0 register

When the timer is used as a timer, the AVR's crystal is used as the source of the frequency. When it is used as a counter, however, it is a pulse outside the AVR that increments the TCNTx register.

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.

The TCNT0 counter counts up as pulses are fed from pin T0. In ATmega32/ATmegal6, T0 is the alternative function of PORTB.0.



AVR TIMER PROGRAMMING IN ASSEMBLY
9.2 COUNTER PROGRAMMING

Example 9-34

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.

Solution:

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

In the case of 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/ATmegal6, T1 is the alternative function of PORTB.1.

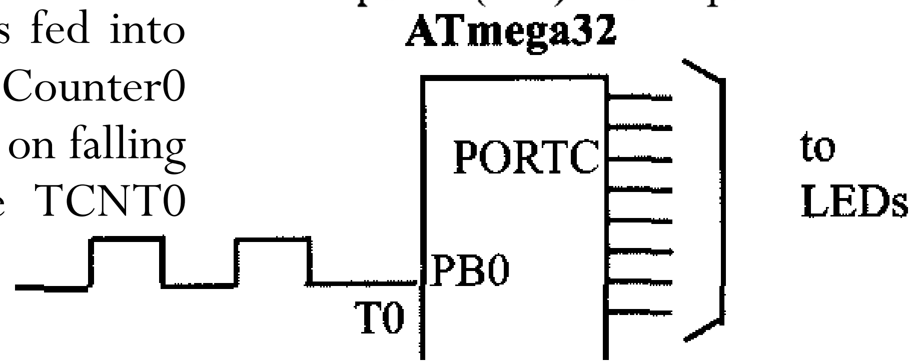
AVR TIMER PROGRAMMING IN ASSEMBLY

9.2 COUNTER PROGRAMMING

Example 9-35

Assuming that a 1Hz 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.

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




```
.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            ;monitor TOV0 flag
        SBRS    R16,TOV0            ;keep doing if Timer() flag is low
        RJMP    AGAIN
        LDI     R16,1<<TOV0
        OUT     TIFR,R16            ;clear TOV0 flag
        RJMP    AGAIN               ;keep doing it
```

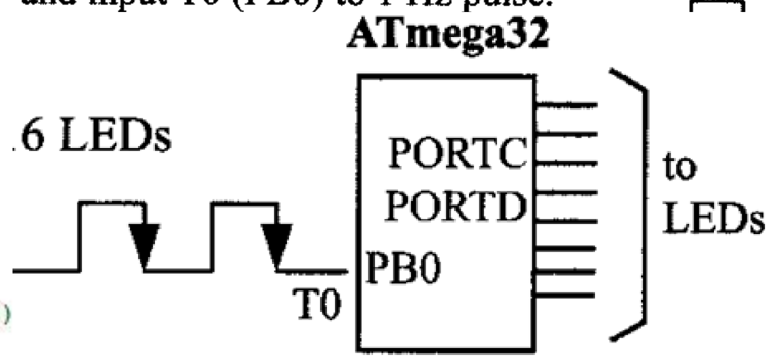
AVR TIMER PROGRAMMING IN ASSEMBLY

9.2 COUNTER PROGRAMMING

Example 9-36

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

PORTC and PORTD are connected to 16 LEDs and input T0 (PB0) to 1 Hz pulse.



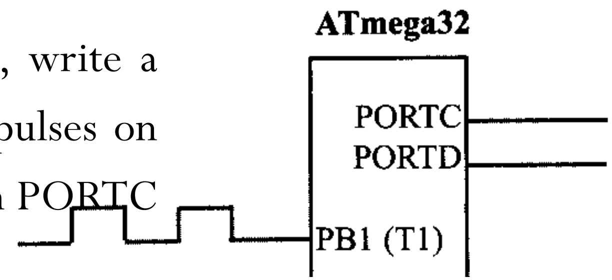

```
.INCLUDE "M32DEF.INC"
        LDI     R19,0           ;R19 = 0
        CBI     DDRB,0          ;make T0 (PB0)
        LDI     R20,0xFF
        OUT     DDRC,R20        ;make PORTC output
        OUT     DDRD,R20        ;make PORTD output
        LDI     R20,0x06
        OUT     TCCR0,R20       ;counter, falling edge
AGAIN:
        IN      R20,TCNT0
        OUT     PORTC,R20       ;PORTC = TCNT0
        IN      R16,TIFR
        SBRS    R16,TOV0
        RJMP    AGAIN           ;keep doing it
        LDI     R16,1<<TOV0     ;clear TOV0 flag
        OUT     TIFR,R16
        INC     R19             ;R19 = R19 + 1
        OUT     PORTD,R19       ;PORTD = R19
        RJMP    AGAIN           ;keep doing it
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.2 COUNTER PROGRAMMING

Example 9-37

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 = TCNTIH
        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
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.2 COUNTER PROGRAMMING

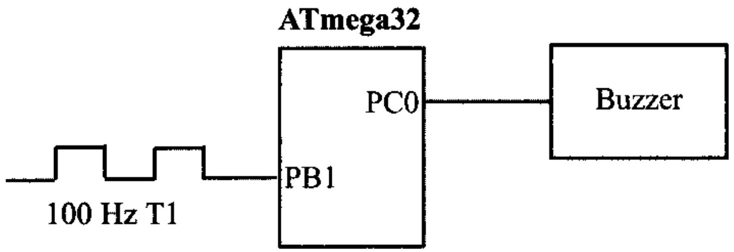
Example 9-38

Assuming that clock pulses are fed into pin T1 (PB1) and a buzzer is connected to pin PORTC.0, write a program for Counter 1 in CTC mode to sound the buzzer every 100 pulses.

Solution:

To sound the buzzer every 100 pulses, we set the OCR1A value to 99 (63 in hex), and then the counter counts up until it reaches OCR1A. Upon compare match, we can sound the buzzer by toggling the PORTC.0 pin.

PC0 is connected to a buzzer and input T1 to a pulse.



AVR TIMER PROGRAMMING IN ASSEMBLY

9.2 COUNTER PROGRAMMING

```
.INCLUDE "M32DEF.INC"
        CBI     DDRB,1          ;make T1 (PB1) input
        SBI     DDRC,0          ;PC0 as an output
        LDI     R16,0x1
        LDI     R17,0
        LDI     R20,0x0
        OUT     TCCR1A,R20      ;CTC, counter, falling edge
        LDI     R20,0x0E
        OUT     TCCR1B,R20
AGAIN:  LDI     R20,0
        OUT     OCR1AH,R20      ;TEMP = 0
        LDI     R20,99
        OUT     OCRIAL,R20      ;ORC1L = R20, OCR1H = TEMP
L1:     IN      R20,TIFR
        SBRS    R20,OCF1A
        RJMP    L1              ;keep doing it
        LDI     R2011<<OCF1A    ;clear OCF1A flag
        OUT     TIFR,R20
        EOR     R17,R16         ;toggle DO of RI7
        OUT     PORTC,R17       ;toggle PC0
        RJMP AGAIN              ;keep doing it
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.3 PROGRAMMING TIMERS IN C

Example 9-39

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.

```c
#include 'Tavr/io.h"
void T0Delay ( );
int main ( )
{
        DDRB = 0xFF;                //PORTB as 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 & 1)==0);      //wait for TF0 to roll over
        TCCR0 = 0;                  //stop Timer
        TIFR = 0x1;                 //clear TF0
}
```

Calculating delay length using timers

As we saw in the last two sections, the delay length depends primarily on two factors:

(a) the crystal frequency, and

(b) the prescaler factor. A third factor in the delay size is the C compiler because various C compilers generate different hex code sizes, and the amount of overhead due to the instructions varies by compiler.

AVR TIMER PROGRAMMING IN ASSEMBLY

9.3 PROGRAMMING TIMERS IN C

Example 9-40

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.

Solution:

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

$\text{Prescaler} = 1{:}8 \Rightarrow T_{\text{clock}} = 8 \times 1/8\ \text{MHz} = 1\,\mu\text{s}$

$70\ \mu\text{s}/1\ \mu\text{s} = 70\ \text{clocks}\ 4\Rightarrow 1 + \text{0xFF} - 70 = \text{0x100} — \text{0x46} = \text{0xBA} = 186$

mashhoun@iust.ac.ir

AVR TIMER PROGRAMMING IN ASSEMBLY
9.3 PROGRAMMING TIMERS IN C

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

void T0Delay ( );
int main ( )
{
        DDRB = 0xFF;                            //PORTS 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

}
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.3 PROGRAMMING TIMERS IN C

Example 9-41

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.

Solution:

XTAL = 8 MHz $\Rightarrow T_{\text{machine cycle}}$ = 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

AVR TIMER PROGRAMMING IN ASSEMBLY

9.3 PROGRAMMING TIMERS IN C

```c
#include "avr/io.h"
void T1Delay ( );
int main ()
{
        DDRB = 0xFF;                    //PORTE output port
        while (1)
        {
                PORTB = PORTB ^ (1<<PB4);   //toggle PB4
                T1Delay();                  //delay size unknown
        }
}

void T1Delay ( )
{
        TCNT1H = 0xC1;                      //TEMP = 0xC1
        TCNTIL = 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
}
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.3 PROGRAMMING TIMERS IN C

Example 9-42 (C version of Example 9-32)

Write a C program to toggle only the PORTB.4 bit continuously every second. Use Timer1, Normal mode, and 1:256 prescaler to create the delay. Assume XTAL = 8 MHz.

Solution:

XTAL = 8 MHz $\Rightarrow T_{\text{machine cycle}} = 1/8\ \text{MHz} = 0.125\ \mu\text{s} = T_{\text{Clock}}$

Prescaler 1:256 $\Rightarrow T_{\text{Clock}} = 256 \times 0.125\ \mu\text{s} = 32\ \mu\text{s}$

1s/32 µs = 31,250 clocks = 0x7Al2 clocks $\Rightarrow$ 1 + 0xFFFF - 0x7Al2 = 0x85EE

AVR TIMER PROGRAMMING IN ASSEMBLY

9.3 PROGRAMMING TIMERS IN C

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

        while(1)
        {
                PORTB = PORTB ^ (1<<PB4);
                T0Delay ();
        }
}

void T0Delay ()
{
        TCNT1H = 0x85;
        TCNT1L = 0xEE;
        TCCR1A = 0x00;
        TCCR1B = 0x04;
        while ((TIFR&(0x1<<TOV1))==0);
        TCCR1B = 0;
        TIFR = 0x1<<TOV1;
}
```

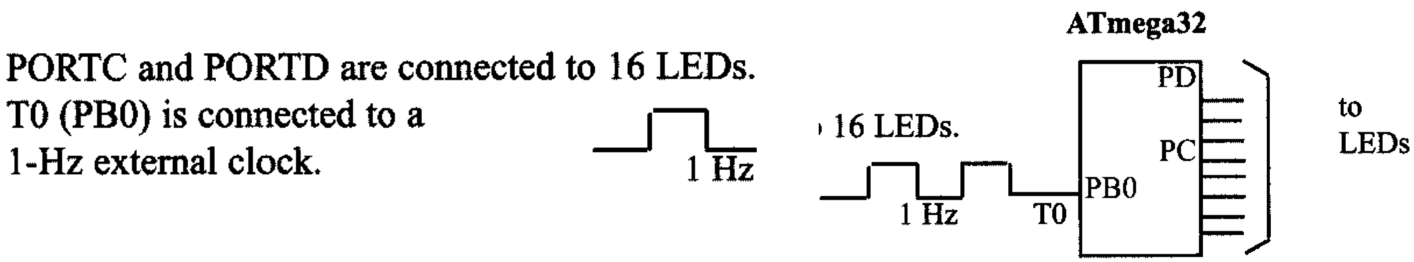
9.3 PROGRAMMING TIMERS IN C

C programming of Timers 0 and 1 as counters

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.

Example 9-43 (C version of Example 9-36)

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.



AVR TIMER PROGRAMMING IN ASSEMBLY

9.3 PROGRAMMING TIMERS IN C

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

int main ()
{
        PORTB = 0x01;           //activate pull-up of PB0
        DDRC = OxFF;            //PORTC as output
        DDRD = OxFF;            //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 TOVO
                PORTD ++;                       //increment PORTD
        }
}
```

AVR TIMER PROGRAMMING IN ASSEMBLY

9.3 PROGRAMMING TIMERS IN C

Example 9-44 (C version of Example 9-37)

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 TCNT11, registers on PORTD and PORTC, respectively.

```
#include "avr/io.h"
int main ()
{
        PORTB = 0x01;                   //activate pull-up of
        DDRC =  0xFF;                   //PORTC as output
        DDRD =  0xFF;                   //PORTD as output

        TCCR1A = 0x00;                  //output clock source
        TCCR1B = 0x06;                  //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(TIFRE&(0x1<<TOV1))==0);   //wait for TOV1

                T1FR = 0x1<<TOV1;               //clear TOV1
        }
}
```