AVR TIMER PROGRAMMING

Hoda Roodaki

hroodaki@kntu.ac.ir

AVR TIMER PROGRAMMING

- Many applications need to count an event or generate time delays.
  - There are counter registers in microcontrollers for this purpose.
    - Its advantage is that the input clock and operation of the timer is independent of the program execution.
- Every timer needs a clock pulse to tick.
- The clock source can be internal or external.
  - 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.*
  - By choosing the external clock option, we feed pulses through one of the AVR's pins.
    - This is called a *counter* to count, detect, and measure the time of events happening outside the AVR.

AVR TIMER PROGRAMMING

- 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.
- 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.

PROGRAMMING TIMERS 0

- **Basic registers of timers**
  - A TCNTn (timer/counter) register.
    - In ATmega32 we have TCNT0, TCNT1, and TCNT2.
    - The TCNTn register is a counter. Upon reset, the TCNTn contains zero. It counts up with each pulse.
    - The contents of the timers/counters can be accessed using the TCNTn.
      - You can load a value into the TCNTn register or read its value.
  - Timer0 is 8-bit in ATmega32; thus, TCNT0 is 8-bit:

PROGRAMMING TIMERS 0

- A TCCRn (timer/counter Control register) register
  - for setting modes of operation.
  - Timer0 can be specified to work as a timer or a counter by loading proper values into the TCCR0.
  - TCCRO (Timer/Counter Control Register) register:

Bit	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

PROGRAMMING TIMERS 0

- TCCRO register:
  - *CS02:CS00 (Timer0 clock source)*
    - These bits in the TCCRO 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.

PROGRAMMING TIMERS 0

D2	D1	D0	Timer0 clock selector
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	freq (No Prescaling)
0	1	0	freq / 8
0	1	1	freq / 64
1	0	0	freq / 256
1	0	1	freq / 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.

PROGRAMMING TIMERS 0

- 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.

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

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

TOV0 (Timer0 Overflow)

- The flag is set when the counter overflows, going from $00 to $FF.
  - 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.
  - In order to clear it we need to write 1 to it.

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

- **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

- To generate a time delay using Timer0 in Normal mode, the following steps are taken:
  - Load the TCNT0 register with the initial count value.
  - Load the value into the TCCR0 register, indicating 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.
  - Keep monitoring the timer overflow flag (TOV0) to see if it is raised. Get out of the loop when TOV0 becomes high.

Steps to program Timer0 in Normal mode

– Stop the timer by disconnecting the clock source, using the following instructions

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

– Clear the TOV0 flag for the next round.

– Go back to Step 1 to load TCNT0 again.

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
;--------------------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 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 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 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
	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.

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   PORTB,R17    ;toggle PB5
      RJMP  BEGIN
DELAY:LDI   R20,0x3E
      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)      ;R20 = 0x01
      OUT   TIFR,R20           ;clear TOV0 flag
      RET
```

Solution:

(a) (FF - 3E + 1) = 0xC2 = 194 in decimal and 194 × 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 × 0.125 µs = 24.25 µs as the width of the pulse.

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:
  - Calculate the period of the timer clock using the following formula

    $$T\ clock = 1 / Ftimer$$

    - where FTimer is the frequency of the clock used for the timer.
  - Divide the desired time delay by T clock· This says how many clocks we need.
  - Perform 256 - *n,* where *n* is the decimal value we got in Step 2.
  - Convert the result of Step 3 to hex, where *xx* is the initial hex value to be loaded into the timer's register.
  - Set TCNTO = *xx.*

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

```
.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    ;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 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 = 62.5 μs 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.

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:
  - the crystal frequency
  - the timer's 8-bit register
- These factors are beyond the control of the AVR programmer.
- The largest time delay is achieved by making TCNT0 zero. We can use the prescaler option in the TCCR0 register to increase the delay by reducing the period. The prescaler option of TCCR0 allows us to divide the instruction clock by a factor of 8 to 1024.

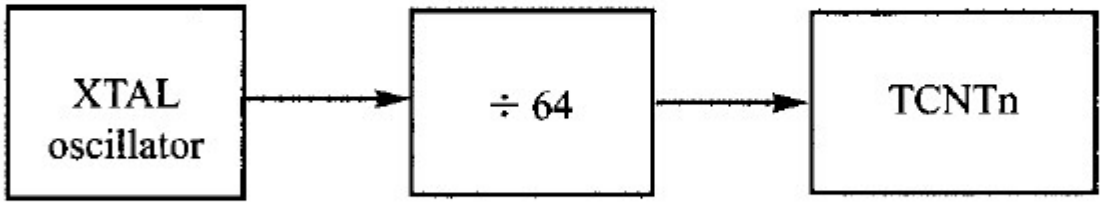
Prescaler and generating a large time delay

- With no prescaler enabled, the crystal oscillator frequency is fed directly into Timer0.
- If we enable the prescaler bit in the TCCR0 register, however, then we can divide the clock before it is fed into Timer0.
- The lower 3 bits of the TCCR0 register give the options of the number we can divide by.
  - This number can be 8, 64, 256, and 1024.
  - Notice that the lowest number is 8 and the highest number is 1024.

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

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.

TCCR0 =

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

Example 9-12

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          ;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   R20,0x10
      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
```

Example 9-12

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

Solution:

TCNT0 = 0x10 = 16 in decimal and 256 – 16 = 240. Now 240 × 64 × 0.125 μs = 1920 μs, or from Example 9-10, we have 240 × 8 μs = 1920 μs.

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) 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.

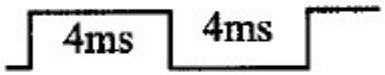
Example 9-14

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

(d) TCNT0 = 83 (hex)

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

Example 9-14

```
        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
```

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.

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
```

Example 9-16

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.

Clear Timer0 on compare match (CTC) mode programming

- OCR0 Register
  - The OCR0 register is used with CTC 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 TCCR0 register should be initialized.
- The OCF0 flag is located in the TIFR register.

Clear Timer0 on compare match (CTC) mode programming

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

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

D2	D1	D0	Timer0 clock selector
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	freq (No Prescaling)
0	1	0	freq / 8
0	1	1	freq / 64
1	0	0	freq / 256
1	0	1	freq / 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.

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.

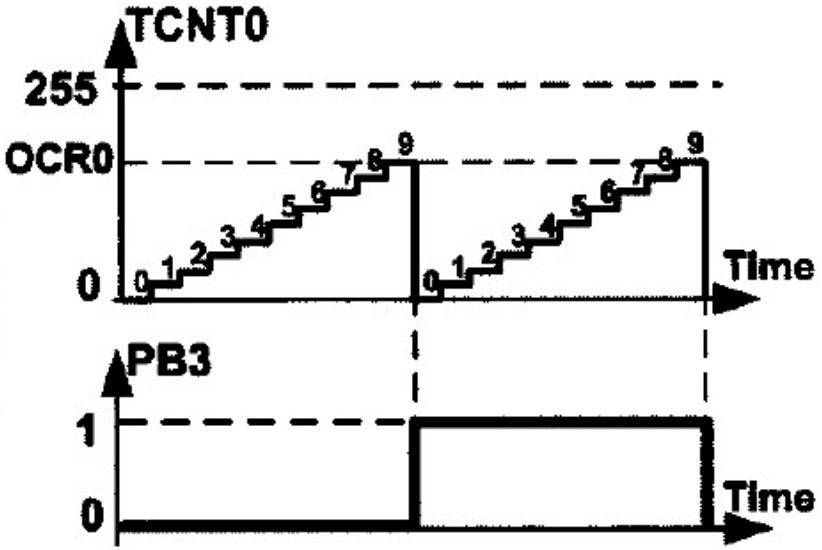
```
.INCLUDE "M32DEF.INC"
       INITSTACK                ;add its definition from Example 9-3
       LDI    R16,0x08
       SBI    DDRB,3           ;PB3 as an output
       LDI    R17,0
BEGIN:OUT     PORTB,R17        ;PORTB = R17
       RCALL  DELAY
       EOR    R17,R16          ;toggle D3 of R17
       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
```

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:

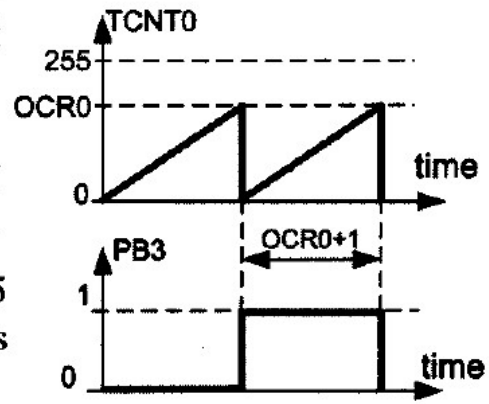
OCR0 is loaded with 9 and TCNT0 is cleared; Thus, after 9 clocks TCNT0 becomes equal to OCR0. On the next clock, the OCF0 flag is set and the reset occurs. That means the TCNT0 is cleared after 9 + 1 = 10 clocks. Because XTAL = 8 MHz, the counter counts up every 0.125 μs. Therefore, we have 10 × 0.125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



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.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
```



Solution:

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 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 × 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 9-21

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

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	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**

Example 9-21

```
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
```

Clear Timer0 on compare match (CTC) mode programming

- 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 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.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
```

Timer2 programming

- Timer2 works the same way as Timer0.
- But there are two differences between Timer0 and Timer2:
  - 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 (Asynchronous Status Register) AS2 bit.

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

Timer2 programming

- 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 programming

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

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
```

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	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**

Example 9-25

```
;----------------- Timer2 Delay
DELAY:LDI   R20,0
      OUT   TCNT2,R20          ;TCNT2 = 0
      LDI   R20,249
      OUT   OCR0,R20           ;OCR0 = 249
      LDI   R20,0x0E
      OUT   TCCR0,R20          ;Timer0,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 programming

- 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 programming

- There are two OCR registers in Timer1:
  - OCR1A
  - OCR1B
- There are two separate flags for each of the OCR registers, which act independently of each other.
  - Whenever TCNT1 equals OCR1A, the OCF1A flag will be set on the next timer clock.
  - When TCNT1 equals OCR1B, the OCF1B flag will be set on the next clock.
  - 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 (OCRIA high byte) and OCR1AL (OCR1lA low byte).

Timer1 programming

- TCCR1A and TCCR1B

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

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

Timer1 programming

- TCCR1A (Timer 1 Control) Register

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

Timer1 programming

CS12:CS10	D2	D1	D0	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.

Timer1 programming

- 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

Example 9-27

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          ;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
      RET
```

Example 9-27

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 × 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
```

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.

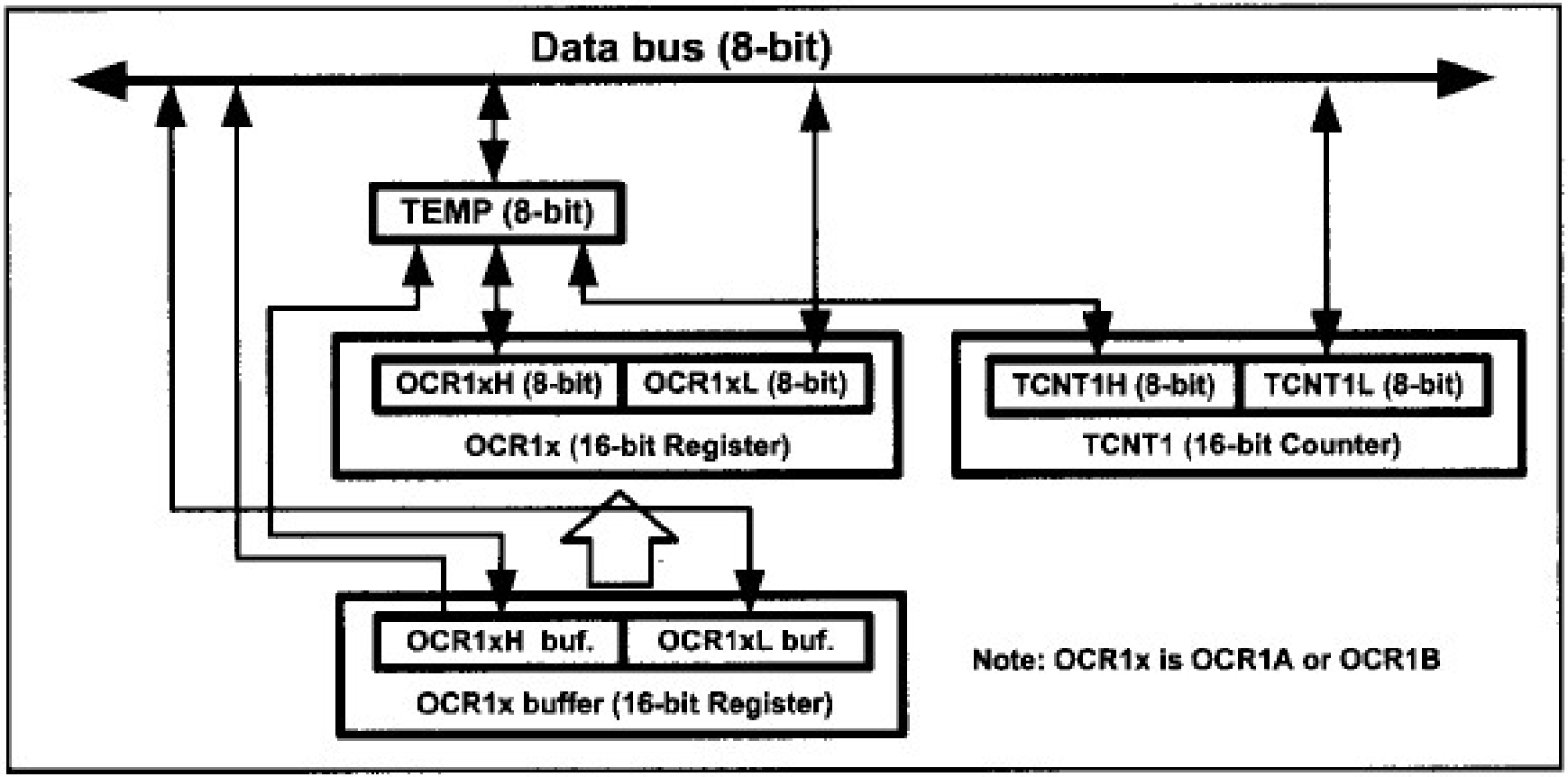
Accessing 16-bit registers

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

Accessing 16-bit registers



Accessing 16-bit registers

- 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
```

- After the execution of "OUT TCNT1H, R16", the content of R16, 0x15, will be stored in the TEMP register.
- When the instruction "OUT TCNTIL, 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.

Accessing 16-bit registers

- 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.

Accessing 16-bit registers

- For example, the following code 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.

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

Accessing 16-bit registers

- 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
```

Accessing 16-bit registers

- Notice that reading the OCR1A and OCR1B registers does not involve using the temporary register.
- It is because the AVR microcontroller does not update the content of OCR1A nor OCR1B unless we update them. For example, consider the following program

Accessing 16-bit registers

- This code reads the low byte of the OCR1A and then the high byte, and between the two readings the content of the register remains unchanged. That is why the AVR does not employ the TEMP register while reading the OCR1A / OCR1B registers.

```
IN    R20,OCR1AL        ;R20 = OCR1L
IN    R21,OCR1AH        ;R21 = OCR1H
```

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 1 ms delay, we need 1 ms/0.125 µs = 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.

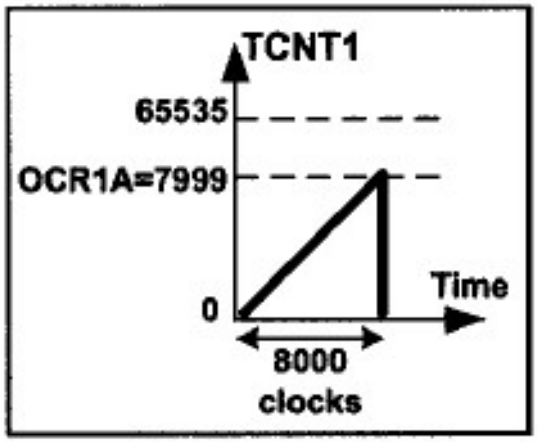
```
.INCLUDE "M32DEF.INC"
      LDI   R16,HIGH(RAMEND)
      OUT   SPH,R16
      LDI   R16,LOW(RAMEND)
      OUT   SPL,R16           ;initialize the stack
      SBI   DDRB,5            ;PB5 as an output
BEGIN:SBI   PORTB,5           ;PB5 = 1
      RCALL DELAY_1ms
      CBI   PORTB,5           ;PB5 = 0
      RCALL DELAY_1ms
      RJMP BEGIN
;----------------------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        ;OCR1AL = 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
```

Example 9-30

```
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
```



Generating a large time delay using prescaler

- We can use the prescaler option in the TCCR1B register to increase the delay by reducing the period. The prescaler option of TCCR1B allows us to divide the instruction clock by a factor of 8 to 1024.

Example 9-32

An LED is connected to PC4. Assuming XTAL = 8 MHz, write a program that toggles the LED once per second.

Solution:

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

Scaler	Timer Clock	Timer Period	Timer Value
None	8 MHz	1/8 MHz = 0.125 µs	1 s/0.125 µs = 8 M
8	8 MHz/8 = 1 MHz	1/1 MHz = 1 µs	1 s/1 µs = 1 M
64	8 MHz/64 = 125 kHz	1/125 kHz = 8 µs	1 s/8 µs = 125,000
256	8 MHz/256 = 31.25 kHz	1/31.25 kHz = 32 µs	1 s/32 µs = 31,250
1024	8 MHz/1024 = 7.8125 kHz	1/7.8125 kHz = 128 µs	1 s/128 µs = 7812.5

From the above calculation we can use only options 256 or 1024. We should use option 256 since we cannot use a decimal point.

```
.INCLUDE "M32DEF.INC"
      LDI   R16,HIGH(RAMEND)  ;initialize stack pointer
      OUT   SPH,R16
      LDI   R16,LOW(RAMEND)
      OUT   SPL,R16
      SBI   DDRC,4            ;PC4 as an output
BEGIN:SBI   PORTC,4           ;PC4 = 1
      RCALL DELAY_1s
      CBI   PORTC,4           ;PC4 = 0
      RCALL DELAY_1s
      RJMP BEGIN
```

Example 9-32

```
;-------------------- Timer1 delay
DELAY_1s:
      LDI   R20,HIGH (31250-1)
      OUT   OCR1AH,R20          ;TEMP = $7A (since 31249 = $7A11)
      LDI   R20,LOW (31250-1)
      OUT   OCR1AL,R20          ;OCR1AL = $11 (since 31249 = $7A11)
      LDI   R20,0
      OUT   TCNT1H,R20          ;TEMP = 0x00
      OUT   TCNT1L,R20          ;TCNT1L = 0x00, TCNT1H = TEMP
      LDI   R20,0x00
      OUT   TCCR1A,R20          ;WGM11:10 = 00
      LDI   R20,0x4
      OUT   TCCR1B,R20          ;WGM13:12 = 00, Normal mode,CS = CLK/256
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 9-33

Assuming XTAL = 8 MHz, write a program to generate 1 Hz frequency on PC4.

Solution:

With 1 Hz we have T = 1 / F = 1 / 1 Hz = 1 second, half of which is high and half low. Thus we need a delay of 0.5 second duration.
Since XTAL = 8 MHz, the different outputs of the prescaler are as follows:

Scaler	Timer Clock	Timer Period	Timer Value
None	8 MHz	1/8 MHz = 0.125 µs	0.5 s/0.125 µs = 4 M
8	8 MHz/8 = 1 MHz	1/1 MHz = 1 µs	0.5 s/1 µs = 500 k
64	8 MHz/64 = 125 kHz	1/125 kHz = 8 µs	0.5 s/8 µs = 62,500
256	8 MHz/256 = 31.25 kHz	1/31.25 kHz = 32 µs	0.5 s/32 µs = 15,625
1024	8 MHz/1024 = 7.8125 kHz	1/7.8125 kHz = 128 µs	0.5 s/128 µs = 3906.25

From the above calculation we can use options 64 or 256. We choose 64 in this Example.

```
.INCLUDE "M32DEF.INC"
      LDI   R16,HIGH(RAMEND)  ;initialize stack pointer
      OUT   SPH,R16
      LDI   R16,LOW(RAMEND)
      OUT   SPL,R16
      SBI   DDRC,4            ;PC4 as an output
BEGIN:SBI   PORTC,4           ;PC4 = 1
      RCALL DELAY_1s
      CBI   PORTC,4           ;PC4 = 0
      RCALL DELAY_1s
      RJMP BEGIN
```

Example 9-33

```
;-------------------- Timer1 delay
DELAY_1s:
      LDI   R20,HIGH (62500-1)
      OUT   OCR1AH,R20          ;TEMP = $F4 (since 62499 = $F423)
      LDI   R20,LOW (62500-1)
      OUT   OCR1AL,R20          ;OCR1AL = $23 (since 62499 = $F423)
      LDI   R20,0x00
      OUT   TCNT1H,R20          ;TEMP = 0x00
      OUT   TCNT1L,R20          ;TCNT1L = 0x00, TCNT1H = TEMP
      LDI   R20,0x00
      OUT   TCCR1A,R20          ;WGM11:10 = 00
      LDI   R20,0x3
      OUT   TCCR1B,R20          ;WGM13:12 = 00, Normal mode, CS = CLK/64
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
```

COUNTER PROGRAMMING

- The AVR timer can also be used to count, detect, and measure the time of events happening outside the AVR.
- 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.
- Notice that, in counter mode, registers such as TCCR, OCR0, and TCNT are the same as for the timer; they even have the same names.

COUNTER PROGRAMMING

- Recall from the previous section that the CS bits (clock selector) in the TCCR0 register decide the source of the clock for the timer.
- IfCS02:00 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).
- InATmega32/ATmega16, T0 is the alternative function of PORTB.0.

COUNTER PROGRAMMING

- 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.
- 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/ ATmegal 6, T1 is the alternative function of PORTB.1.

Example 9-35

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.

Solution:

```
.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
```

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.

Solution:

```
.INCLUDE "M32DEF.INC"
      LDI   R19,0             ;R19 = 0
      CBI   DDRB,0            ;make T0 (PB0) input
      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
```

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.

Solution:

```
.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
```

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.

```
.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
      LDI   R20,0x0E
      OUT   TCCR1B,R20      ;CTC, counter, falling edge
AGAIN:
      LDI   R20,0
      OUT   OCR1AH,R20      ;TEMP = 0
      LDI   R20,99
      OUT   OCR1AL,R20      ;ORC1L = R20, OCR1H = TEMP
L1:   IN    R20,TIFR
      SBRS  R20,OCF1A
      RJMP  L1              ;keep doing it
      LDI   R20,1<<OCF1A    ;clear OCF1A flag
      OUT   TIFR, R20

      EOR   R17,R16         ;toggle D0 of R17
      OUT   PORTC,R17       ;toggle PC0
      RJMP  AGAIN           ;keep doing it
```