CHW 469 : Embedded Systems

Instructor:

Dr. Ahmed Shalaby

http://bu.edu.eg/staff/ahmedshalaby14#

AVR Programming in C

Chapter 7

The AVR microcontroller and embedded systems

using assembly and c

Topics

- Data Types
- Time Delays
- IO Programming in C
- Logic Operation in C
- Data serialization in C

Data Types

- Use unsigned whenever you can
- unsigned char instead of unsigned int if you can

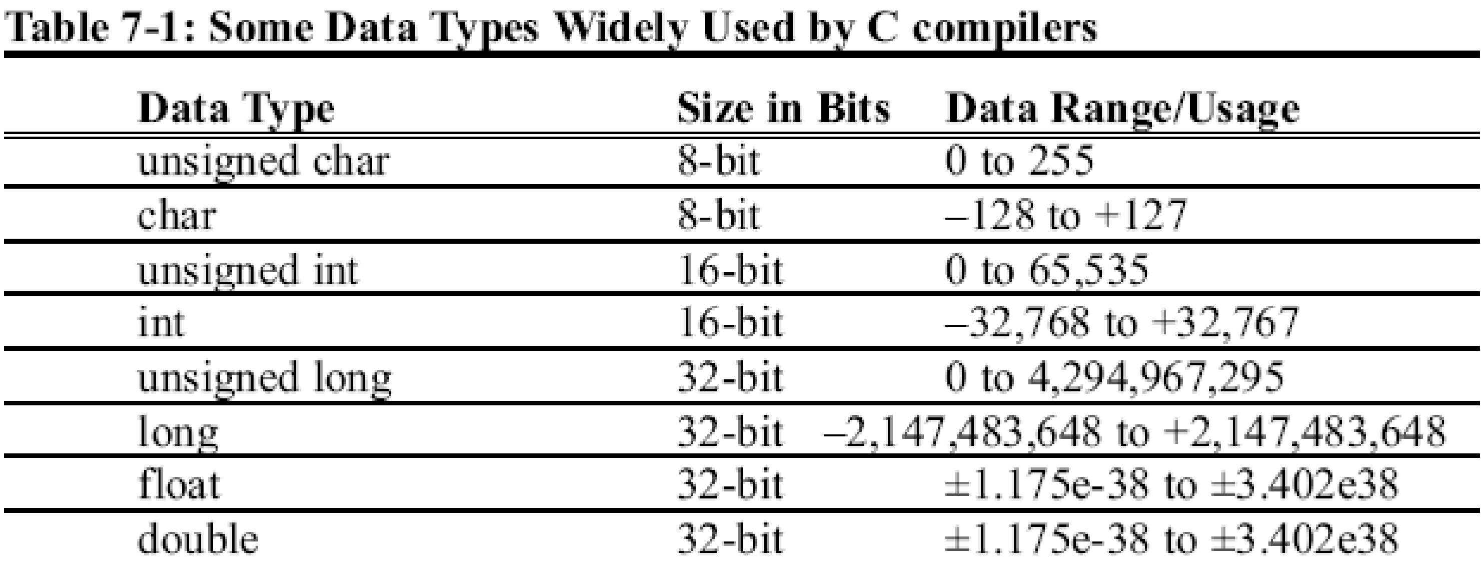
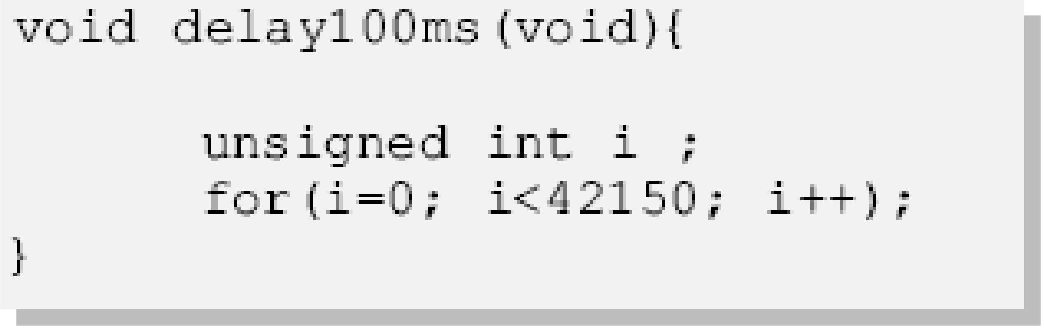


Table 7-1: Some Data Types Widely Used by C compilers

Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 255
char	8-bit	–128 to +127
unsigned int	16-bit	0 to 65,535
int	16-bit	–32,768 to +32,767
unsigned long	32-bit	0 to 4,294,967,295
long	32-bit	–2,147,483,648 to +2,147,483,648
float	32-bit	±1.175e-38 to ±3.402e38
double	32-bit	±1.175e-38 to ±3.402e38

Time Delays in C

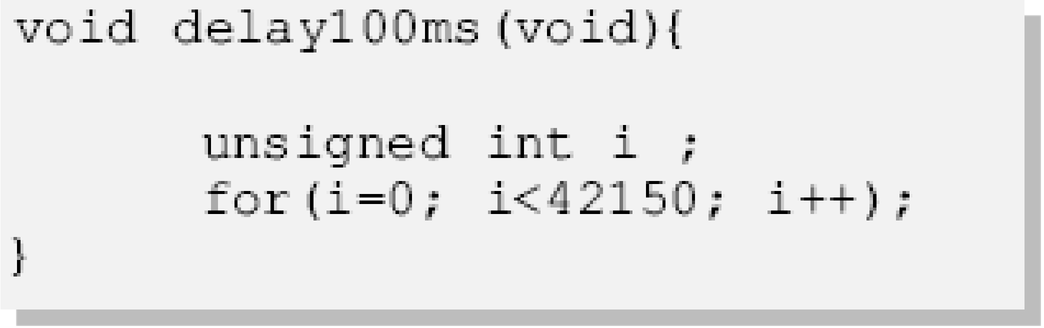
- You can use for to make time delay



```
void delay100ms(void){

        unsigned int i ;
        for(i=0; i<42150; i++);
}
```

Time Delays in C

- You can use for to make time delay

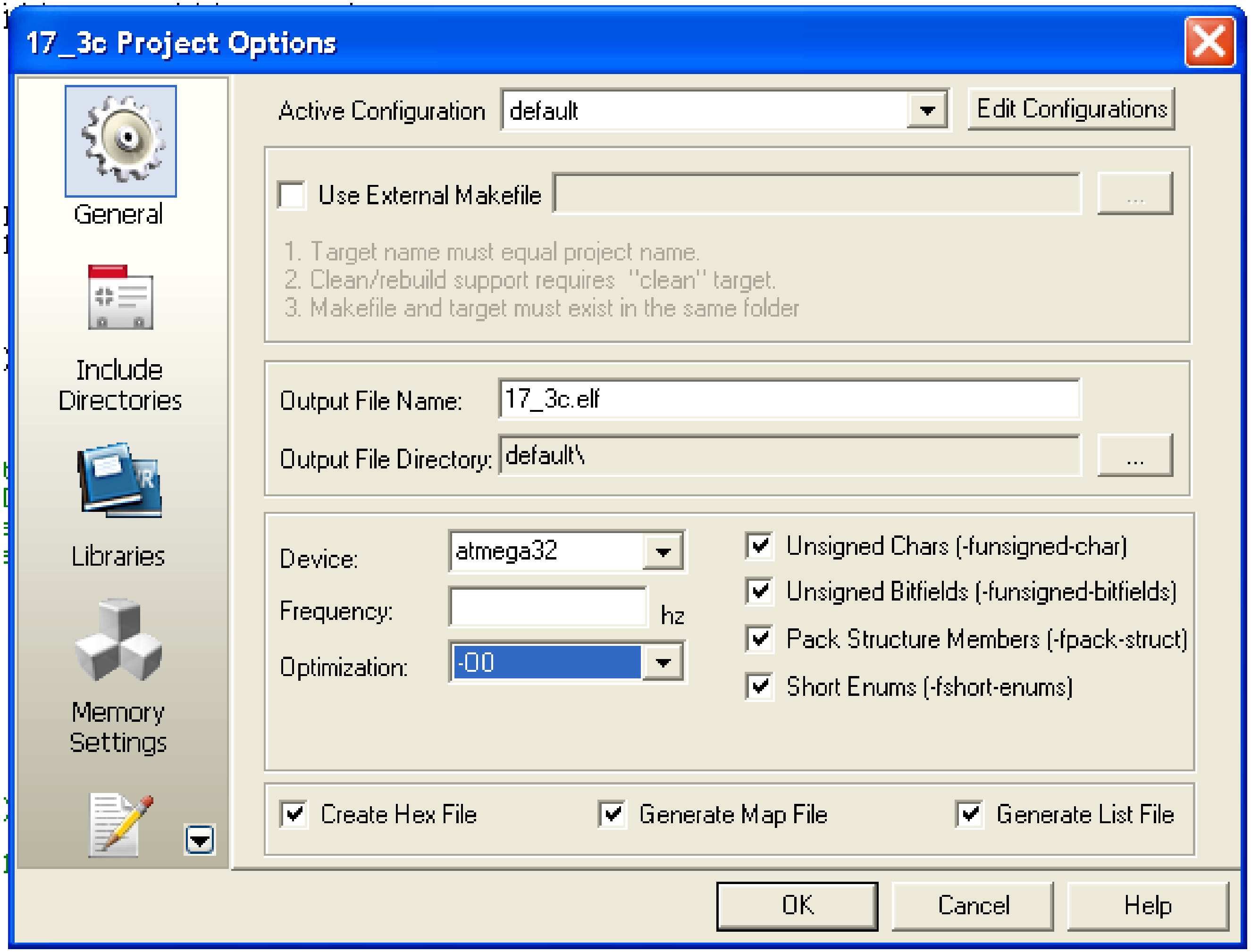


```
void delay100ms(void){

        unsigned int i ;
        for(i=0; i<42150; i++);
}
```

If you use for loop

- The clock frequency can change your delay duration !
- The compiler has direct effect on delay duration!
- You MUST set the optimization level to O0 !

How to set optimization level to O0



17_3c Project Options

Active Configuration: default

Use External Makefile: □

1. Target name must equal project name.
2. Clean/rebuild support requires "clean" target.
3. Makefile and target must exist in the same folder

Output File Name: 17_3c.elf

Output File Directory: default\

Device: atmega32

Frequency: hz

Optimization: -O0

☑ Unsigned Chars (-funsigned-char)

☑ Unsigned Bitfields (-funsigned-bitfields)

☑ Pack Structure Members (-fpack-struct)

☑ Short Enums (-fshort-enums)

☑ Create Hex File

☑ Generate Map File

☑ Generate List File

Time Delays in C

- You can use pre defined functions of compilers to make time delay

IN WinAVR :

First you should include:

```
#include <util/delay.h>
```

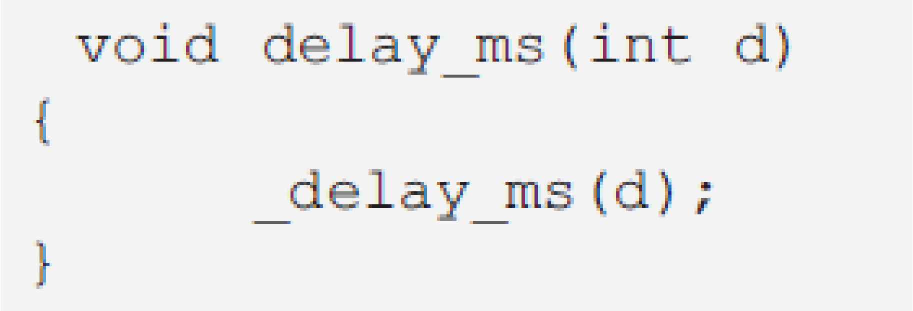
and then you can use

```
        delay_ms(1000);
        delay_us(1000);
```

- It is compiler dependant not hardware dependant

Time Delays in C

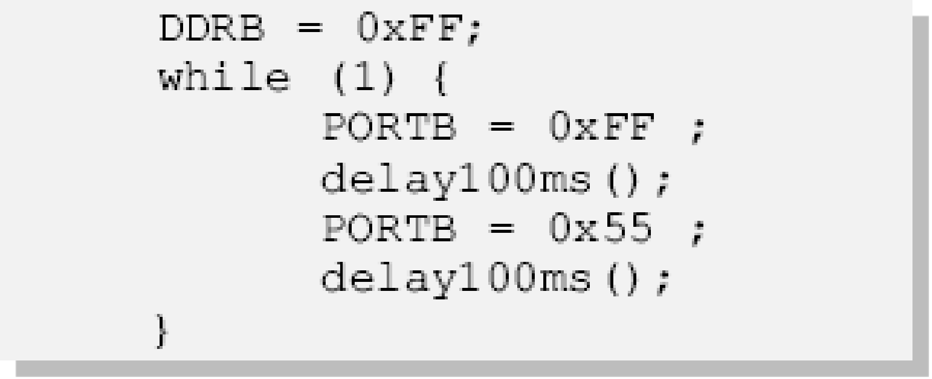
- To overcome the portability problem, you can use macro or wrapper function. So to change the compiler you need to change only a simple function.



```
  void delay_ms(int d)
{
        _delay_ms(d);
}
```

I/O programming in C

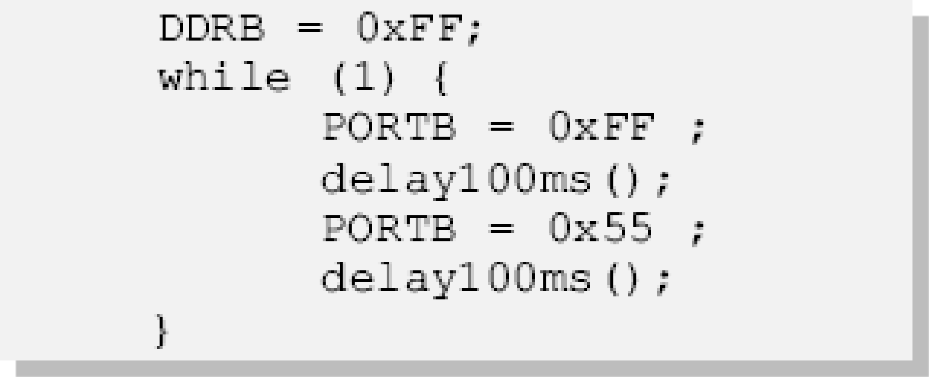
Byte size IO programming in C



```
DDRB = 0xFF;
while (1) {
        PORTB = 0xFF ;
        delay100ms();
        PORTB = 0x55 ;
        delay100ms();
}
```

I/O programming in C

Byte size IO programming in C

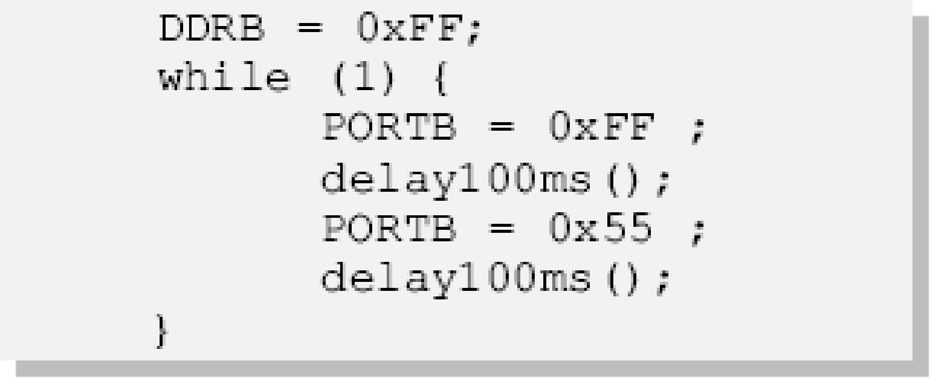


```
DDRB = 0xFF;
while (1) {
        PORTB = 0xFF ;
        delay100ms();
        PORTB = 0x55 ;
        delay100ms();
}
```

Different compilers have different syntax for bit manipulations!

I/O programming in C

Byte size IO programming in C



```
DDRB = 0xFF;
while (1) {
        PORTB = 0xFF ;
        delay100ms();
        PORTB = 0x55 ;
        delay100ms();
}
```

Different compilers have different syntax for bit manipulations!

Masking is the best way

Logical Operations in C

1110 1111 && 0000 0001 = True AND True = True

1110 1111 || 0000 0000 = True OR False = True

!(1110 1111) = Not (True) = False

Bit-Wise logical operators

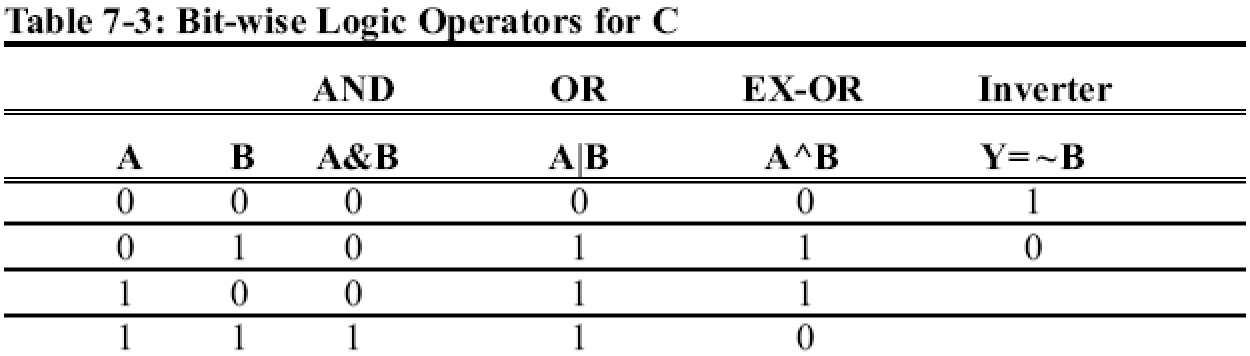


Table 7-3: Bit-wise Logic Operators for C

		AND	OR	EX-OR	Inverter
A	B	A&B	A|B	A^B	Y=~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

```
  1110 1111
& 0000 0001
 ---------------
  0000 0001
```

```
  1110 1111
| 0000 0001
 ---------------
  1110 1111
```

```
~ 1110 1011
 ---------------
  0001 0100
```

Shift operations in C

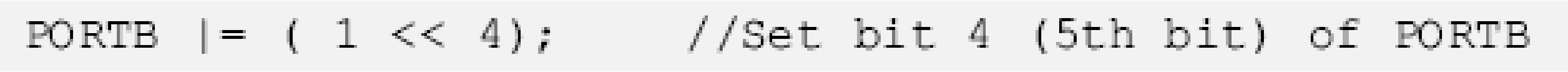
- data >> number of bits to be shifted right
- data << number of bits to be shifted left

```
 1110 0000 >> 3           0000 0001 <<2
---------------          ---------------
 0001 1100                0000 0100
```

Setting a bit in a Byte to 1

- We can use | operator to set a bit of a byte to 1

```
     xxxx xxxx                          xxxx xxxx
|    0001 0000          OR         |    1 << 4
     --------------                     --------------
     xxx1 xxxx                          xxx1 xxxx
```

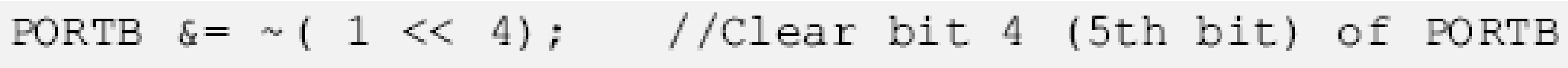


```
PORTB |= ( 1 << 4);       //Set bit 4 (5th bit) of PORTB
```

Clearing a bit in a Byte to 0

- We can use | operator to set a bit of a byte to 1

```
       xxxx xxxx                      xxxx xxxx
&      1110 1111          OR      &   ~(1 << 4)
       --------------                 --------------
       xxx0 xxxx                      xxx0 xxxx
```



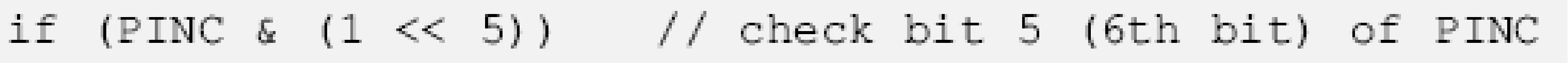
```
PORTB &= ~( 1 << 4);    //Clear bit 4 (5th bit) of PORTB
```

See Example 7-18

Checking a bit in a Byte

- We can use & operator to see if a bit in a byte is 1 or 0

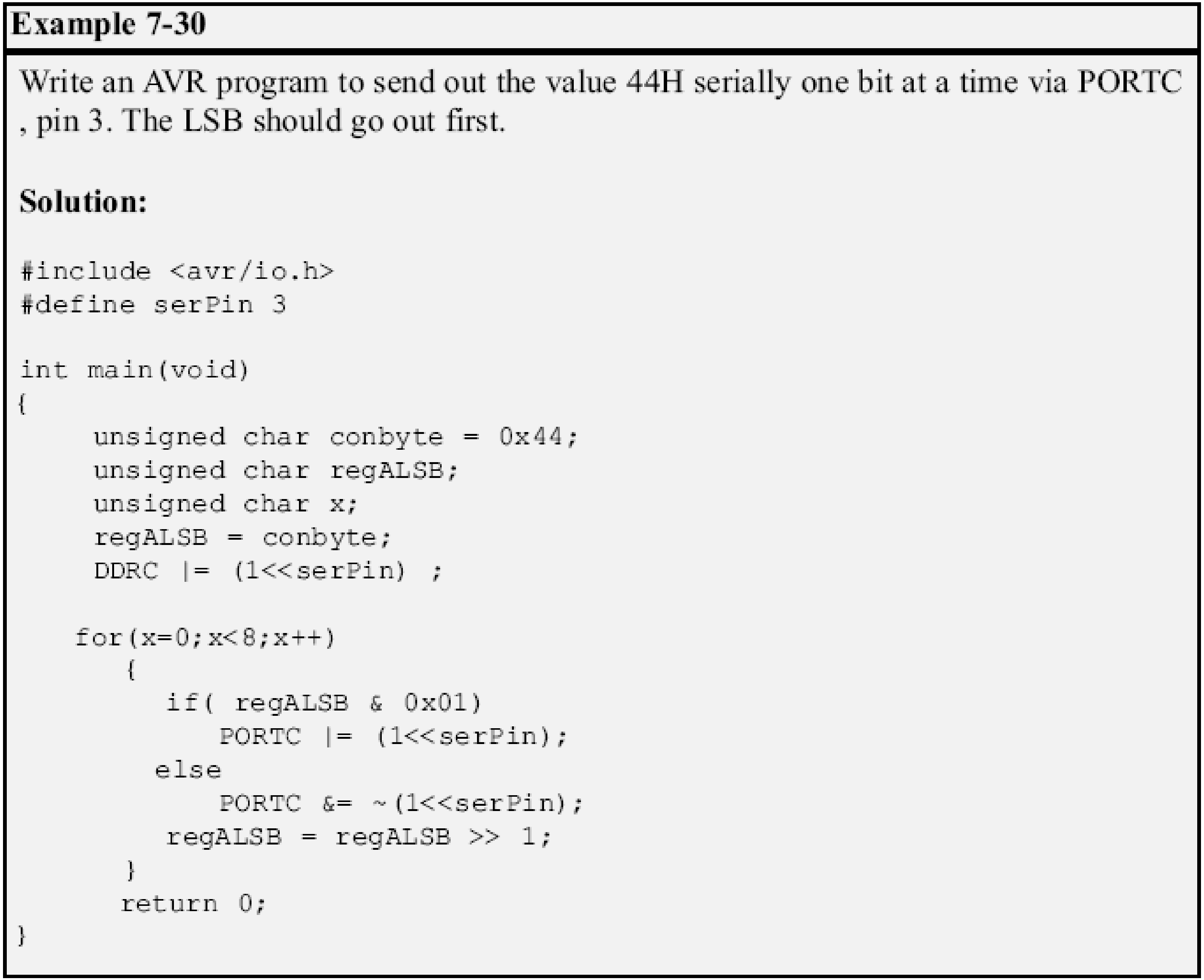
```
         xxxx xxxx                         xxxx xxxx
&        0001 0000           OR        &   (1 << 4)
         --------------                    --------------
         000x 0000                         00x0 0000
```



```
if (PINC & (1 << 5))    // check bit 5 (6th bit) of PINC
```

Data Serialization in C

- Any of serial ports (USART, SPI, I2C, JTAG,…)
- Do it yourself !



Example 7-30

Write an AVR program to send out the value 44H serially one bit at a time via PORTC , pin 3. The LSB should go out first.

Solution:

```
#include <avr/io.h>
#define serPin 3

int main(void)
{
    unsigned char conbyte = 0x44;
    unsigned char regALSB;
    unsigned char x;
    regALSB = conbyte;
    DDRC |= (1<<serPin) ;

   for(x=0;x<8;x++)
      {
         if( regALSB & 0x01)
            PORTC |= (1<<serPin);
        else
            PORTC &= ~(1<<serPin);
         regALSB = regALSB >> 1;
      }
      return 0;
}
```

Memory Types In AVR

- Flash Memory
  - Not deleted when power is off
  - Big in size
  - Suitable for codes, tables and fixed data
- EEPROM
  - Not deleted when power is off
  - Not very big in size
  - Suitable for small data that may be modified but should not be lost when power is off
- RAM
  - deleted when power is off
  - Suitable for storing the data we want to manipulate because we have fast access read or modify them.

AVR Hardware Connections and Flash Loading

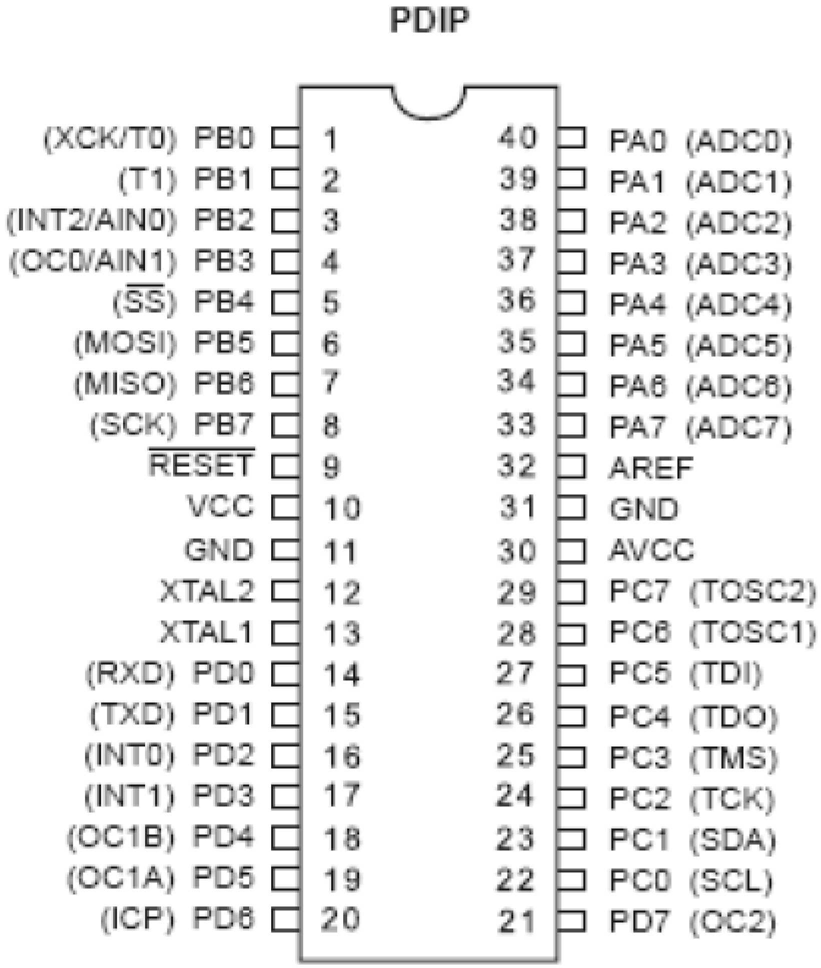
The AVR microcontroller and embedded systems

using assembly and c

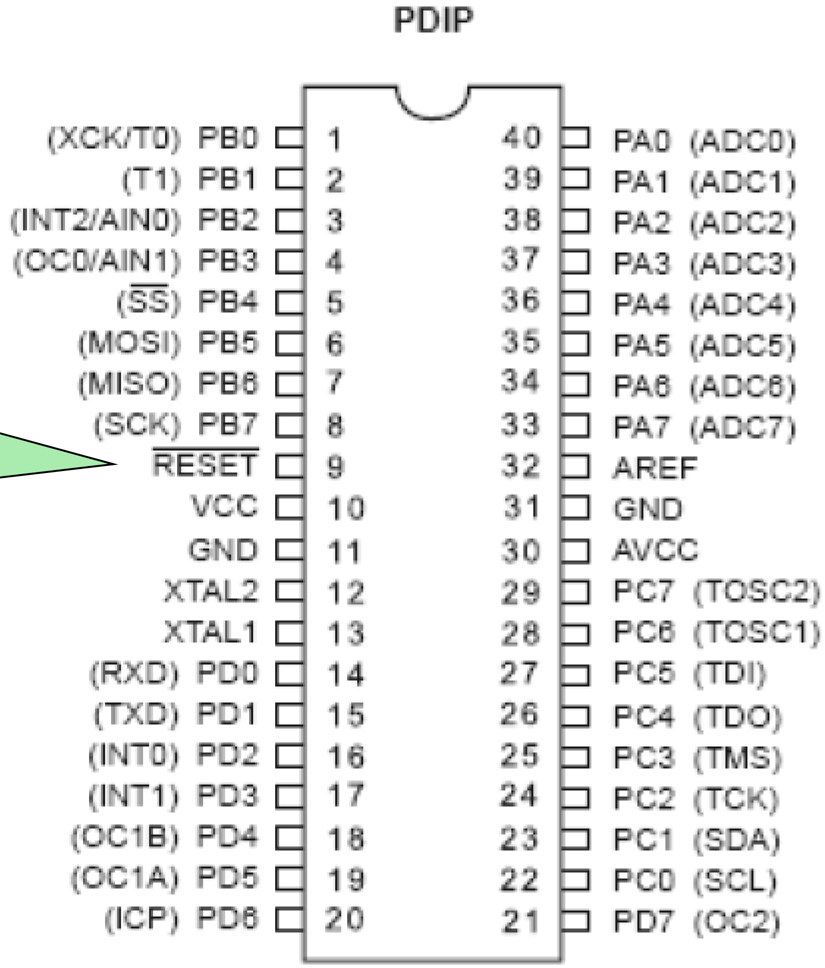
Topics

- AVR Pins
- AVR simplest connections
- Fuse bits and clock source
- Fuse bits and startup time
- What is inside a hex file?
- Loading a hex file into flash
- Start with MDE AVR 32 Trainer board

ATmega 32 pins

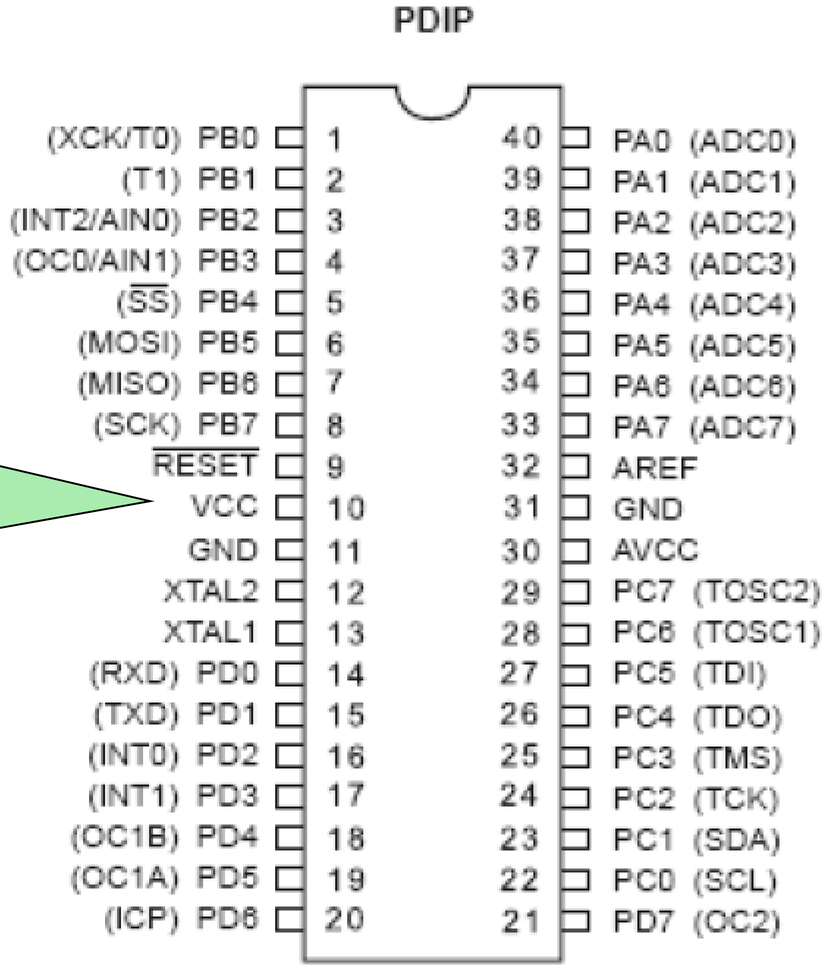


ATmega 32 pins



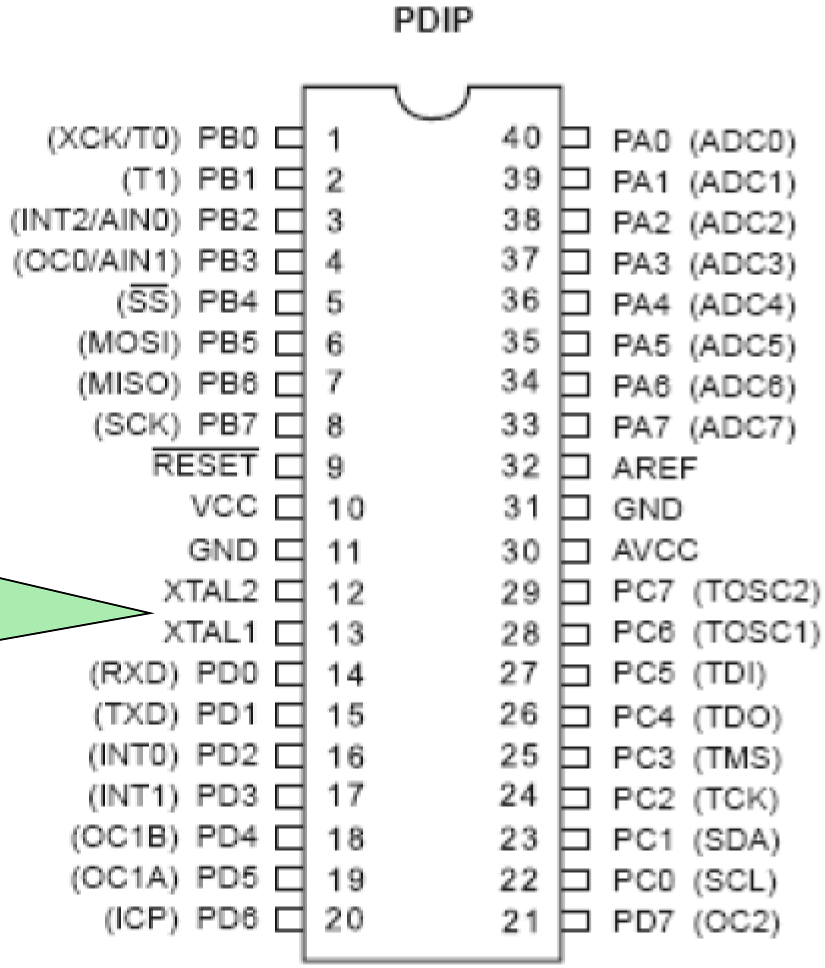
Clears all the registers and restart the execution of program

ATmega 32 pins



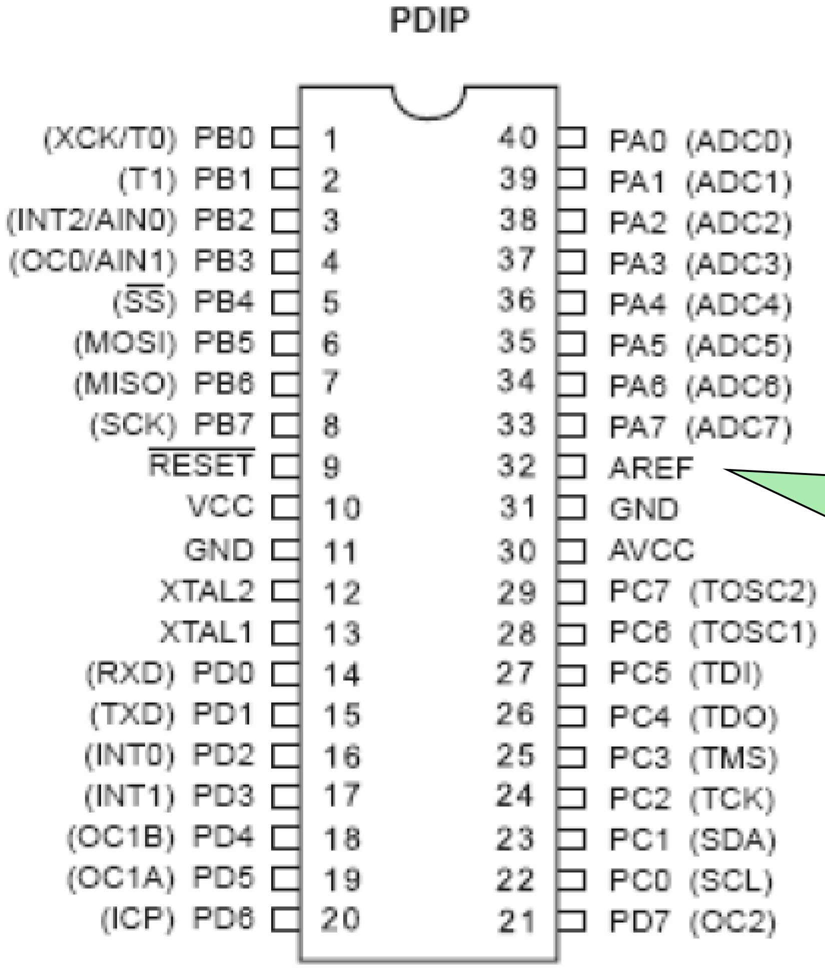
Provides supply voltage to the chip. It should be connected to +5

ATmega 32 pins



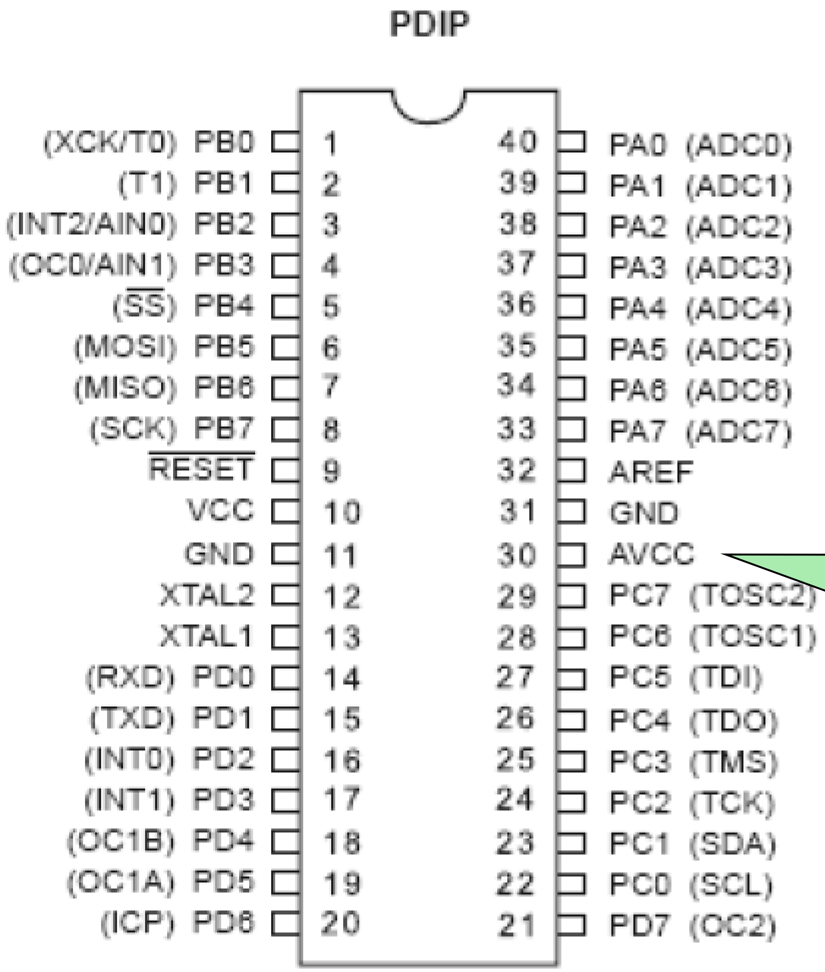
These pins are used to connect external crystal or RC oscillator

ATmega 32 pins



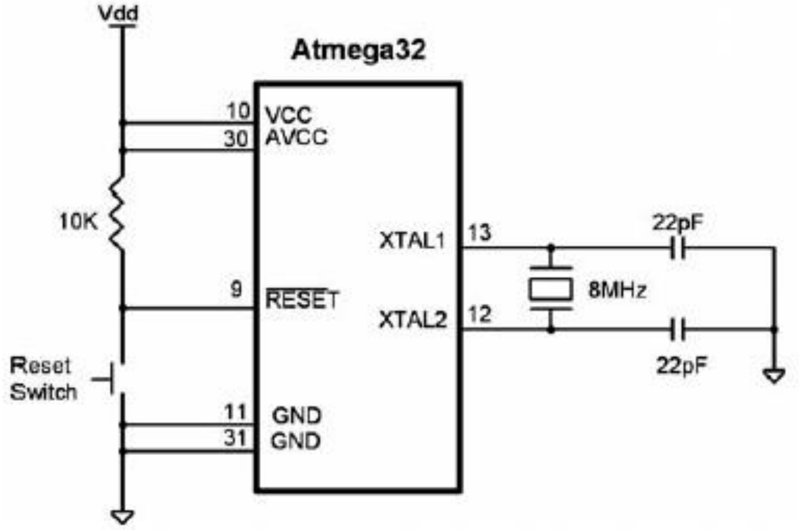
Reference voltage for ADC

ATmega 32 pins



Supply voltage for ADC and portA. Connect it to VCC

AVR simplest connection



Fuse bytes of ATmega 32

Values of fuses is needed before system starts

They control some features of hardware

There are Fuse bytes in Atmega32

Fuse bytes of ATmega 32

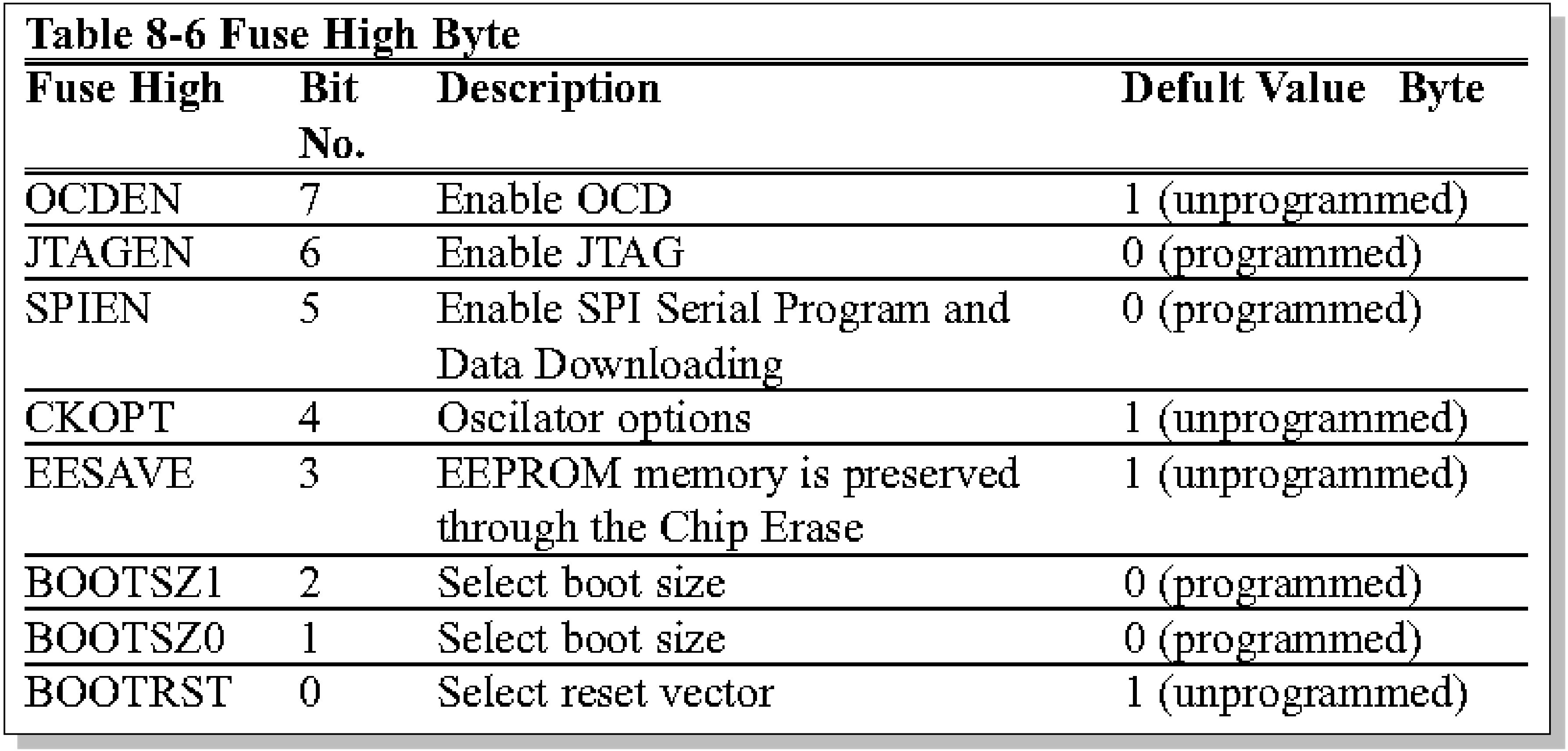


Table 8-6 Fuse High Byte

Fuse High	Bit No.	Description	Defult Value Byte
OCDEN	7	Enable OCD	1 (unprogrammed)
JTAGEN	6	Enable JTAG	0 (programmed)
SPIEN	5	Enable SPI Serial Program and Data Downloading	0 (programmed)
CKOPT	4	Oscilator options	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed)
BOOTSZ1	2	Select boot size	0 (programmed)
BOOTSZ0	1	Select boot size	0 (programmed)
BOOTRST	0	Select reset vector	1 (unprogrammed)

Fuse bytes of ATmega 32

Values of fuses is needed before system starts

They control some features of hardware

There are Fuse bytes in Atmega32

Fuse bytes of ATmega 32

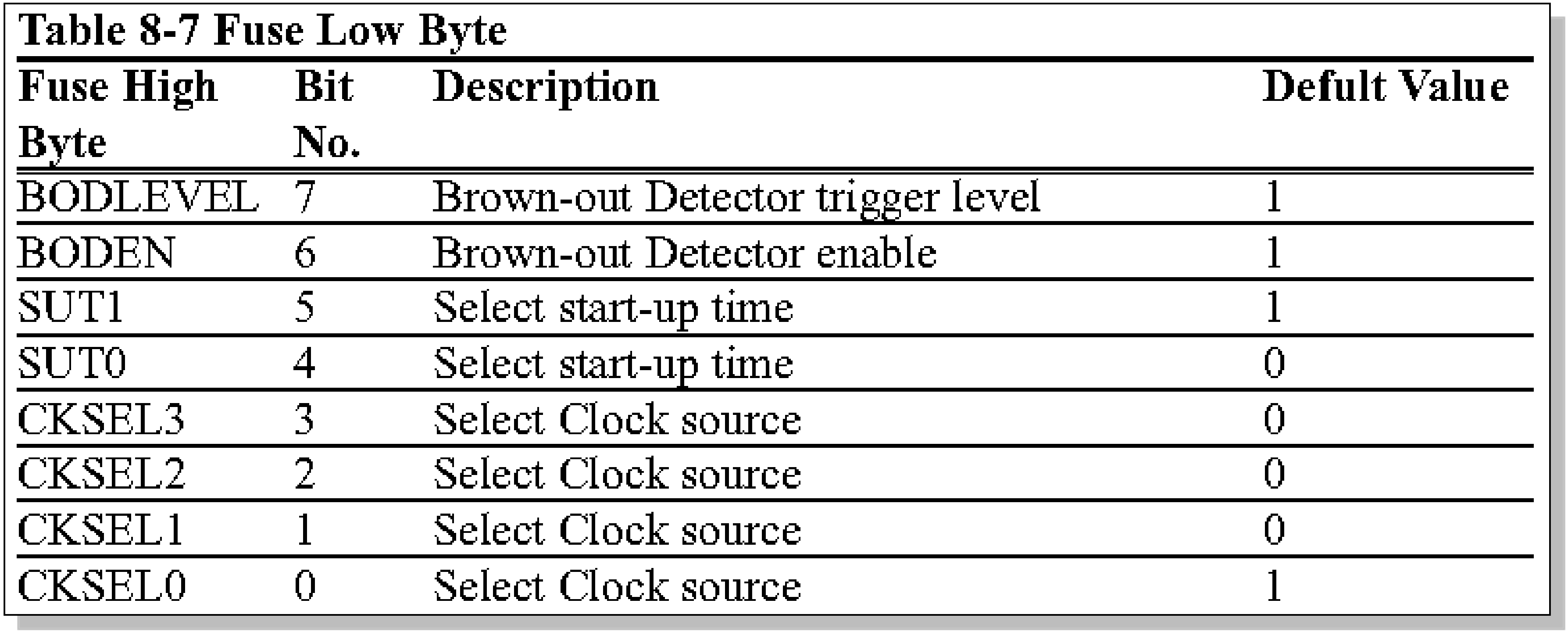


Table 8-7 Fuse Low Byte

Fuse High Byte	Bit No.	Description	Defult Value
BODLEVEL	7	Brown-out Detector trigger level	1
BODEN	6	Brown-out Detector enable	1
SUT1	5	Select start-up time	1
SUT0	4	Select start-up time	0
CKSEL3	3	Select Clock source	0
CKSEL2	2	Select Clock source	0
CKSEL1	1	Select Clock source	0
CKSEL0	0	Select Clock source	1

Fuse bytes of ATmega 32

Values of fuses is needed before system starts

They control some features of hardware

There are Fuse bytes in Atmega32

Clock source in ATmega 32

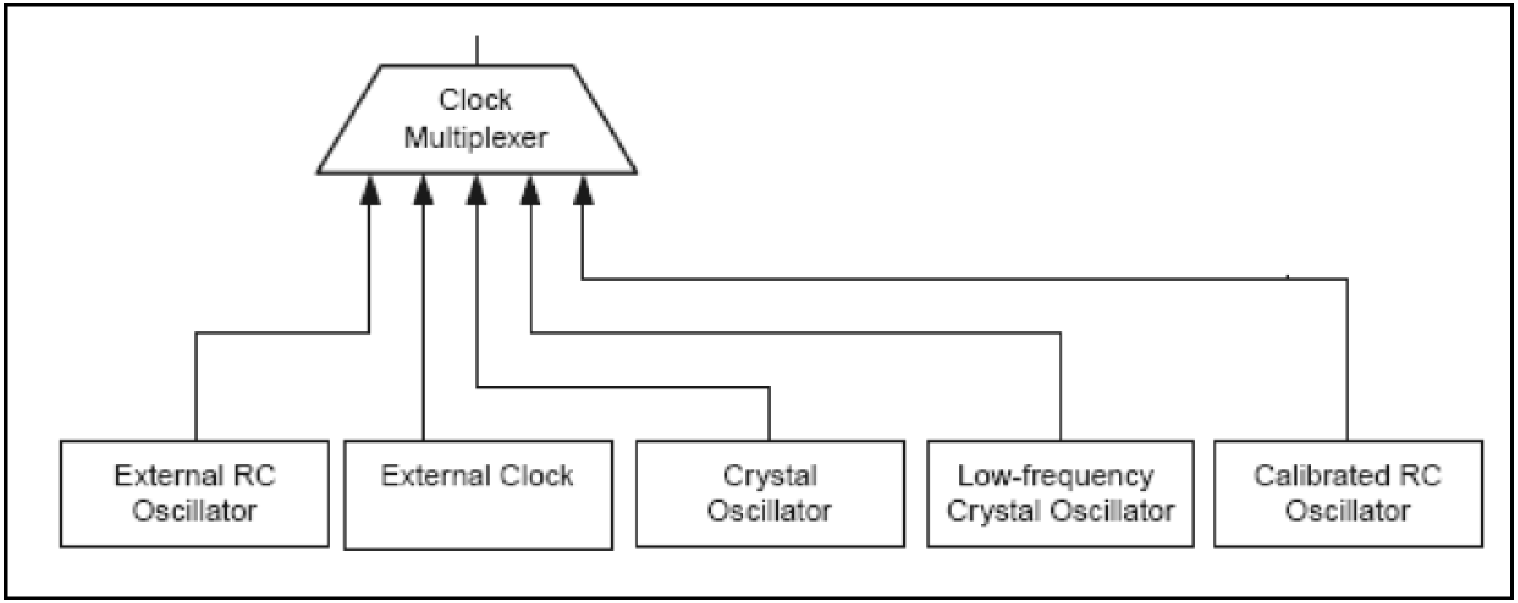


Figure 8-4. Atmega32 Clock Sources

Clock source in ATmega 32

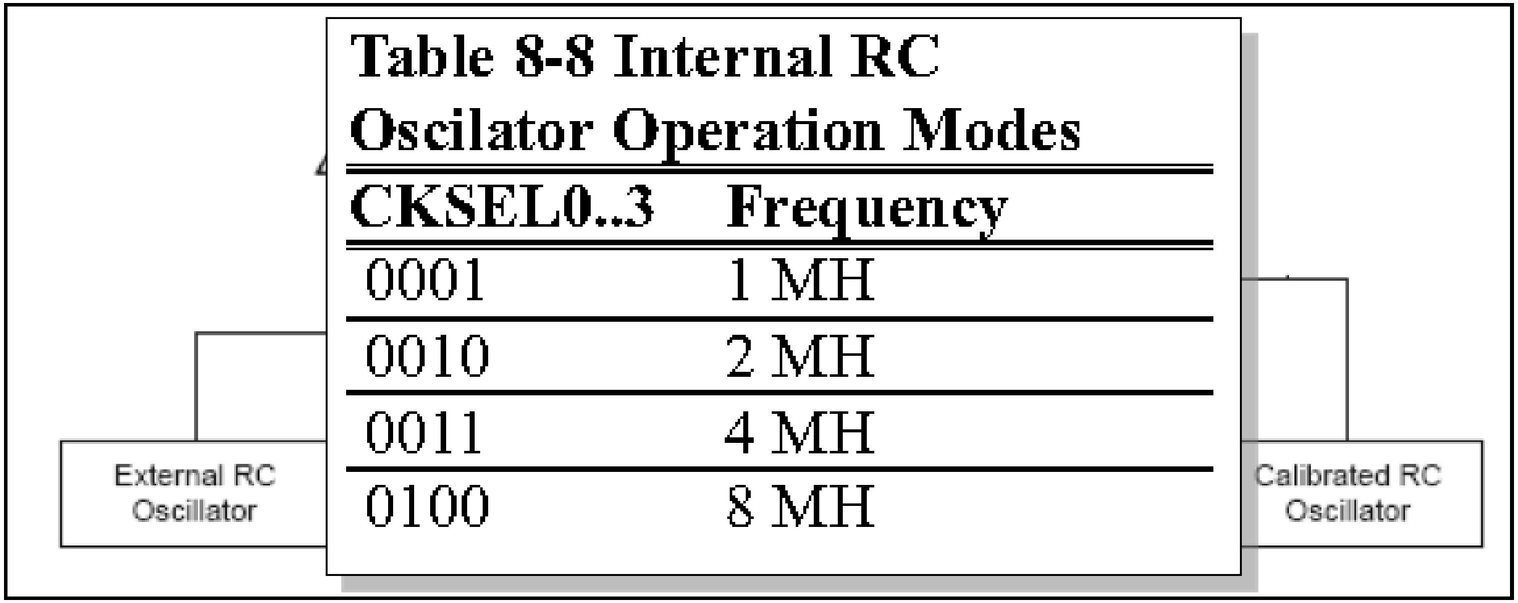


Table 8-8 Internal RC Oscilator Operation Modes

CKSEL0..3	Frequency
0001	1 MH
0010	2 MH
0011	4 MH
0100	8 MH

Figure 8-4. Atmega32 Clock Sources

Clock source in ATmega 32

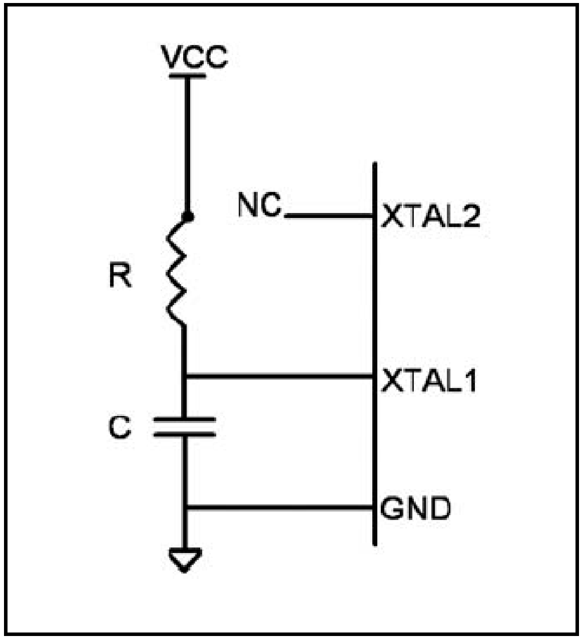


Figure 8-5 External RC

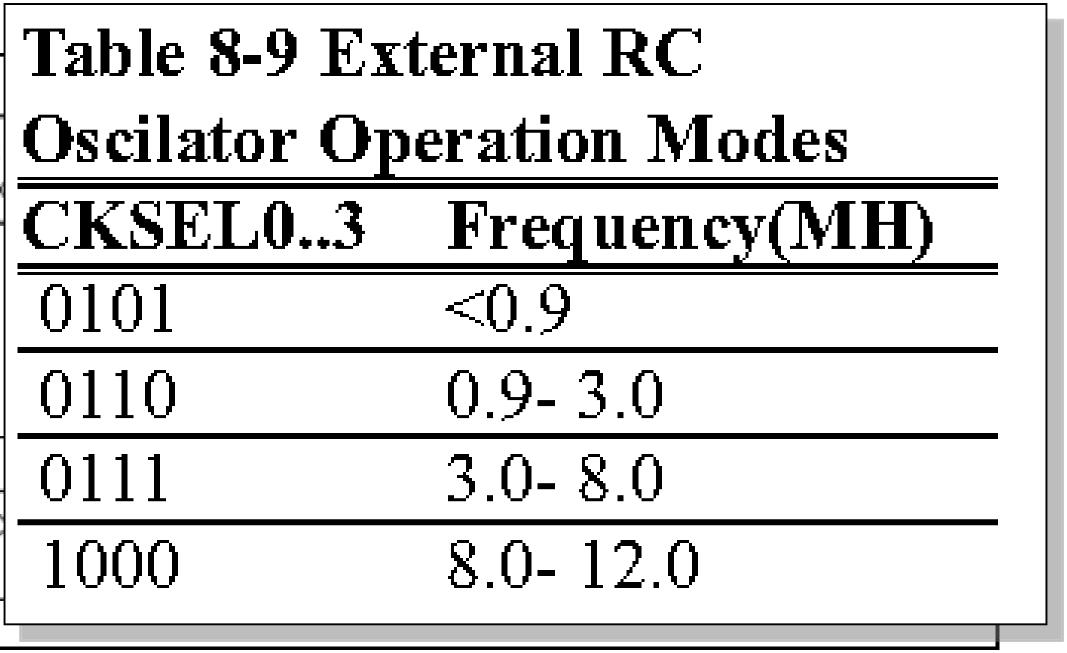


Table 8-9 External RC Oscilator Operation Modes

CKSEL0..3	Frequency(MH)
0101	<0.9
0110	0.9- 3.0
0111	3.0- 8.0
1000	8.0- 12.0

lock Sources

Clock source in ATmega 32

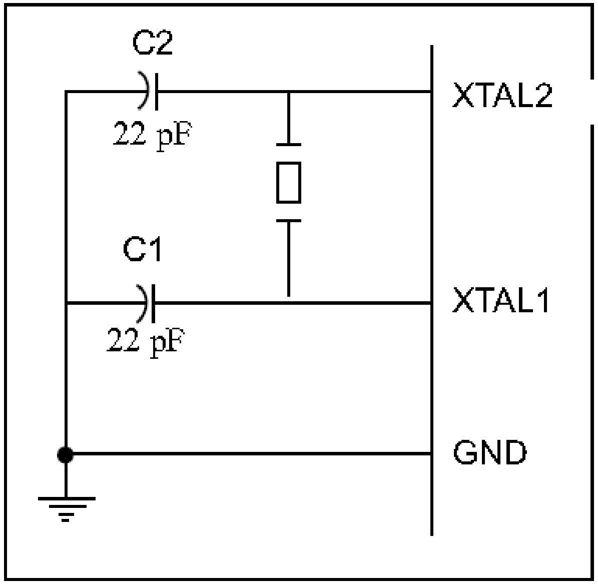
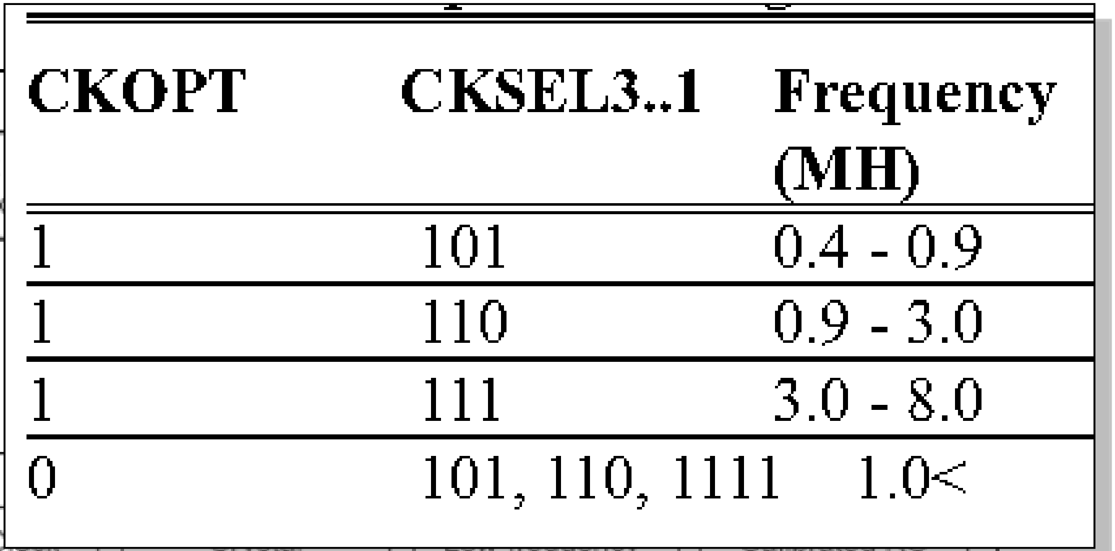


Figure 8-6a. XTAL1–XTAL2 Connection to Crystal Oscillator



CKOPT	CKSEL3..1	Frequency (MH)
1	101	0.4 - 0.9
1	110	0.9 - 3.0
1	111	3.0 - 8.0
0	101, 110, 1111	1.0<

lock Sources

Power on Reset and Burn on Detection

- Burn on Detection (BOD): Monitors the level of VCC and reset the system if (VCC<BOD level)
- The most difficult time for a system is during power up. To pass this time, In AVR when RESET pin becomes high, program does not starts running. It starts running after a specified time has elapsed. SUT0 and SUT1 define this time.

Power on Reset and Burn on Detection

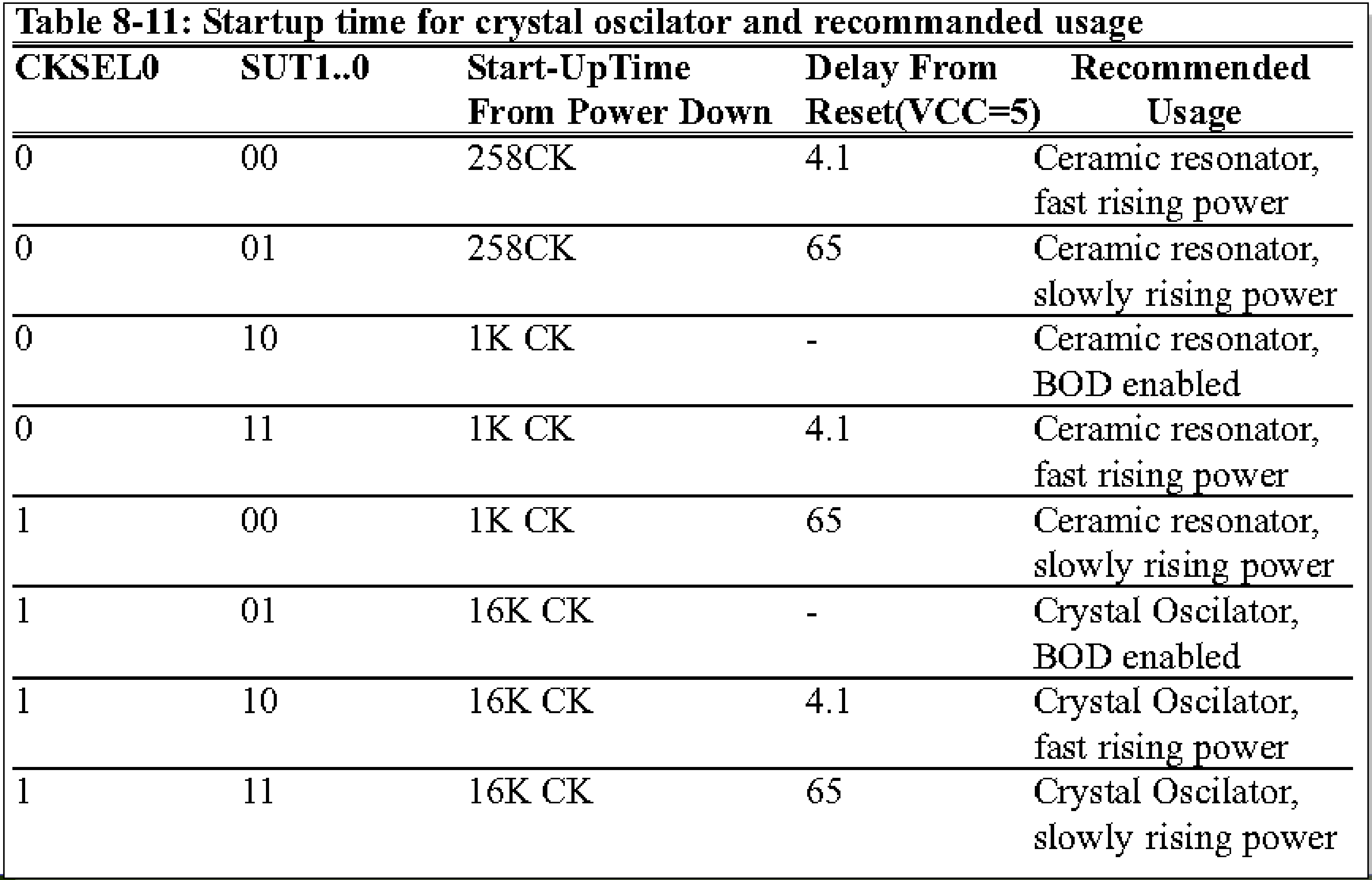


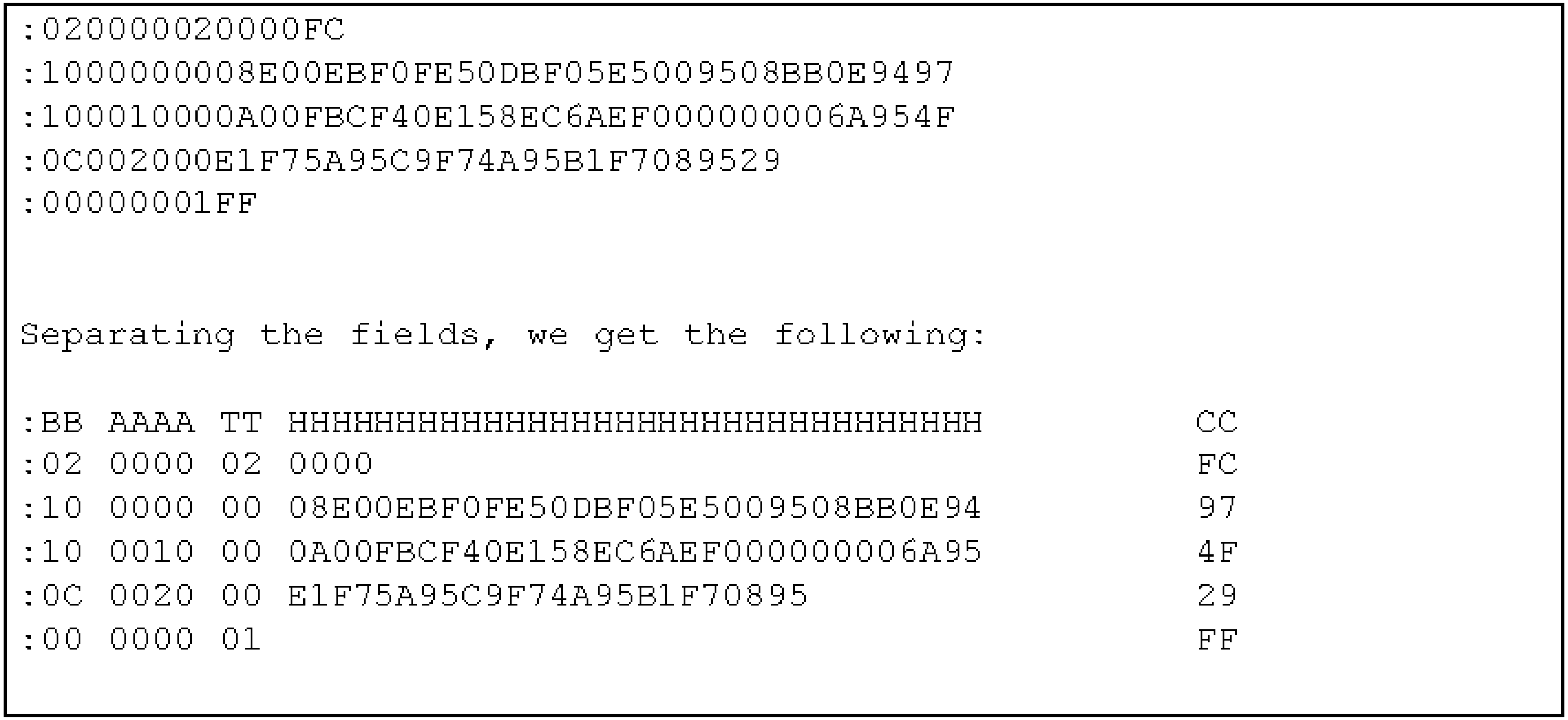
Table 8-11: Startup time for crystal oscilator and recommanded usage

CKSEL0	SUT1..0	Start-UpTime From Power Down	Delay From Reset(VCC=5)	Recommended Usage
0	00	258CK	4.1	Ceramic resonator, fast rising power
0	01	258CK	65	Ceramic resonator, slowly rising power
0	10	1K CK	-	Ceramic resonator, BOD enabled
0	11	1K CK	4.1	Ceramic resonator, fast rising power
1	00	1K CK	65	Ceramic resonator, slowly rising power
1	01	16K CK	-	Crystal Oscilator, BOD enabled
1	10	16K CK	4.1	Crystal Oscilator, fast rising power
1	11	16K CK	65	Crystal Oscilator, slowly rising power

Golden Rule of Fuse bits

If you are using an external crystal with a frequency more 1MH you can set all of the CKSEL3, CKSEL2, CKSEL1, SUT1 and SUT0 to 1 and clear CKOPT to 0.

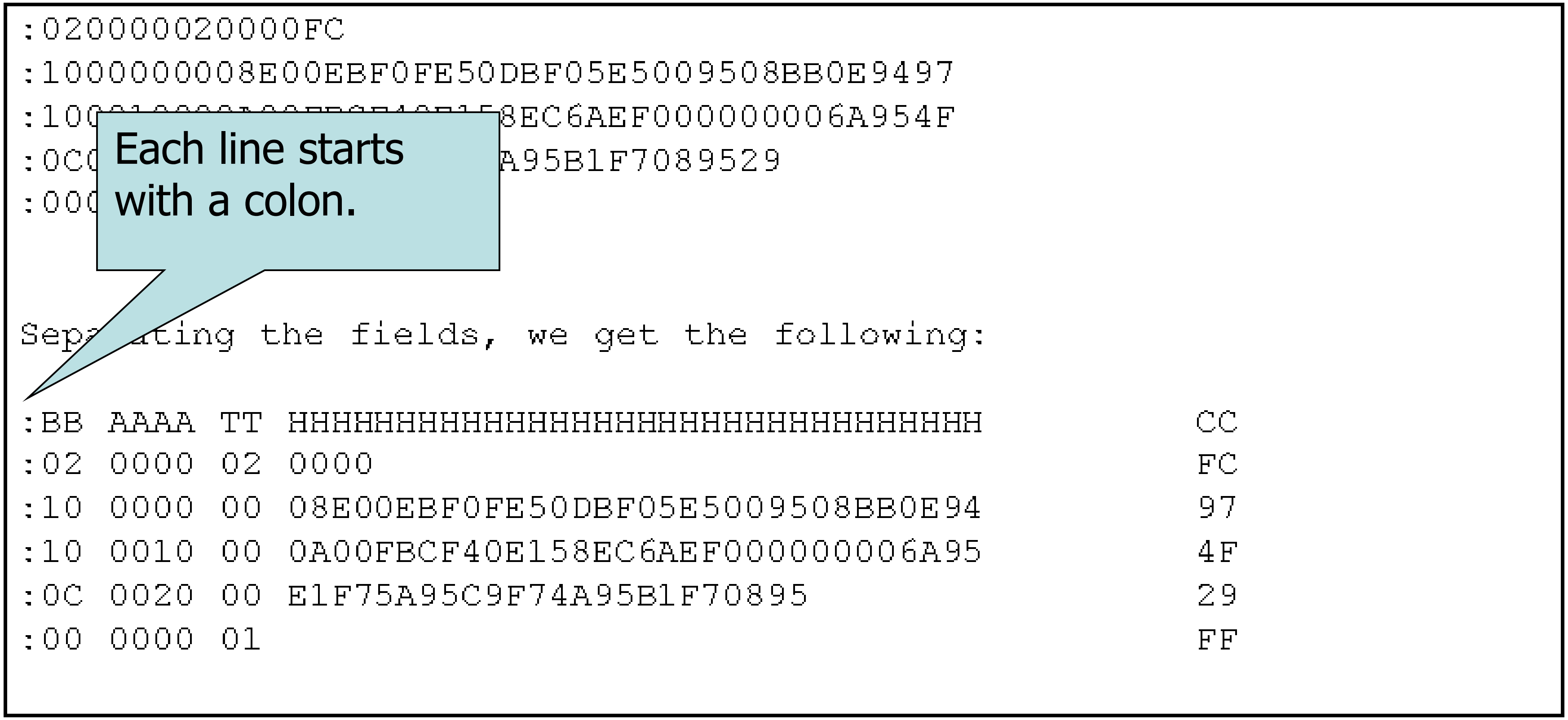
Inside an HEX file



```
:020000020000FC
:1000000008E00EBF0FE50DBF05E5009508BB0E9497
:100010000A00FBCF40E158EC6AEF000000006A954F
:0C002000E1F75A95C9F74A95B1F7089529
:00000001FF


Separating the fields, we get the following:

:BB  AAAA  TT  HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH          CC
:02  0000  02  0000                                     FC
:10  0000  00  08E00EBF0FE50DBF05E5009508BB0E94         97
:10  0010  00  0A00FBCF40E158EC6AEF000000006A95         4F
:0C  0020  00  E1F75A95C9F74A95B1F70895                 29
:00  0000  01                                           FF
```

Figure 8-7. Intel Hex File Test Program with the Intel Hex Option

Inside an HEX file

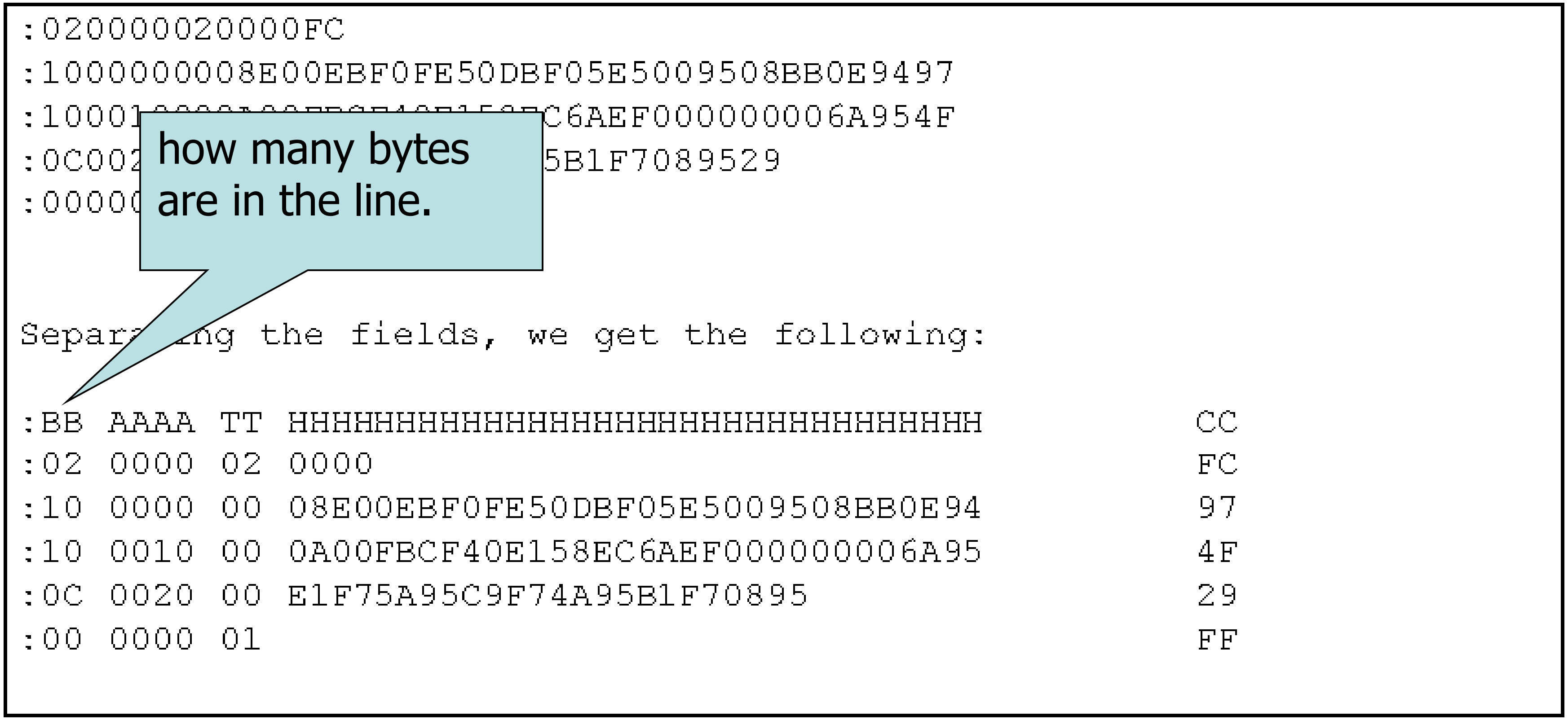


```
:020000020000FC
:1000000008E00EBF0FE50DBF05E5009508BB0E9497
:10                      8EC6AEF000000006A954F
:0C                      A95B1F7089529
:00


Separating the fields, we get the following:

:BB AAAA TT HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH          CC
:02 0000 02 0000                                      FC
:10 0000 00 08E00EBF0FE50DBF05E5009508BB0E94          97
:10 0010 00 0A00FBCF40E158EC6AEF000000006A95          4F
:0C 0020 00 E1F75A95C9F74A95B1F70895                  29
:00 0000 01                                           FF
```

Each line starts with a colon.

Figure 8-7. Intel Hex File Test Program with the Intel Hex Option

Inside an HEX file

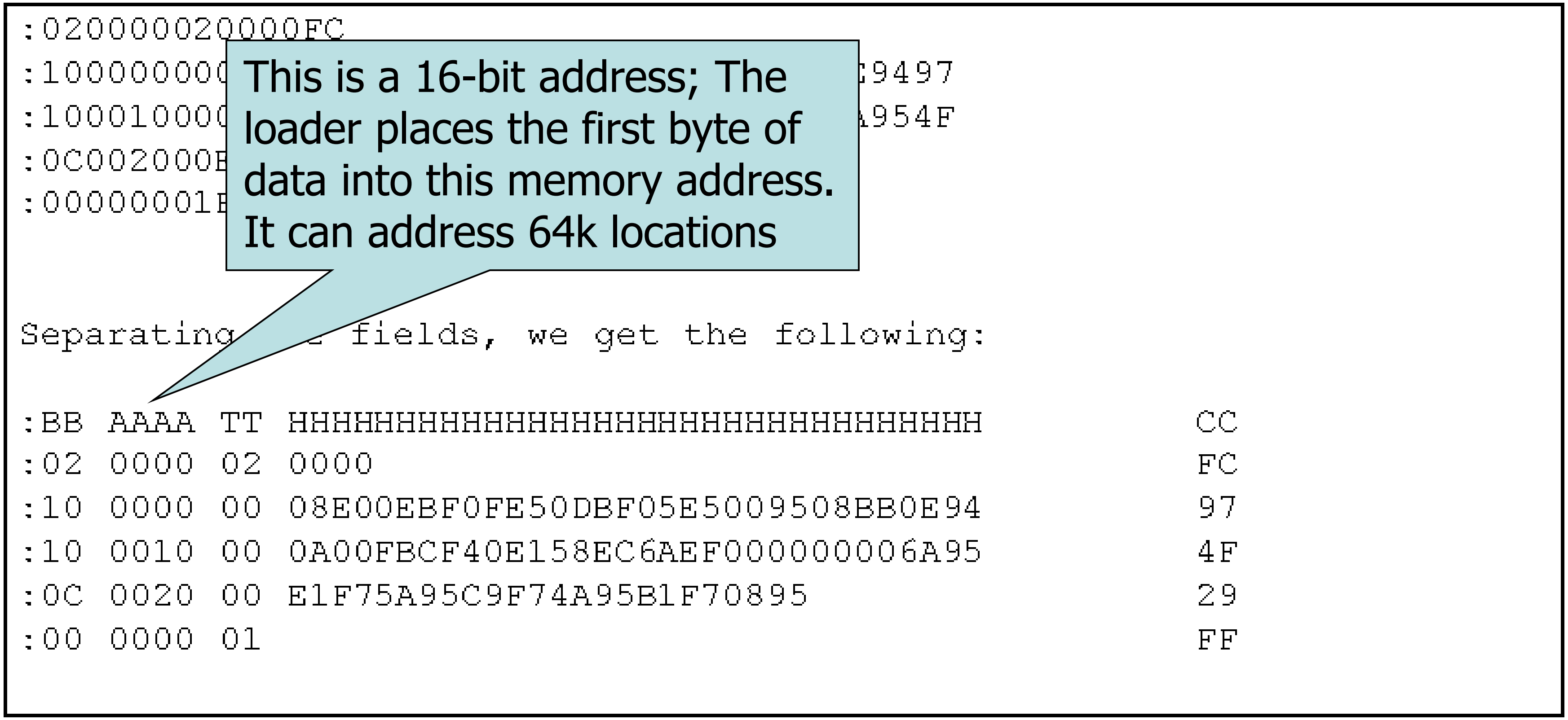


```
:020000020000FC
:1000000008E00EBF0FE50DBF05E5009508BB0E9497
:10001                    C6AEF000000006A954F
:0C002                    5B1F7089529
:00000


Separ    ng the fields, we get the following:

:BB AAAA TT HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH         CC
:02 0000 02 0000                                    FC
:10 0000 00 08E00EBF0FE50DBF05E5009508BB0E94        97
:10 0010 00 0A00FBCF40E158EC6AEF000000006A95        4F
:0C 0020 00 E1F75A95C9F74A95B1F70895                29
:00 0000 01                                         FF
```

how many bytes are in the line.

Figure 8-7. Intel Hex File Test Program with the Intel Hex Option

Inside an HEX file

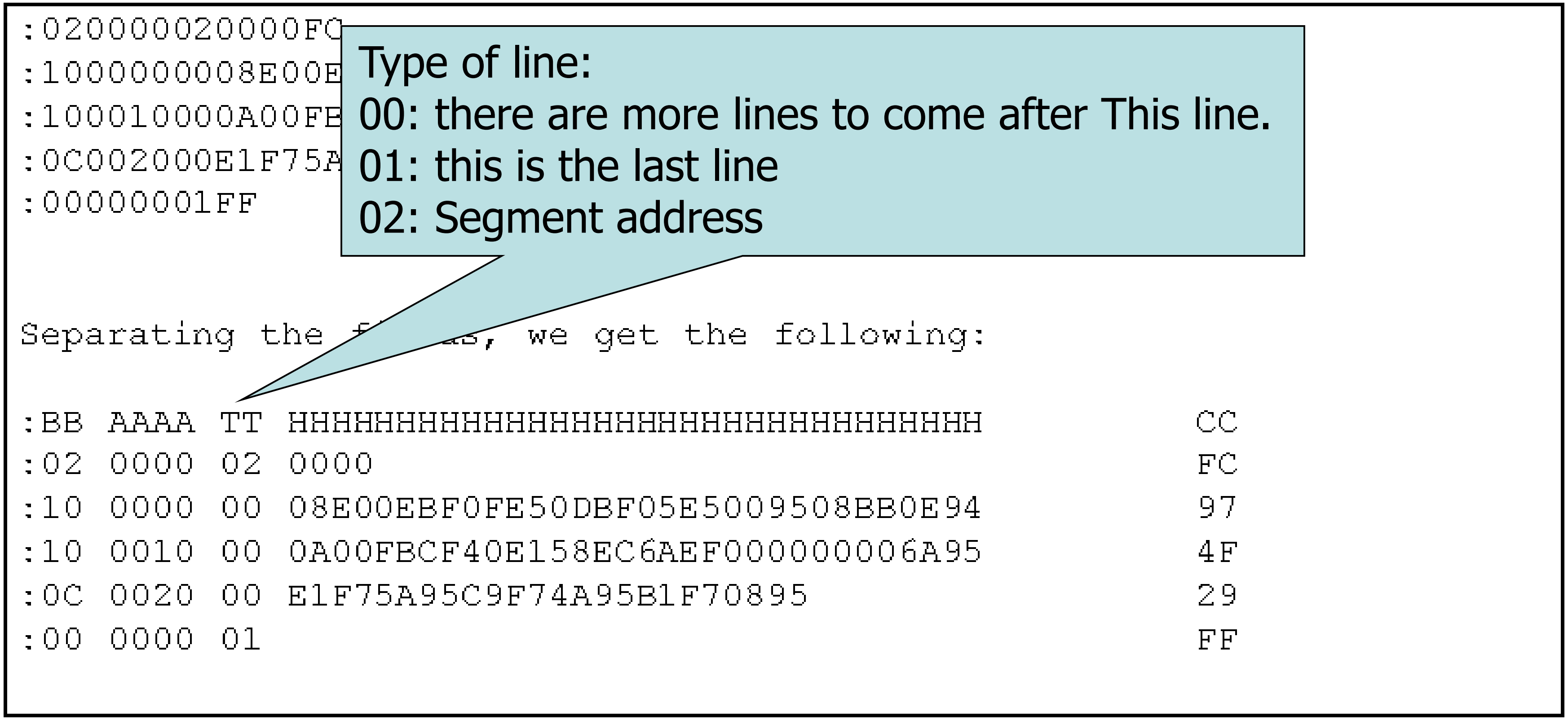


```
:020000020000FC
:100000000                              C9497
:100010000                              A954F
:0C002000E
:00000001F


Separating                fields, we get the following:

:BB AAAA TT HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH         CC
:02 0000 02 0000                                     FC
:10 0000 00 08E00EBF0FE50DBF05E5009508BB0E94         97
:10 0010 00 0A00FBCF40E158EC6AEF000000006A95         4F
:0C 0020 00 E1F75A95C9F74A95B1F70895                 29
:00 0000 01                                          FF
```

This is a 16-bit address; The loader places the first byte of data into this memory address. It can address 64k locations

Figure 8-7. Intel Hex File Test Program with the Intel Hex Option

Inside an HEX file

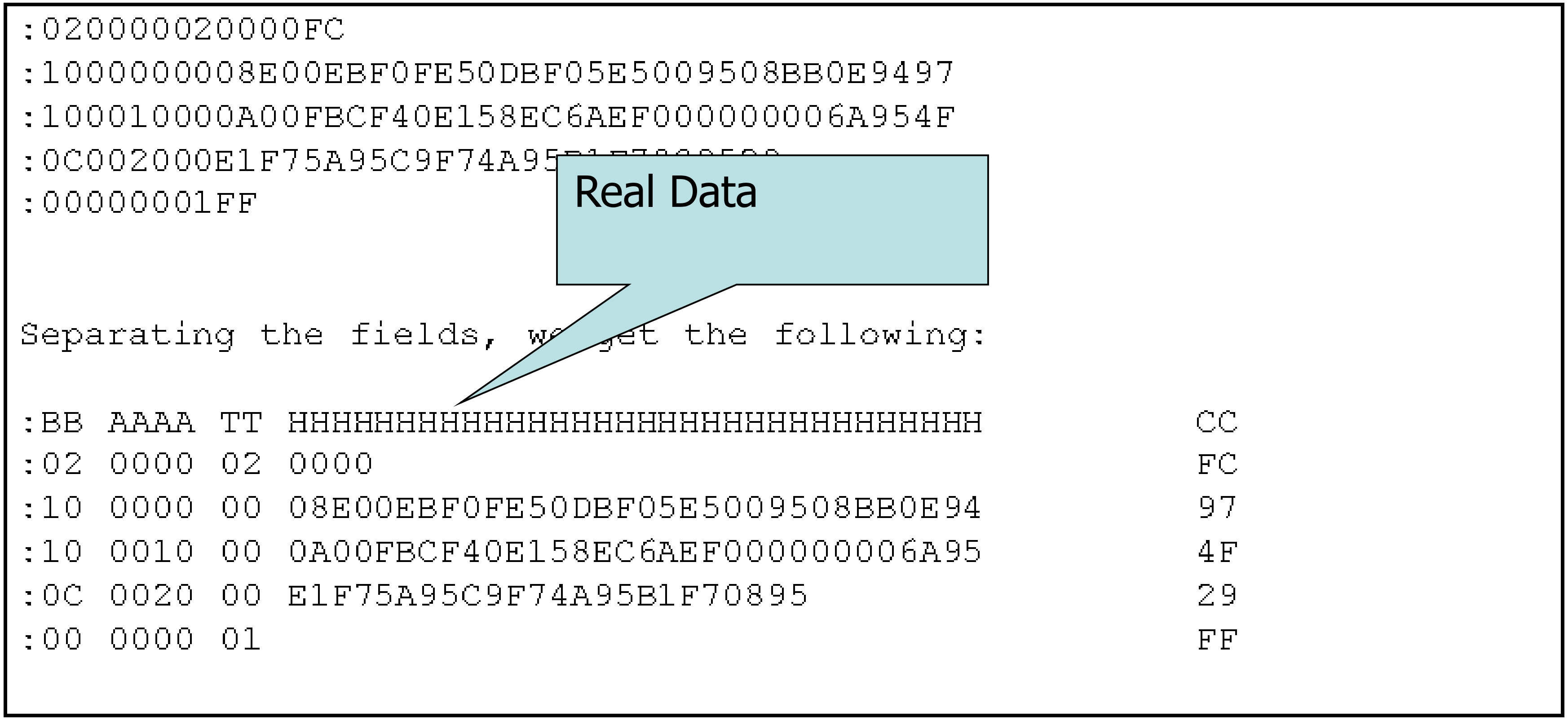


```
:020000020000FC
:1000000008E00E
:100010000A00FB
:0C002000E1F75A
:00000001FF


Separating the fields, we get the following:

:BB AAAA TT HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH          CC
:02 0000 02 0000                                      FC
:10 0000 00 08E00EBF0FE50DBF05E5009508BB0E94          97
:10 0010 00 0A00FBCF40E158EC6AEF000000006A95          4F
:0C 0020 00 E1F75A95C9F74A95B1F70895                  29
:00 0000 01                                           FF
```

Type of line:
00: there are more lines to come after This line.
01: this is the last line
02: Segment address

Figure 8-7. Intel Hex File Test Program with the Intel Hex Option

Inside an HEX file

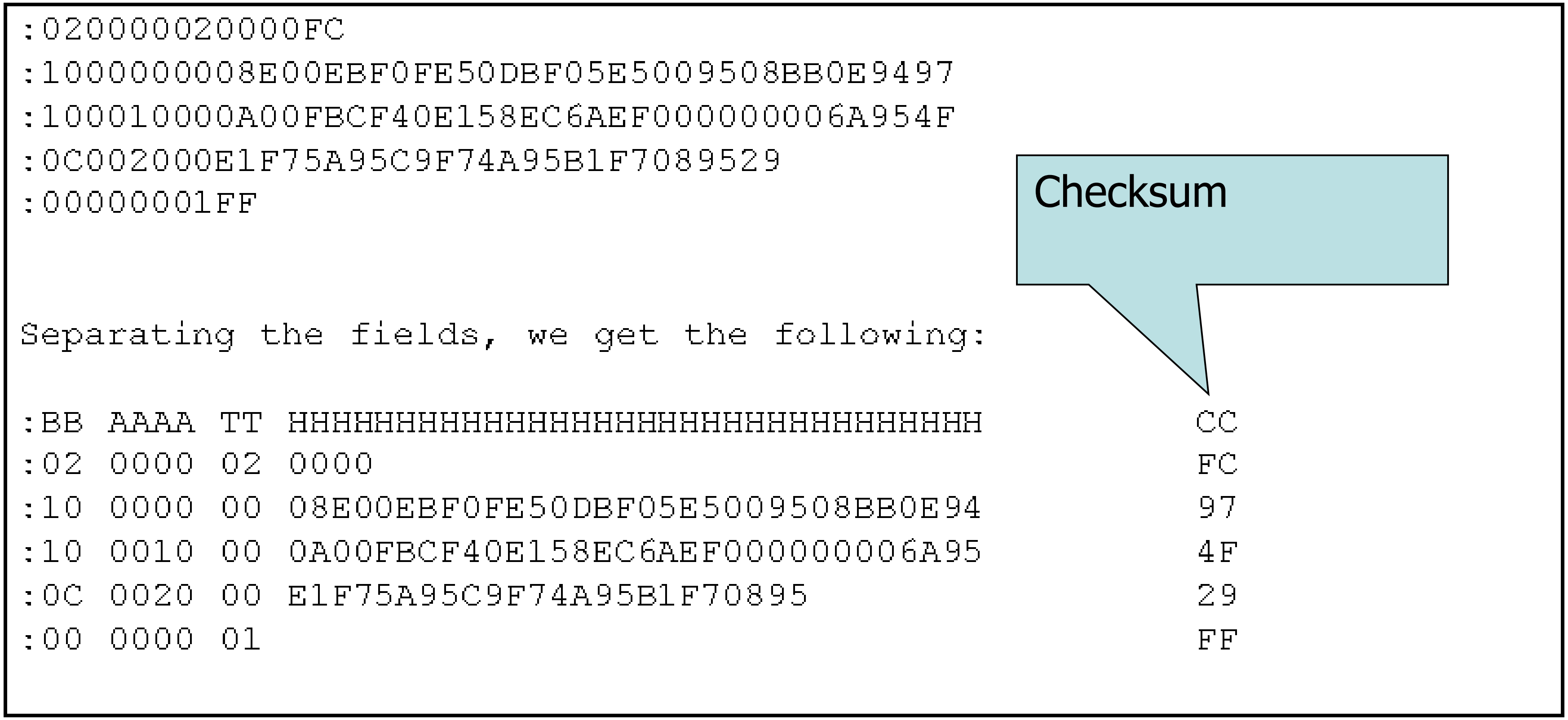


```
:020000020000FC
:1000000008E00EBF0FE50DBF05E5009508BB0E9497
:100010000A00FBCF40E158EC6AEF000000006A954F
:0C002000E1F75A95C9F74A95
:00000001FF


Separating the fields, we get the following:

:BB AAAA TT HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH           CC
:02 0000 02 0000                                       FC
:10 0000 00 08E00EBF0FE50DBF05E5009508BB0E94           97
:10 0010 00 0A00FBCF40E158EC6AEF000000006A95           4F
:0C 0020 00 E1F75A95C9F74A95B1F70895                   29
:00 0000 01                                            FF
```

Real Data

Figure 8-7. Intel Hex File Test Program with the Intel Hex Option

Inside an HEX file



```
:020000020000FC
:1000000008E00EBF0FE50DBF05E5009508BB0E9497
:100010000A00FBCF40E158EC6AEF000000006A954F
:0C002000E1F75A95C9F74A95B1F7089529
:00000001FF


Separating the fields, we get the following:

:BB AAAA TT HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH          CC
:02 0000 02 0000                                      FC
:10 0000 00 08E00EBF0FE50DBF05E5009508BB0E94          97
:10 0010 00 0A00FBCF40E158EC6AEF000000006A95          4F
:0C 0020 00 E1F75A95C9F74A95B1F70895                  29
:00 0000 01                                           FF
```

Checksum

Figure 8-7. Intel Hex File Test Program with the Intel Hex Option

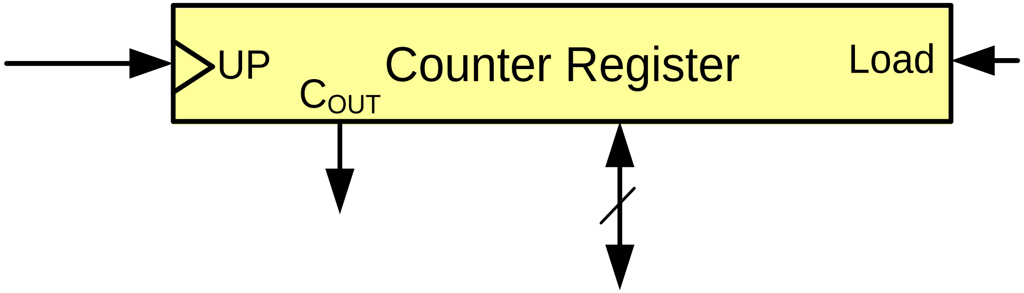
AVR Programming

- Parallel programming
- ISP
  - SPI
  - JTAG
- Boot loader

Timer/counter

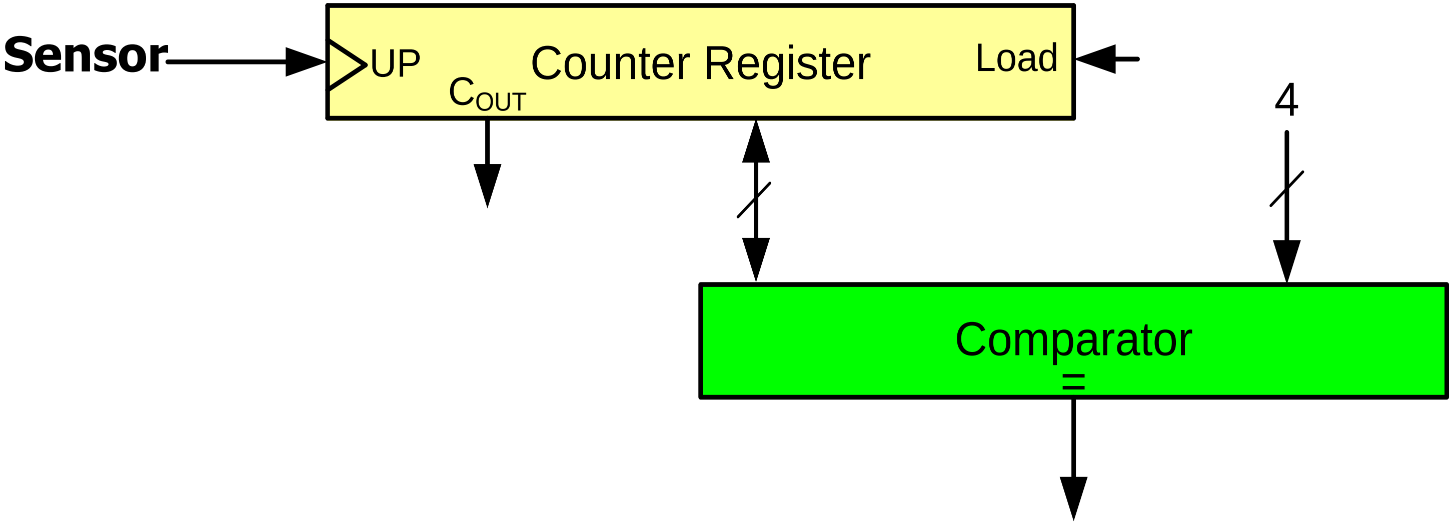
The AVR microcontroller
and embedded
systems
using assembly and c

A counter register



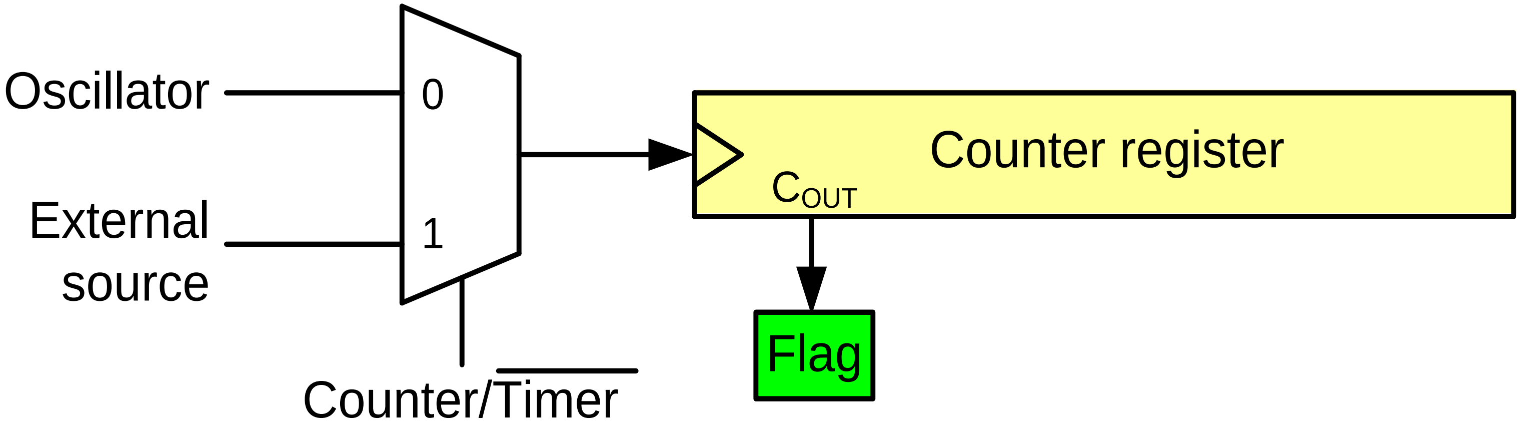
A simple design (counting people)

First design



A generic timer/counter

- Delay generating
- Counting
- Wave-form generating
- Capturing

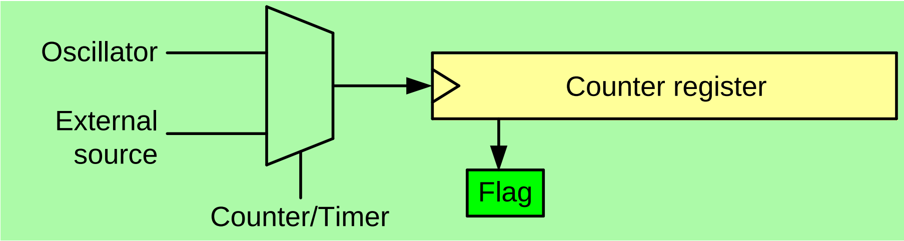


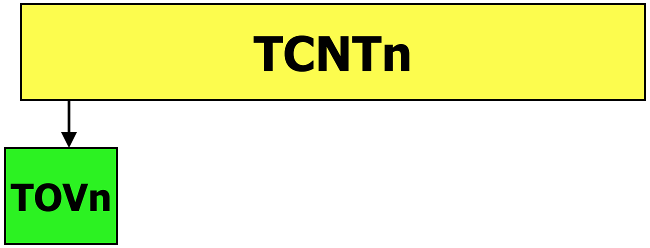
Timers in AVR

- 1 to 6 timers
  - 3 timers in ATmega32
- 8-bit and 16-bit timers
  - two 8-bit timers and one 16-bit timer in ATmega32

Timer in AVR

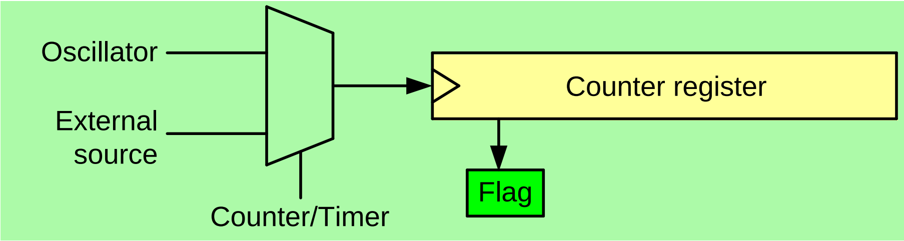
- TCNTn (Timer/Counter register)
- TOVn (Timer Overflow flag)
- TCCRn (Timer Counter control register)
- OCRn (output compare register)
- OCFn (output compare match flag)

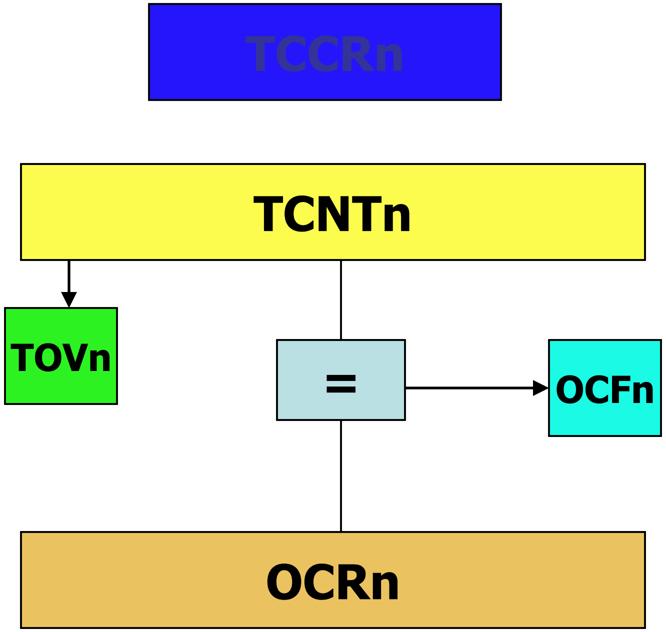




Timer in AVR

- TCNTn (Timer/Counter register)
- TOVn (Timer Overflow flag)
- TCCRn (Timer Counter control register)
- OCRn (output compare register)
- OCFn (output compare match flag)





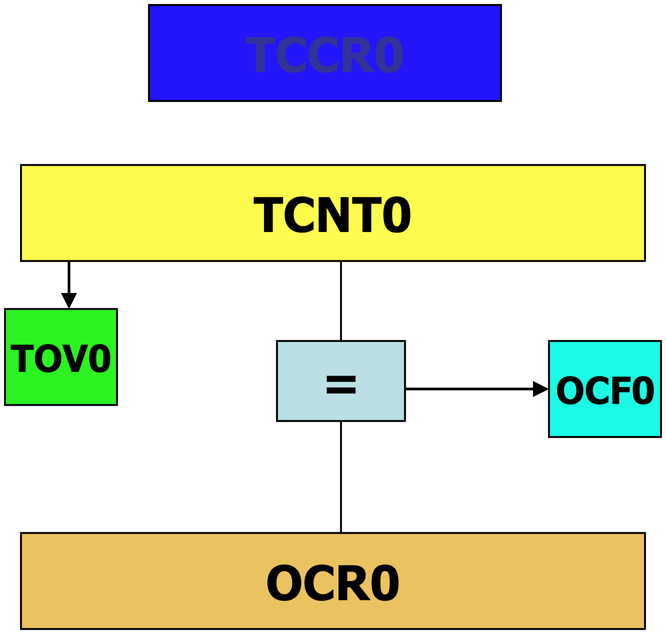
Comment:

All of the timer registers are byte-addressable I/O registers

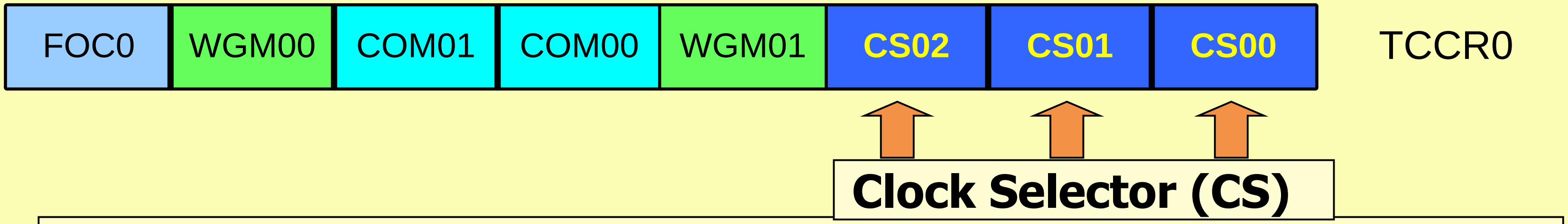
Timer 0 (an 8-bit timer)

The AVR microcontroller
and embedded
systems
using assembly and c

Timer 0



OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR



FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0

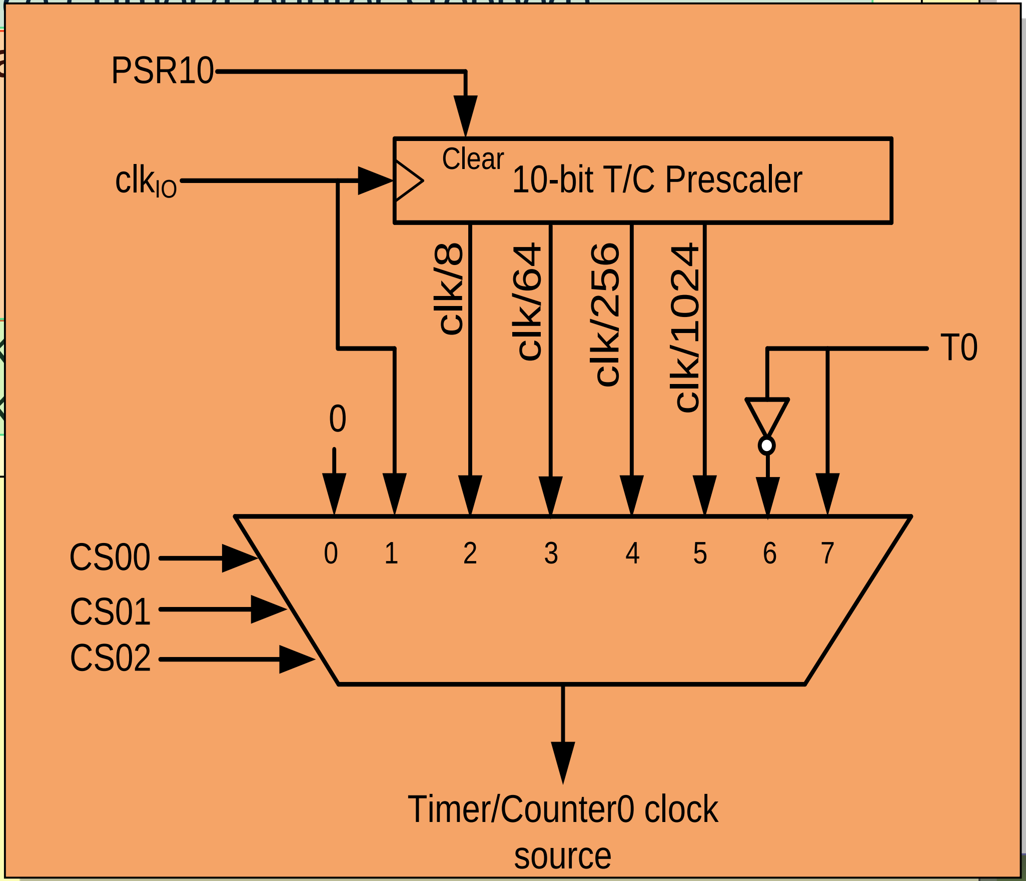
Clock Selector (CS)

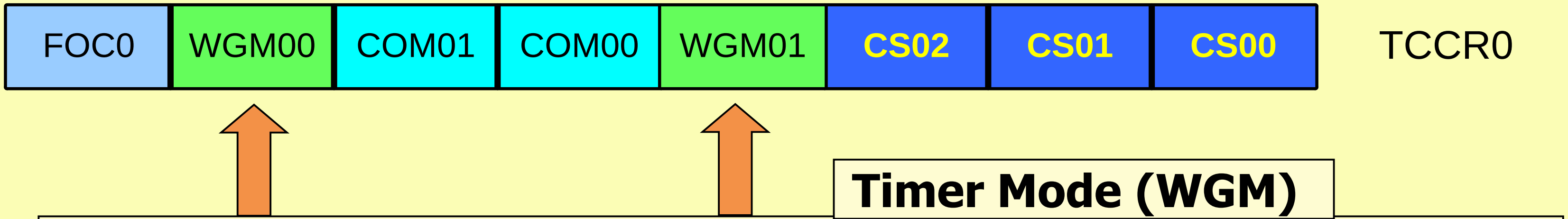
CS02	CS01	CS00	Comment
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 64
1	0	0	clk / 256
1	0	1	clk / 1024
1	1	0	External clock source on T0 pin. Clock on falling edge
1	1	1	External clock source on T0 pin. Clock on rising edge

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0

Clock Selector (CS)

CS02	CS01	CS00	Comment
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk (No Presca
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
1	1	1	External clock



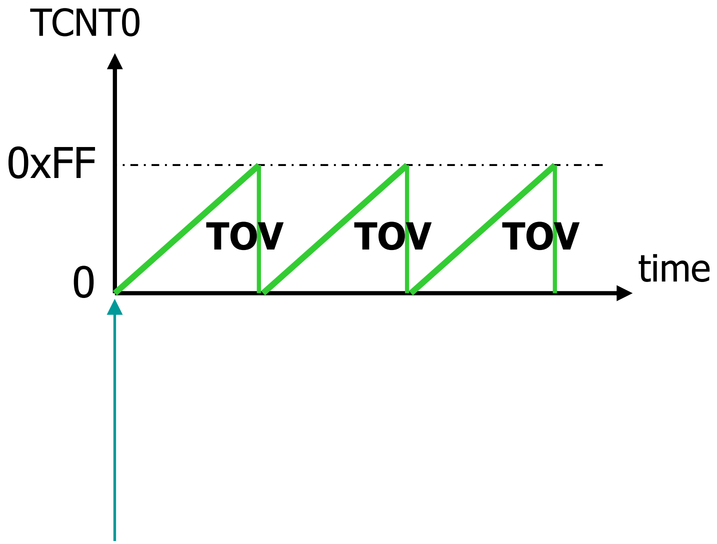


FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0

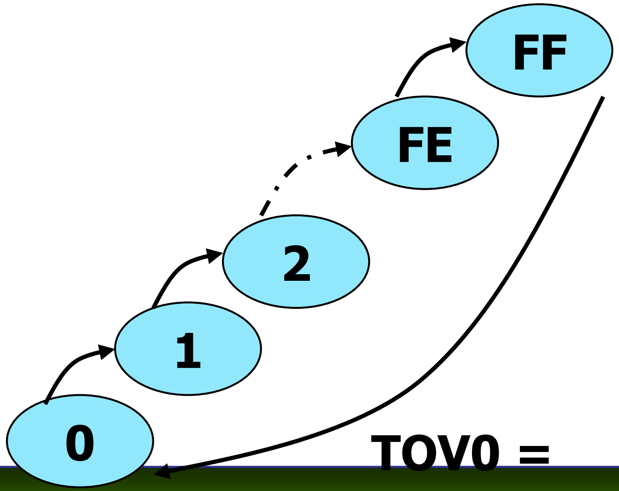
Timer Mode (WGM)

WGM00	WGM01	Comment
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

Normal mode

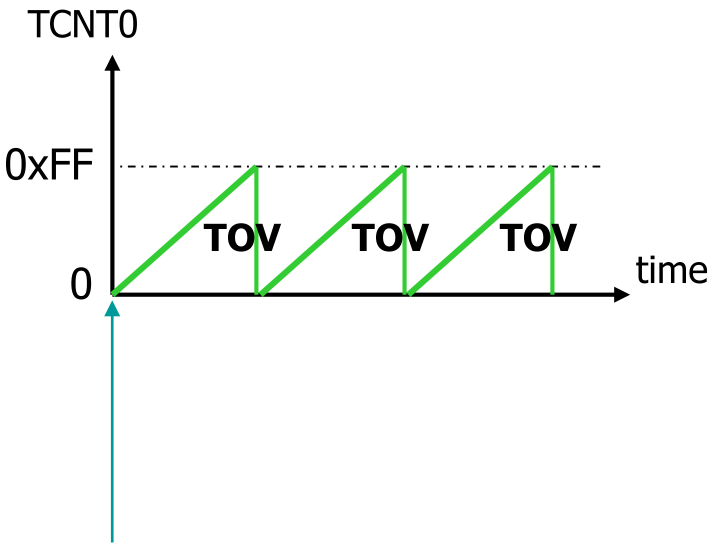


TOV0: 0

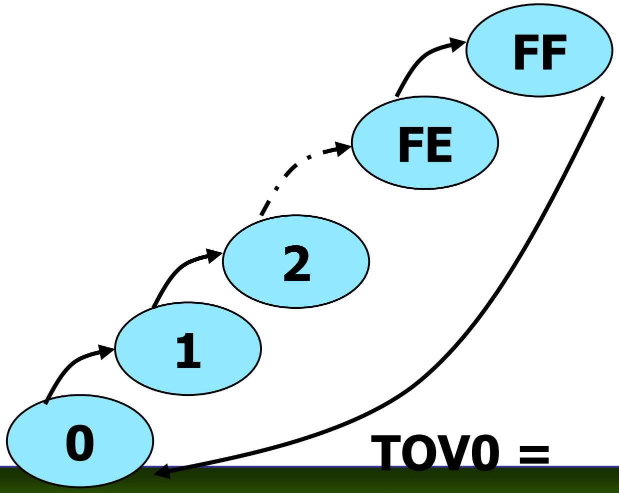


TOV0 = 1

Normal mode



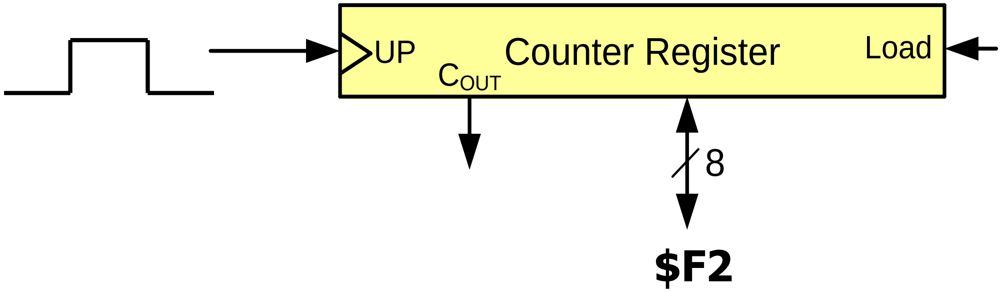
TOV0: 1



TOV0 = 1

Example 1: Write a program that waits 14 machine cycles in Normal mode.

```
14 = $0E

 $100
-$0E
-----
 $F2
```

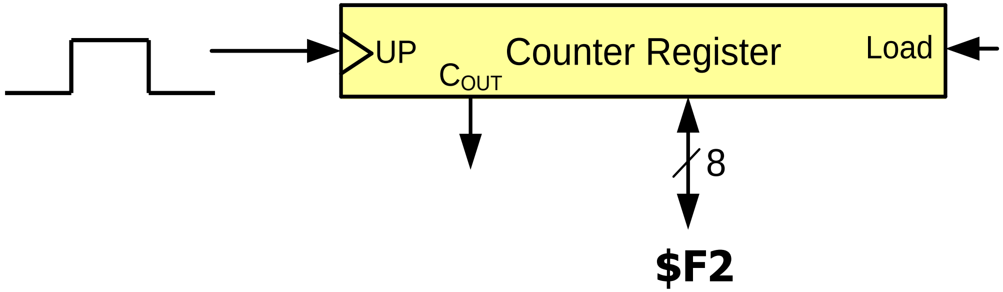


FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0

Example 1: Write a program that waits 14 machine cycles in Normal mode.

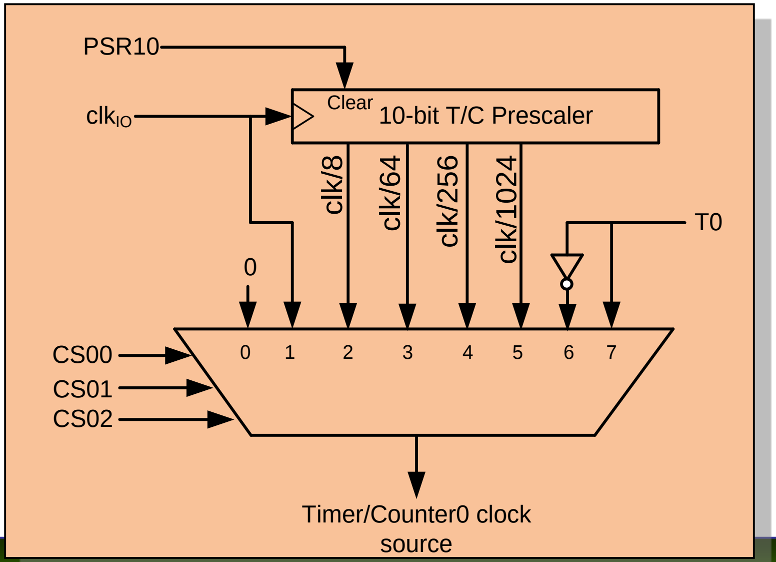
14 = $0E

```
 $100
-$0E
-----
 $F2
```



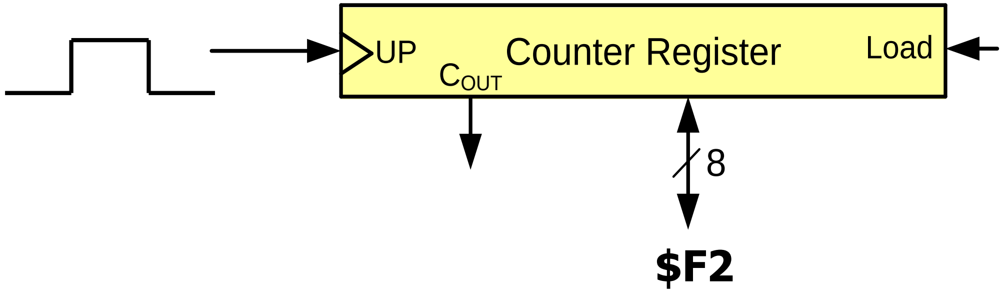
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
					0	0	1

TCCR0



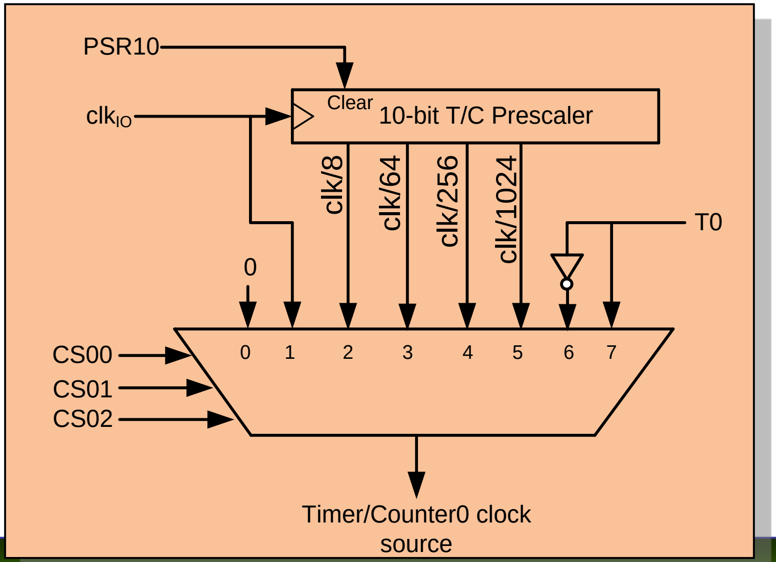
Example 1: Write a program that waits 14 machine cycles in Normal mode.

```
14 = $0E

 $100
 -$0E
 ----
 $F2
```



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

WGM00	WGM01	Comment
0	0	Normal
0	1	CTC
1	0	PWM, phase correct
1	1	Fast PWM



Example 1: write a program that waits 14 machine cycles in Normal mode.

```
  $100
 -$0E
 -----
  $F2
```

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR

```
.INCLUDE "M32DEF.INC"

        LDI     R16,0x20
        SBI     DDRB,5   ;PB5 as an output
        LDI     R17,0
        OUT     PORTB,R17
BEGIN:  LDI     R20,0xF2
        OUT     TCNT0,R20         ;load timer0
        LDI     R20,0x01
        OUT     TCCR0,R20  ;Timer0,Normal mode,int clk
AGAIN:  IN      R20,TIFR          ;read TIFR
        SBRS    R20,0  ;if TOV0 is set skip next inst.
        RJMP    AGAIN
        LDI     R20,0x0
        OUT     TCCR0,R20         ;stop Timer0
        LDI     R20,(1<<TOV0)     ;R20 = 0x01
        OUT     TIFR,R20          ;clear TOV0 flag

        EOR     R17,R16           ;toggle D5 of R17
        OUT     PORTB,R17         ;toggle PB5
        RJMP    BEGIN
```

```
DDRB = 1<<5;
PORTB &= ~(1<<5);   //PB5=0
while (1)
{
  TCNT0 = 0xF2;
  TCCR0 = 0x01;
 while((TIFR&(1<<TOV0))==0);
  TCCR0 = 0;
  TIFR = (1<<TOV0);
  PORTB = PORTB^(1<<5);
}
```

Example 1: write a program that waits 14 machine cycles in Normal mode.

```
 $100
-$0E
-----
 $F2
```

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR

```
.INCLUDE "M32DEF.INC"

         LDI      R16,0x20
         SBI      DDRB,5   ;PB5 as an output
         LDI      R17,0
         OUT      PORTB,R17
BEGIN:   LDI      R20,0xF2
         OUT      TCNT0,R20        ;load timer0
         LDI      R20,0x01
         OUT      TCCR0,R20  ;Ti
AGAIN:   IN       R20,TIFR
         SBRS     R20,0 ;if TOV
         RJMP     AGAIN
         LDI      R20,0x0
         OUT      TCCR0,R20
         LDI      R20,(1<<TOV0)
         OUT      TIFR,R20

         EOR      R17,R16
         OUT      PORTB,R17
         RJMP     BEGIN
```

```
DDRB = 1<<5;
PORTB &= ~(1<<5);  //PB5=0
while (1)
{
   TCNT0 = 0xF2;
```

Question: How to calculate the delay generated by the timer?

Answer:

1) Calculate how much a machine clock lasts.
   T= 1/f
2) Calculate how many machine clocks it waits.
3) Delay = T * number of machine cycles

In example 1 calculate the delay. Imagine XTAL = 10 MHz.

Solution 1 (inaccurate)**:**

1) **Calculating T:**

T = 1/f = 1/10M = 0.1µs

2) **Calculating num of machine cycles:**

$$\begin{array}{r} \$100 \\ -\$F2 \\ \hline \$0E \; = \; 14 \end{array}$$

3) **Calculating delay**

14 * 0.1µs = 1.4 0µs

```
.INCLUDE "M32DEF.INC"

        LDI     R16,0x20
        SBI     DDRB,5   ;PB5 as an output
        LDI     R17,0
        OUT     PORTB,R17
BEGIN:  LDI     R20,0xF2
        OUT     TCNT0,R20         ;load timer0
        LDI     R20,0x01
        OUT     TCCR0,R20 ;Timer0,Normal mode,int clk
AGAIN:  IN      R20,TIFR          ;read TIFR
        SBRS    R20,0 ;if TOV0 is set skip next inst.
        RJMP    AGAIN
        LDI     R20,0x0
        OUT     TCCR0,R20         ;stop Timer0
        LDI     R20,0x01
        OUT     TIFR,R20          ;clear TOV0 flag

        EOR     R17,R16           ;toggle D5 of R17
        OUT     PORTB,R17         ;toggle PB5
        RJMP    BEGIN
```

Accurate calculating

Other than timer, executing the instructions consumes time; so if we want to calculate the accurate delay a program causes we should add the delay caused by instructions to the delay caused by the timer

```
        LDI     R16,0x20
        SBI     DDRB,5
        LDI     R17,0
        OUT     PORTB,R17
BEGIN:  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
        EOR     R17,R16                     1
        OUT     PORTB,R17                   1
        RJMP    BEGIN                       2
                                            18
```

Delay caused by timer = 14 * 0.1µs = 1.4 µs **Delay caused by instructions = 18 * 0.1µs = 1.8**

Total delay = 3.2 µs ➔ wave period = 2*3.2 µs = 6.4 µs ➔ wave frequency = 156.25 KHz

Finding values to be loaded into the timer

1. Calculate the period of clock source.
   - Period = 1 / Frequency
     - E.g. For XTAL = 8 MHz → T = 1/8MHz
2. Divide the desired time delay by period of clock.
3. Perform 256 - n, where n is the decimal value we got in Step 2.
4. Set TCNT0 = 256 - n

Example 2: Assuming that XTAL = 10 MHz, write a program to generate a square wave with a period of 10 ms on pin PORTB.3.

- For a square wave with T = 10 µs we must have a time delay of 5 µs. Because XTAL = 10 MHz, the counter counts up every 0.1 µs. This means that we need 5 µs / 0.1 µs = 50 clocks. 256 - 50 = 206.

```
.INCLUDE "M32DEF.INC"

         LDI      R16,0x08
         SBI      DDRB,3   ;PB3 as an output
         LDI      R17,0
         OUT      PORTB,R17
BEGIN:   LDI      R20,206
         OUT      TCNT0,R20        ;load timer0
         LDI      R20,0x01
         OUT      TCCR0,R20 ;Timer0,Normal mode,int clk
AGAIN:   IN       R20,TIFR         ;read TIFR
         SBRS     R20,TOV0 ;if TOV0 is set skip next
         RJMP     AGAIN
         LDI      R20,0x0
         OUT      TCCR0,R20        ;stop Timer0
         LDI      R20,0x01
         OUT      TIFR,R20         ;clear TOV0 flag
         EOR      R17,R16          ;toggle D3 of R17
         OUT      PORTB,R17        ;toggle PB3
         RJMP     BEGIN
```

```
DDRB = 1<<3;
PORTB &= ~ (1<<3);
while (1)
{
  TCNT0 = 206;
  TCCR0 = 0x01;
  while((TIFR&0x01) == 0);
  TCCR0 = 0;
  TIFR = 1<<TOV0;
  PORTB = PORTB ^ (1<<3);
}
```

Example 3: Modify TCNT0 in Example 2 to get the largest time delay possible with no prescaler. Find the delay in µs. In your calculation, do not include the overhead due to instructions.

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

```
.INCLUDE "M32DEF.INC"

         LDI      R16,1<<3
         SBI      DDRB,3     ;PB3 as an output
         LDI      R17,0
         OUT      PORTB,R17
BEGIN:   LDI      R20,0x0
         OUT      TCNT0,R20        ;load Timer0
         LDI      R20,0x01
         OUT      TCCR0,R20 ;Timer0,Normal mode,int clk
AGAIN:   IN       R20,TIFR         ;read TIFR
         SBRS     R20,TOV0  ;if TOV0 is set skip next
         RJMP     AGAIN
         LDI      R20,0x0
         OUT      TCCR0,R20        ;stop Timer0
         LDI      R20,0x01
         OUT      TIFR,R20         ;clear TOV0 flag
         EOR      R17,R16          ;toggle D3 of R17
         OUT      PORTB,R17        ;toggle PB3
         RJMP     BEGIN
```

```
DDRB = 1 << 3;
PORTB &= ~(1<<3);
while (1)
{
  TCNT0 = 0x0;
  TCCR0 = 0x01;

while((TIFR&(1<<TOV0))==0);
  TCCR0 = 0;
  TIFR = 0x01;
  PORTB = PORTB^(1<<3);
}
```

Example 3: Modify TCNT0 in Example 2 to get the largest time delay possible with no prescaler. Find the delay in µs. In your calculation, do not include the overhead due to instructions.

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

```
.INCLUDE "M32DEF.INC"

        LDI     R16,1<<3
        SBI     DDRB,3     ;PB3 as an output
        LDI     R17,0
        OUT     PORTB,R17
BEGIN:  LDI     R20,0x0
        OUT     TCNT0,R20         ;load Timer0
        LDI     R20,0x01
        OUT     TCCR0,R20  ;Timer0,Normal mode,int clk
AGAIN:  IN      R20,TI
        SBRS    R20,TO
        RJMP    AGAIN
        LDI     R20,0x
        OUT     TCCR0,
        LDI     R20,0x
        OUT     TIFR,R
        EOR     R17,R1
        OUT     PORTB,
        RJMP    BEGIN
```

```
DDRB = 1 << 3;
PORTB &= ~(1<<3);
while (1)
{
  TCNT0 = 0x0;
  TCCR0 = 0x01;

while((TIFR&(1<<TOV0))==0);
  TCCR0 = 0;
  TIFR = 0x01;
  PORTB = PORTB^(1<<3);
}
```

Solution

1) Calculating T:

T = 1/f = 1/10MHz = 0.1µs

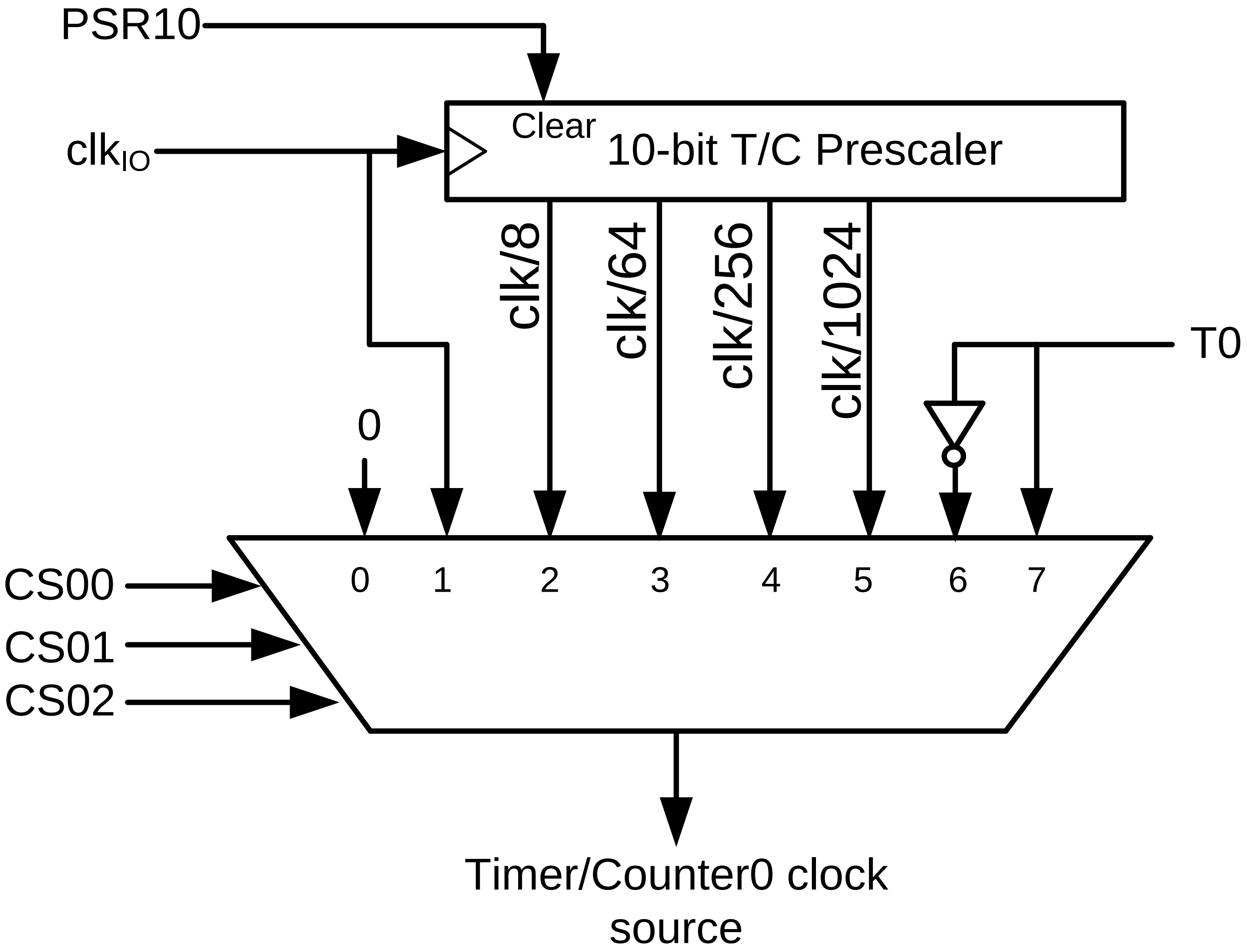
2) Calculating delay

256 * 0.1µs = 25.6µs

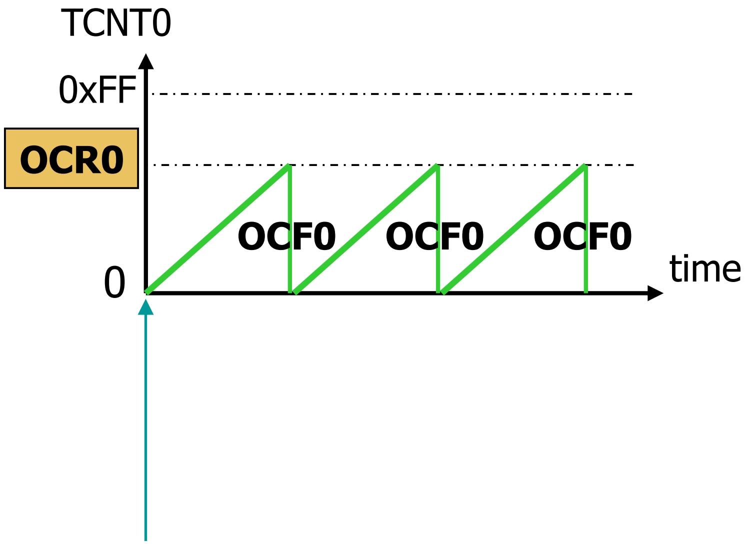
Generating Large Delays

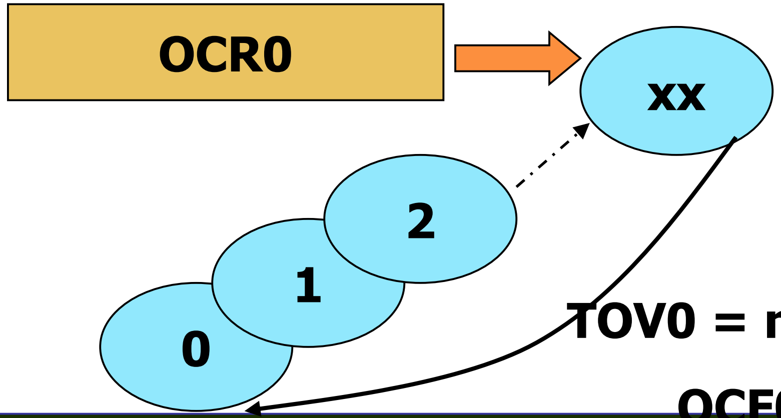
- Using loop
- Prescaler
- Bigger counters

Prescaler and generating a large time delay



CTC (Clear Timer on Compare match) mode





TOV0 = no change

OCF0 = 1

TOV0: 0

OCF0: 1

Rewrite example 2 using CTC

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR

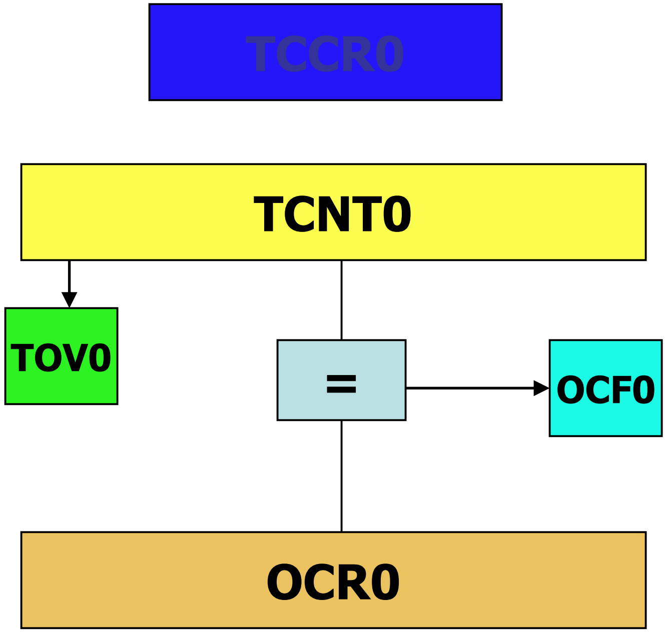
- For a square wave with T = 10 µs we must have a time delay of 5 µs. Because XTAL = 10 MHz, the counter counts up every 0.1 µs. This means that we need 5 µs / 0.1 µs = 50 clocks. Therefore, we have OCR0= 49.

```
.INCLUDE "M32DEF.INC"
        LDI     R16,0x08
        SBI     DDRB,3   ;PB3 as an output
        LDI     R17,0
        OUT     PORTB,R17
        LDI     R20,49
        OUT     OCR0,R20 ;load timer0
BEGIN:  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
        RJMP    AGAIN
        LDI     R20,0x0
        OUT     TCCR0,R20         ;stop Timer0
        LDI     R20,0x02
        OUT     TIFR,R20          ;clear TOV0 flag
        EOR     R17,R16           ;toggle D3 of R17
        OUT     PORTB,R17         ;toggle PB3
        RJMP    BEGIN
```

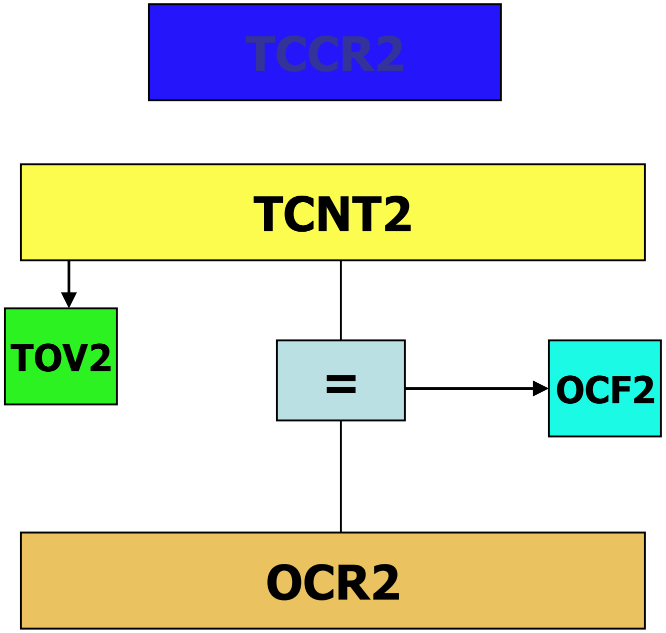
```
DDRB |= 1<<3;
PORTB &= ~(1<<3);
while (1)
{
   OCR0 = 49;
   TCCR0 = 0x09;

while((TIFR&(1<<OCF0))==0);
   TCCR0 = 0; //stop timer0
   TIFR = 0x02;
   PORTB.3 = ~PORTB.3;
}
```

Timer2

- Timer0



- Timer2



OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR

The difference between Timer0 and Timer2

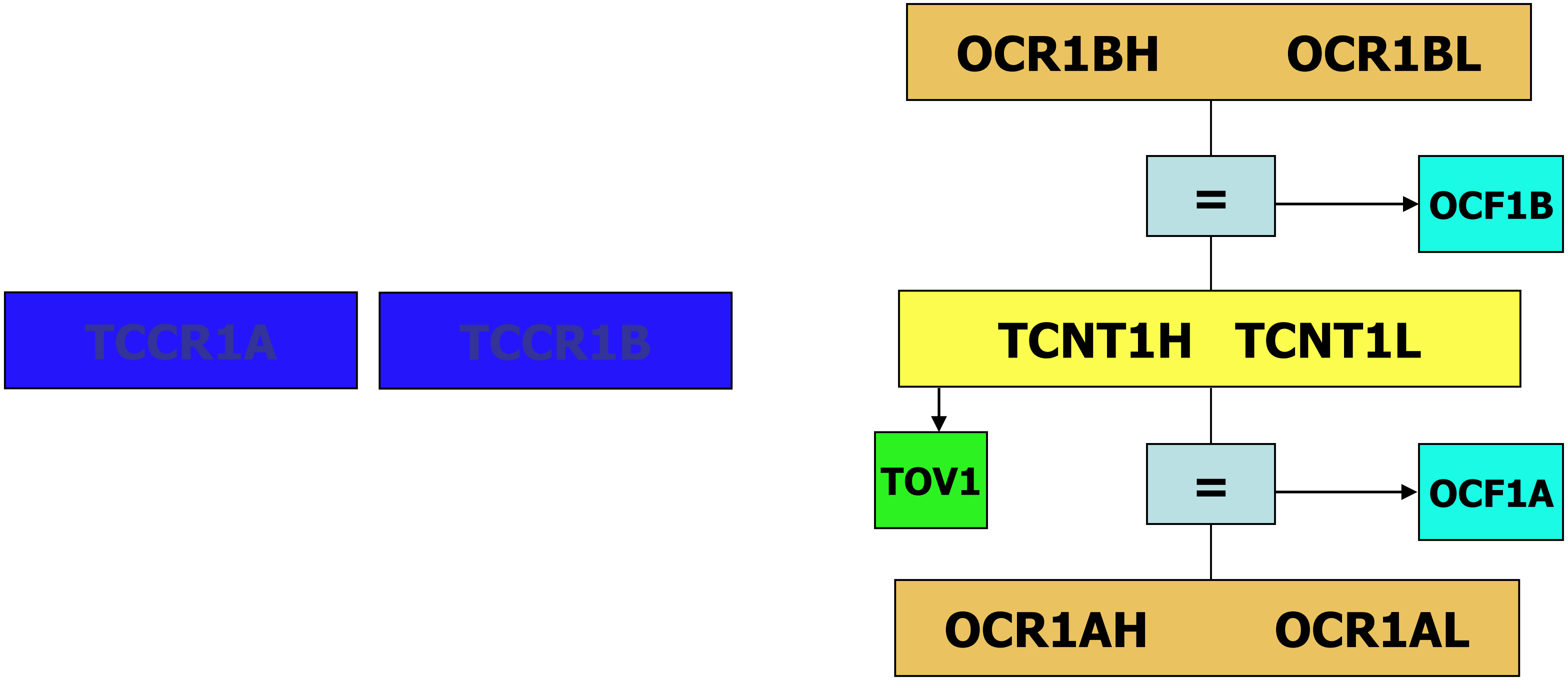
- Timer0

CS02	CS01	CS00	Comment
0	0	0	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 (falling edge)
1	1	1	External clock (rising edge)

- Timer2

CS22	CS21	CS20	Comment
0	0	0	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

Timer 1



OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

TIFR

COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B

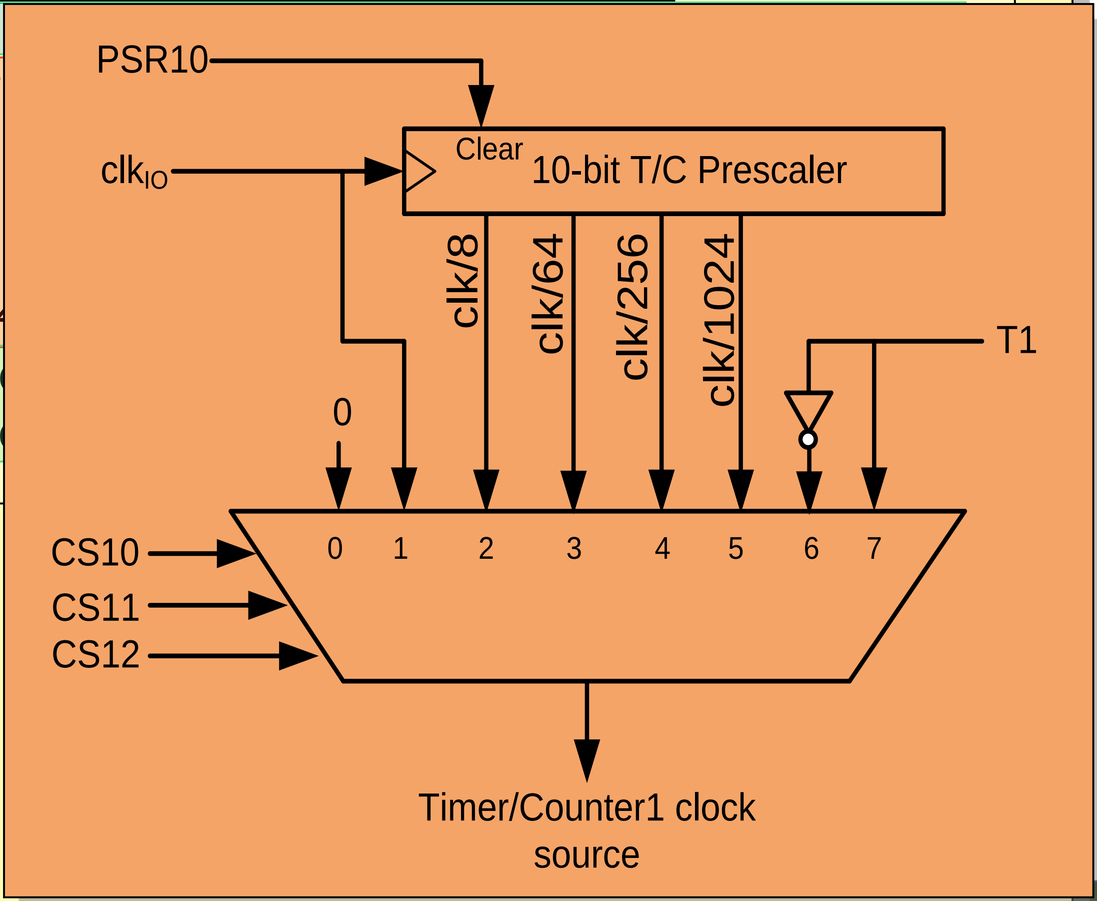
Clock Selector (CS)

CS12	CS11	CS10	Comment
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

COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B

Clock Selector (CS)

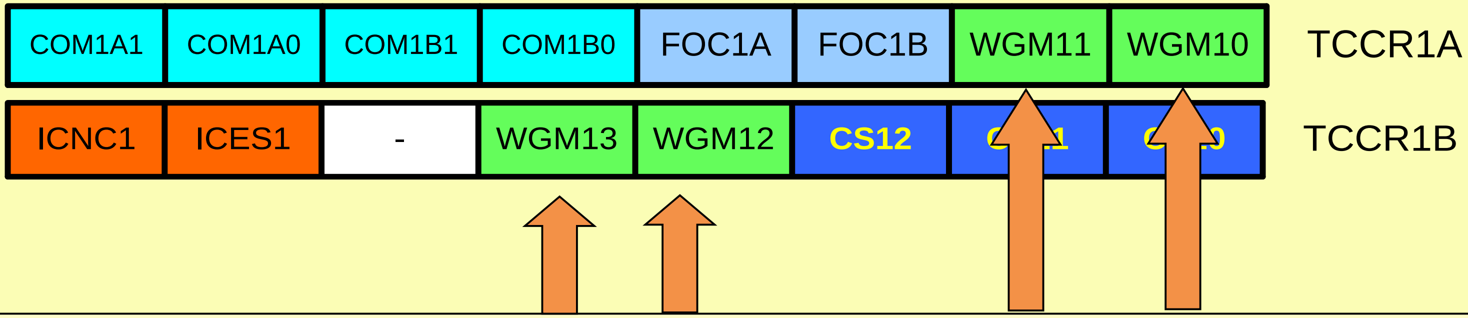
CS12	CS11	CS10	Comment
0	0	0	No clock
0	0	1	clk (No P
0	1	0	clk / 8
0	1	1	clk / 64
1	0	0	clk / 256
1	0	1	clk / 102
1	1	0	External c
1	1	1	External c



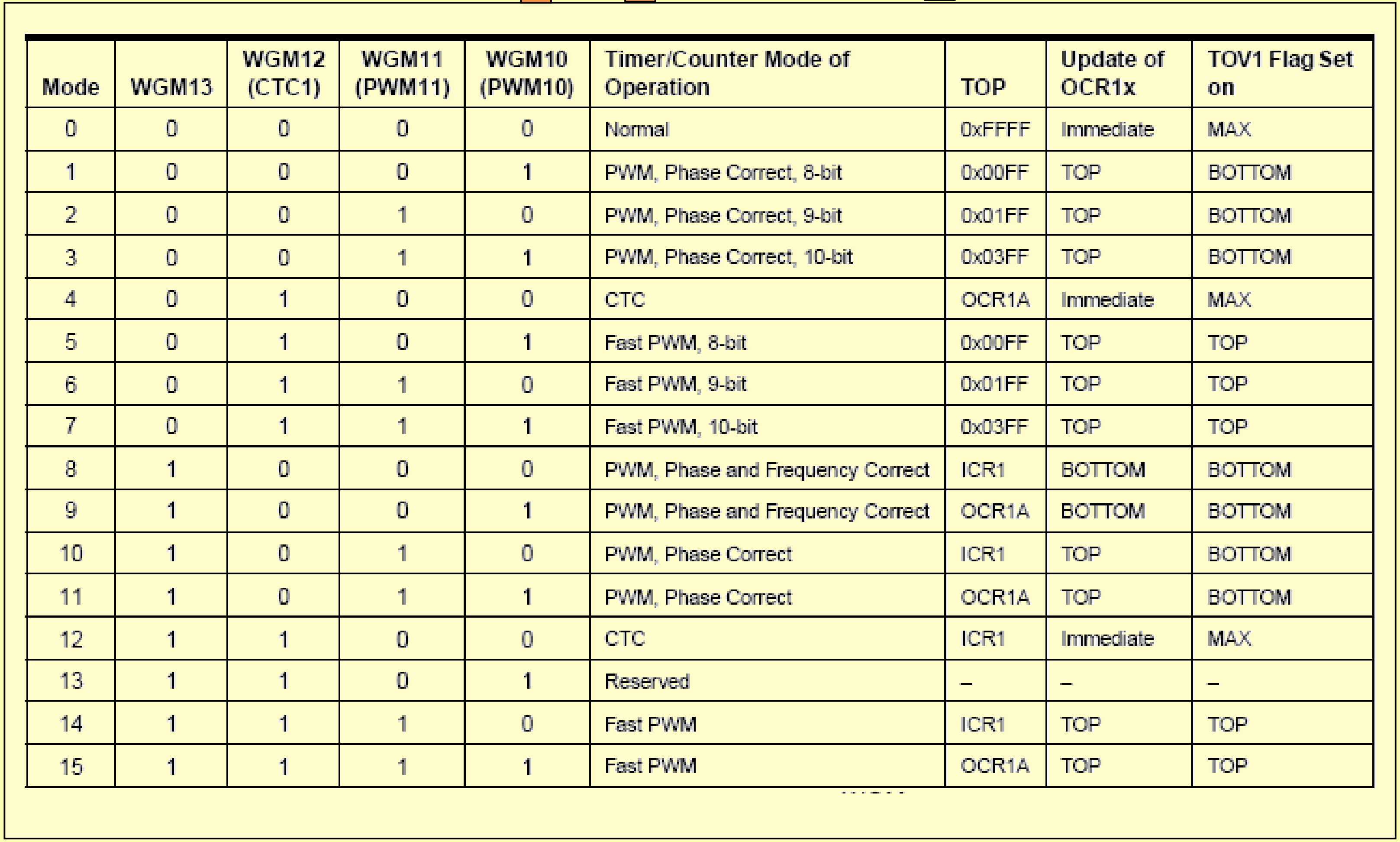
COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B

Clock Selector (CS)

CS12	CS11	CS10	Comment
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



COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B



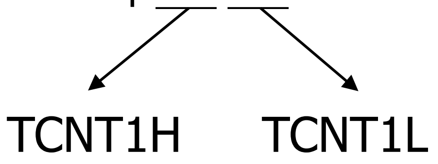
Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	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

Assuming XTAL = 10 MHz write a program that toggles PB5 once per millisecond, using Normal mode.

XTAL = 10 MHz → 1/10 MHz = 0.1 μs

Num. of machine cycles = 1 ms / 0.1 μs = 10,000

TCNT1 = 65,536 – 10,000 = 55,536 = $D8F0



TCNT1H TCNT1L

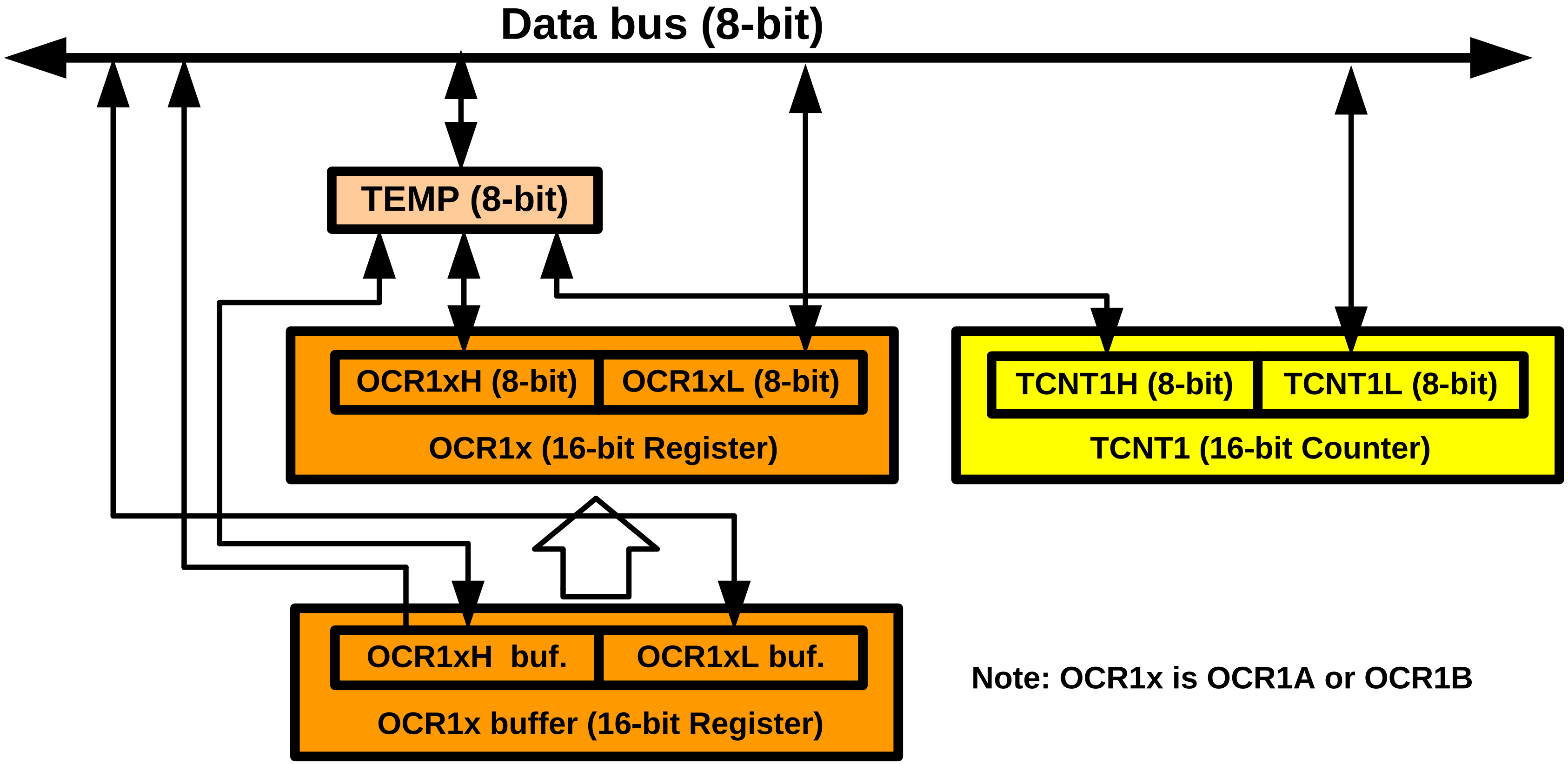
Assuming XTAL = 10 MHz write a program that toggles PB5 once per millisecond, using Normal mode.

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

DELAY_1ms:
    LDI   R20,0xD8
    OUT   TCNT1H,R20          ;TEMP = 0xD8
    LDI   R20,0xF0
    OUT   TCNT1L,R20          ;TCNT1L = 0xF0, TCNT1H = TEMP
    LDI   R20,0x0
    OUT   TCCR1A,R20          ;WGM11:10=00
    LDI   R20,0x1
    OUT   TCCR1B,R20          ;WGM13:12=00,CS=CLK
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
```

Assuming XTAL = 10 MHz write a program that toggles PB5 once per millisecond, using Normal mode.

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

DELAY 1ms:
    LDI    R20,HIGH(-10000)
    OUT    TCNT1H,R20
    LDI    R20, ,LOW(-10000)
    OUT    TCNT1L,R20          ;Timer1 overflows after 10000 machine cycles
    LDI    R20,0x0
    OUT    TCCR1A,R20          ;WGM11:10=00
    LDI    R20,0x1
    OUT    TCCR1B,R20          ;WGM13:12=00,CS=CLK
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
```

TEMP register

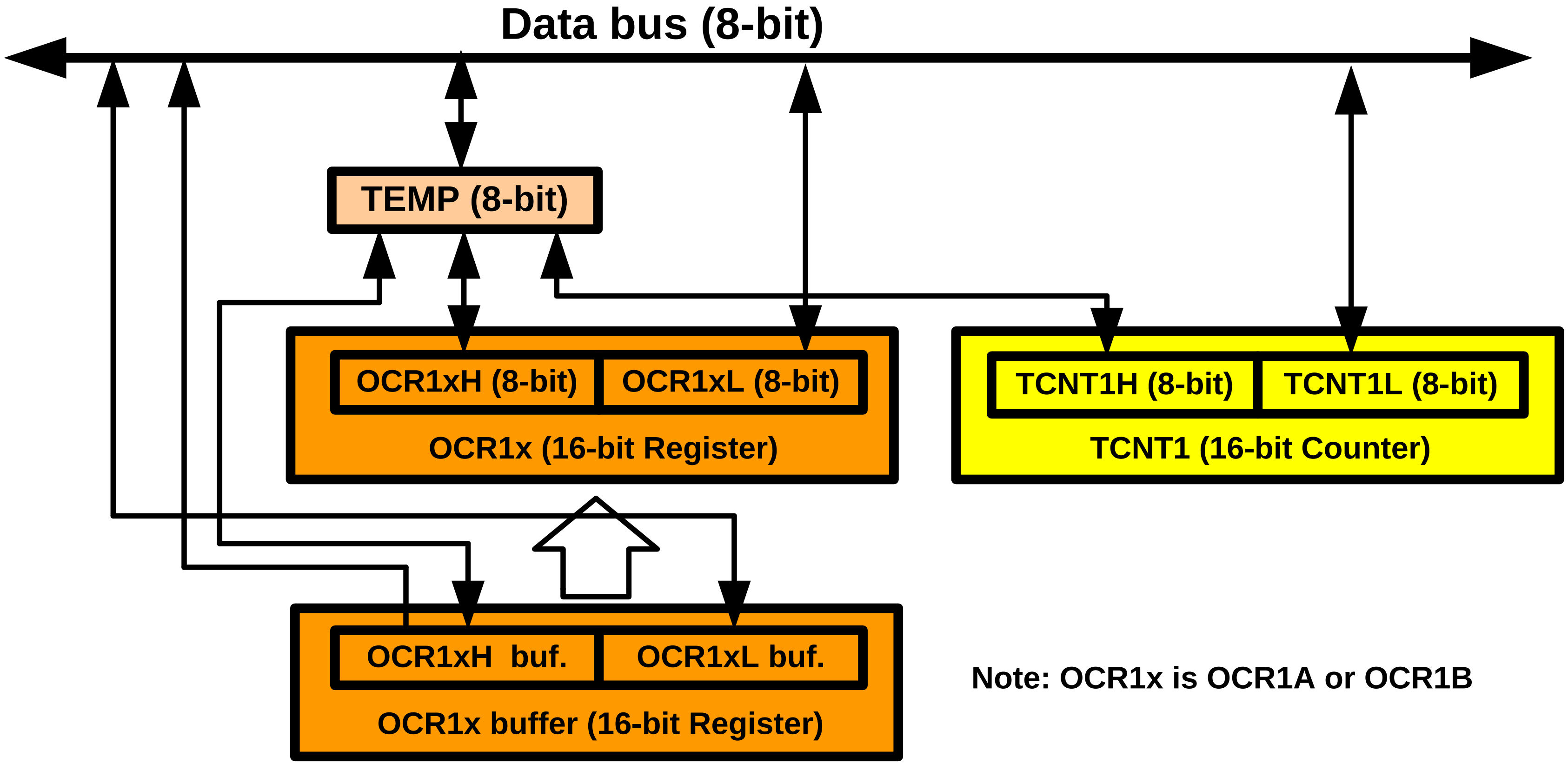


Note: OCR1x is OCR1A or OCR1B

```
LDI   R20,0xF3
OUT   TCNT1H,R20
LDI   R20,0x53
OUT   TCNT1L,R20
```

```
TCNT1H = 0xF3;
TCNT1L = 0x53;
```

TEMP register



Note: OCR1x is OCR1A or OCR1B

```
IN   R20,TCNT1L
IN   R21,TCNT1H
```

```
a = TCNT1L;
b = TCNT1H;
```

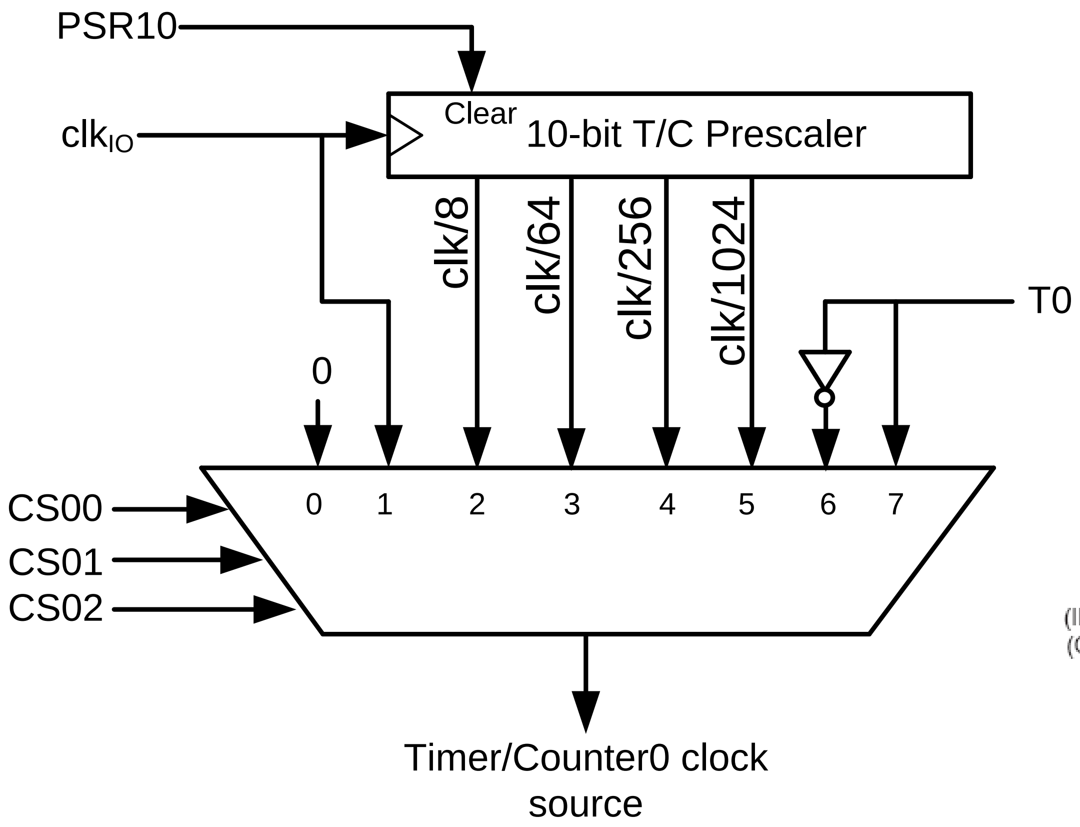
Assuming XTAL = 10 MHz write a program that toggles PB5 once per millisecond, using CTC mode.

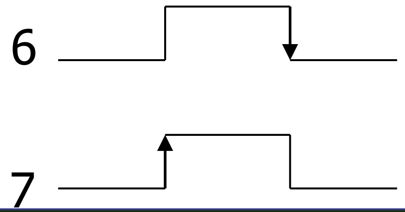
```
.INCLUDE "M32DEF.INC"
     LDI     R16,HIGH(RAMEND)
     OUT     SPH,R16
     LDI     R16,LOW(RAMEND)
     OUT     SPL,R16
     SBI     DDRB,5                    ;PB5 as an output
BEGIN:SBI    PORTB,5                   ;PB5 = 1
     RCALL   DELAY_1ms
     CBI     PORTB,5                   ;PB5 =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

DELAY_1ms:
     LDI     R20,0x00
     OUT     TCNT1H,R20                ;TEMP = 0
     OUT     TCNT1L,R20                ;TCNT1L = 0, TCNT1H = TEMP

     LDI     R20,0x27
     OUT     OCR1AH,R20                ;TEMP = 0x27
     LDI     R20,0x0F
     OUT     OCR1AL,R20                ;OCR1AL = 0x0F, OCR1AH = TEMP

     LDI     R20,0x3
     OUT     TCCR1A,R20                ;WGM11:10=11
     LDI     R20,0x19
     OUT     TCCR1B,R20                ;WGM13:12=11,CS=CLK
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
```

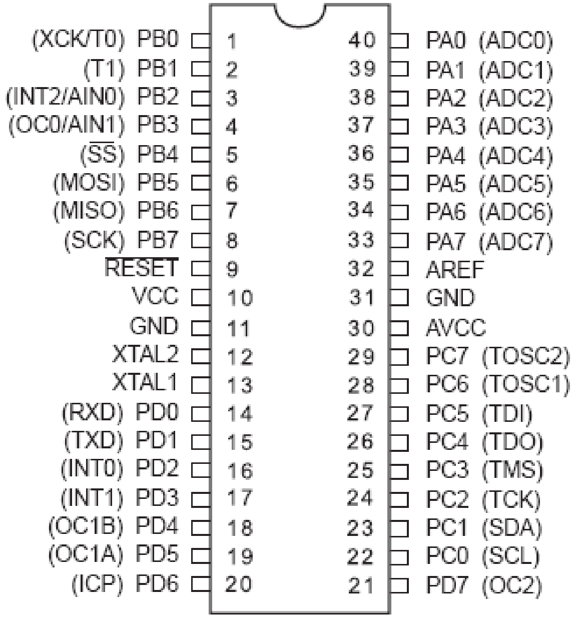
Counting

The AVR microcontroller
and embedded
systems
using assembly and c

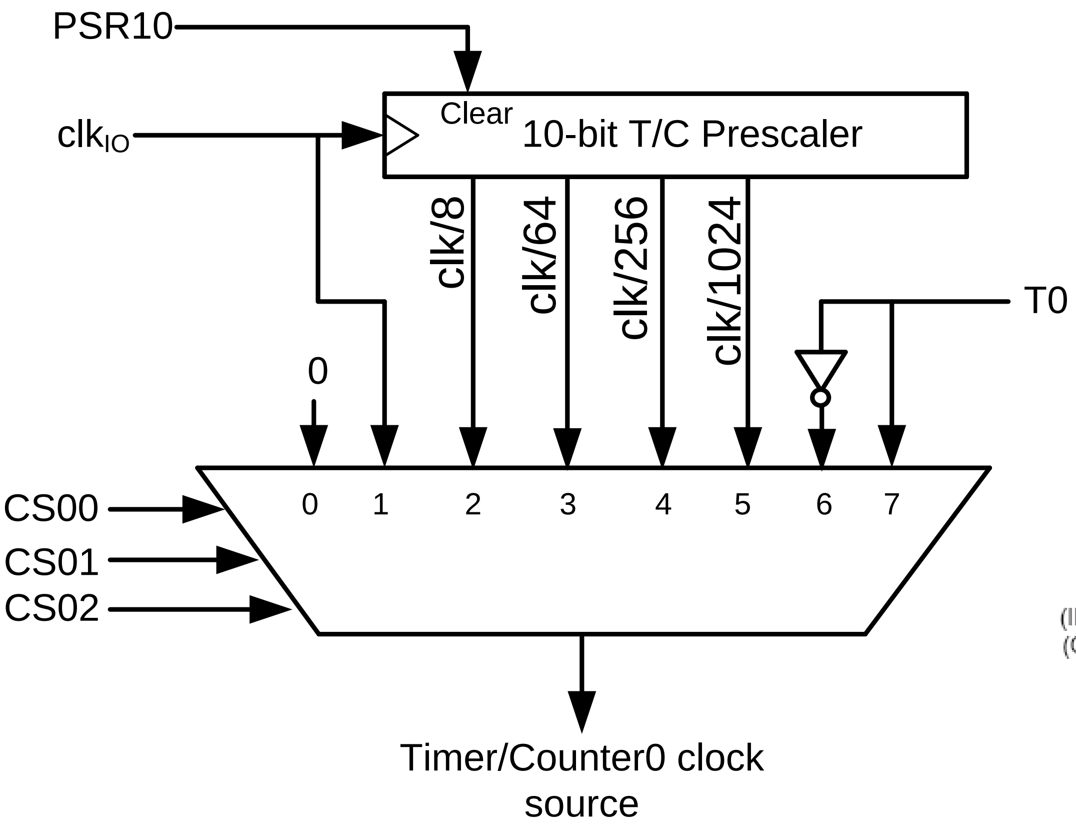
Counting

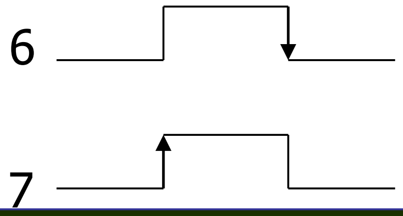

PSR10
clk$_{IO}$
Clear
10-bit T/C Prescaler
clk/8
clk/64
clk/256
clk/1024
T0
0
CS00
CS01
CS02
0
1
2
3
4
5
6
7
Timer/Counter0 clock source

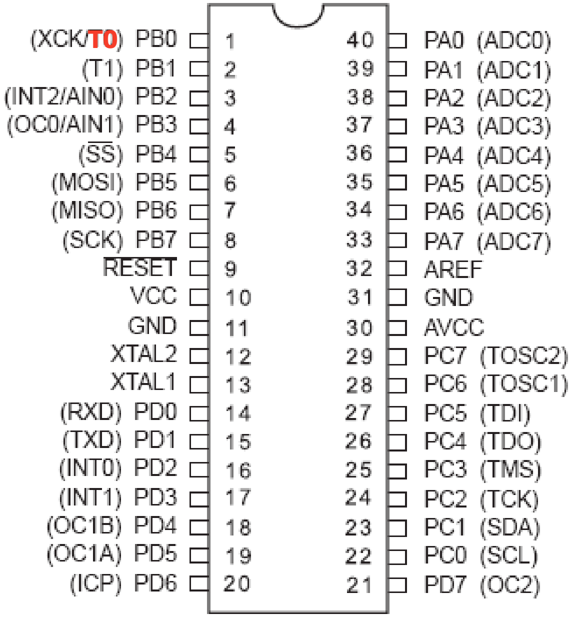

6
7


(XCK/T0) PB0 1
(T1) PB1 2
(INT2/AIN0) PB2 3
(OC0/AIN1) PB3 4
(SS) PB4 5
(MOSI) PB5 6
(MISO) PB6 7
(SCK) PB7 8
RESET 9
VCC 10
GND 11
XTAL2 12
XTAL1 13
(RXD) PD0 14
(TXD) PD1 15
(INT0) PD2 16
(INT1) PD3 17
(OC1B) PD4 18
(OC1A) PD5 19
(ICP) PD6 20
40 PA0 (ADC0)
39 PA1 (ADC1)
38 PA2 (ADC2)
37 PA3 (ADC3)
36 PA4 (ADC4)
35 PA5 (ADC5)
34 PA6 (ADC6)
33 PA7 (ADC7)
32 AREF
31 GND
30 AVCC
29 PC7 (TOSC2)
28 PC6 (TOSC1)
27 PC5 (TDI)
26 PC4 (TDO)
25 PC3 (TMS)
24 PC2 (TCK)
23 PC1 (SDA)
22 PC0 (SCL)
21 PD7 (OC2)

Counting


PSR10
clkIO
Clear
10-bit T/C Prescaler
clk/8
clk/64
clk/256
clk/1024
T0
0
0
1
2
3
4
5
6
7
CS00
CS01
CS02
Timer/Counter0 clock source


6
7


(XCK/T0) PB0 1
(T1) PB1 2
(INT2/AIN0) PB2 3
(OC0/AIN1) PB3 4
(SS) PB4 5
(MOSI) PB5 6
(MISO) PB6 7
(SCK) PB7 8
RESET 9
VCC 10
GND 11
XTAL2 12
XTAL1 13
(RXD) PD0 14
(TXD) PD1 15
(INT0) PD2 16
(INT1) PD3 17
(OC1B) PD4 18
(OC1A) PD5 19
(ICP) PD6 20
40 PA0 (ADC0)
39 PA1 (ADC1)
38 PA2 (ADC2)
37 PA3 (ADC3)
36 PA4 (ADC4)
35 PA5 (ADC5)
34 PA6 (ADC6)
33 PA7 (ADC7)
32 AREF
31 GND
30 AVCC
29 PC7 (TOSC2)
28 PC6 (TOSC1)
27 PC5 (TDI)
26 PC4 (TDO)
25 PC3 (TMS)
24 PC2 (TCK)
23 PC1 (SDA)
22 PC0 (SCL)
21 PD7 (OC2)

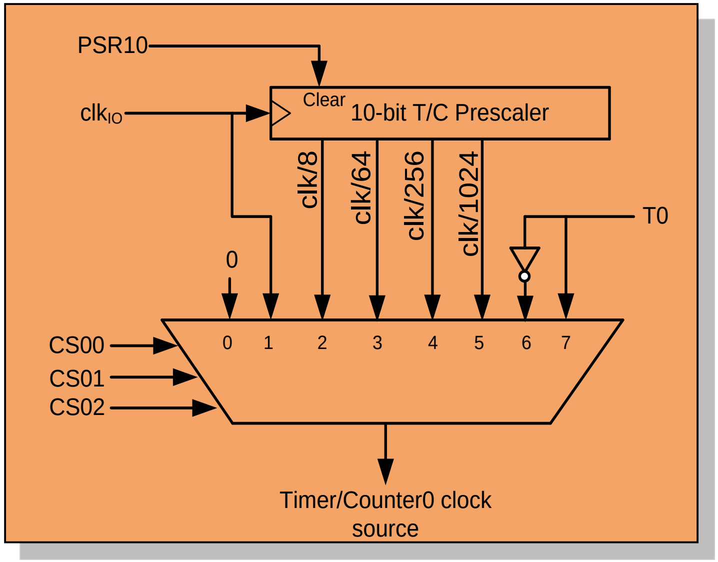
Example Assuming that clock pulses are fed into pin T0, write a program for counter 0 in normal mode to count the pulses on falling edge and display the state of the TCNT0 count on PORTC.

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

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0

Example Assuming that clock pulses are fed into pin T0, write a program for counter 0 in normal mode to count the pulses on falling edge and display the state of the TCNT0 count on PORTC.

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



FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

TCCR0

Assuming that clock pulses are fed into pin T1. Write a program for counter 1 in CTC mode to make PORTC.0 high every 100 pulses.

```
.INCLUDE "M32DEF.INC"

    CBI   DDRB,1               ;make T1 (PB1) input

    SBI   DDRC,0               ;PC0 as an output

    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

    SBI   PORTC,0              ;PC0 = 1
    CBI   PORTC,0              ;PC0 = 0
    RJMP  AGAIN                ;keep doing it
```