

Mechatronics Engineering

Tutorial #3

Interrupts in ATmega328P

Tutorial Contents

- Interrupts
- Seven Segment Display
- Timing Events: An Example on Displays Multiplexing

Tutorial Contents

- **Interrupts**
- Seven Segment Display
- Timing Events: An Example on Displays Multiplexing

Interrupts

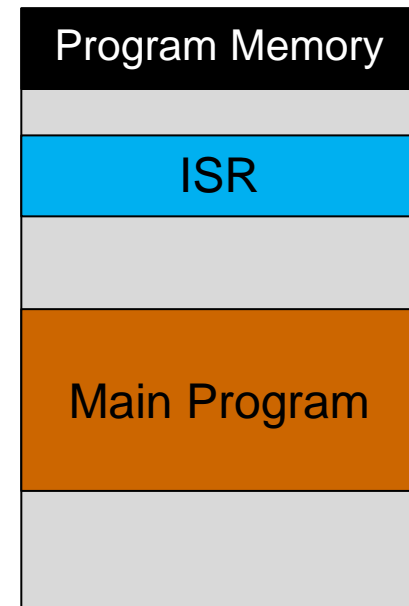
- Microcontrollers do not always have the minimum specifications for using a multitasking Operating System (OS).
- In this case, the programmer might need to handle multiple events in the program.
- Two possible strategies are:
 1. Check for an event repeatedly (**Polling**). Example:

```
while(1) // Main Loop
{
    if(buttonIsPressed()) // check a button (polling)
    {
        PerformAction(); // take this action if button is pressed
    }
    OtherEventsHandling(); // handle other events in main loop
}
```

- Polling Issues:
 - ❖ High priority event handling may be delayed for large programs.
 - ❖ Requires frequent checks, otherwise an event may be not detected.
 - ❖ Takes some computational time to perform the checks.

Interrupts

- The second possible strategy is the use of **Interrupts**:
- Interrupt Operation Sequence:
 1. When an event occurs, the interrupt subsystem will **notify** the CPU.
 2. The CPU will then **jump** to a special program memory location, an Interrupt Service Routine (**ISR**), after executing the current instruction.
 3. The CPU will **execute** event handling code found in this **ISR**.
 4. Once the ISR is executed, the CPU **returns** to where it left off.



Interrupts

- To use any interrupt in the ATmega328P, the global interrupt bit must be set to 1.
- In addition to the global interrupt bit, All interrupts are assigned individual interrupt enable bits which must be written logic one to use the corresponding ISR.

SREG – AVR Status Register

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
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	

- **Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

Interrupts

- Interrupt Types in the ATmega328P:
 - ❖ Internally Triggered Interrupts:
 - Respond to events related to the internal AVR peripherals.
 - This includes: timer interrupts, ADC, UART, SPI.
 - ❖ Externally Triggered Interrupts:
 - Respond to events related to external signals.
 - This includes: Input pin voltage change.
- There are 26 different interrupt branching locations (Interrupt vectors) in the ATmega328P program memory. Each one is for a different interrupt source.
- At the **start** of the ISR the global interrupt bit is **cleared**, disabling all interrupts during execution of the ISR.
- At the **return** from the ISR the global interrupt bit is automatically **set**, reenabling the interrupts.

Interrupts

- Interrupt Vectors:

VectorNo.	Program Address ⁽²⁾	Source	Interrupt Definition
1	0x0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

Interrupts

- Usage of Interrupt in C (AVR-GCC):

- ❖ Include <avr/interrupt.h>
- ❖ Use the Interrupt function definition macro: `ISR(vector){}`.
- ❖ Example (timer 1 overflow interrupt handling function):

```
ISR(TIMER1_OVF_vect)
{
    // global interrupt disabled by hardware at start of ISR
    // interrupt handling code here for timer 1 overflow
    // This ISR is linked by compiler to program memory location 0x001A
} // global interrupt reenabled by hardware at end of ISR
```

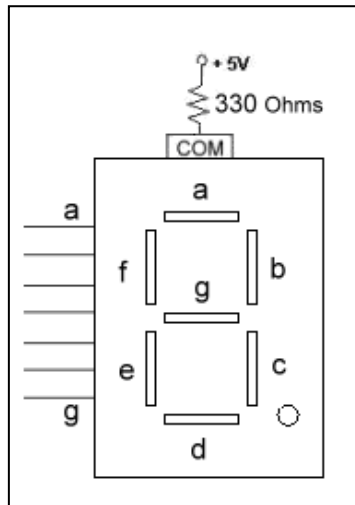
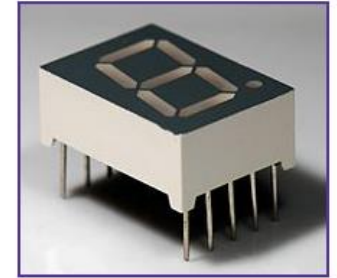
- ❖ Here `vector` is a symbol that indicates the type of interrupt and accordingly, the corresponding interrupt vector for this ISR.
- ❖ **AVR-libc** online documentation includes full list of interrupt vector symbols.

Tutorial Contents

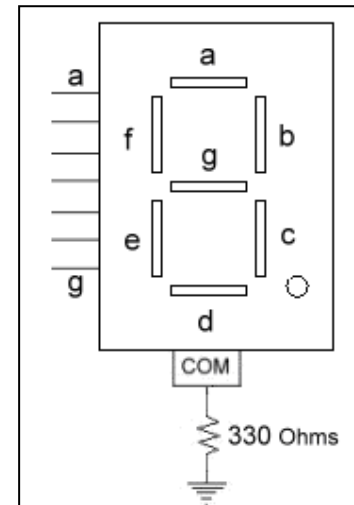
- Interrupts
- **Seven Segment Display**
- Timing Events: An Example on Displays Multiplexing

Seven-Segment Display

- The segments in a 7-segment display are arranged to form a single digit from 0 to F
- Even though LCD displays are more comfortable to work with, 7-segment displays are still standard in the industry.
- The 8 LEDs inside each display can be arranged with a common cathode or common anode.



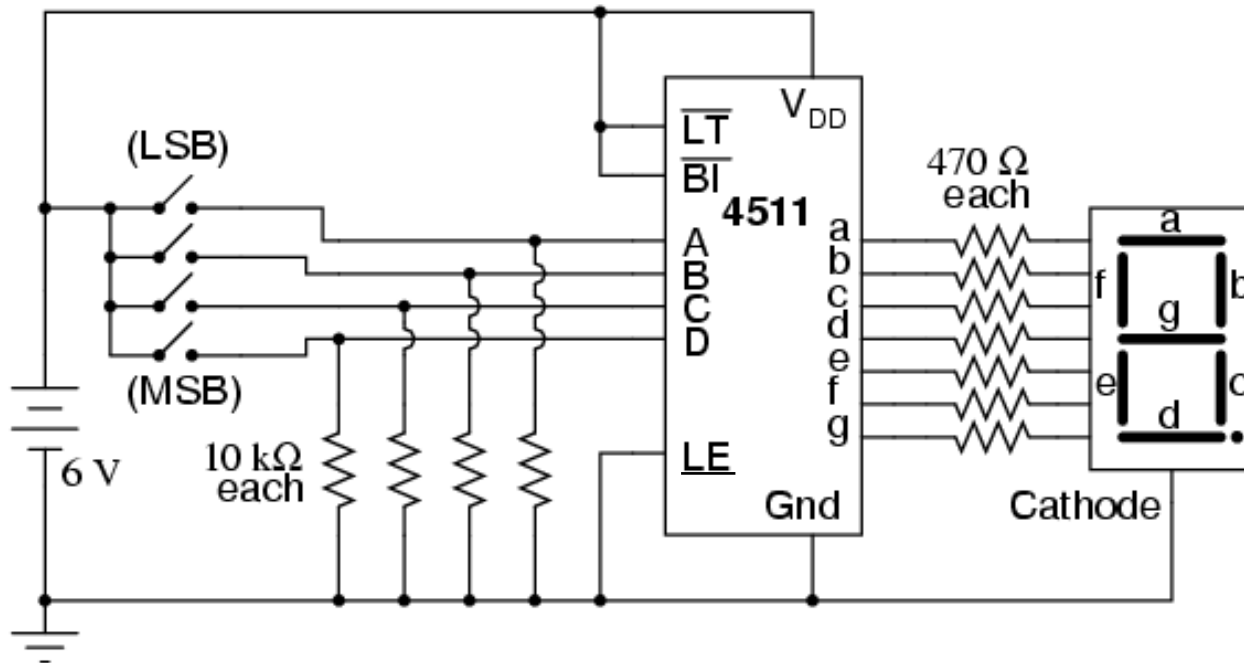
Circuit diagram for Common Anode 7-Segment Display



Circuit diagram for Common Cathode 7-Segment Display

7-segment driver

The simplest way to drive a display is via a display driver. These are available for up to 4 displays.



\overline{LT} = Lamp Test,

\overline{BI} = Blanking Input,

\overline{LE} = Latch Enable

Multiplexing

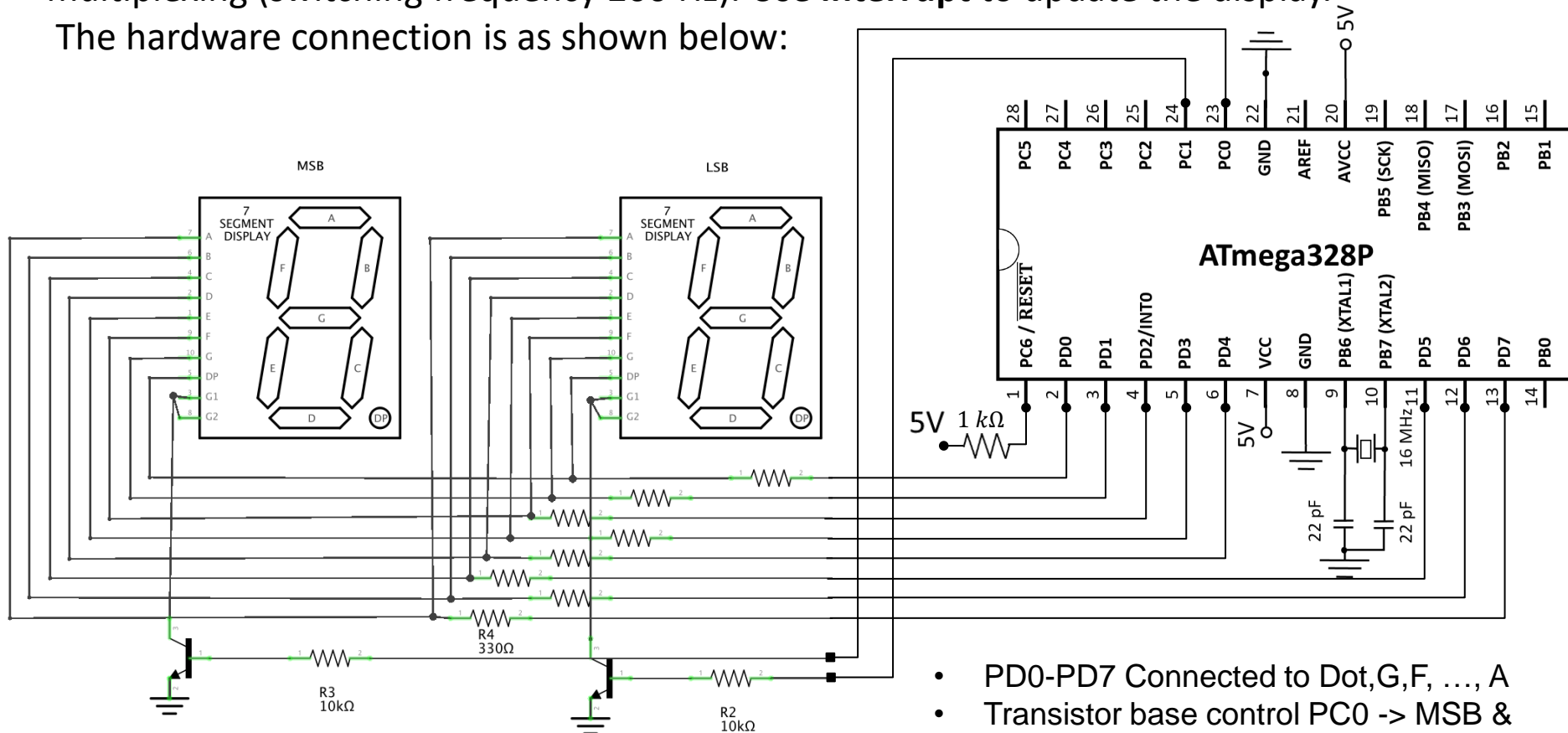
- Alternatively displays can be driven by a microcontroller and if more than one display is required, the method of driving them is called ***"multiplexing."***
- If a single display is to be driven from a microcontroller, 7 lines will be needed plus one for the decimal point. For each additional display, only one extra line is needed.
- To produce a 4, 5 or 6 digit display, all the 7-segment displays are connected in parallel. The common line (the common-cathode line) is taken out separately and this line is taken low for a short period of time to turn on the display.
- Each display is turned on at a rate above 100 times per second, and it will appear that all the displays are turned on at the same time. As each display is turned on, the appropriate information must be delivered to it so that it will give the correct reading.

Tutorial Contents

- Interrupts
- Seven Segment Display
- **Timing Events: An Example on Displays Multiplexing**

Problem 1:

Write an assembly code that displays the number 21 on the common cathode 7-segment displays. You should use **timer 0** in CTC mode to manage the timing of display multiplexing (switching frequency 200 Hz). Use **interrupt** to update the display. The hardware connection is as shown below:



Solution:

- Set timer 0 in Clear Timer on Compare match (CTC) mode.

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 14-8. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on ⁽¹⁾⁽²⁾
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Notes: 1. MAX = 0xFF

2. BOTTOM = 0x00

Solution:

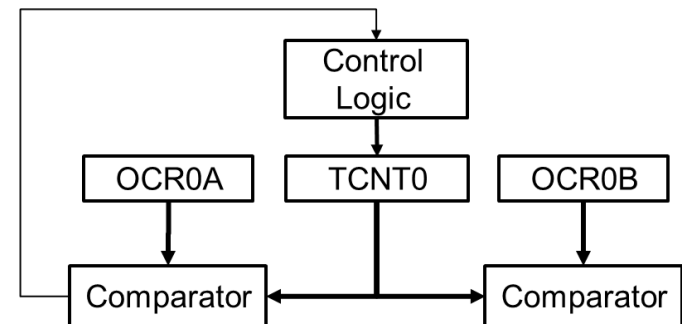
- In CTC mode, when the value in TCNT0 matches the value in OCR0A, TCNT0 resets to zero.
- Clock source should be selected to result in a compare match every (1/200 = 0.005) seconds.
- Period of compare match with OCR0A:

$$T_{match} = N (1 + OCR0A) T_{clk_{IO}}$$

- N is the prescaler value selected using CS0x bits.
- The prescaler allows the timer to increment every Nth clock cycle.

Table 14-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$ (no prescaling)
0	1	0	$clk_{I/O}/8$ (from prescaler)
0	1	1	$clk_{I/O}/64$ (from prescaler)
1	0	0	$clk_{I/O}/256$ (from prescaler)
1	0	1	$clk_{I/O}/1024$ (from prescaler)
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.



Solution:

$$T_{match} = N (1 + OCR0A) T_{clk_IO}$$

- A possible way to determine the required prescaler N is to perform an initial calculation by setting the OCR0A value to its maximum (255).

- Given that the required T_{match} is 0.005 s:

$$N = \frac{T_{match}}{T_{clk_IO}(256)} = \frac{0.005}{6.25 * 10^{-8} (256)} = 312.5$$

- To achieve the required T_{match} , the following condition must be satisfied:
 $N \geq 312.5$. Therefore, the only suitable prescaler value is $N = 1024$.
- To find the OCR0A value that corresponds to $T_{match} = 0.005$ s, We use the T_{match} equation with $N = 1024$.

$$OCR0A = \frac{T_{match}}{N * T_{clk_IO}} - 1 = \frac{0.005}{1024 * 6.25 * 10^{-8}} - 1 = 77.125 \approx 77$$


- Nearest value to calculated **OCR0A** is 77 which produces $T_{match} = 0.00499$ s
Very close to 0.005 s. To get higher resolution we may use the 16-bit **Timer 1** !

Solution:

- To enable the interrupt on timer 0 compare match A, we must enable the global interrupt enable bit “I”.
- We must also set the output compare match A interrupt enable bit (**OCIE0A**) in the **TIMSK0** register.

TIMSK0 – Timer/Counter Interrupt Mask Register

Bit (0x6E)	7	6	5	4	3	2	1	0	
	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



- **Bits 7:3 – Reserved**

These bits are reserved bits in the ATmega48A/PA/88A/PA/168A/PA/328/P and will always read as zero.

- **Bit 2 – OCIE0B: Timer/Counter Output Compare Match B Interrupt Enable**

When the OCIE0B bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter Compare Match B interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter occurs, i.e., when the OCF0B bit is set in the Timer/Counter Interrupt Flag Register – TIFR0.

- **Bit 1 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable**

When the OCIE0A bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Compare Match A interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter0 occurs, i.e., when the OCF0A bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

Solution:

TIFR0 – Timer/Counter 0 Interrupt Flag Register

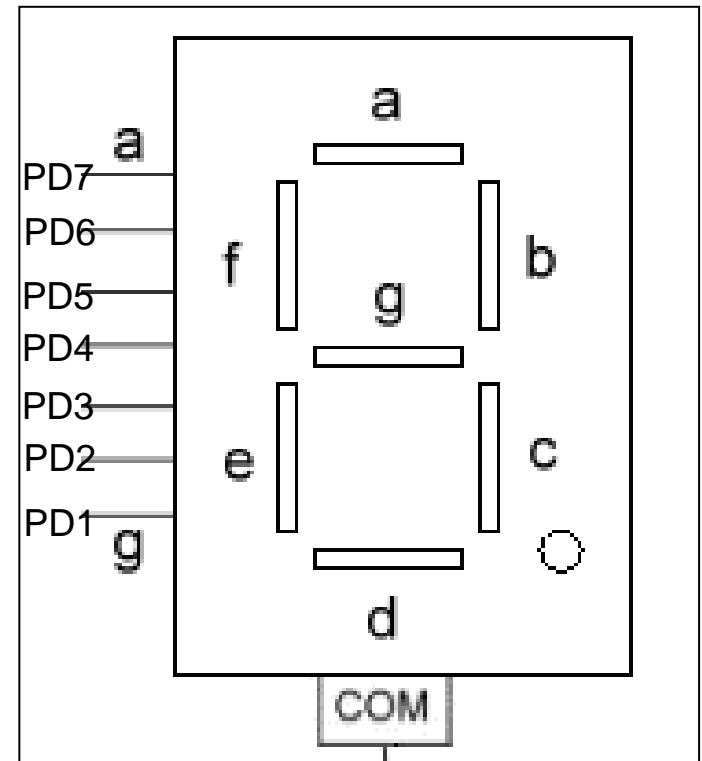
Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- When a compare match between OCR0A and TCNT0 occurs, a flag **OCF0A** is set.
- The flag is cleared automatically by hardware after executing the corresponding interrupt handling ISR (TIMER0 COMPA).
- If not using interrupts, the flag may need to be cleared manually by writing logic one to the flag.

Solution:

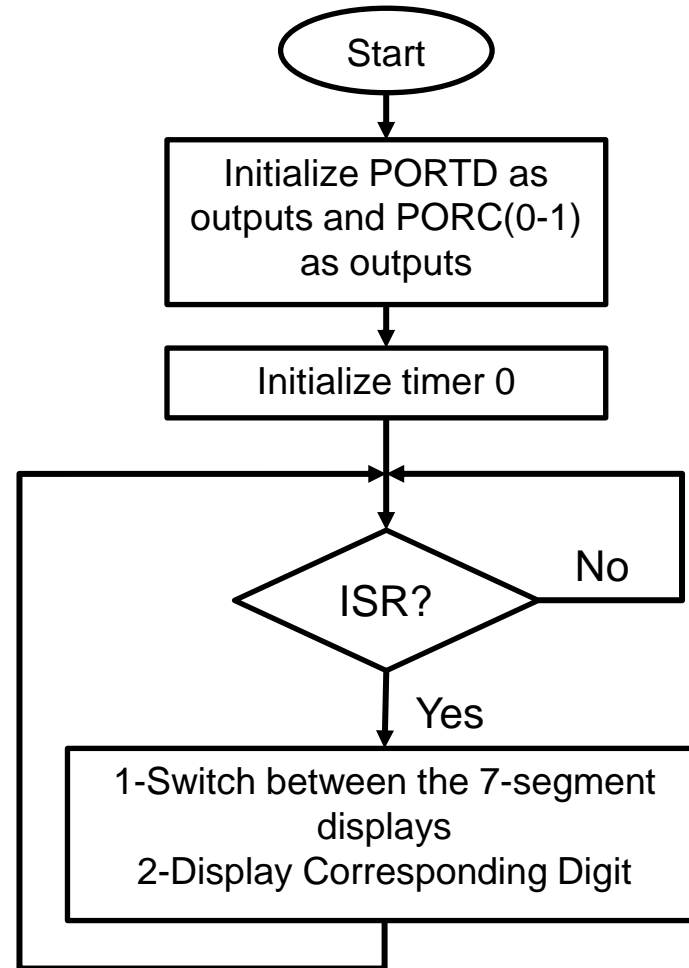
- It is required to decode the values for each digit to match the connection shown on the seven-segment display.
- A look-up table can be used to decode each digit (0-9) into the corresponding pin values for PD1 – PD7 According to the figure shown.

```
// Lookup table.  
const unsigned char bcd_lookup[] = {0b11111100,  
0b01100000, 0b11011010, 0b11110010, 0b01100110,  
0b10110110, 0b11111010, 0b11100000, 0b11111110,  
0b11110110}; // Each elements holds the value of  
PORTD that would display the digit corresponding  
to its index.  
//Example:  
PORTD = bcd_lookup[2]; // Displays 2 on 7-segment
```



Solution:

- Flow chart for the seven-segment multiplexing code:



Code:

```
#include <avr/interrupt.h>
#include <avr/io.h>

const unsigned char disp_lookup[]={0b11111100,0b01100000, 0b11011010,
                                   0b11110010, 0b01100110,0b10110110,
                                   0b11111010, 0b11100000, 0b11111110,
                                   0b11110110};

volatile unsigned char val,temp; // flag to switch between MSB and LSB

int main(void)
{
    val = 21;
    DDRD = 0xFF; // all pins on PORTD are outputs
    DDRC = 0b00000011; // PC0-PC1 are outputs
    PORTD = disp_lookup[(val%10)]; // display the Least significant bit on PORTD
    PORTC= 0b00000010; // set pin 1 to logic high (LSB) and the other MSB to logic low

    OCR0A = 77; // set value for clear on compare match
    SREG |= 0b10000000; // enable global interrupt
    TIMSK0 |= 0b00000010; // enable OCIE0A interrupt flag for OCR0A and TCNT match
    TCCR0A = 0b00000010; // set mode to CTC
    TCCR0B = 0b00000101; // set clock prescaler to be 1024

    while (1)
    {
        // wait for interrupt
    }
}
```

Code, continued:

Every 0.005 seconds this code should be executed.

```
ISR(TIMER0_COMPA_vect)
{
    PORTD = 0; // turn off all LEDs during switching
    PORTC ^= 0b00000011; // toggle the last two bits of PORTC 01 <-> 10
    if(PORTC & 0b00000001) // MSB active
    {
        temp = val/10; // extract MSB
    }else // LSB active
    {
        temp = val; // keep LSB
    }
    PORTD = disp_lookup[temp%10]; // set PORTD according to lookup table
}
```