Unit III- PIC Interrupts & Interfacing-I

Interrupt Vs Polling

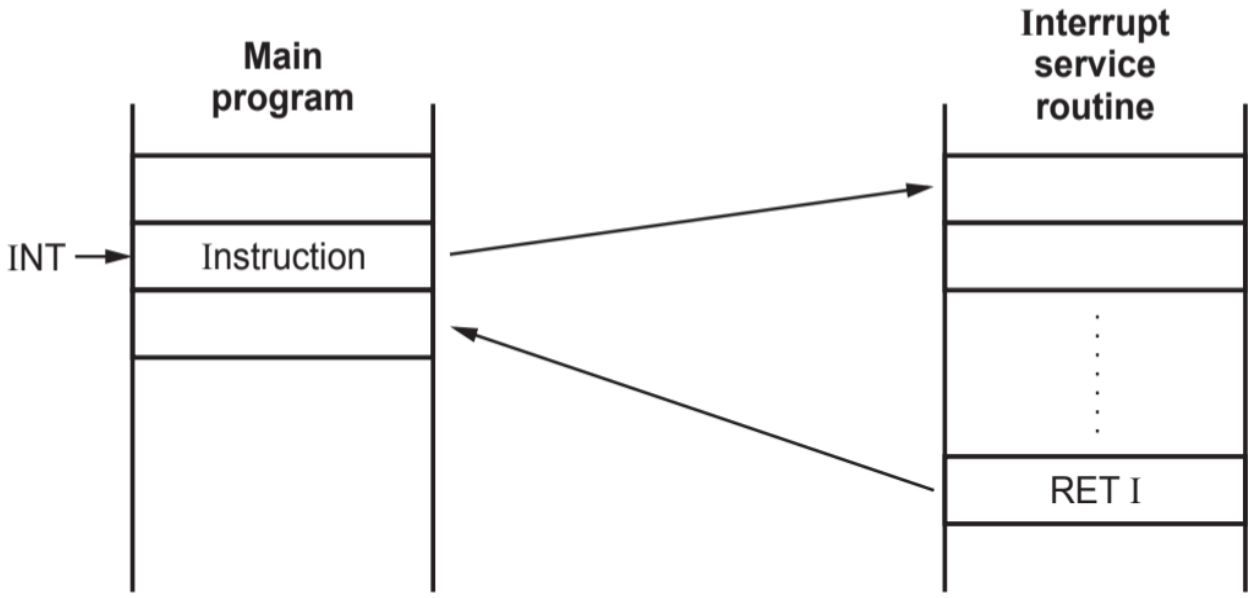
- When one or more I/O devices are connected to a microcontroller system, any one of them may demand service at any time. The microcontroller can service these devices in one of the two ways : **polling** or **interrupt**.
- In polling, the microcontroller's program simply checks each of the I/O devices to see if any device needs servicing. If so, it performs the service.
- In the interrupt method, whenever any device needs microcontroller's service, it tells this to microcontroller by sending an interrupt signal. Upon receiving an interrupt signal, the microcontroller stops executing its current program and calls a subroutine called **Interrupt service routine** which services the interrupt. Once the interrupt service routine is completed, the microcontroller resumes the execution of the interrupted program.

Sr. No.	Polling method	Interrupt method
1.	In polling, a lot of microcontroller's processing power is wasted in checking whether the I/O devices need service.	In interrupt method, microcontroller's time is not wasted in checking whether the I/O devices need service.
2.	Since all the devices are polled in a sequential manner, there is no explicit priority mechanism in the polling method.	Interrupt method can assign priorities to the interrupts so that they can be served according to priority basis.
3.	In polling, there is no possibility of ignoring a device request for service.	In interrupt method, the microcontroller can ignore (mask) a device request for service.
4.	Inefficient way of handling I/O devices.	Efficient way of handling I/O devices.
5.	No external hardware signal required.	External hardware signal is required to interrupt microcontroller.

Steps in Executing the Interrupt

- In response to the interrupt, the microcontroller goes through following steps :

1. It completes the execution of current instruction and save the address of the next instruction, i.e. the contents of PC on the stack.
2. It also saves the current status of all the interrupts internally i.e. not on the stack.
3. It then transfers the program control to a fixed location in memory called the **Interrupt Vector Table (IVT)**. IVT holds the address of the interrupt service routine.
4. The microcontroller gets the address of the Interrupt Service Routine (ISR) from IVT and transfers program control to the ISR. It executes instructions within the interrupt service routine in sequence till it reaches the RETI (Return From The Interrupt) instruction.
5. Upon executing the RETI instruction, it gets the Program Counter (PC) address from the stack and restores the interrupt logic. Since PC is loaded with the address of the next instruction from where the interrupt was initiated, it starts to execute from that address.



Interrupt service routine processing

IVT (Interrupt Vector Table)

- IVT stands for **Interrupt Vector Table.** It holds the addresses of the interrupt service routines for various interrupts supported by the microcontroller.
- Table shows the interrupt vector table for the PIC18.

Interrupt	Vector Address (RAM Location)
Power-on Reset	0000H
High Priority Interrupt	0008H
Low Priority Interrupt	0018H

Classification of Interrupts

- An interrupt caused by an external signal is referred as a **hardware interrupt** or **external interrupt**. Conditional interrupts or interrupts caused by special instructions are called **software interrupts or internal interrupts**.
- The interrupts are further classified as maskable interrupts, non-maskable interrupts, vector interrupts and non-vector interrupts.

1. **Maskable Interrupts :** Maskable interrupts are enabled and disabled under program control. By setting or resetting particular flip-flops in the processor, interrupts can be masked or unmasked, respectively. When masked, processor does not respond to the interrupt even though the interrupt is activated. Such interrupts are called Maskable interrupts.

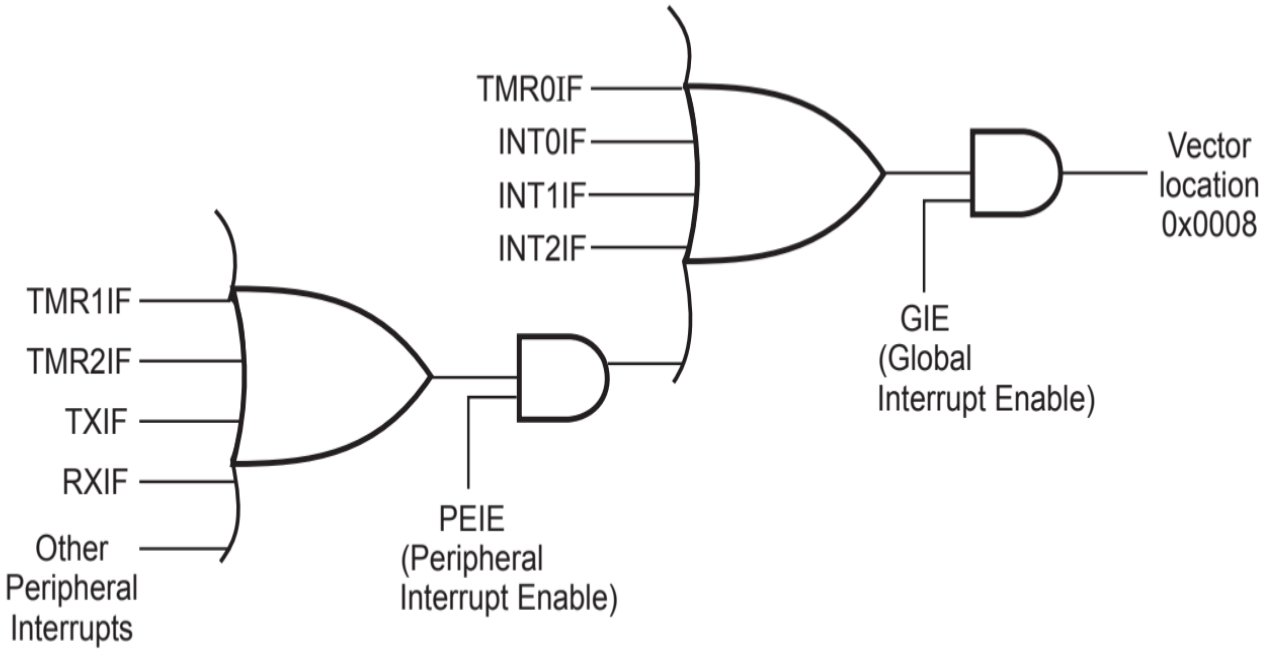
2. **Non-Maskable Interrupts :** The interrupts which can not be masked under software control are called non-maskable interrupts.

3. **Vectored Interrupts :** The interrupt for which the starting address of the interrupt service routine is predefined is called vectored interrupts. Since the starting address of interrupt service routine is predefined the interrupt response time for vector interrupts is less compared to non-vectored interrupts.

4. **Non-vectored Interrupts :** The interrupt for which the starting address of the interrupt service routine is not predefined and it is provided by the external hardware at the time of interrupt is called non-vectored interrupt.

Sources of Interrupts

- The PIC18 devices have multiple interrupt sources depending on which peripherals are incorporated in the chip. Some of the most widely used interrupt sources of PIC18 are :
  - External hardware interrupts : INT0, INT1 and INT2
  - Timer interrupts : TMR0IF, TMR1IF, TMR2IF and TMR3IF
  - Serial communication interrupts : TXIF and RCIF
  - RB Port change interrupt : RBIF
  - A/D converter interrupt : ADIF
  - Compare, capture, PWM interrupt : CCP1IF



Simplified interrupt structure of PIC18

- Each interrupt source has three bits to control its operation. The functions of these bits are :
  - Flag bit to indicate that an interrupt event occurred.
  - Enable bit that allows program execution to branch to the interrupt vector address when the flag bit is set.
  - Priority bit to select high priority or low priority.

Enabling and Disabling Interrupts

- When PIC18 is reset, all interrupts are disabled. These are enabled by software. All of the bits that generate interrupts can be set or cleared by software.
- Fig. 6.6.1 shows the bit pattern for the INTCON register.

bit 7 **GIE :** Global Interrupt Enable bit

1 = Enables all unmasked interrupts

0 = Disables all interrupts

bit 6 **PEIE :** Peripheral Interrupt Enable bit

1 = Enables all unmasked peripheral interrupts

0 = Disables all peripheral interrupts

bit 5 **TMR0IE :** TMR0 Overflow Interrupt Enable bit

1 = Enables the TMR0 overflow interrupt

0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE :** INT0 External Interrupt Enable bit

1 = Enables the INT0 external interrupt

0 = Disables the INT0 external interrupt

bit 3 **RBIE :** RB Port Change Interrupt Enable bit

1 = Enables the RB port change interrupt

0 = Disables the RB port change interrupt

- The PEIE bit (INTCON register) enables/disables all peripheral interrupt sources. The GIE bit (INTCON register) enables/disables all interrupt sources.

Steps in Enabling Interrupt

1. Make GIE bit (INTCON register : Bit 7) = 1 to enable all maskable interrupts.
2. Set the dedicated interrupt enable flag bit for the interrupt to be used.
3. If we want to use peripheral interrupts, enable them by making PEIE (Peripheral Interrupt Enable bit) of INTCON register high.

Interrupt Registers

- For the PIC18F458 microcontroller, 13 registers are used to control interrupt operation. These registers are :
  - RCON
  - INTCON
  - INTCON2
  - INTCON3
  - PIR1, PIR2, PIR3
  - PIE1, PIE2, PIE3
  - IPR1, IPR2, IPR3

Reset Control Register: RCON

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IPEN	SBOREN	-	R1	TO	PD	POR	BOR

bit 7 **IPEN:** Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (PIC16CXXX Compatibility mode)

bit 6 **SBOREN:** BOR Software Enable bit

bit 5 **Unimplemented:** Read as '0'

bit 4 **$\overline{RI}$:** RESET Instruction Flag bit

bit 3 **$\overline{TO}$:** Watchdog Time-out Flag bit

bit 2 **$\overline{PD}$:** Power-Down Detection Flag bit

bit 1 **$\overline{por}$:** Power-on Reset Status bit

bit 0 **$\overline{BOR}$:** Brown-out Reset Status bit

INTCON Registers

- The INTCON registers are readable and writable registers that contain various enable, priority and flag bits. Because of the number of interrupts to be controlled, PIC18FXX8 devices have three INTCON registers.

INTCON Register

GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF

bit 7 bit 0

bit 7 **GIE/GIEH :** Global Interrupt Enable bit

When IPEN (RCON < 7) = 0;

1 = Enables all unmasked interrupts

0 = Disables all interrupts

When IPEN (RCON < 7 >) = 1;

1 = Enables all high priority interrupts

0 = Disables all priority interrupts

bit 6 **PEIE/GIEL :** Peripheral Interrupt Enable bit

When IPEN (RCON < 7 >) = 0;

1 = Enables all unmasked peripheral interrupts

0 = Disables all peripheral interrupts

When IPEN (RCON < 7 >) = 1;

1 = Enables all low priority peripheral interrupts

0 = Disables all low priority peripheral interrupts

bit 5 **TMR0IE :** TMR0 Overflow Interrupt Enable bit

1 = Enables the TMR0 overflow interrupt

0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE :** INT0 External Interrupt Enable bit

1 = Enables the INT0 external interrupt

0 = Disables the INT0 external interrupt

bit 3 **RBIE :** RB Port Change Interrupt Enable bit

1 = Enables the RB port change interrupt

0 = Disables the RB port change interrupt

bit 2 **TMR0IF :** TMR0 Overflow Interrupt Flag bit

1 = TMR0 register has overflowed (must be cleared in software)

0 = TMR0 register did not overflow

bit 1 **INT0IF :** INT0 External Interrupt Flag bit

1 = The INT0 external interrupt occurred (must be cleared in software by reading PORTB)

0 = The INT0 external interrupt did not occur

bit 0 **RBIF :** RB Port Change Interrupt Flag bit

1 = At least one of the RB7 : RB4 pins changed state (must be cleared in software)

0 = None of the RB7 : RB4 pins have changed state

INTCON2 Register

$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	–	–	TMR0IP	–	RBIP

bit 7 bit 0

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit

1 = All PORTB pull-ups are disabled

0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG0 :** External Interrupt 0 Edge Select bit

1 = Interrupt on rising edge

0 = Interrupt on falling edge

bit 5 **INTEDG1 :** External Interrupt 1 Edge Select bit

1 = Interrupt on rising edge

0 = Interrupt on falling edge

bit 4-3 **Unimplemented :** Read as '0'

bit 2 **TMR0IP :** TMR0 Overflow Interrupt Priority bit

1 = High priority

2 = Low priority

bit 1 **RBIP :** RB Port Change Interrupt Priority bit

1 = High priority

0 = Low priority

INTCON3 Register

INT2IP	INT1IP	–	INT2IE	INT1IE	–	INT2IF	INT1IF

bit 7 bit 0

bit 7 **INT2IP :** INT2 External Interrupt Priority bit

1 = High priority

0 = Low priority

bit 6 **INT1IP :** INT1 External Interrupt Priority bit

1 = High priority

0 = Low priority

bit 5 **Unimplemented :** Read as '0'

bit 4 **INT2IE :** INT2 External Interrupt Enable bit

1 = Enables the INT2 external interrupt

0 = Disables the INT2 external interrupt

bit 3 **INT1IE :** INT1 External Interrupt Enable bit

1 = Enables the INT1 external interrupt

0 = Disables the INT1 external interrupt

bit 2 **Unimplemented :** Read as '0'

bit 1 **INT2IF :** INT2 External Interrupt Flag bit

1 = The INT2 external interrupt occurred (must be cleared in software)

0 = The INT2 external interrupt did not occur

bit 0 **INT1IF :** INT1 External Interrupt Flag bit

1 = The INT1 external interrupt occurred (must be cleared in software)

0 = The INT1 external interrupt did not occur

PIR Registers

- The Peripheral Interrupt Request (PIR) registers contain the individual flag bits for the peripheral interrupts.
- Due to the number of peripheral interrupt sources, there are three Peripheral Interrupt Request (Flag) registers (PIR1, PIR2, PIR3).

PIR1: Peripheral Interrupt Request (Flag) Register 1

PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

bit 7 bit 0

bit 7 **PSPIF :** Parallel Slave Port Read/Write Interrupt Flag bit

1 = A read or a write operation has taken place (must be cleared in software)

0 = No read or write has occurred

bit 6 **ADIF :** A/D Converter Interrupt Flag bit

1 = An A/D conversion completed (must be cleared in software)

0 = The A/D conversion is not complete

bit 5 **RCIF :** USART Receive Interrupt Flag bit

1 = The USART receive buffer, RCREG is full (cleared when RCREG is read)

0 = The USART receive buffer is empty

bit 4 **TXIF :** USART Transmit Interrupt Flag bit

1 = The USART transmit buffer, TXREG is empty (cleared when TXREG is written)

0 = The USART transmit buffer is full

bit 3 **SPPIF :** Master Synchronous Serial Port Interrupt Flag bit

1 = The transmission/reception is complete (must be cleared in software)

0 = Waiting to transmit/receive

bit 2 **CCP1IF :** CCP1 Interrupt Flag bit

Capture mode :

1 = A TMR1 register capture occurred (must be cleared in software)

0 = No TMR1 register capture occurred

Compare mode :

1 = A TMR1 register compare match occurred (must be cleared in software)

0 = No TMR1 register compare match occurred

PWM mode :

Unused in this mode.

bit 1 **TMR2IF :** TMR2 to PR2 Match Interrupt Flag bit

1 = TMR2 to PR2 match occurred (must be cleared in software)

0 = No TMR2 to PR2 match occurred

bit 0 **TMR1IF :** TMR1 Overflow Interrupt Flag bit

1 = TMR1 register overflowed (must be cleared in software)

0 = TMR1 register did not overflow

PIR2 : Peripheral Interrupt Request (Flag) Register 2

–	CMIF	–	EEIF	BCLIF	LVDIF	TMR3IF	ECCP1IF

bit 7 bit 0

bit 7 **Unimplemented :** Read as '0'

bit 6 **CMIF :** Comparator Interrupt Flag bit

1 = Comparator input has changed

0 = Comparator input has not changed

bit 5 **Unimplemented :** Read as '0'

bit 4 **EEIF :** EEPROM Write Operation Interrupt Flag bit

1 = Write operation is complete (must be cleared in software)

0 = Write operation is not complete

bit 3 **BCLIF :** Bus Collision Interrupt Flag bit

1 = A has collision occurred (must be cleared in software)

0 = No bus collision occurred

bit 2 **LVDIF :** Low-Voltage Detect Interrupt Flag bit

1 = A low-voltage condition occurred (must be cleared in software)

0 = The device voltage is above the Low-Voltage Detect trip point

bit 1 **TMR3IF :** TMR3 Overflow Interrupt Flat bit

1 = TMR3 register overflowed (must be cleared in software)

0 = TMR3 register did not overflow

bit 0 **ECCP1IF :** ECCP1 Interrupt Flag bit

Capture mode :

1 = A TMR1 (TMR3) register capture occurred (Must be cleared in software)

0 = No TMR1 (TMR3) register capture occurred

PWM mode :

Unused in this mode.

PIR 3 : Peripheral Interrupt Request (Flag) Register 3

IRXIF	WAKIF	ERRIF	TXB2IF	TXB1IF	TXB0IF	RXB1IF	RXB0IF

bit 7 bit 0

bit 7 **IRXIF :** Invalid Message Received Interrupt Flag bit

1 = An invalid message has occurred on the CAN bus

0 = An invalid message has not occurred on the CAN bus

bit 6 **WAKIF :** Bus Activity Wake-up Interrupt Flag bit

1 = Activity on the CAN bus has occurred

0 = Activity on the CAN bus has not occurred

bit 5 **ERRIF :** CAN bus Error Interrupt Flag bit

1 = An error has occurred in the CAN module (multiple sources)

0 = An error has not occurred in the CAN module

bit 4 **TXB2IF :** Transmit Buffer 2 Interrupt Flag bit

1 = Transmit Buffer 2 has completed transmission of a message and may be reloaded

0 = Transmit Buffer 2 has not completed transmission of a message

bit 3 **TXB1IF :** Transmit Buffer 1 Interrupt Flag bit

1 = Transmit Buffer 1 has completed transmission of a message and may be reloaded

0 = Transmit Buffer 1 has not completed transmission of a message

bit 2 **TXB0IF :** Transmit Buffer 0 Interrupt Flag bit

1 = Transmit Buffer 0 has completed transmission of a message and may be reloaded

0 = Transmit Buffer 0 has not completed transmission of a message

bit 1 **RXB1IF :** Receiver Buffer 1 Interrupt Flag bit

1 = Transmit Buffer 1 has completed transmission of a message and may be reloaded

0 = Transmit Buffer 1 has not completed transmission of a message

bit 0 **TXB1IF :** Receiver Buffer 0 Interrupt Flag bit

1 = Transmit Buffer 0 has completed transmission of a message and may be reloaded

0 = Transmit Buffer 0 has not completed transmission of a message

PIE Registers

- The Peripheral Interrupt Enable (PIE) registers contain the individual enable bits for the peripheral interrupts.
- Due to the number of peripheral interrupt sources, there are three Peripheral Interrupt Enable registers (PIE1, PIE2, PIE3).
- When IPEN is clear, the PEIE bit must be set to enable any of these peripheral interrupts.

PIE1: Peripheral Interrupt Enable Register 1

PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE

bit 7 bit 0

bit 7 **PSPIE :** Parallel Slave Port Read/Write Interrupt Enable bit

1 = Enables the PSP read/write interrupt

0 = Disables the PSP read/write interrupt

bit 6 **ADIE :** A/D Converter Interrupt Enable bit

1 = Enables the A/D interrupt

0 = Disables the A/D interrupt

bit 5 **RCIE :** USART Receive Interrupt Enable bit

1 = Enables the USART receiver interrupt

0 = Disables the USART receive interrupt

bit 4 **TXIE :** USART Transmit Interrupt Enable bit

1 = Enables the USART transmit interrupt

0 = Disables the USART transmit interrupt

bit 3 **SSPIE :** Master Synchronous Serial Port Interrupt Enable bit

1 = Enables the MSSP interrupt

0 = Disables the MSSP interrupt

bit 2 **CCP1IE :** CCP1 Interrupt Enable bit

1 = Enables the CCP1 interrupt

0 = Disables the CCP1 interrupt

bit 1 **TMR2IE :** TMR2 to PR2 Match Interrupt Enable bit

1 = Enables the TMR2 to PR2 match interrupt

0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE :** TMR1 Overflow Interrupt Enable bit

1 = Enables the TMR1 overflow interrupt

0 = Disables the TMR1 overflow interrupt

PIE2 : Peripheral Interrupt Enable Register 2

–	CMIE	–	EEIE	BCLIE	LVDIE	TMR2IE	ECCP1IE

bit 7 bit 0

bit 7 **Unimplemented :** Read as '0'

bit 6 **CMIE :** Comparator Interrupt Enable bit

1 = Enables the comparator interrupt

0 = Disables the comparator interrupt

bit 5 **Unimplemented :** Read as '0'

bit 4 **EEIE :** EEPROM Write Interrupt Enable bit

1 = Enabled

0 = Disabled

bit 3 **BCLIE :** Bus Colliion Interrupt Enable bit

1 = Enabled

0 = Disabled

bit 2 **LVDIE :** Low-Voltage Detect Interrupt Enable bit

1 = Enabled

0 = Disabled

bit 1 **TMR3IE :** TMR3 Overflow Interrupt Enable bit

1 = Enables the TMR3 overflow interrupt

0 = Disables the TMR3 overflow interrupt

bit 0 **ECCP1IE :** ECCP1 Interrupt Enable bit

1 = Enables the ECCP1 interrupt

0 = Disables the ECCP1 interrupt

PIE3: Peripheral Interrupt Enable Register 3

IRXIE	WAKIE	ERRIE	TXB2IE	TXB1IE	TXB0IE	RXB1IE	RXB0IE

bit 7 bit 0

bit 7 **IRXIE :** Invalid CAN Message Received Interrupt Enable bit

1 = Enables the invalid CAN message received interrupt

0 = Disables the invalid CAN message received interrupt

bit 6 **WAKIE :** Bus Activity Wake-up Interrupt Enable bit

1 = Enables the bus activity wake-up interrupt

0 = Disables the bus activity wake-up interrupt

bit 5 **ERRIE :** CAN bus Error Interrupt Enable bit

1 = Enables the CAN bus error interrupt

0 = Disables the CAN bus error interrupt

bit 4 **TXB2IE :** Transmit Buffer 2 Interrupt Enable bit

1 = Enables the Transmit Buffer 2 interrupt

0 = Disables the Transmit Buffer 2 interrupt

bit 3 **TXB1IE :** Transmit Buffer 1 Interrupt Enable bit

1 = Enables the Transmit Buffer 1 interrupt

0 = Disables the Transmit Buffer 1 interrupt

bit 2 **TXB0IE :** Transmit Buffer 0 Interrupt Enable bit

1 = Enables the Transmit Buffer 0 interrupt

0 = Disables the Transmit Buffer 0 interrupt

bit 1 **RXB1IE :** Receive Buffer 1 Interrupt Enable bit

1 = Enables the Receive Buffer 1 interrupt

0 = Disables the Receive Buffer 1 interrupt

bit 0 **RXB0IE :** Receive Buffer 0 Interrupt Enable bit

1 = Enables the Receive Buffer 0 interrupt

0 = Disables the Receive Buffer 0 interrupt

IPR Registers

- The Interrupt Priority (IPR) registers contain the individual priority bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are three Peripheral Interrupt Priority registers (IPR1, IPR2 and IPR3).
- The operation of the priority bits requires that the Interrupt Priority Enable bit (IPEN) be set.

IPR1: Peripheral Interrupt Priority Register 1

PSPIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP

bit 7 bit 0

bit 7 **PSPIP :** Parallel Slave Port Read/Write Interrupt Priority bit

1 = High priority

0 = Low priority

bit 6 **ADIP :** A/D Converter Interrupt Priority bit

1 = High priority

0 = Low priority

bit 5 **RCIP :** USART Transmit Interrupt Priority bit

1 = High priority

0 = Low priority

bit 4 **TXIP :** USART Transmit Interrupt Priority bit

1 = High priority

0 = Low priority

bit 3 **SSPIP :** Master Synchronous Serial Port Interrupt Priority bit

1 = High priority

0 = Low priority

bit 2 **CCP1IP :** CCP1 Interrupt Priority bit

1 = High priority

0 = Low priority

bit 1 **TMR2IP :** TMR2 to PR2 Match Interrupt bit

1 = High priority

0 = Low priority

bit 0 **TMR1IP :** TMR1 Overflow Interrupt Priority bit

1 = High priority

0 = Low priority

IPR2 : Peripheral Interrupt Priority Register 2

–	CMIP	–	EEIP	BCLIP	LVDIP	TMR3IP	ECCP1IP

bit 7 bit 0

bit 7 **Unimplemented :** Read as '0'

bit 6 **CMIP :** Comparator Interrupt Priority bit

1 = High priority

0 = Low priority

bit 5 **Unimplemented :** Read as '0'

bit 4 **EEIP :** EEPROM Write Interrupt Priority bit

1 = High priority

0 = Low priority

bit 3 **BCLIP :** Bus Collision Interrupt Priority bit

1 = High priority

0 = Low priority

bit 2 **LVDIP :** Low-Voltage Detect Interrupt Priority bit

1 = High priority

0 = Low priority

bit 1 **TMR3IP :** TMR3 Overflow Interrupt Priority bit

1 = High priority

0 = Low priority

bit 0 **ECCP1IP :** ECCP1 Interrupt Priority bit

1 = High priority

0 = Low priority

IPR3 : Peripheral Interrupt Priority Register 3

IRXIP	WAKIP	ERRIP	TXB2IP	TXB1IP	TXB0IP	RXB1IP	RXB0IP

bit 7 bit 0

bit 7 **IRXIP :** Invalid Message Received Interrupt Priority bit

1 = High priority

0 = Low priority

bit 6 **WAKIP :** Bus Activity Wake-up Interrupt Priority bit

1 = High priority

0 = Low priority

bit 5 **ERRIP :** CAN bus Error Interrupt Priority bit

1 = High priority

0 = Low priority

bit 4 **TXB2IP :** Transmit Buffer 2 Interrupt Priority bit

1 = High priority

0 = Low priority

bit 3 **TXB1IP :** Transmit Buffer 1 Interrupt Priority bit

1 = High priority

0 = Low priority

bit 2 **TXB0IP :** Transmit Buffer 0 Interrupt Priority bit

1 = High priority

0 = Low priority

bit 1 **RXB1IP :** Receive Buffer 1 Interrupt Priority bit

1 = High priority

0 = Low priority

bit 0 **RXB0IP :** Receive Buffer 0 Interrupt Priority bit

1 = High priority

0 = Low priority

RCON Register

- The Reset Control (RCON) register contains the IPEN bit which is used to enable prioritized interrupts.

IPEN	–	–	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$

bit 7 bit 0

bit 7 **IPEN :** Interrupt Priority Enable bit

1 = Enable priority levels on interrupts

0 = Disable priority levels on interrupts (PIC 16CXXX Compatibility mode)

bit 6-5 **Unimplemented :** Read as '0'

bit 4 $\overline{RI}$: RESET Instruction Flag bit

bit 3 $\overline{TO}$: Watchdog Time-out Flag bit

bit 2 $\overline{PD}$: Power-down Detection Flag bit

bit 1 $\overline{POR}$: Power-on Reset Status bit

bit 0 $\overline{BOR}$: Brown-out Reset bit

Priority of Interrupts

- There are three Peripheral Interrupt Priority registers (IPR1, IPR2 and IPR3).
- The Interrupt Priority (IPR) registers contain the individual priority bits for the peripheral interrupts. They specify the priority of a particular request either High or Low.

- Priority Bit = 1 High Priority
- Priority Bit = 0 Low Priority

- The high priority signals are located at program memory location 0008h and the low priority signals at 0018h.
- If the priority is disabled, by default, all the interrupts branch to 0008h memory location.
- Once the execution reaches the ISR, the interrupt is determined by polling the interrupt flags and associated code sequence is executed. After execution of code, the flag bit must be cleared in software to avoid recursion.

Programming of Timer using Interrupts

- The timer interrupt is generated when the timer register overflows from FFh to 00h in 8-bit mode or FFFFh to 0000h in 16-bit mode. This overflow sets the TMRxIF bit. The timer interrupt can be masked by clearing the TMRxIE bit.
- The TMRxIF bit must be cleared in software by the Timer module Interrupt Service Routine before re-enabling this interrupt.
- **Note :** x may take value : 0, 1, 2, 3.
- shows the timers and their corresponding interrupt enable flag bits and interrupt flag bits.

Timer	Interrupt Enable		Interrupt Flag	
	Bit	Register	Bit	Register
Timer 0	TMR0IE	INTCON	TMR0IF	INTCON
Timer 1	TMR1IE	PIE1	TMR1IF	PIR1
Timer 2	TMR2IE	PIE1	TMR2IF	PIR1
Timer 3	TMR3IE	PIE2	TMR3IF	PIR2

Steps for Programming PIC18 Timer using Interrupt

1. Enable GIE, PEIE, TMRxIE.
2. Configure the TxCON register.
3. Clear TMRxIF Timer interrupt flag.
4. Load the count in Timer register TMRx
5. Set TMRxON to start the Timer operation.
6. When TMRxIF = 1, code will jump to ISR to execute it, and after execution, control returns to the main program.

Write a C program for the PIC18 to generate a square wave of 25 Hz frequency on RB4 using Timer0 ISR. Assume XTAL = 10 MHz.

Solution :

Sol. : **Given :** F_{OSC} = 10 MHZ, No Prescaler

$$F_{TIMER} = \frac{F_{OSC}}{4} = \frac{10}{4} = 2.5 \text{ MHZ}$$

$$\text{Period} = \frac{1}{F_{TIMER}} = \frac{1}{2.5 \text{ MHz}} = 0.4 \text{ µs}$$

$$T = \frac{1}{25 \text{Hz}} = 0.04 \text{ s}$$

$$T_{ON} = T_{OFF} = \frac{T}{2} = 0.02 \text{ s}$$

$$\text{Count} = \frac{\text{Desired Delay}}{\text{Timer Priod}} = \frac{0.02 \text{ s}}{0.4 \text{ µs}} = 50000$$

The value to be loaded in 16-bit Timer register is :

65536 – Count = 65536 – 50000 = 15536 = 3CB0H

```
#include <P18F458.h>
void Timer0_ISR(void);
void CHK_ISR (void);
#define   PORTBit  PORTBbits.RB4
#pragma interrupt CHK_ISR
void CHK_ISR (void)
{
      If (INTCONbits.TMR0IF = 1)  // If Timer0 caused interrupt execute Timer0_ISR
            Timer0_ISR();
}
#pragma code HiPrio_Int = 0x08         // High priority interrupt
void HiPrio_Int (void)

 {
      _asm
      GOTO CHK_ISR      ; High priority interrupt land at address 0008H. To avoid using
                        ; limited memory space allocated to IVT and to have more space
                        ; for ISR we use GOTO instruction to CALL ISR written at
                        ; different place
      _endasm
 }
 #pragma code
```

```
void main(void)
{
      TRISBbits.TRISB4=0;              // configure pin 4 of Port B as output
      INTCONbits.GIE = 1 ;             // Enable all interrupts globally
      INTCONbits.PEIE = 1 ;            // Enable all peripheral interrupts
      T0CON=0x08;                      // Timer0, 16 bit mode, no prescaler
      INTCONbits.TMR0IF = 0 ;          // Clear Timer0 interrupt flag bit
      TMR0H=0x3C;                      // Load Higher byte in TMR0H
      TMR0L= 0xB0;                     // Load Lower byte in TMR0L
      INTCONbits.TMR0IE = 1 ;          // Enable Timer0 interrupt
      T0CONbits.TMR0ON = 1;            // Start the Timer0
      while(1);                        // Wait for the interrupt
}

void Timer0_ISR(void)
 {
    PORTBit = ~PORTBit;               // Toggle the bit RB4
    TMR0H=0x3C;                       // Load Higher byte in TMR0H
     TMR0L= 0xB0;                     // Load Lower byte in TMR0L
     INTCONbits.TMR0IF = 0 ;          // Clear Timer0 interrupt flag bit
 }
```

Programming of External Hardware Interrupts

- PIC18 microcontroller has three external hardware interrupts - INT0, INT1, and INT2. They are on PORTB pins RB0, RB1, and RB2.
- These interrupts are edge-triggered interrupts i.e. triggered by either a rising edge or by a falling edge.

INTCON2 : Interrupt Control Register

$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	-	TMR0IP	-	RBIP

bit7 bit 0

INTEDG0 : External Interrupt 0 Edge select bit
1= Interrupt on Rising Edge
0= Interrupt on Falling Edge

INTEDG1 : External Interrupt 1 Edge select bit
1= Interrupt on Rising Edge
0= Interrupt on Falling Edge

INTEDG2 : External Interrupt 2 Edge select bit
1= Interrupt on Rising Edge
0= Interrupt on Falling Edge

Interrupt	Interrupt Enable		Interrupt Flag	
	Bit	Register	Bit	Register
INT0 (RB0)	INT0IE	INTCON	INT0IF	INTCON
INT1 (RB1)	INT1IE	INTCON3	INT1IF	INTCON3
INT2 (RB2)	INT2IE	INTCON3	INT2IF	INTCON3

Programming of Serial Communication Interrupt

- The UART Interrupt is used to detect the completion of data transmission or reception without continuously polling the flag bit, thus saving processor time.
- The interrupt is detected with the help of the **TXIF** and **RCIF** bits.
- To use these interrupts, GIE (Global Interrupt Enable), PEIE (Peripheral Interrupt Enable) needs to be enabled along with RCIE (Receive Interrupt Enable) and TXIE (Transmit Interrupt Enable).
- shows the serial communication interrupts and their corresponding interrupt enable flag bits and interrupt flag bits.

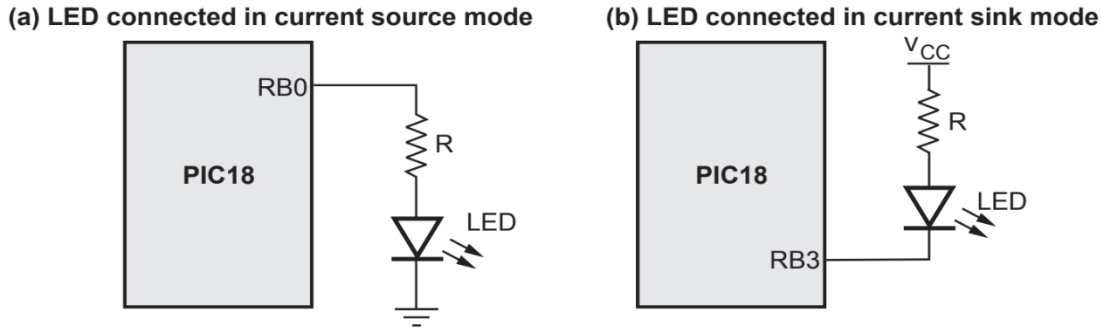
Interrupt	Interrupt Enable		Interrupt Flag	
	Bit	Register	Bit	Register
TXIF	TXIE	PIE1	TXIF	PIR1
RCIF	RCIE	PIE1	RCIF	PIR1

Steps for Programming PIC18 USART using Interrupt

1. Initialize the Baud Rate by loading value to the SPBRG register.
2. Set bit SPEN in the RCSTA for Serial Port enable.
3. Set bit BRGH in the TXSTA for low or high speed.
4. Clear bit SYNC in the TXSTA registers for asynchronous communication.
5. Set bit TXEN in the TXSTA register to enable transmission.
6. Set bit CREN in the RCSTA register to enable reception.
7. Clear RCIF and TXIF
8. Enable GIE, PEIE, RCIE and TXIE for ISR.

Interfacing of LED

- Since the current sinking and sourcing capacities in the PIC18 family is about 25 mA, we can drive LED using PORT pins in two ways, as shown in
  - Current sourcing mode
  - Current sinking mode



For current source mode $R = \frac{V_{out} - V_{LED}}{I_{LED}}$

For current sink mode $R = \frac{V_{CC} - V_{LED}}{I_{LED}}$

where V_{LED} is the voltage across LED

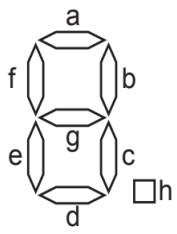
I_{LED} is the current through LED

V_{out} is the voltage at output pin

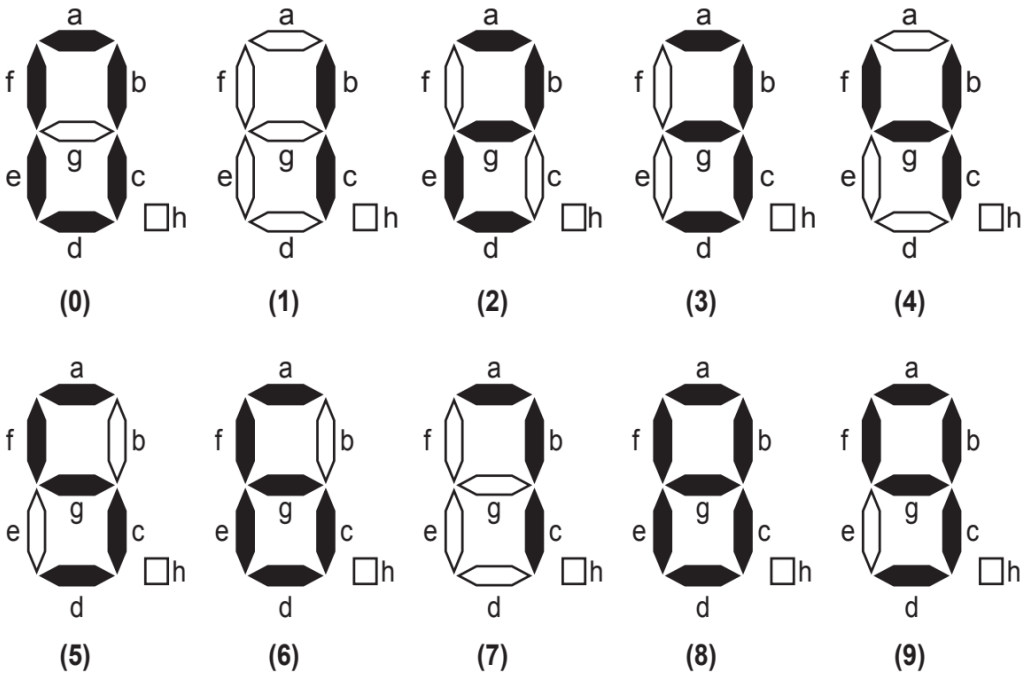
- Since I_{LED} is the range of 10-15 mA, V_{LED} = 1.5 V, V_{CC} = 5 V, $V_{out} \approx$ 5 V, suitable value for R is in the range of 270 Ω - 330 Ω.

Interfacing Multiplexed 7-Segment Display

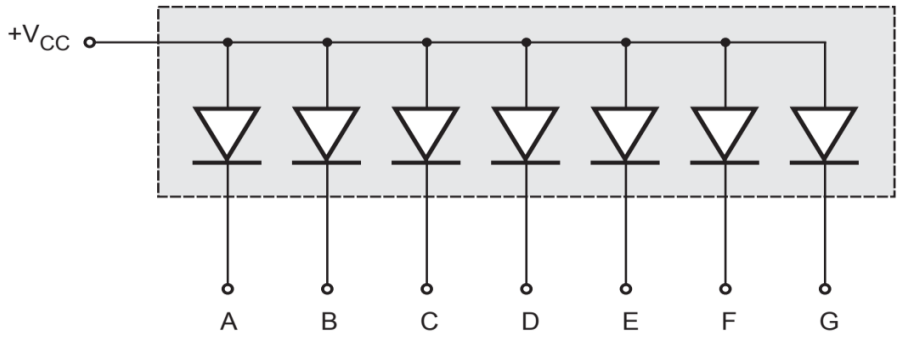
- Seven-segment display : Seven-segment displays are generally used as numerical indicators and consists of a number of LEDs arranged in seven-segments as shown in the Fig.
- Any number between 0 and 9 can be indicated by lighting the appropriate segments.
- The seven-segments are labelled a to g and dot is labelled as h. By forward biasing different LED segments, we can display the digits 0 through 9. For instance, to display 0, we need to light up a, b, c, d, e, and f. To light up 5, we need segments a, f, g, c and d.



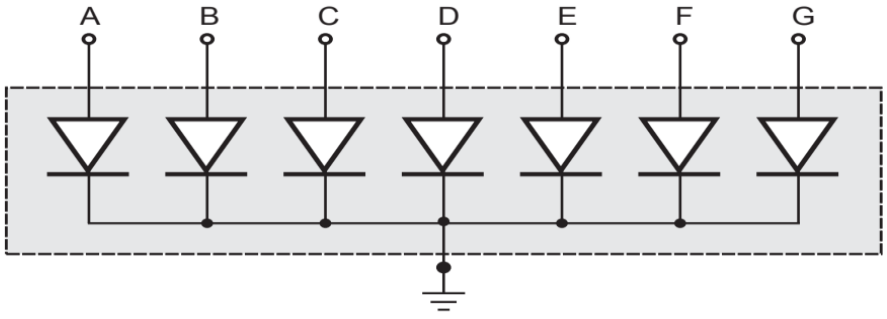
Seven-segment



- These 7-segment displays are of two types :
  - Common anode type
  - Common cathode type
- In common anode, all anodes of LEDs are connected together as shown in Fig. and in common cathode, all cathodes are connected together, as shown in Fig. .



(a) Common anode type

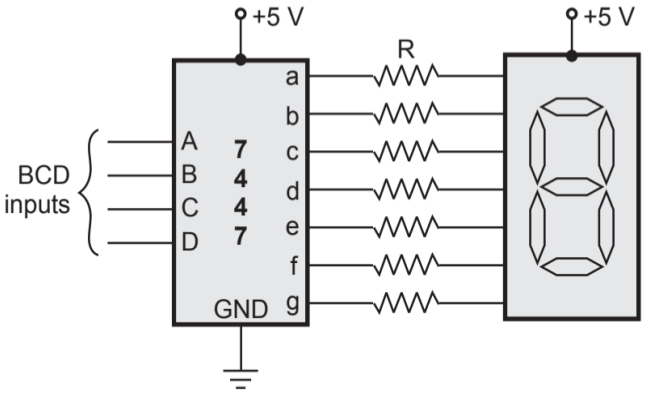


(b) Common cathode type

Interfacing LED Displays

Static display

- shows a circuit to drive a single, seven-segment, common anode LED display. For common anode, when anode is connected to positive supply, a low voltage is applied to a cathode to turn it on. Here, BCD to seven-segment decoder, IC 7447 is used to apply low voltages at cathodes according to BCD input applied to IC 7447.



Circuit for driving single seven-segment LED display

- To limit the current through LED segments resistors are connected in series with the segments. This circuit connection is referred to as a static display because current is being passed through the display at all times.
- The value of the resistor in series with the segment can be calculated as follows :
  We know, V_{CC} – drop across LED segment – IR = 0

  Drop across LED segment is nearly 1.5 V.

$$\therefore \quad IR = V_{CC} - 1.5\ V = 5 - 1.5\ V = 3.5\ V$$

- Each LED segment requires a current of between 5 and 30 mA to light. Let's assume that current through LED segment is 15 mA.

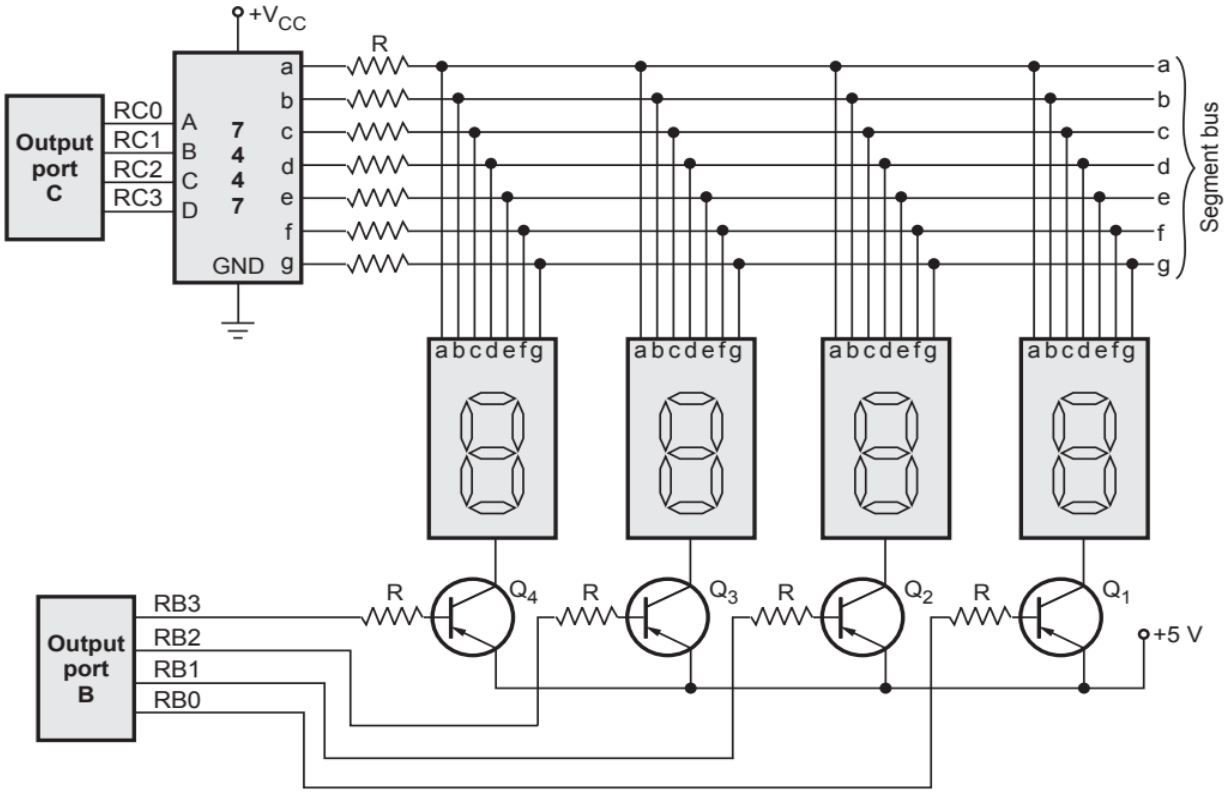
$$\therefore \quad R = \frac{3.5\ V}{15\ mA} = 233\ \Omega$$

- In practice, the voltage drop across the LED and the output of IC 7447 are not exactly predictable and the exact current through the LED is not critical as long as we don't exceed its maximum current rating. Therefore, a standard value 220 Ω can be used.
- The static display circuits work well for driving just one or two LED digits. However, these circuits are not suitable for driving more LED digits, say 8 digits. When there are more number of digits, the first problem is power consumption. For worst case calculations, assume that all eight digits with all segments are lit. Therefore, worst case current required is

  I = 8 (digits) × 7 (segment) × 15 mA (current per segment) = 840 mA
- A second problem of the static approach is that each display digit requires a separate BCD to 7-segment decoder.

Multiplexed display

- To solve the problems of the static display approach, multiplexed display method is used the 4 seven-segment displays connected using multiplexed method. Here, common anode seven-segment LEDs are used.
- Anodes are connected to +5 V through transistors. Cathodes of all seven-segments are connected in parallel and then to the output of 7447 IC through resistors. Looking at the Fig. the question may occur in our mind that, "Aren't all of the digits going to display the same number?" The answer is that they would show the same number only if all the digits are turned on at the same time. However, in multiplexed display the segment information is sent for all digits on the common lines (output lines of IC 7447), but only one display digit is turned on at a time. The PNP transistors connected in series with the common anode of each digit act as an ON and OFF switch for that digit. Here's how the multiplexing process works.



- The BCD code for digit 1 is first output from PORT C, to the IC 7447. The IC 7447, BCD to seven-segment decoder outputs the corresponding seven-segment code on the segment bus lines. The transistor Q_1 connected to digit 1 is then turned on by outputting a low to that bit of PORT B. All of the rest of the bits of PORTB are made high to ensure no other digits are turned on. After 2 ms, digit 1 is turned OFF outputting all highs to PORTB. The BCD code for digit 2 is then output to the PORT C, and bit pattern to turn on digit 2 is output on PORT B. After 2 ms, digit 2 is turned off and the process is repeated for digit 3 and digit 4. After completion of turn for each digit, all the digits are lit again in turn.

- With 4 digits and 2 ms per digit we get back to digit 1 every 8 ms or about 125 times a second. This refresh rate is fast enough that, to our eye and due to persistence of vision all digits will appear to be lit all the time.
- In multiplexed display, the segment current is kept in between 40 mA to 60 mA so that they will appear as bright as they would if not multiplexed. Even with this increased segment current, multiplexing gives a large saving in power and hardware components.

shows the multiplexed 8-digit 7-segment LED display connected in PIC18 system using port C and port B. In this circuit port C and port B are used as a latch, i.e. output port. Port C provides the segment data inputs to the display and port B provides a means of selecting a display position at a time for multiplexing the displays. Here, instead of BCD to seven-segment decoder (IC 7447) transistors are used to drive the LED segments. Due to this we can also display HEX characters on the display; however in this case we have to send the proper 7-segment code of a particular digit that is to be displayed on the port C.

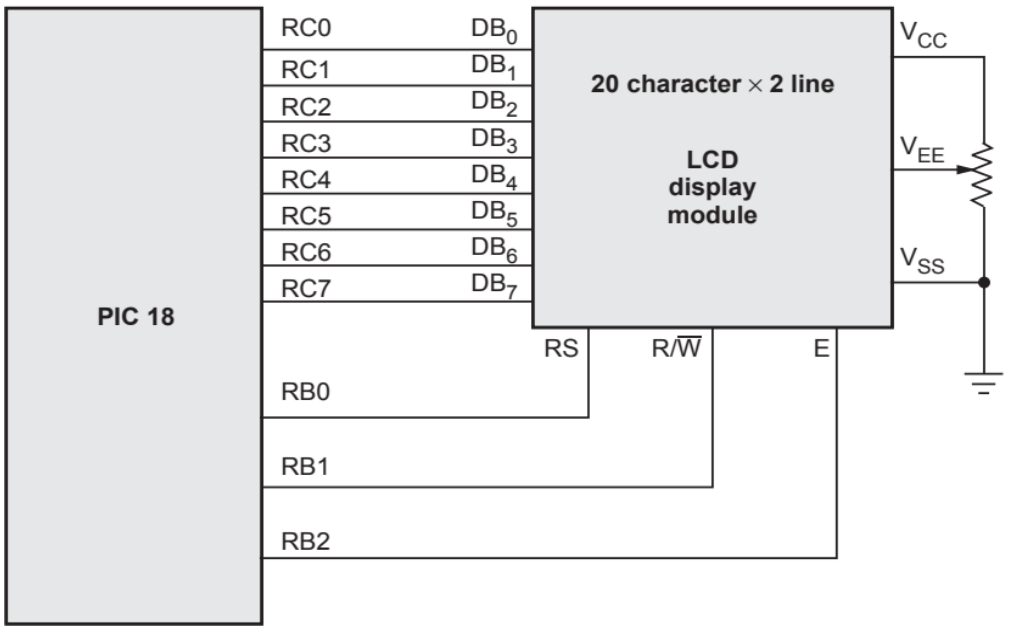
Number	g	f	e	d	c	b	a	Hexadecimal
0	0	1	1	1	1	1	1	3F
1	0	0	0	0	1	1	0	06
2	1	0	1	1	0	1	1	5B
3	1	0	0	1	1	1	1	4F
4	1	1	0	0	1	1	0	66
5	1	1	0	1	1	0	1	6D
6	1	1	1	1	1	0	1	7D
7	0	0	0	0	1	1	1	07
8	1	1	1	1	1	1	1	7F
9	1	1	0	1	1	1	1	6F

Interfacing of LCD

- Now-a-days, many LCD modules are available which have built-in drivers for LCD and interfacing circuitry to interface them to microprocessor/microcontroller systems. These LCD modules allow display of characters as well as numbers. They are available in 16 × 2, 20 × 1, 20 × 2, 20 × 4 and 40 × 2 sizes. The main advantage of using LCD is modules is that they require very less power. The first figure represents number of character in each line and second figure represents number of lines the display has. In this section, we see the interfacing of 20 × 2 LCD display module to microprocessor/microcontroller system through 8255. In this module the display is organized as two lines, each of 20 characters. The module has 14-pins.

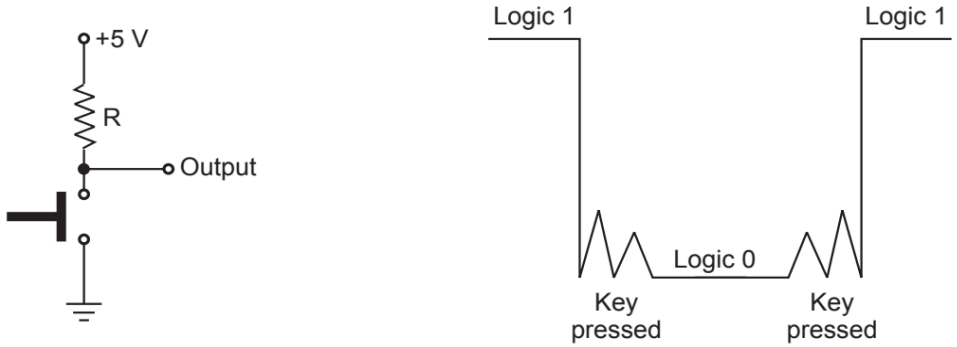
Pin	Symbol	I/O	Description
1	V_{SS}	-	Ground
2	V_{CC}	-	+ 5 V power supply
3	V_{EE}	-	Used for controlling LCD contrast
4	RS	I	Register select input is used to select either of the two available registers in the module : Data register or command register. When RS = 0 Data register is selected. When RS = 1 Command register is selected.
5	$R/\overline{W}$	I	Allows the user to write information to the LCD or read information from it. For reading $R/\overline{W}$ = 1 and for writing $R/\overline{W}$ = 0
6	E	I	This pin is used by LCD to latch information available at its data pins.
7-14	DB_0-DB_7	I/O	This 8-bit data bus is used to send information to the LCD or read the contents of the internal registers of the LCD.

- The Fig. shows the interfacing of a 20 character × 2 line LCD module with the PIC 18. As shown in the Fig. the data lines are connected to the PORTC of PIC 18 and control lines RS, $R/\overline{W}$ and E are driven by RB0, RB1 and RB2 lines of PORTB, respectively. The voltage at V_{EE} pin is adjusted by a potentiometer to adjust the contrast of the LCD.



Interfacing Key Board (4 × 4 Matrix)

- For interfacing keyboard to the microprocessor/microcontroller based systems, usually push button keys are used. These push button keys when pressed, bounces a few times, closing and opening the contacts before providing a steady reading, as shown in the Fig. Reading taken during bouncing period may be faulty. Therefore, microprocessor/microcontroller must wait until the key reach to a steady state; this is known as **key debounce**.

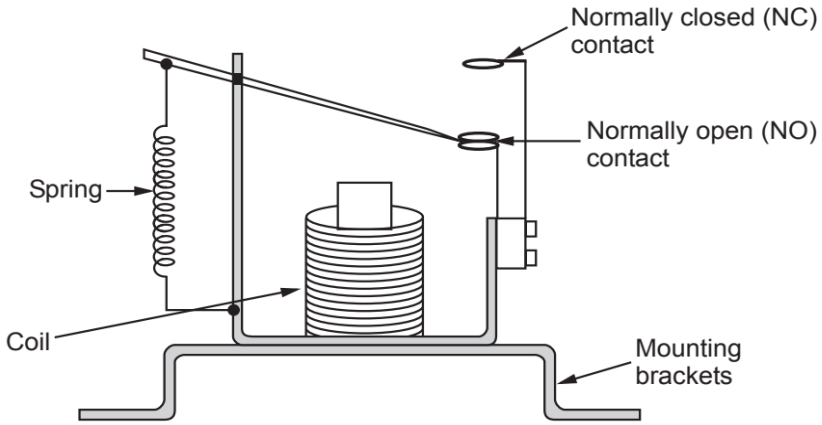


Bouncing of key switch

Key	Keycode							
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
K_1	1	1	1	1	1	1	1	0
K_2	1	1	1	1	1	1	0	1
K_3	1	1	1	1	1	0	1	1
K_4	1	1	1	1	0	1	1	1
K_5	1	1	1	0	1	1	1	1
K_6	1	1	0	1	1	1	1	1
K_7	1	0	1	1	1	1	1	1

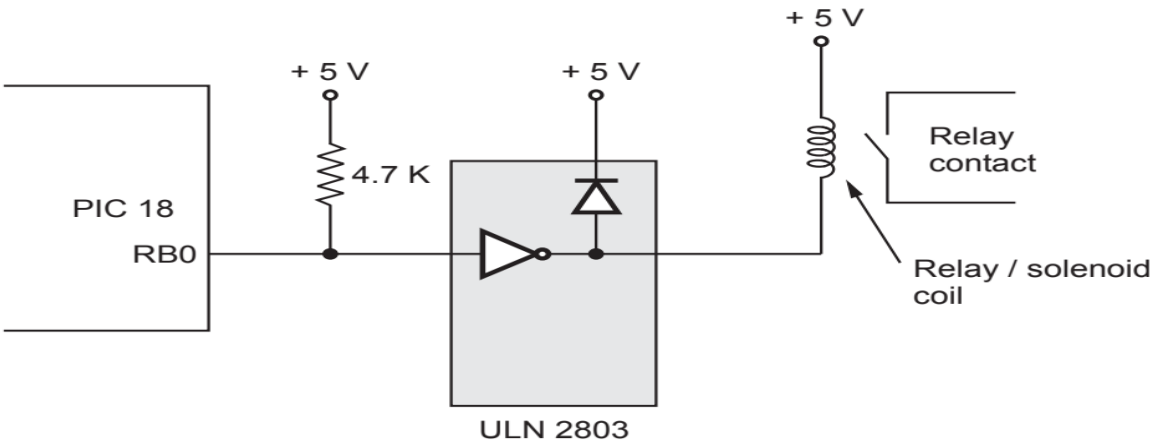
Interfacing Relay

- To control ON/OFF operation of ac devices we can use electromagnetic relay. This relay has both normally open and closed contacts. When a current is passed through the coil of the relay, the switch arm is pulled down, opening the top contact and closing the bottom contact,



Electromagnetic relay

- The relay contacts are rated for maximum current of about 20 A - 25 A. These relays are also called mechanical relays because mechanical contact makes the circuit to ON or OFF. The mechanical relays having higher current ratings are sometimes called **contactors**.
- The output current of PIC18 microcontroller pins is not sufficient to drive the relay, or solenoid. The relay/solenoid coil needs around 50 mA - 100 mA to be energized and microcontroller's pin can provide this current. For this reason, we have to use driver such as the ULN2803 or a power transistor or a power MOSFET between the microcontroller and the relay/solenoid



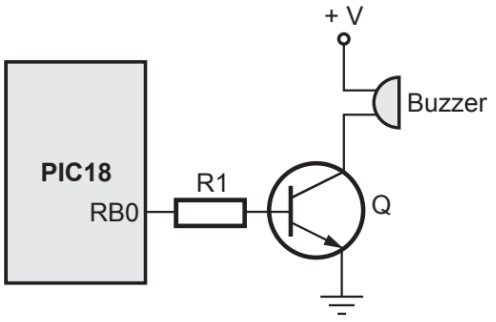
(a) Relay driving circuit

Interfacing Buzzer

- Buzzers are mainly divided into two types -
  - Active buzzers and
  - Passive buzzers
- Active buzzer produces a single type of sound or tone. It requires a dc power source for generating the sound or beep.
- On the other hand, passive buzzer can generate different sounds or beep. It depends upon the frequency which is provided to the buzzer. It required an ac power source for generating the sound or beep.
- the interfacing of a simple active buzzer with PIC18 microcontroller.

- Usually, buzzer takes more current and hence it could not possibly directly driven by microcontroller pin. In this condition, an NPN or PNP transistor is added in the circuit for safely turning ON or OFF the buzzer.

- The embedded C program given can be used to turn ON and turn OFF the buzzer



Interfacing of a simple active buzzer with PIC18 microcontroller

****