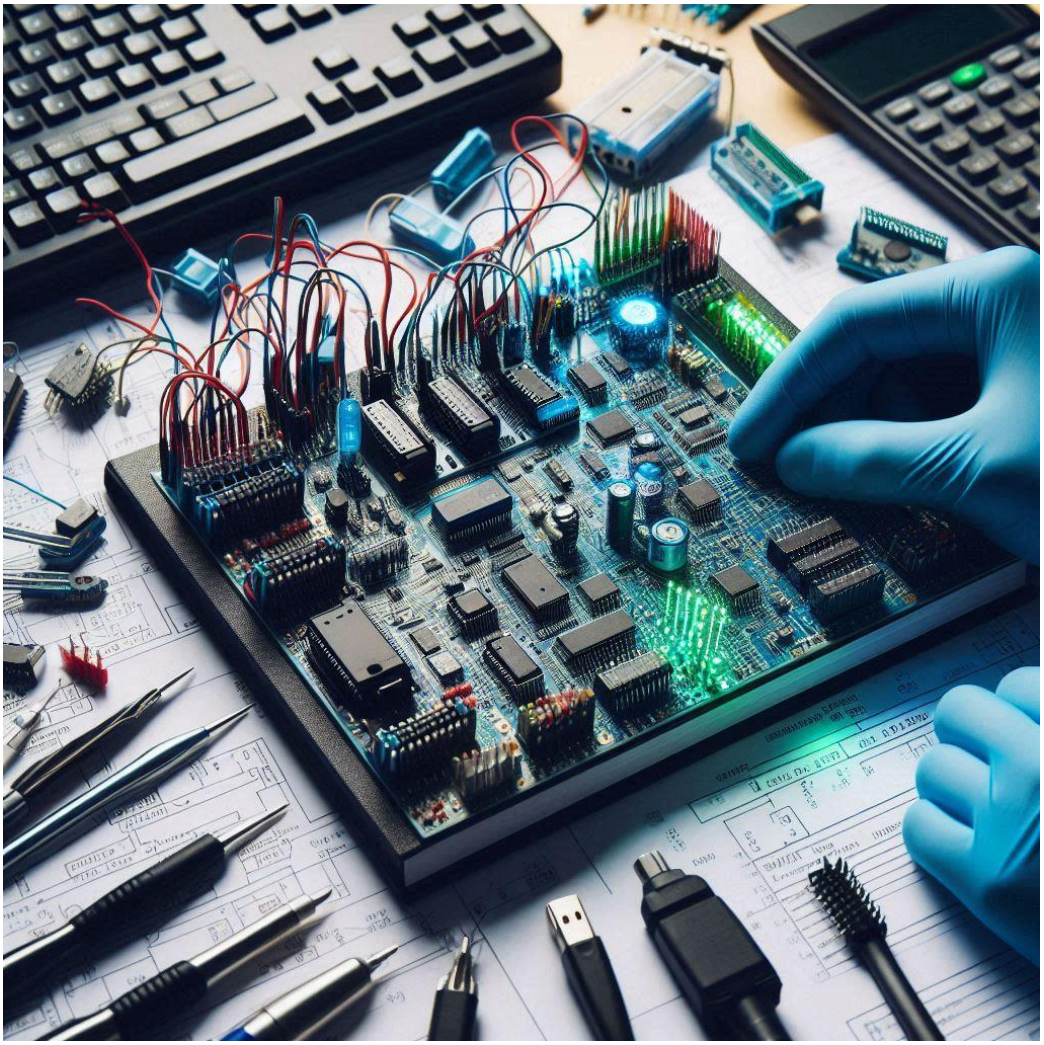
p

Microcontrollers and Embedded Electronics

Laboratory Manual-BECS 31421

Dr. P A N S Priyadharshana and Dr. W G C Kumarage



Department of Physics and Electronics
University of Kelaniya

Microcontrollers and Embedded Electronics
Laboratory Manual-BECS 31421

Dr. P A N S Priyadharshana and Dr. W G C Kumarage

Department of Physics and Electronics

Faculty of Science

University of Kelaniya

Kelaniya 11600

Sri Lanka

Table of Contents

Introduction

At the core of contemporary technology are microcontrollers, which serve as the command centers for innumerable embedded systems that drive everything from smart gadgets and industrial automation to consumer electronics and automobile systems. They are crucial in advancing the creation of intelligent, networked, and energy-efficient systems because of their adaptability and effectiveness. Gaining a solid foundation in areas like electronics, robotics, automation, and industrial systems requires an understanding of microcontroller programming and applications in addition to academic accomplishment.

The goal of the Microcontrollers and Embedded Electronics (BECS 31421) course is to provide students the academic understanding and real-world skills they need to succeed in this quickly changing industry. Students may develop, simulate, and implement microcontroller-based systems with the help of this lab manual, which offers organized, practical experiences. Students are guaranteed to go beyond theory and acquire the skills and methods required to succeed in embedded systems engineering because of the focus on practical problem-solving.

Microcontrollers make it possible to integrate and operate a wide range of systems and devices in today's technologically advanced world, from sophisticated industrial machinery to domestic appliances. Proficiency in microcontroller-based systems is highly regarded in a variety of businesses and is necessary for academic and research endeavors. Through the development of essential abilities in programming, system design, and embedded application development, this course equips students for success in the workplace.

Objectives

The primary aim of this course is to bridge the gap between theoretical knowledge and practical application, ensuring that students develop the technical skills needed to implement microcontroller solutions in real-world scenarios. Upon completing the course, students will be proficient in:

- Programming microcontrollers for various applications.
- Simulating and testing embedded systems using industry-standard tools.
- Interfacing microcontrollers with external devices such as sensors, displays, and actuators to create functional prototypes.
- Designing and implementing embedded systems using modular design principles.
- Building circuits with switches, LEDs, resistors, potentiometers, and liquid crystal displays.
- Synchronizing hardware and software inputs/outputs with peripherals like switches, sensors, motors, and LCDs.
- Debugging systems using tools like oscilloscopes, logic analyzers, and software instrumentation.

This comprehensive approach ensures that students acquire the skills necessary for a wide range of engineering applications, enabling them to confidently design and implement embedded systems.

Tools and Techniques

To successfully complete the lab exercises and develop a robust understanding of microcontroller applications, students will work with the following key tools:

- **MikroC Pro**
  A powerful Integrated Development Environment (IDE) tailored for PIC microcontroller programming. MikroC Pro simplifies the coding process through its intuitive interface and built-in libraries, making it accessible to both beginners and advanced users.

- **Proteus**
  An essential simulation tool for designing, simulating, and testing embedded systems before physical implementation. Proteus allows students to validate their designs in a virtual environment, reducing errors and saving time during the development process.

- **PicKit 3 Programmer**
  A critical hardware tool that enables students to upload their compiled programs from MikroC Pro or MPLAB IPE to the PIC16F628A microcontroller. This enables real-time testing and debugging of the embedded systems created during the lab sessions.

Students will be better equipped to handle the demands of both academic research and industrial applications by using these tools to obtain practical experience in the whole embedded system development process, from coding and simulation to hardware integration and testing.

Laboratory - 01
Introduction to Software Tools: MikroC, Proteus, and PicKit 3 Programmer

Objectives

1. To be familiar with MikroC Pro, PROTEUS, and PicKit 3 programmer tools.
2. To acquire the knowledge and skills necessary to complete a project using MikroC Pro and generate the corresponding hex file.
3. To learn how to create a simulation project in PROTEUS.
4. To learn how to program a PIC using MPLAB IPE and PicKit 3.

Introduction

This manual aims to provide a comprehensive understanding of three necessary tools commonly employed in microcontrollers and embedded systems: MikroC, Proteus, and PicKit 3 Programmer. MikroC, a versatile integrated development environment (IDE) for microcontroller programming, is developed by MikroElektronika and is widely regarded as one of the best compilers for beginners due to its default built-in functions tailored to commonly used tasks. Recognized for its user-friendly interface, extensive features, and smooth integration with PIC microcontroller development environments, MikroC significantly facilitates the efficient development of projects. Furthermore, its compatibility with Proteus simulator, a strong platform for designing and testing embedded systems, enables virtual testing of projects before real-world implementation. PicKit 3 Programmer assumes a vital role in programming microcontroller devices (PIC 10xxx, PIC 12xxx, PIC 14xxx and PIC 16xxx ect.), ensuring the smooth transfer of compiled code to hardware. For this laboratory manual PIC 16F628A, an 8 bit microcontroller is employed. Understanding the pin configuration is crucial for effective interfacing. Figure 1.1 illustrates the pin configuration of the PIC 16F628A microcontroller, including I/O pins, analog input pins, power supply pins, and communication interface pins. Through its inclusion in this manual, users will gain the necessary insights and proficiency to effectively utilize MikroC, Proteus, and PicKit 3 Programmer in their embedded systems projects.

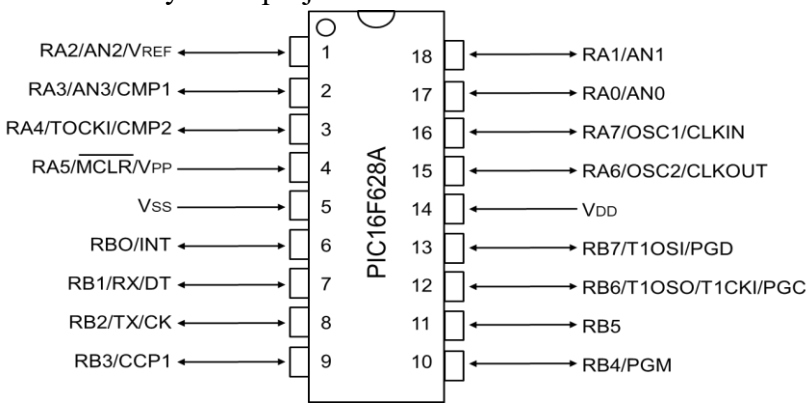


Figure 1.1: PIC16F628A PIC Microcontroller pin configuration.

- **Getting Started with MikroC Pro**

Follow the essential steps given below to initiate the MikroC programming process:

1. **Launch MikroC Pro for PIC**: Open your computer's MikroC Pro for PIC software.
2. **Create a New Project**: Navigate to the 'File' menu and select 'New Project'.
3. **Choose Project Type**: Select 'Standard project' and proceed by clicking 'Next'.
4. **Project Settings - New Project Wizard**: To begin, enter the project name and select the path where you have created the project folder. Specify the clock frequency, which denotes the oscillator frequency utilized with the microcontroller. For this project, we use the PIC 16F628A microcontroller with a 4 MHz crystal. After entering these details, tick the "Open Edit Project window to set Configuration bits" option. Click 'Next' to proceed.
5. **Configuration Bit**: This section allows you to incorporate configuration settings, such as Oscillator Selection, Watchdog Timer, and Power-up Timer. Choose the configuration settings as shown in Figure 1.2 and click 'Next'.

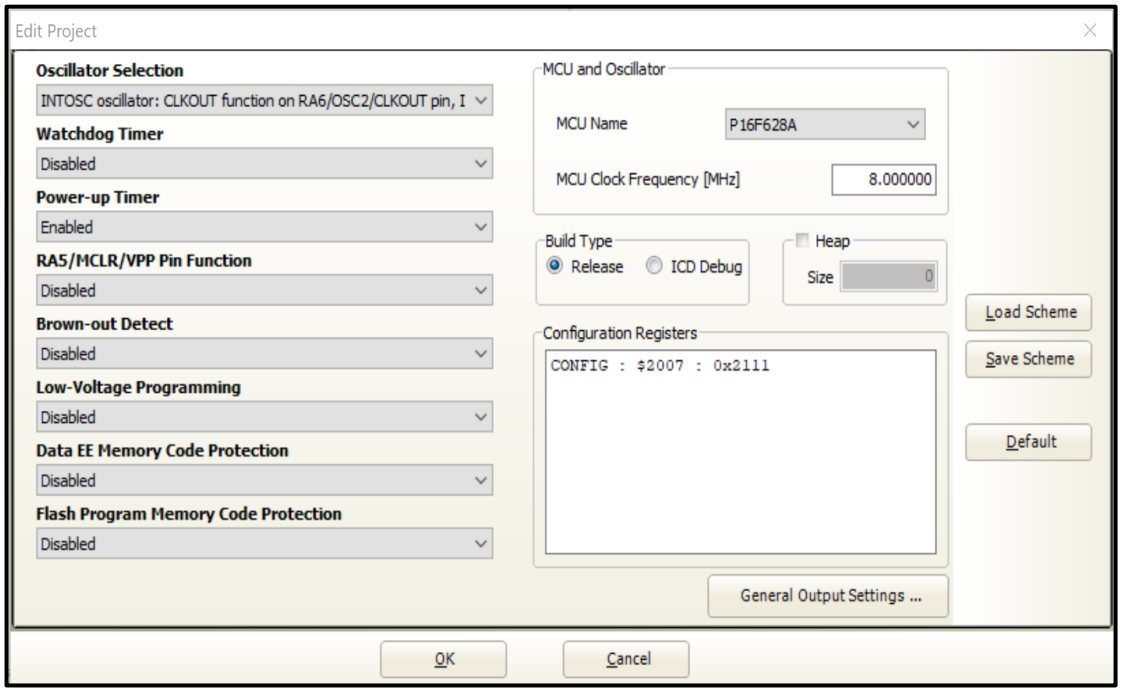


Figure 1.2: Configuration Bit.

6. **Finalizing the New Project Wizard**: Click on the 'Save' icon in the toolbar to finalize the New Project Wizard and proceed.

7. **Editor Interface**: Now, you will be directed to the editor interface, where you can start writing the MikroC code for your project.

8. **Library functions**: Library functions in MikroC Pro play a crucial role when interfacing inputs and peripherals with a PIC microcontroller by simplifying the process of sending commands and data, managing timing requirements, and ensuring proper communication between the microcontroller and the devices. MikroC Pro offers a comprehensive suite of library functions that facilitate the integration of various inputs and peripherals, including LCDs, keypads, ADC modules, and UART communication, among others. For instance, the integration and configuration of appropriate library functions for an LCD display can be achieved through the following steps.

    I. **Include the LCD Library:** Navigate to the 'Library Manager' tab on the right side of the MikroC interface. Check the box next to the "LCD" library to include it in your project.

    II. **Find the LCD Configuration Code:** Look for an example code provided with MikroC that demonstrates interfacing an LCD with a PIC microcontroller (Ex.NO:04). In the example code, you will typically find the configuration code that specifies the pin connections between the LCD and the microcontroller.

    III. **Add the LCD Configuration Code to Your Project:** Once you locate the configuration code in the example, copy it. Open your source code file (e.g., **main.c**) where you want to configure the LCD. Paste the copied configuration code into your file, typically at the beginning before the "**main"** function or any other function where you plan to initialize the LCD.

- **Getting Started with PROTEUS**

To initiate the simulation process in Proteus, follow these essential steps:

1. **Open Proteus**: Launch the Proteus software application on your computer.
2. **Create or Open Project**: Create a new project or access a previously saved project file.
3. **New Project Wizard**: To begin, enter the project name and select the path where you have created the project folder. Click 'Next' to proceed.
4. **Design Template**: Select the necessary page setting. These page settings typically include parameters such as paper size, orientation (portrait or landscape), margins, and other printing options. Click 'Next' to proceed.
5. **PCB Layout**: Users are prompted to select the appropriate PCB layout option. One option is to choose a specific PCB layout from available templates, while the other option is to opt-out and select "Do not create a PCB layout". In this instance, we will choose the latter option, indicating that we do not intend to create a PCB layout for our current project. Click 'Next' to proceed.

6. **Firmware Project**: You will encounter three options: “No Firmware Project”, “Create Firmware Project”, and “Create Flowchart Project”. Opting for the first choice, “No Firmware Project” means you are not starting a separate firmware project with your current circuit design. Picking this option shows you do not need a standalone firmware project immediately. After choosing “No Firmware Project”, click “Next” to continue your project without adding a firmware project into the process. Then, you will see a summary page where you can review all your settings. Click “Finish” to go to the workspace (Figure 1.3).
7. **Place Components**: Place necessary components onto the workspace. Ensure you have included the PIC microcontroller, any required peripherals, power sources, and other components relevant to your project.
8. **Wiring Connections**: Connect the components appropriately using wires, ensuring that all connections are correctly established as per your circuit design.
9. **Compile Code**: Ensure your code is compiled and the “HEX file” is loaded onto the microcontroller in Proteus.
10. **Run Simulation**: Initiate the simulation process by clicking on Proteus’s “Play” or “Run” button. This action will start simulating your PIC microcontroller circuit.
11. **Observe Results**: Monitor your circuit’s behavior within the Proteus simulation environment. You can observe how the microcontroller interacts with other components and peripherals per your programmed logic.

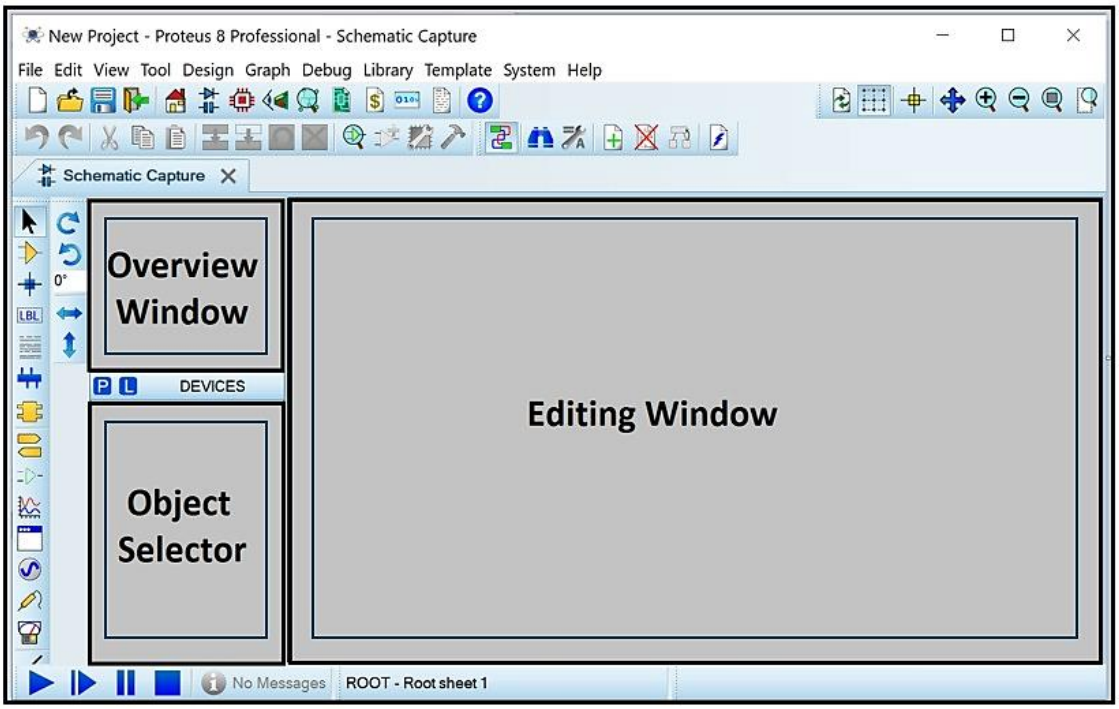


Figure 1.3: Workspace window.

12. **Debug (if necessary)**: If you encounter any unexpected behavior or errors during simulation, use Proteus's debugging tools to identify and rectify issues within your circuit or code.

13. **Analyze and Iterate**: Analyze the simulation results and adjust your circuit design or code. Iterate through the simulation process until you achieve the desired functionality and performance.

- **Getting Start with Pic Kit 3**

The basic steps to burn a PIC microcontroller using MPLAB IPE (Integrated Programming Environment) are given as follows.

1. **Download and Install MPLAB IPE (V3):** If you haven't already, download and install MPLAB IPE from the official Microchip website. Ensure that you have the appropriate drivers installed for your programmer hardware.

2. **Connect Hardware:** Connect your PIC programmer hardware (such as Pickit 3) to your computer via USB. Connect the appropriate cables from the programmer to your PIC microcontroller on your development board or target system.

3. **Open MPLAB IPE:** Launch MPLAB IPE software on your computer (Figure 1.4).

4. **Configuration:** Navigate to the settings menu and opt for advanced mode, then log in using the default password.

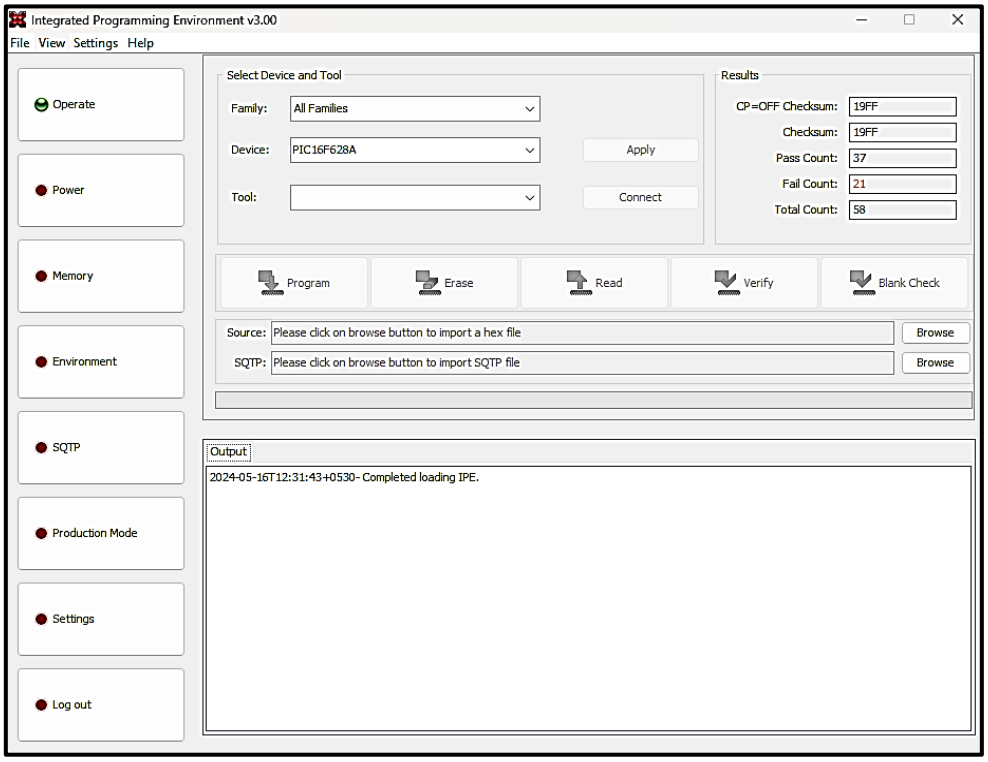


Figure 1.4: MPLAB IPE v3.00 programmer interface.

5. **Power Configuration**: Access the power settings and designate the voltage (VDD) as 4.5V. Additionally, ensure to enable the "Power target Circuit from Tool" option. Finally, click on "Operate" to proceed.

6. **Select Device**: In MPLAB IPE, go to the "Device" tab and select the specific PIC microcontroller you want to program from the dropdown menu. Ensure that the selected device matches the one on your development board.

7. Select Programmer: In the "Tool" tab, select the programmer hardware you are using from the dropdown menu. For example, if you are using Pickit 3, select "Pickit 3" from the list.

8. **Load Hex File**: Click on the "File" tab and select "Import Hex" to load the hex file containing your program code. Navigate to the location where your hex file is saved and select it.

9. **Set Configuration Bits (Optional)**: If necessary, configure the device configuration bits according to your application requirements. This step may vary depending on your specific PIC microcontroller.

10. **Program Device**: Once everything is set up correctly, click on the "Program" button in MPLAB IPE to start programming the PIC microcontroller. The software will then erase the device, program the hex file onto the microcontroller, and verify the programming.

11. **Verify Programming**: After programming is complete, MPLAB IPE will automatically verify the contents of the programmed memory against the hex file to ensure successful programming.

12. **Disconnect Hardware**: Once programming and verification are successful, disconnect the programmer hardware from your computer and the target system.

Note: During your laboratory session, detailed guidance will be provided on the proper connection of the PIC microcontroller to the programmer and the requisite jumper settings on the programmer.

Laboratory - 02
Introduction to Input and Output in PIC Programming

Objectives

1. To learn how to configure input and output pins on the PIC16F628A microcontroller.
2. To understand how to set pins as inputs or outputs using the TRIS register.
3. To develop skills in interfacing with external devices, such as switches and LEDs, through input and output operations.

Apparatus

Software: MikroC PRO, PROTEUS, MPLAB IPE (V3)

Hardware: PIC 16F628A, PicKit 3, 12 V (~3 A) power adaptor, LM 7805, breadboard, tactile switch, 8 × LED's, 4 × 470 Ω and 1 kΩ resistors.

Theory

Understanding the role of the TRIS (TRIState) register is essential to effectively configuring input and output pins in PIC microcontroller programming. The TRIS register determines whether a pin functions as an input or output. Typically, two registers are used, TRISA and TRISB, each with a binary representation. When a bit in the TRIS register is set (logic 1), the corresponding pin is configured as an input. Conversely, when the bit is cleared (logic 0), the corresponding pin is configured as an output. For example, TRISA and TRISB may be defined as 0b00000100 and 0b00000000, respectively. In this configuration, a pin, such as RA2, is designated as an input, while PORTB is designated as the output. This precise binary notation ensures accurate pin allocation, facilitating smooth interaction with external peripherals like switches and LEDs.

Employing software-defined bits, like the 'sbit' declaration for 'sw', enhances dynamic input state monitoring. Through conditional statements within the code, the behavior of the output state (on PORTB) is governed by the input state (RA2). For instance, when a switch is pressed (sw == 1), the output (PORTB) can transition to a high state, represented by the binary value 0b11111111. Conversely, when the switch is released (sw == 0), the outputs are terminated, denoted by 0b00000000 (PORTB).

This instructional approach provides a comprehensive understanding of configuring input and output operations in PIC microcontroller programming, advancing proficiency in embedded systems development.

Procedure

1. Write a code in C language to turn “ON” the LEDs, interfaced with PORTB, upon turning “ON” the tactile switch, and “OFF” the LEDs upon turning “OFF” the tactile switch engagement. A guided template for this code is in the “Template Code” section.

2. Create a schematic representation of the LED ON/OFF circuit shown in Figure 1 within the PROTEUS simulation environment.

3. Validate the code’s functionality by integrating the corresponding “HEX” file into the PROTEUS Simulator.

4. Employ the Micro Pik Kit 3 to transfer the compiled program code (.hex) onto the PIC 16F628A microcontroller.

5. Set up the circuit (Figure 2.1) on a breadboard to physically exemplify its operational dynamics.

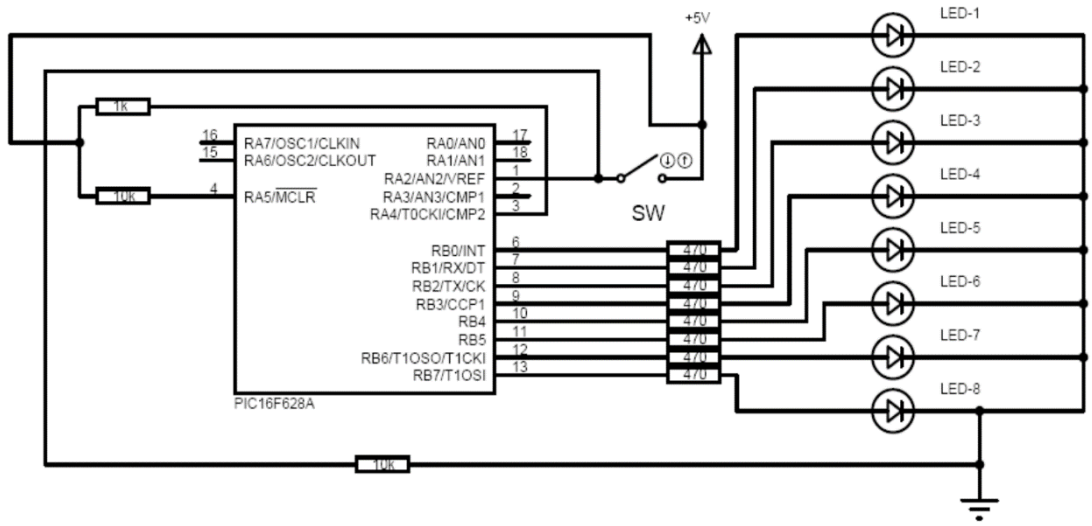


Figure 2.1: LED ON/OFF circuit diagram for PIC16F268A.

Template Code:

```
// Define the address and bit for the switch
sbit sw at RA2_bit; // Define sw at RA2 bit

// Main function
void main() {

    // Step 1: Initialize configuration settings
  CMCON = **** ;  // Hint: Disable Comparator
  TRISA =   **** ;  // Hint: Configure TRISA register
  TRISB =   **** ;  // Hint: Configure TRISB register
  PORTB =  **** ;  // Hint: Initialize PORTB register
  RA2_bit = **** ;  // Hint: Set RA2_bit to low state

    // Step 2: Enter the first loop
  do {
    // Step 3: Check the state of the switch
      if(sw == **** ) {
    // Step 4: If the switch is pressed, set PORTB to be low
        PORTB = ****;
      }
      else {
// Step 5: If the switch is not pressed, set PORTB to be high
        PORTB = ****;
      }
  } while(**** ); // Hint: Enter a condition for the infinite loop
}
```

Execute the **** sections in the code to finalize the program.

Laboratory - 03
Microcontroller-Implemented LED Chaser

Objectives

1. To comprehend the versatility of input and output pins on the PIC 16F628A microcontroller.
2. To configure microcontroller pins for driving external devices.
3. To execute the deployment of a chaser effect on LEDs employing a PIC microcontroller.

Apparatus

Software: MikroC PRO, PROTEUS, MPLAB IPE (V3).

Hardware: PIC 16F628A, PicKit 3, 12 V (~3 A) power adaptor, LM 7805 voltage regulator, breadboard, 7 × LED's, 1 kΩ and 7 × 470 Ω resistors.

Theory

The LED chaser, also known as the Knight Rider circuit, presents an engaging lighting effect widely employed in automotive lighting and decorative displays. This application, comprising a row of LEDs arranged to mimic a scanning motion, underscores the critical role of timing considerations in embedded systems design. Precise control over time intervals is indispensable for achieving desired functionalities, a task facilitated by the delay function "**Delay_ms()**". The delay function regulates timing and synchronization within the Knight Rider circuit and enhances visual appeal by introducing subtle delays between LED shifts, typically spanning a few tens of milliseconds. Furthermore, adjustments of the parameter within the "Delay_ms()" function allow for fine-tuning the LED illumination duration, thereby affording more control over the circuit's behavior.

In microcontroller programming, integers serve as foundational elements for representing numerical values and executing arithmetic operations. In this experiment, the integer variable, 'i', assumes the role of a loop counter, enabling sequential control of LED lights. The illuminated LED can traverse the sequence by iteratively incrementing or decrementing 'i' within a loop structure, highlighting the adaptability of integers in managing iterative processes. This fundamental utilization of integers underscores their pivotal role in directing program flow and managing data within embedded systems.

Disabling the comparator module (CMCON = 7) is crucial to prevent interference with the microcontroller's intended functionality. This preventive measure is imperative, as the comparator module has the potential to introduce extraneous noise or impede the microcontroller's performance. Following the comparator module's deactivation, all pins of PORTB are configured as outputs, priming them for driving LEDs and ensuring seamless operation of the Knight Rider circuit.

Procedure

1. Write a code in the C language to control the LED chaser.

2. Create a schematic representation of the LED ON/OFF circuit shown in Figure 3.1 within the PROTEUS simulation environment.

3. Validate the code's functionality by integrating the corresponding "HEX" file into the PROTEUS simulator.

4. Employ the Micro PikKit 3 to transfer the compiled program code onto the PIC 16F628A microcontroller.

5. Set up the circuit on a breadboard to physically exemplify its operational dynamics.

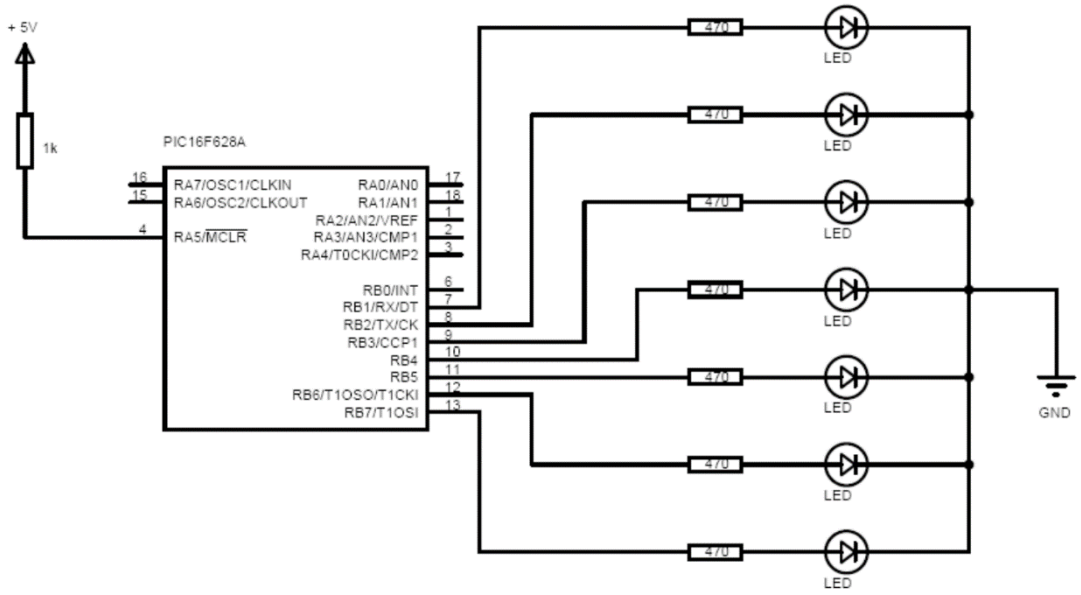


Figure 3.1: LED Chasing circuit diagram for PIC16F268A.

Template Code

```
void knightrider(void) {
   int i;
  // Step 1: Set all pins of PORTB as outputs
   TRISB = ****; // Hint: Set all pins of PORTB as outputs
  // Step 2: Initialize PORTB with the first LED lit
   PORTB = ****; // Hint: Initialize PORTB to light the first LED (RB0)
  // Step 3: Define the left shift loop
   for (i = ****; i <= ****; i++) {
   PORTB = (PORTB **** 1); // Hint: Shift the lit LED to the left
  // Step 4: Delay for smoother animation
      Delay_ms(****); // Hint: Delay for smoother animation
   }
  // Step 5: Define the right shift loop
   for (i = ****; i >= ****; i--) {
      PORTB = (PORTB ****); // Hint: Shift the lit LED to the right
  // Step 6: Delay for smoother animation
      Delay_ms(****);
   }
}
void main() {
   CMCON = ****; // Hint: Disable comparators
   TRISA = ****; // Hint: Set all PORTA pins as digital I/O
   while (****) { Hint: Enter a condition for the infinite loop
   knightrider(); // Call the knightrider function
   }
}
```

Execute the **** sections in the code to finalize the program.

Laboratory - 04
Interrupt handling with a PIC Microcontroller

Objectives

1. To comprehend the basic principles and mechanisms of interrupts in microcontrollers.
2. To gain practical experience in interrupt service routines (ISRs) and configuring the microcontroller to respond to interrupts.
3. To learn how to use interrupts to improve the efficiency and responsiveness of an embedded system.

Apparatus

Software: MikroC PRO, PROTEUS, MPLAB IPE (V3)

Hardware: PIC 16F628A, PicKit 3, 12 V (~3 A) power adaptor, LM 7805 voltage regulator, breadboard, 4 × LED's, 0.1 μF capacitor, tactile switch, 4 × 470 Ω, and 10 kΩ resistors

Theory

This laboratory experiment investigates the implementation of interrupt handling in a PIC microcontroller. In the PIC16F628A microcontroller, interrupts can be classified into two main categories: internal interrupts and external interrupts.

- Internal Interrupts

Internal interrupts are generated by the internal hardware peripherals of the microcontroller. These interrupts are triggered by events occurring within the microcontroller itself. Primary internal interrupts in the PIC16F628A are given bellow.

1. Timer0 Overflow Interrupt (T0IF)
2. EEPROM Write Complete Interrupt (EEIF)
3. USART Receive Interrupt (RCIF)
4. USART Transmit Interrupt (TXIF)
5. Comparator Interrupt (CMIF)

- External Interrupts

External interrupts are generated by events that occur outside the microcontroller, such as a change in state on an external pin. The primary external interrupts in the PIC16F628A are given bellow.

1. INT External Interrupt (INTF)
2. Port B Change Interrupt (RBIF)

- Interrupt Control and Handling

Interrupts in the PIC16F628A are controlled through various registers.

1. INTCON (Interrupt Control Register)

   Controls enabling and flagging for most interrupts.
   GIE (Global Interrupt Enable): Master control for all interrupts.
   PEIE (Peripheral Interrupt Enable): Enables peripheral interrupts.
   T0IE, INTE, RBIE: Enable individual interrupts for Timer0, INT pin, and Port B change.
   T0IF, INTF, RBIF: Flag bits indicating that the corresponding interrupt condition has occurred.

2. PIE1 (Peripheral Interrupt Enable Register 1) and PIR1 (Peripheral Interrupt Request Register 1)

   PIE1: Enables individual peripheral interrupts like Timer1, USART, EEPROM, and comparators.
   PIR1: Contains flag bits for peripheral interrupts.

By configuring these registers, the PIC16F628A can effectively manage and respond to both internal and external interrupt events, ensuring timely and efficient handling of critical events. In this experiment, we specifically investigate the handling of external interrupts using the RB0/INT pin. Upon receiving an interrupt signal, the microcontroller executes a predefined Interrupt Service Routine (ISR) to handle the event. This mechanism allows the main program to operate independently while promptly responding to critical external inputs.

Procedure

1. Develop a C language program that toggles the state of PORTA every 50 ms within the main loop. When an external interrupt is triggered on RB0, the Interrupt Service Routine (ISR) should execute, toggling PORTB pins RB1 and RB2 with a 200 ms delay. Ensure that RA5 is disabled in bitconfiguration.

2. Using the PROTEUS simulation environment, develop a schematic representation of the interrupt circuit, as illustrated in Figure 4.1.

3. Validate the code's functionality by integrating the corresponding "HEX" file into the PROTEUS simulator.

4. Employ the Micro PikKit 3 and MPLAB IPE to transfer the compiled program code onto the PIC 16F628A microcontroller.

5. Set up the circuit on a breadboard to physically exemplify its operational dynamics.

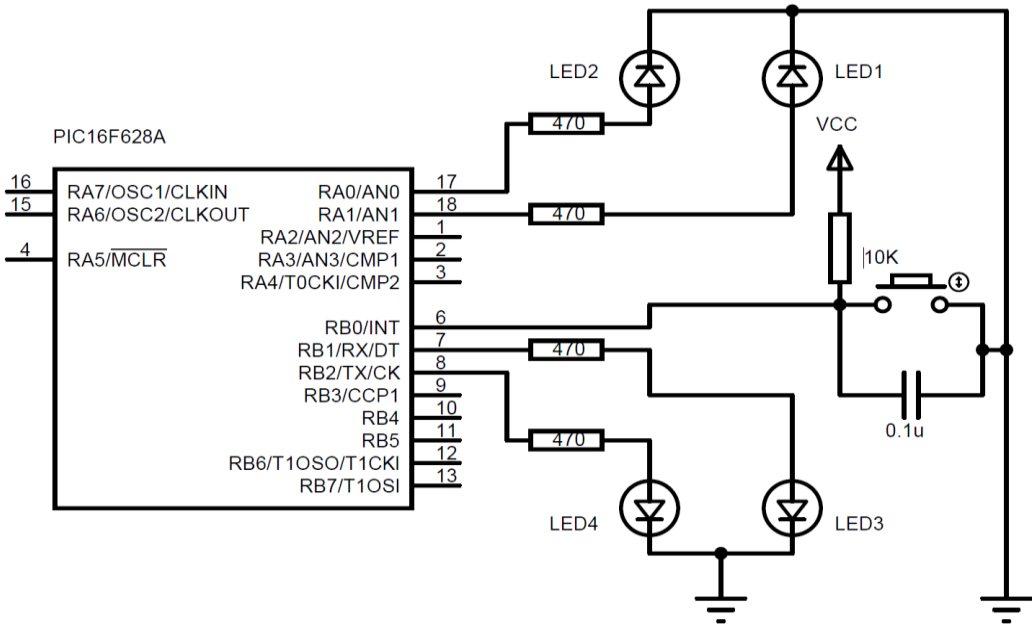


Figure 4.1: LED ON/OFF circuit diagram for PIC16F268A.

Template Code:

```
// Step 1: Declare the main function
    void main() {
// Step 2: Initialize configuration settings
   TRISB = ****; // Hint: Set RB0 as input, others as output
   TRISA = ****; // Hint: Set all port A pins as output
   CMCON = ****; // Hint: Disable comparators
   OPTION_REG = ****; // Hint: Configure option register
// Step 3: Enable interrupts
   INTCON.GIE = ****; // Hint: Enable global interrupts
   INTCON.PEIE = ****; // Hint: Enable peripheral interrupts
   INTCON.INTE = ****; // Hint: Enable RB0/INT interrupt
// Step 4: Define the infinite loop
    while (****) { // Hint: Enter an appropriate condition for the loop
// Step 5: Set initial PORT values
    PORTB.RB2 = ****; // Hint: Set RB2 to low
    PORTA.RA0 = ****; // Hint: Set RA0 to high
    PORTA.RA1 = ****; // Hint: Set RA1 to low
    delay_ms(****); // Hint: Pauses the execution for 100 milliseconds
// Step 6: Toggle PORT values
    PORTA.RA0 = ****; // Hint: Set RA0 to low
    PORTA.RA1 = ****; // Hint: Set RA1 to high
    delay_ms(****); // Hint: Pauses the execution for 100 milliseconds
    INTCON.INTF = ****; // Hint: Clear the external interrupt flag
   }
}
// Step 7: Interrupt service routine
   void interrupt() {
   if (****) {  // Hint: Check the external interrupt flag (INTCON.INTF)
// Step 8: Set PORT values upon interrupt
    PORTB.RB1 = ****; // Hint: Set RB1 to high
    PORTB.RB2 = ****; // Hint: Set RB2 to low
    delay_ms(****); // Hint: Pauses the execution for 100 milliseconds
// Step 9: Toggle PORT values
    PORTB.RB1 = ****; // Hint: Set RB1 to low
    PORTB.RB2 = ****; // Hint: Set RB2 to high
    delay_ms(****); // Hint: Pauses the execution for 100 milliseconds
    INTCON.INTF = ****; // Hint: Clear the external interrupt flag
   }
}
```

Execute the **** sections in the code to finalize the program.

Laboratory - 05
Interfacing a 16×2 LCD with a PIC Microcontroller

Objectives

1. To understand the Interfacing of a 16×2 LCD with a PIC Microcontroller.
2. To configure microcontroller pins for driving external devices.
3. To become proficient in utilizing library functions in MikroC Pro software.

Apparatus

Software: MikroC PRO, PROTEUS, MPLAB IPE (V3)

Hardware: PIC 16F628A, PicKit 3, 12 V (~3 A) power adaptor, LM 7805 voltage regulator, breadboard, 16×2 LCD, and 10 kΩ potentiometer

Theory

Liquid crystal displays (LCDs) are extensively utilized devices available in various forms, differing in size, shape, and numerous other features. This experiment focuses on interfacing a 16×2 LCD with a PIC microcontroller to display and manipulate text, a fundamental skill in embedded systems design. A 16×2 LCD, commonly used in microcontroller projects, operates typically at 5V, although some models can work at 3.3V. The display features 16 columns and 02 rows, allowing for the display of up to 32 characters at a time. It has several pins for data and control, as illustrated in Figure 5.1.

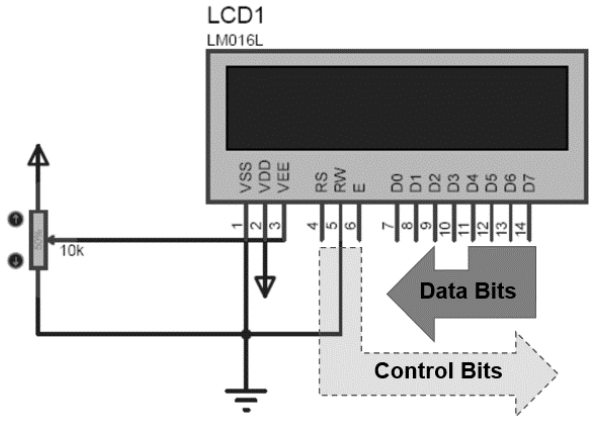


Figure 5.1: LCD Display Layout with Pin Configuration.

Typically, a standard 16×2 LCD features three primary control pins: Register Select (RS), Read/Write (RW), and Enable (E). The RS pin determines whether the sent data is interpreted as command instructions or character data. The RW pin specifies the data transfer direction, allowing the microcontroller to either write data to the LCD or read data from it, although it is often grounded to simplify operations to write-only mode. The Enable pin plays a crucial role in initiating data transactions between the microcontroller and the LCD, activating data transfer on the falling edge of the signal.

The LCD's data pins (DB0-DB7) can be configured for either 8-bit or 4-bit data transfer modes. In the 8-bit mode, all eight data lines (DB0-DB7) are utilized, enabling faster data transfer as the entire byte of data is conveyed to the LCD in a single operation. However, this mode requires more I/O pins from the microcontroller, which may be impractical in systems with limited available pins. Conversely, in the 4-bit mode as illustrated in Figure 1, only the upper four data lines (DB4-DB7) are used, conserving microcontroller I/O pins by requiring only four data lines. The trade-off for this conservation is a slightly slower data transfer rate, as each byte of data must be sent in two 4-bit operations: first, the higher bite (4 bits), followed by the lower bite (4 bits).

Concerning the power connections of the LCD, VSS should be grounded (0V), and VDD should be maintained within the range of 4.5V to 5V. The VEE pin is designated for contrast adjustment on the display. VEE is recommended to be connected to a variable power supply for precise control and optimal display performance.

Procedure

1. Develop a C language program to display the following texts across two rows with different effects.

   Text #1: "WELCOME"
   Text #2: "BECS 31421"
   Text #3: "EX:LCD"
   Text #4: "NO:06"

   Note: Here, the first two texts should be displayed initially, then cleared after a short delay (500 ms). The last two texts should be displayed with a swinging effect, including shifting right, shifting left, and centering.

2. Develop a schematic representation of the LCD interfacing circuit, as illustrated in Figure 5.2, using the PROTEUS simulation environment.

3. Validate the code's functionality by integrating the corresponding "HEX" file into the PROTEUS simulator.

4. Employ the Micro PikKit 3 and MPLAB IPE to transfer the compiled program code onto the PIC 16F628A microcontroller.

5. Set up the circuit on a breadboard to physically exemplify its operational dynamics. Ensure to supply power to the anode (A) and cathode (K) pins of the LCD to activate the backlight.

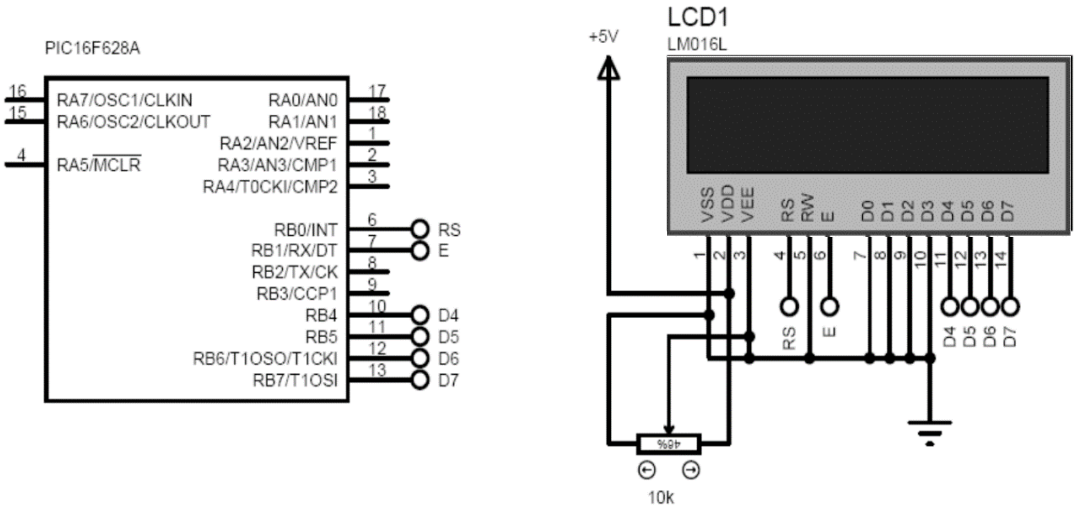


Figure 5.2: LCD interfacing circuit with PIC16F268A.

Template Code

```
// Step 1: Include necessary libraries for LCD
    **** // Hint: The LCD module connections

// Step 2: Define text to be displayed on the LCD
char txt1[] = "****"; // Text #1
char txt2[] = "****"; // Text #2
char txt3[] = "****"; // Text #3
char txt4[] = "****"; // Text #4
```

```
char i; // Hint: Loop variable

// Step 3: Define a function to add delay for text moving
void Move_Delay() {
  Delay_ms(****); // Hint: Adjust the delay for 500 milliseconds
}

// Step 4: Define the main function
void main() {
   CCP1CON = ****; // Hint: Disable CCP1 module (Capture/Compare/PWM)
   T1CON = ****; // Hint: Disable Timer1
   VRCON = ****; // Hint: Disable Voltage Reference module
   INTCON = ****;   // Disable interruptions
   CMCON = ****;    // Disable comparators

// Step 5: Configure LCD settings
    Lcd_Init(); // Hint: Initialize the LCD
    Lcd_Cmd(****); // Hint: Clear display
    Lcd_Cmd(****); // Hint: Turn off cursor
    Delay_ms(****); // Hint: Adjust the delay for 10 milliseconds
    Lcd_Out(****,****,txt1); // Hint: Display Text #1 starting at fifth column in the first row
    Lcd_Out(****,****,txt2); // Hint: Display Text #2 starting at third column in the second row
    Delay_ms(****); // Hint: Adjust the delay for 1000 milliseconds
    Lcd_Cmd(****); // Hint: Clear the display again
    Lcd_Out(****,****,txt3); // Hint: Display Text #3 starting at sixth column in the first row
    Lcd_Out(****,****,txt4); // Hint: Display Text #4 starting at sixth column in the second row
    Delay_ms(****); // Hint: Adjust the delay for 1000 milliseconds

// Step 6: Scrolling text display on LCD
    for(i=0; i<****; i++) { // Hint: shift the text to right 4 times
        Lcd_Cmd(****); // Hint: Shift text right
        Move_Delay(); // Hint: Call delay function
    }

// Step 9: Move the text to left 9 times
// Step 10: Move text to the center by shifting right 5 times
// Step 11: Enter the endless loop
    while(****) { // Hint: Enter a condition for the infinite loop
    }
}
```

Execute the **** sections in the code to finalize the program.

Laboratory - 06
Interfacing DC Motor with PIC Microcontroller

Objectives

1. To develop skills for defining bit-level access for switches and motor control pins.
2. To learn how to interface a DC motor with a PIC microcontroller.
3. To understand the process of interfacing a PIC microcontroller with a motor driver integrated circuit (IC).

Apparatus

Software: MikroC PRO, PROTEUS Simulator, MPLAB IPE (V3).

Hardware: PIC 16F628A, PicKit 3, 12 V (~3 A) power adaptor, LM 7805 voltage regulator, L293D motor driver, 3 × tactile switches, 3 V DC motor, breadboard, 4 × 10 kΩ and 3 × 1 kΩ resistors.

Theory

This experiment focuses on the fundamentals of interfacing a DC motor with a PIC microcontroller to achieve bi-directional rotation. However, a microcontroller cannot directly drive a DC motor because DC motors require higher current and voltage levels than microcontrollers provide. For instance, microcontrollers usually operate at 3.3 V or 5 V power supplies, with I/O pins handling up to 25 mA of current. In contrast, a standard simple DC motor may require a 3 V, 300 mA power supply. Directly connecting a DC motor to a microcontroller would overload and potentially damage the microcontroller due to the back EMF generated by the motors. These mismatches in electrical characteristics necessitate using an intermediate device to control the motor safely.

Utilizing the L293D integrated circuit (IC) (Figure 6.1) is a standard method for addressing this issue. The L293D is a dual H-bridge motor driver IC that allows control of two DC motors in clockwise and counterclockwise directions. It can provide bidirectional drive currents of up to 600 mA at voltages ranging from 4.5 V to 36 V. Designed to drive inductive loads such as DC motors, bipolar stepping motors, relays, and solenoids, the L293D ensures safe operation by incorporating built-in output clamp diodes for suppressing inductive transients. All inputs of this IC are Time-to-live (TTL) compatible, making it suitable for interfacing with microcontrollers. Furthermore, its compact design simplifies the overall circuit layout, making it an efficient choice for various motor control applications.

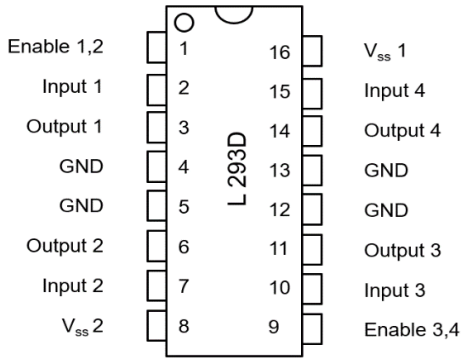


Figure 6.1: Pin configuration of L293D motor drive.

Procedure

1. Write code in the C language to implement forward and backward rotation of a DC motor using switches.

2. Create a schematic representation of the motor rotation circuit as shown in Figure 6.2 within the PROTEUS simulation environment and validate the code's functionality by integrating the corresponding "HEX" file into the PROTEUS simulator.

3. Use the Microchip PicKit 3 to transfer the compiled program code onto the PIC16F628A microcontroller.

4. Construct the circuit on a breadboard to demonstrate how to control the rotation of the DC motor.

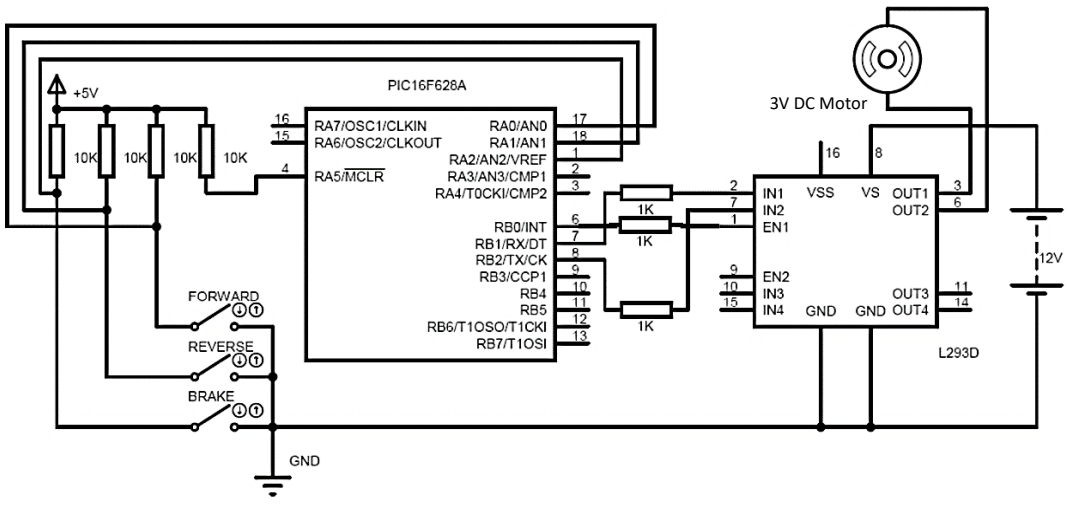


Figure 6.2. Schematic of a PIC-based DC motor controller circuit.

Template Code

```
 // Step 1: Define pin assignments for motor control
      sbit Forward at RA0_bit;
      sbit Reverse at RA1_bit;
      sbit Brake at RA2_bit;
      void main() {

// Step 2: Initialize configuration settings
     CMCON = ****; // Hint: Disable comparators
     TRISA = ****;    // Hint: Set RA0, RA1 and RA2 as input, others as output
     TRISB = ****;    // Hint: Set all port B pins as output
     PORTB = ****;   // Hint: Initialize PORTB

// Step 3: Define the infinite loop
      while(****) { // Hint: Enter an appropriate condition for the loop

// Step 4: Handling "Forward" Conditions
    if (!Forward) {
       PORTB = ****;  // Hint: Clear Port B output latch
       RB0_bit = ****; // Hint: Set RB0 high
       RB1_bit = ****; // Hint: Set RB1 high
       RB2_bit = ****; // Hint: Set RB2 low
    }
// Step 5: Handling "Reverse" Conditions
    else if (!Reverse) {
       PORTB = ****;  // Hint: Clear Port B output latch
       RB0_bit = ****; // Hint: Set RB0 high
       RB1_bit = ****; // Hint: Set RB1 low
       RB2_bit = ****; // Hint: Set RB to high
    }
// Step 6: Handling "Break" Conditions
       else if (!Brake) {
// Step 7: Perform actions for braking
       PORTB = ****;  // Hint: Ensure PORTB is cleared
    }
  }
}
```

Execute the **** sections in the code to finalize the program.

Laboratory - 07
Implementing a PIC Microcontroller-Based Quiz Buzzer

Objectives

1. To demonstrate the integration of buzzers with a PIC microcontroller for real-time event handling.
2. To explore using transistors as current amplifiers to drive high-power components, such as buzzers, with a PIC microcontroller.
3. To implement a reset function that clears and prepares the system for the next phase.

Apparatus

Software: MikroC PRO, PROTEUS, MPLAB IPE (V3).

Hardware: PIC 16F628A, Pick Kit 3, 12 V (~3 A) power adaptor, LM 7805 voltage regulator, 3 × tactile switches, 2 × active buzzer (5V), 2 × BC 547 transistors, breadboard, 2 × 1 kΩ and 2 × 470 Ω resistors.

Theory

Buzzers are commonly utilized in various applications such as alarms, timers, and user interface feedback systems due to their simplicity and effectiveness. This experiment demonstrates the integration of buzzers with a PIC microcontroller in implementing a fundamental game known as “Fastest Finger-First.” In this game, multiple players participate, each equipped with their buzzer. Participants compete to swiftly press their respective buttons in response to a prompt to demonstrate their agility. A master reset button for the overseeing judge facilitates a smooth transition to the game’s next phase.

PIC microcontrollers cannot generally provide high voltage and current levels to external circuitry. For instance, the PIC16F628A microcontroller is limited to a maximum current output of 25 mA per individual I/O pin and operates within a maximum voltage range of 5.0V. These specifications are often inadequate for driving electronic components such as buzzers, which may require higher voltage and current levels for optimal operation. Consequently, transistors have become a viable solution to address this limitation due to their inherent amplification capabilities.

Transistors offer electrical isolation and efficient switching capabilities. When a microcontroller outputs a signal to the base of a transistor, it switches the connected load ON or OFF without exposing the microcontroller directly to the high currents or voltages required by the load. This isolation protects the microcontroller from potential electrical noise and voltage spikes, enhancing the circuit’s overall reliability and longevity.

This experiment aims to develop the primary control logic based on the above concepts to detect the initial button press, disable any additional inputs until resetting the system, and activate the corresponding LED or buzzer to signal the winning contestant.

Procedure

1. Write a code in the C language to implement the fastest finger-first quiz buzzer functionality.

2. Create a schematic representation of the quiz buzzer circuit within the PROTEUS simulation environment, as shown in Figure 7.1.

3. Validate the code's functionality by integrating the corresponding "HEX" file into the PROTEUS simulator.

4. Employ the Micro PikKit 3 to transfer the compiled program code onto the PIC 16F628A microcontroller.

5. Construct the circuit on a breadboard to demonstrate the quiz buzzer operation.

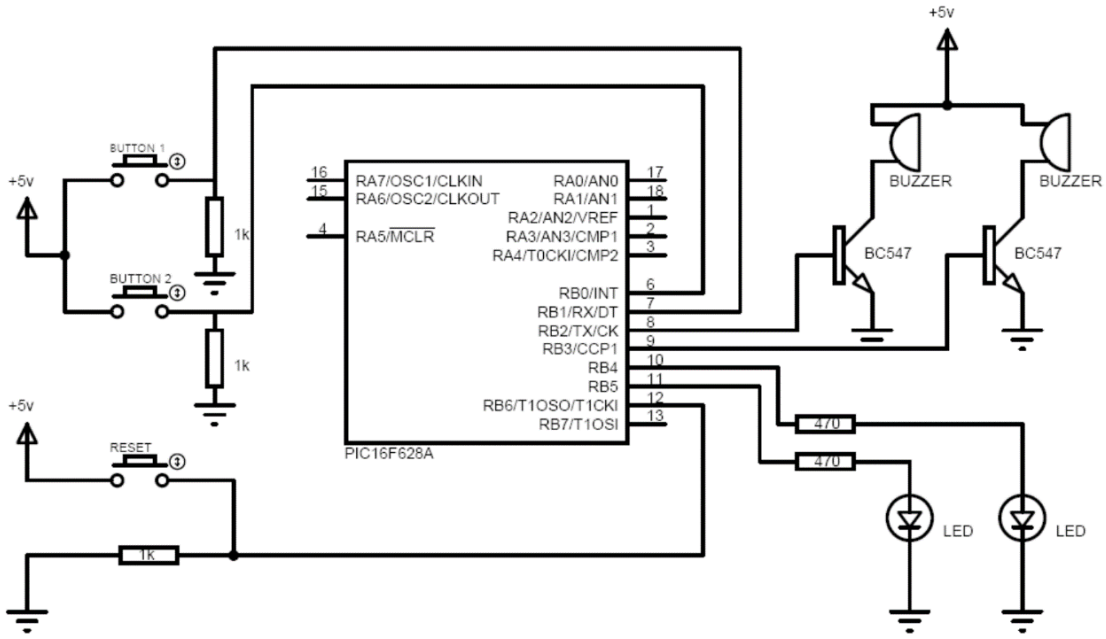


Figure 7.1: LED Chasing circuit diagram for PIC16F268A.

Template Code

```
// Step 1: Declare the main function
    void main() {
    int i = ****; // Hint: Initialize a variable(i) to control the loop

// Step 2: Configure input and output pin configurations
    TRISB = ****;       // Hint: Set RB0, RB1 and RB6 as input, others as output
    PORTB = ****;     // Hint: Initialize PORTB

// Step 3: Define the infinite loop
    while(****) { // Hint: Enter an appropriate condition for the loop

// Step 4: Check player 1 button (RB0)
    if(PORTB == ****) { // Hint: Check if Player 1 presses RB0
    PORTB = ****;   // Hint: Set 3rd bit (RB2-Buzzer) and 5th bit (RB4-LED) of port B to high
    while(i == ****) { // Hint: Enter the condition to continue as long as 'i' is 0
    if(PORTB == ****) { // Hint: Check if portB has changed to indicate the reset (RB6) is
    pressed
    PORTB = ****; // Hint: If the reset button is pressed, set all bits of port B to low to turn off
                        the Buzzer and LED
    i = ****; // Hint: Set the appropriate conditions to exit the inner loop
            }
        }
    i = ****; // Hint: After exiting the inner loop, reset the 'i' for the next iteration
      }
// Step 5: Check player 2 button (RB1)
    else if(PORTB == ****) { // Hint: Check if Player 2 presses RB1
    PORTB = ****;  // // Hint: Set 4th bit (RB3-Buzzer) and 6th bit (RB5-LED) of port B to
    high
    while(i == ****) { // Hint: Enter the condition to continue as long as 'i' is 0
    if(PORTB == ****) { // Hint: Check if portB has changed to indicate the reset (RB6) is
    pressed
    PORTB = ****; // Hint: If the reset button is pressed, set all bits of port B to low to turn off
                        the Buzzer and LED
    i = ****; // Hint: Set the appropriate conditions to exit the inner loop
        }
    }
    i = ****; Hint: After exiting the inner loop, reset the 'i' for the next iteration
            }
       }
    }
```

Execute the **** sections in the code to finalize the program.

Laboratory - 08
Pulse width modulation (PWM) with microcontroller

Objectives

1. To understand the concept and application of PWM.
2. To learn how to configure and use the PWM module in the PIC16F628A.
3. To observe the effect of varying PWM duty cycle on LED brightness.

Apparatus

Software: MikroC PRO, PROTEUS, MPLAB IPE (V3)

Hardware: PIC 16F628A, PicKit 3, 12 V (~3 A) power adaptor, LM 7805 voltage regulator, breadboard, LED, oscilloscope, and 470Ω resistor

Theory

PWM (Pulse Width Modulation) is a powerful technique that allows control over analog devices with a digital signal. This experiment demonstrates using PWM to control the brightness of an LED with a PIC16F628A microcontroller. Adjusting the duty cycle of the PWM signal allows us to control the power delivered to the LED effectively, thus varying its brightness.

The PIC16F628A microcontroller supports PWM functionality through its built-in CCP (Capture/Compare/PWM) modules. Typically, CCP modules can be configured for a range of tasks, including capturing external events, comparing two values, and generating PWM signals. The duty cycle of the signal, representing the ratio of the on-time to the total period, determines the average voltage level over time. This duty cycle manipulation enables precise control over the output intensity of PWM signals, as represented by the following equation for average power (P_{avg}) as a function of maximum power (p_{max}).

$$P_{avg} = p_{max} \times \left(\frac{Duty\ cycle\ (\%)}{100}\right)$$

In PWM control, the duty cycle parameter is typically represented by an 8-bit value, allowing for precise control with 256 levels ranging from 0 to 255 on the PIC16F628A microcontroller. Here, a value of 0 corresponds to a 0% duty cycle, meaning the PWM signal remains constantly low, resulting in an average power output of 0% ($P_{avg} = 0$). Conversely, a value of 255 represents a 100% duty cycle, where the PWM signal is constantly high, resulting in the maximum average power output ($P_{avg} = P_{max}$).

Similarly, each step within this range between 0 to 255 corresponds to a slight adjustment in the duty cycle, allowing for precise tuning of the PWM signal's output intensity according to the above equation. MikroC Pro's PWM library functions facilitate the initialization and control of PWM signals on PIC microcontrollers. The manual "Introduction to Software Tools" extensively documents comprehensive instructions for incorporating and configuring the library functions.

Procedure

1. Write a code in C language to create a PWM-based LED brightness control system using the PIC microcontroller. Ensure that the necessary library functions for PWM initialization are included and that RA5 is disabled in the bit configuration.

2. Create a schematic representation of the PWM-based LED brightness control circuit shown in Figure 8.1 within the PROTEUS simulation environment.

3. Validate the code's functionality by integrating the corresponding "HEX" file into the PROTEUS simulator.

4. Employ the Micro PikKit 3 to transfer the compiled program code onto the PIC 16F628A microcontroller.

5. Set up the circuit on a breadboard to physically exemplify its operational dynamics.

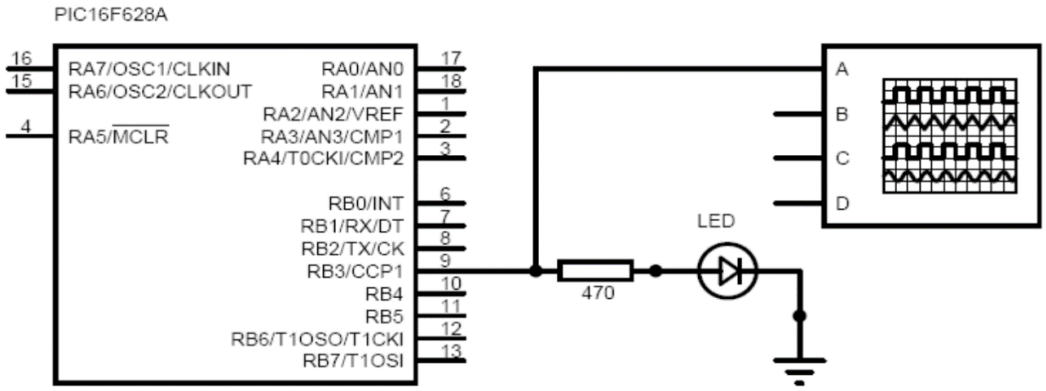


Figure 8.1: The PWM-based LED brightness control circuit diagram.

Template Code

```
// Step 1: Declare the main function
      void main() {
// Step 2: Configurationsettings: output and initialization
      CMCON = ****;   // Hint: Disable comparators
      TRISB = ****; // Hint: Set all port B pins as output
      PORTB = ****; // Hint: Initialize PORTB

// Step 3: PWM initialization and start
      PWM1_Init(****);   // Hint: Initialize PWM1 module with the frequency of 5kHz
      PWM1_Start(); // Hint: Starts the PWM1 module operation

// Step 4: Define the infinite loop
      while(****) { / /Hint: Enter an appropriate condition for the loop

// Step 5: PWM duty cycle variation
      unsigned short duty_cycle; / /Hint: Declare an unasigned short variable 'duty_cycle'

       // Step 5.1: Increasing duty cycle loop
      for (duty_cycle = ****; duty_cycle <= ****; duty_cycle += ****) { // Hint: Start from
                  minimum duty cycle and increment up to maximum, increasing by 5
       PWM1_Set_Duty(duty_cycle); // Hint: Set the duty cycle
       Delay_ms(****); // Hint: Delay to observe the change in LED brightness
}

      // Step 5.2: decreasing duty cycle loop

      for (duty_cycle = ****; duty_cycle >= ****; duty_cycle -= ****) { // Hint: Start from
                  maximum duty cycle and decrement down to minimum, decreasing by 5

       PWM1_Set_Duty(duty_cycle); // Hint: Set the duty cycle
      Delay_ms(****); // Hint: Delay to observe the change in LED brightness

      }

   }
}
```

Execute the **** sections in the code to finalize the program.

Laboratory - 09
Interfacing 7-Segment Display

Objectives

1. To comprehend the process of interfacing the 7-segment with the PIC microcontroller.
2. To be familiar with the counter function and handle overflow conditions to ensure accurate counting within the specified range.
3. To understand how to use this knowledge in real-world applications.

Apparatus

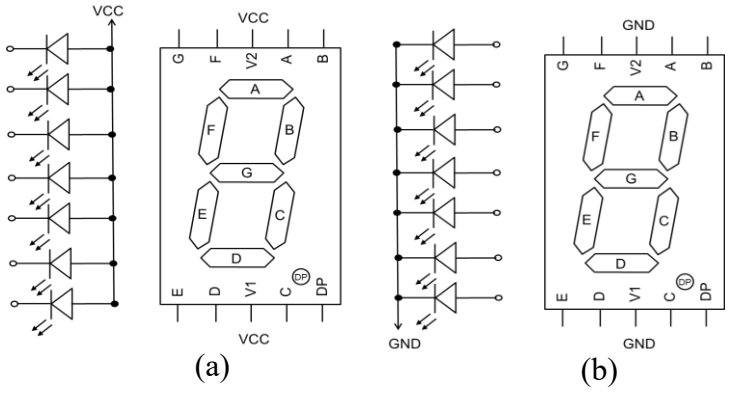
Software: MikroC PRO, PROTEUS Simulator, MPLB IPE (V3).

Hardware: PIC 16F628A, 12 V (~3 A) power adaptor, LM 7805 voltage regulator, Pick Kit 3, breadboard, tactile switch, seven-segment display, 10 kΩ and 7 × 330 Ω resistors.

Theory

The seven-segment display is considered one of the simplest and most basic electronic display devices. Its operation is easy to understand, and its interface with the controller is relatively straightforward. As the name suggests, it consists of seven LEDs arranged in a specific pattern, each designated by a letter from 'A' to 'G'. Additionally, some displays feature an eighth LED called 'DP', which illuminates a dot or decimal point.

Seven-segment displays have two configuration types: Common anode and Common cathode. A common pin is present to determine the display type. As illustrated in Figure 9.1(a), a common anode display features interconnected positive terminals for all LEDs, necessitating a 'HIGH' signal for operation. Conversely, in a common cathode display, depicted in Figure 9.1(b), all cathode connections are interconnected and must be grounded. Figure 9.2 illustrates a typical method for connecting the common cathode seven-segment display to the port B pins of the PIC 16F628A microcontroller.



(a) (b)

Figure 9.1: Configurations of seven-segment displays: (a) common anode, and (b) common cathode.

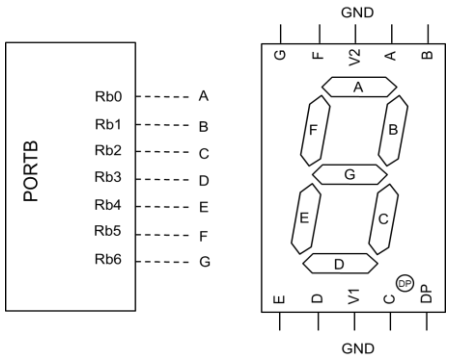


Figure 9.2: Pin interfacing of common cathode 7-segments to port B of PIC 16F628A.

Procedure

1. Write a code in the C language for interfacing the seven-segment (BCD input).

2. Create a schematic representation of the PIC16F268A-based counter circuit shown in Figure 9.3 within the PROTEUS simulation environment.

3. Validate the code's functionality by integrating the corresponding "HEX" file into the PROTEUS simulator.

4. Employ the Micro PikKit 3 to transfer the compiled program code onto the PIC 16F628A microcontroller.

5. Set up the circuit on a breadboard to physically exemplify its counter operation, showcasing numbers 0 through 9.

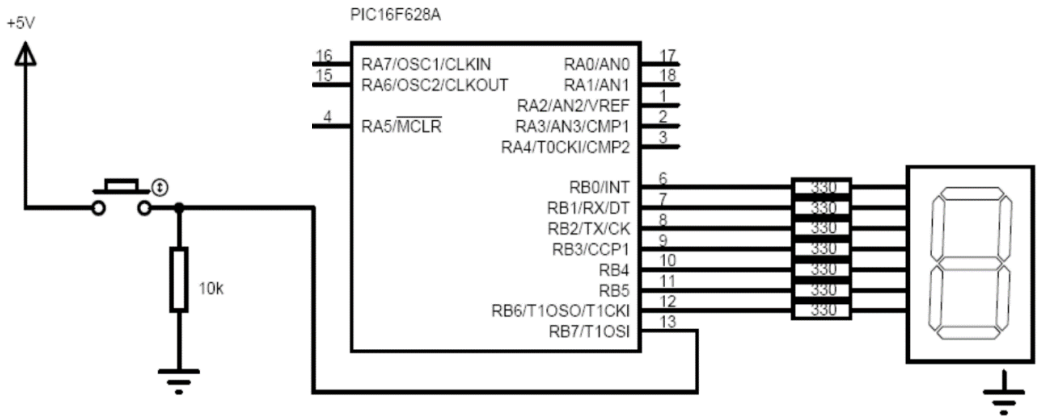


Figure 9.3: Circuit diagram of PIC16F268A-based counter.

Template Code

```
// Step 1: Declare a constant array
        const unsigned char numbers [10] = {**********, ……................};
        // Hint: Declare a constant array 'numbers' containing 10 elements of type 'unsigned
        char'.

// Step 2: Gloabal variables decleration
        int stop = ****;   // Hint: Initialize an integer variable 'stop' with an initial value of zero
        int count = ****; // Hint: Initialize an integer variable 'count' with an initial value of
        zero

// Step 3: Declare the main function
        void main() {

// Step 4: Initialize configuration settings
        TRISA = ****;   // Hint: Set RA0 as input, others as output
        TRISB = ****;   // Hint: Set RB7 as input, others as output
        PORTA = ****;  // Hint: Initialize all pins of PORTA to low
        PORTB = ****;  // Hint: Initialize all pins of PORTB to low

// Step 5: Define the infinite loop
        while (**** ) {  // Hint: Enter an appropriate condition for the loop

// Step 6: Condition checking and display logic
        if (PORTB.RB7 == **** && stop == ****) { // Hint: Check RB7 is high and 'stop'is
        zero
        count ****; // Hint: If true, increment the 'count'
        if (count == ****) {  // Hint: Check if 'count' has reached its maximum value
        count = ****;   // Hint: Reset 'count' to its minimum value
        }
        PORTB =; // Hint: Set PORTB to display the digits corresponding to 'numbers[count]'
        stop = ****;  // Hint: Set 'stop' to high until RB7 low
        Delay_ms(****); // Hint: Delay for 10 milliseconds
        }
        else if (PORTB.RB7 == ****) { // Hint: Check if RB7 is in low
        stop = ****; // Hint: If true Reset the 'stop' to zero
        }
    }
}
```

Execute the **** sections in the code to finalize the program.

Laboratory - 10
Implementing Relay Control with PIC Microcontroller

Objectives

1. To demonstrate the integration of relays with a PIC microcontroller in controlling high-power electrical devices.
2. To explore transistors as current amplifiers to drive high-power components, such as relays, with a PIC microcontroller.
3. To design and implement circuits that control high-power devices safely and effectively using low-power microcontroller outputs.

Apparatus

Software: MikroC PRO, PROTEUS Simulator, MPLB IPE (V3)

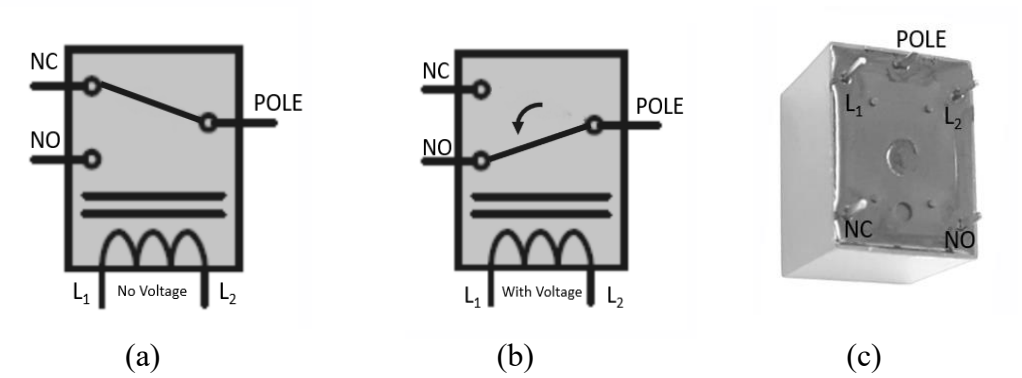
Hardware: PIC 16F628A, Pick Kit 3, 12 V (~ 3 A) power adaptor, LM 7805 voltage regulator, tactile switch, JQC-3F(T73) relay, 12 V bulb, In 4007 diode, 01 kΩ, and 10 kΩ resistors

Theory

This laboratory experiment employs a relay with the PIC16F628A microcontroller to control electrical devices. Relays function similarly to conventional switches but utilize an electromagnetic coil to automate their operation. When an adequate current is applied across this coil, it becomes energized, triggering the switching of the relay's contacts and thus opening or closing the connected circuit. Relays offer the advantage of isolating the input and output circuits, enhancing their adaptability in various applications.

The single pole double throw (SPDT) relays are widely used in microcontroller programming due to their versatility and ease of integration. These relays are available in different voltage ranges, including 5V, 6V, 12V, and 18V. However, the 5V relay is the most suitable for aligning with the PIC16F628A microcontroller's operating voltage. Despite its compact size, this 5V relay exhibits strong potential for handling significant loads such as a 220V (AC) bulb.

The JQC-3F(T73), a 5V SPDT relay that features five pins, each serving a specific function. Pins L1 and L2 precisely correspond to the internal electromagnetic coil, which controls the relay's activation (Figure 10.1). Additionally, the relay features POLE, NO (Normally Open), and NC (Normally Closed) pins. Under typical settings, POLE is connected to NC. However, POLE switches its connection to NO upon energizing the relay, facilitating the switching operation.



(a) (b) (c)

Figure 10.1: SPDT relay (a, b) Working mechanism with and without voltage; (c) Pin connection.

PIC microcontrollers, such as the PIC16F628A, are often incapable of providing sufficient power for high-power external components due to their limited current and voltage output. These limitations render them unsuitable for directly driving devices such as relays. In such instances, a transistor can be utilized to amplify the microcontroller's output, enabling it to drive higher-power components. The transistor functions as the relay's driver, amplifying the control signal from the microcontroller to energize the relay coil. Furthermore, a diode, typically a clamp diode like the 1N4007, is incorporated to protect the circuit from voltage spikes generated during relay deactivation.

Procedure

1. Develop a C language program to control the operation of a relay, thereby regulating a 12V lamp.

2. As shown in Figure 10.2, create a schematic representation of the relay control circuit within the PROTEUS simulation environment.

3. Validate the code's functionality by integrating the corresponding "HEX" file into the PROTEUS simulator.

4. Employ the Micro PikKit 3 to transfer the compiled program code onto the PIC 16F628A microcontroller.

5. Construct the circuit on a breadboard to demonstrate the relay operation.

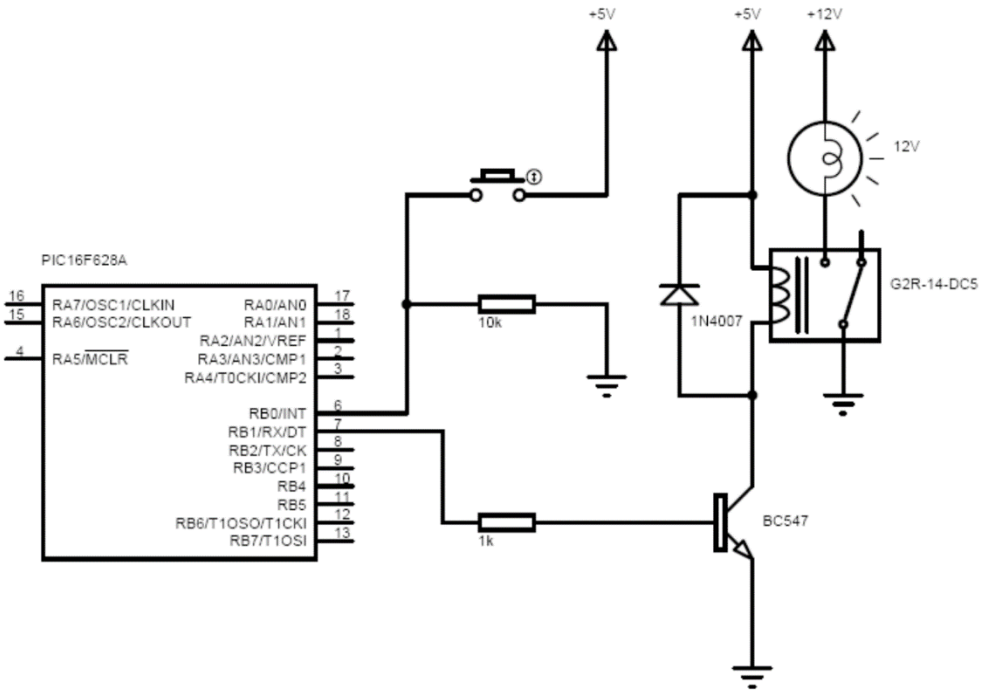


Figure 10.2: Relay control circuit diagram.

Template Code

```
// Step 1: Define pin assignments for switch and relay control
sbit sw1 at RB0_bit;
sbit relay at RB4_bit;

// Step 2: Declare a variable to store the current state of the relay
char cue1 = 0;

void main() {

 // Step 3: Initialize configuration settings
   CMCON = ****; // Hint: Disable comparators
   TRISA = ****; // Hint: Set RA0 as input, others as output
   TRISB = ****; // Hint: Set RB0 as input, RB4 as output
   PORTA = ****; // Hint: Initialize PORTA
   PORTB = ****; // Hint: Initialize PORTB

       // Step 4: Define the infinite loop
   while (****) { // Hint: Enter an appropriate condition for the loop

        // Step 5: Check if the switch is pressed
      if (sw1 == ****) { // Hint: Enter the correct condition to detect the state of the switch

         // Step 6: Toggle the relay state
         cue1 = ****; // Hint: Toggle the value of cue1

         // Step 7: Control the relay based on cue1
         relay = ****; // Hint: Assign cue1 to relay

         // Step 8: Delay to debounce the switch
         Delay_ms(****); // Hint: Set the delay time to 50 milliseconds
      }
   }
}
```

Execute the ******** sections in the code to finalize the program.

Laboratory - 11
Microcontroller-Based Simple Calculator

Objectives

1. To understand the Interfacing of a 16×2 LCD and 4×4 keypad with a microcontroller
2. To become proficient in using library functions within the MikroC Pro software
3. To implement arithmetic operations using a microcontroller

Apparatus

Software: MikroC PRO, PROTEUS, MPLAB IPE (V3)

Hardware: PIC 16F628A, PicKit 3, 12 V (~3 A) power adaptor, LM 7805 voltage regulator, breadboard, 16×2 LCD, 4×4 keypad, 10 kΩ potentiometer, and 10 kΩ resistor

Theory

This section offers an overview of interfacing a 16×2 LCD and a keypad with a PIC microcontroller. For detailed instructions on adding and configuring library functions specific to the LCD in MikroC, refer to the "Introduction to Software Tools" manual. To begin this experiment, it is essential to add the LCD library function following the outlined three-step process. Additionally, integrate other essential library functions such as Conversion, C_String, C_Type, Keypad 4×4, Sprinti, Sprintl, and LCD_Constants by selecting each library in the Library Manager. These integrations are crucial for enabling efficient communication between the microcontroller and the input devices, ensuring accurate representation of characters and symbols on the LCD screen.

ASCII (American Standard Code for Information Interchange) is a character encoding standard used to represent text in computers and some other devices that use text. Using ASCII codes for LCD communication has several advantages. ASCII provides a standardized way to represent characters, ensuring compatibility across different systems and devices. Each character, including letters, digits, and special symbols, is represented by a unique 7-bit or 8-bit code, simplifying the process of displaying text on the LCD. When sending data to the LCD, the microcontroller can send the ASCII code of the desired character. The LCD controller interprets these codes and displays the corresponding characters.

This experiment used a 4×4 keypad with four rows and four columns (Figure 11.1). The keypad is scanned for key inputs through a method called Polling, which involves setting one row low at a time and reading the status of the column pins as input at that instant. This process is repeated for all rows, enabling the microcontroller to detect which key is pressed. The

following examples illustrate how the keypad input is processed and displayed on the LCD using ASCII codes.

Keypad Input and LCD Display

- When a user presses the '1' key on the keypad, the microcontroller sends the ASCII code for '1' (which is 49) to the LCD. The LCD interprets this code and displays the character '1' on the screen.
- Similarly, pressing the '2' key sends the ASCII code for '2' (which is 50) to the LCD, displaying the character '2'.
- This process is repeated for other numerical keys, where '3' sends 51, '4' sends 52, '5' sends 53, '6' sends 54, '7' sends 55, '8' sends 56, '9' sends 57, and '0' sends 48.

Control Characters

- Specific keys on the keypad correspond to arithmetic operations, with specific ASCII codes representing these operations. For instance, the '+' key sends the ASCII code for plus (which is 43), the '-' key sends 45, the '*' key sends 42, and the '/' key sends 47.
- The '=' key, used to signify the end of input and to perform the calculation, sends the ASCII code 61.

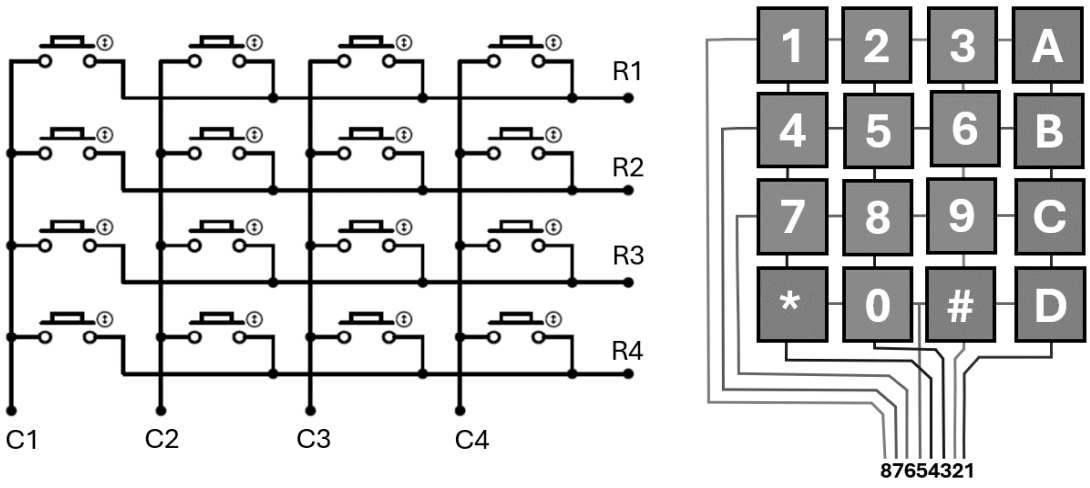


Figure 11.1: Schematic representation of the 4×4 keypad utilized in the setup.

Procedure

1. Write a code in C language to develop a calculator capable of performing basic arithmetic operations using two-digit inputs.

2. Develop a schematic representation of the microcontroller-based calculator circuit shown in Figure 11.2 within the PROTEUS simulation environment.

3. Validate the code's functionality by integrating the corresponding "HEX" file into the PROTEUS simulator.

4. Employ the Micro PikKit 3 and MPLAB IPE to transfer the compiled program code onto the PIC 16F628A microcontroller.

5. Set up the circuit on a breadboard to physically exemplify its operational dynamics.

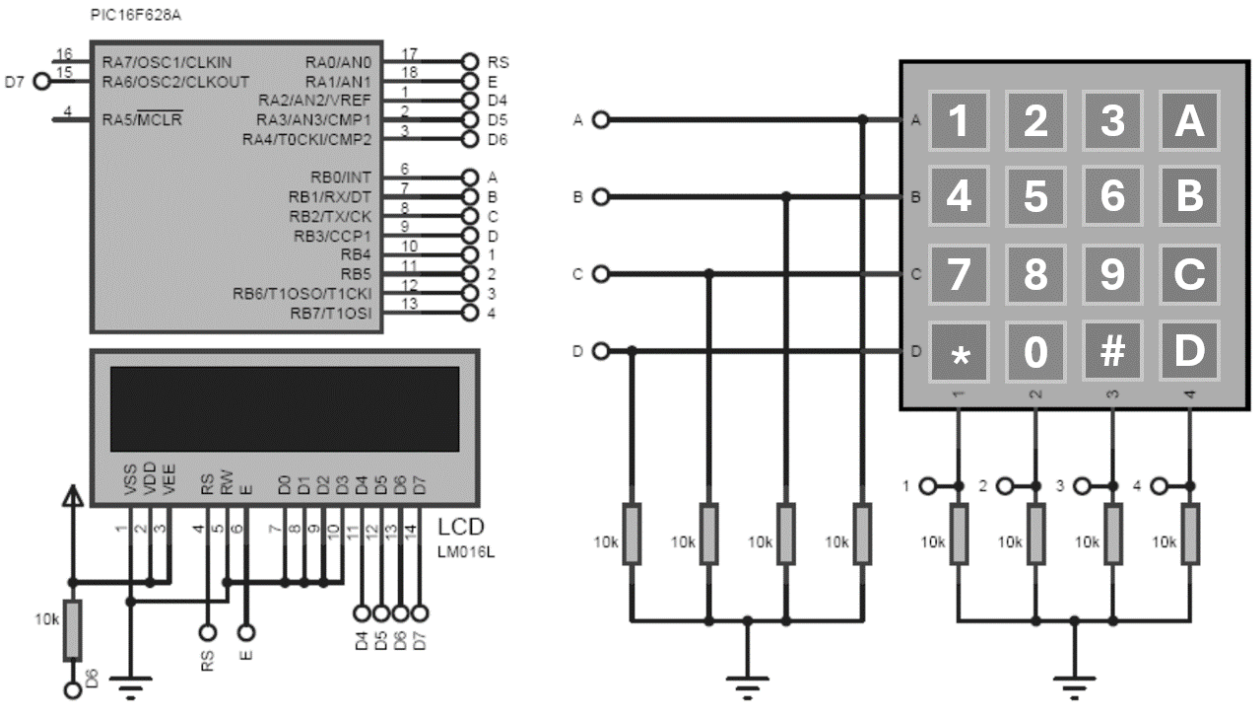


Figure 11.2: LCD interfacing circuit with PIC16F268A.

Template Code

```
// Step 1: Define variables
    // Hint: These variables will store keypad input, calculation results, and LCD display values
    unsigned short kp = ****; // Hint: Keypad press value
    int PW1 = ****; // Hint: First operand part 1
    int PW2 = ****; // Hint: First operand part 2
    float PW3 = ****; // Hint: Operator (Division, Multiply, Subtraction, Addition)
    int PW4 = ****; // Hint: Second operand part 1
    int PW5 = ****; // Hint: Second operand part 2
    int val = ****; // Hint: Value corresponding to the key pressed
    int number1 = ****; // Hint: First operand value
    int number2 = ****; // Hint: Second operand value
    int digit1 = ****; // Hint: Flag for first digit input
    int digit2 = ****; // Hint: Flag for second digit input
    int digit3 = ****; // Hint: Flag for third digit input
    int digit4 = ****; // Hint: Flag for fourth digit input
    int digit5 = ****; // Hint: Flag for fifth digit input
    int Division = ****; // Hint: Division operator value
    int Multiply = ****; // Hint: Multiplication operator value
    int Substractive = ****; // Hint: Subtraction operator value
    int Add = ****; // Hint: Addition operator value
    unsigned char answerx[15]; // Hint: Array to hold the result for display
    long Answer = ****; // Hint: Result of the calculation

// Step 2: Keypad and LCD module connections
    char keypadPort at PORTB; // Hint: Define the port used for the keypad
    **** // Hint: Include necessary libraries for LCD

void main() {
  // Step 3: Initialize configuration settings
  CMCON = ****; // Hint: Disable comparators
  CCP1CON = ****; // Hint: Disable CCP1 (Capture/Compare/PWM module)
  VRCON = ****; // Hint: Disable Voltage Reference module
  T1CON = ****; // Hint: Disable Timer1
```

```
// Step 4: Initialize Keypad and LCD

  Keypad_Init(); // Hint: Initialize Keypad
  Lcd_Init(); // Hint: Initialize LCD
  Lcd_Cmd(_LCD_CLEAR); // Clear LCD display
  Lcd_Cmd(_LCD_CURSOR_OFF); // Turn off cursor
  Lcd_Out(1, 1, "Enter Values"); // Display prompt message
  Lcd_Cmd(_LCD_SECOND_ROW); // Move cursor to second row
  Lcd_Cmd(_LCD_BLINK_CURSOR_ON); // Turn on cursor blinking for user interaction

  // Step 5: Define the infinite loop
  while (****) {// Hint: Enter an appropriate condition for the loop
    kp = 0; // Reset keypad press value

  // Step 6: Wait for key to be pressed and released
        do {
      kp = Keypad_Key_Click(); // Read keypad press
    } while (!kp);

    // Step 7: Convert keypad press to corresponding value
      switch (kp) {
      case 1: kp = 49; val = 1; break; // Key '1'

     ****// Hint: Add cases for “2 to 9, 0, *, /, +, -” according to the keypad in Figure 1
    }

    // Step 8: Handle input and update display
    // Hint: Manage digit inputs and perform calculations based on the operator
    if (digit1 == 0) {
      if (val == Add || val == Substractive || val == Multiply || val == Division) {
        digit1 = 0; // Reset if an operator is detected
      } else if (val) {
        Lcd_Chr(2, 1, kp); // Display pressed key on LCD
        PW1 = (val == 99) ? 0 : val; // Update PW1 based on the key pressed
        digit1 = 1; // Set digit1 flag
      }
    }

    // Step 9: Process additional digits
    ****// Hint: Continue similar logic for digit 2, digit 3, digit 4, and digit 5
```

```
      // Step 10: Perform calculation when '=' is pressed
         if (val == 61) {
         Lcd_Cmd(****); // Hint: Turn off cursor
         number1 = (PW1 * 10) + PW2; // Prepare operands
         number2 = (PW4 * 10) + PW5; // Prepare operands

         // Perform calculation based on the operator
         if (PW3 == 10) Answer = number1 / number2; // Division
         else if (PW3 == 13) Answer = number1 + number2; // Addition
         else if (PW3 == 12) Answer = number1 - number2; // Subtraction
         else if (PW3 == 11) Answer = number1 * number2; // Multiplication

         Lcd_Cmd(****); // Hint: Clear display
         Lcd_Cmd(****); // Hint: Turn off cursor
         Lcd_Out(1, 1, "Answer="); // Display result label
         IntToStr(Answer, answerx); // Convert result to string
         Lcd_Out(2, 1, answerx); // Display result on LCD

         // Wait for key press to continue
         while (val == 0) {
            do {
               kp = Keypad_Key_Click(); // Read keypad press
            } while (!kp);

            // Check for reset key (e.g., key '4') to restart input
            if (kp == 4) {
               Lcd_Cmd(****); // Hint: Clear display
               Lcd_Cmd(****); // Hint: Turn off cursor
               Lcd_Out(1, 1, "Enter Values"); // Display prompt
               val = 1; // Reset val for new input
            }
         }
         // Reset variables for next calculation
         val = number1 = number2 = PW1 = PW2 = PW3 = PW4 = PW5 = digit1 = digit2 = digit3 = digit4
= digit5 = Add = Answer = 0;
      }
   }
}
```

Execute the **** sections in the code to finalize the program.

References

1. Wilmshurst, T., (2009), Designing Embedded Systems with PIC Microcontrollers, Principles and Applications, 2nd Edition, Newnes
2. Valvano, J. W., (2012), Embedded Microcomputer Systems: Real Time Interfacing, 3rd Edition, CL Engineering
3. Mazidi, M. A., Mazidi, J. G., & McKinlay, R. D., (2007), The 8051 Microcontroller and Embedded Systems Using Assembly and C, 2nd Edition, Pearson
4. Lipovski, G. J. & Irwin, J. D., (2000), Embedded Microcontroller Interfacing for M COMPULSORY Systems, 1st Edition, Academic Press

Department of Physics and Electronics

University of Kelaniya

Kelaniya

Sri Lanka

2024