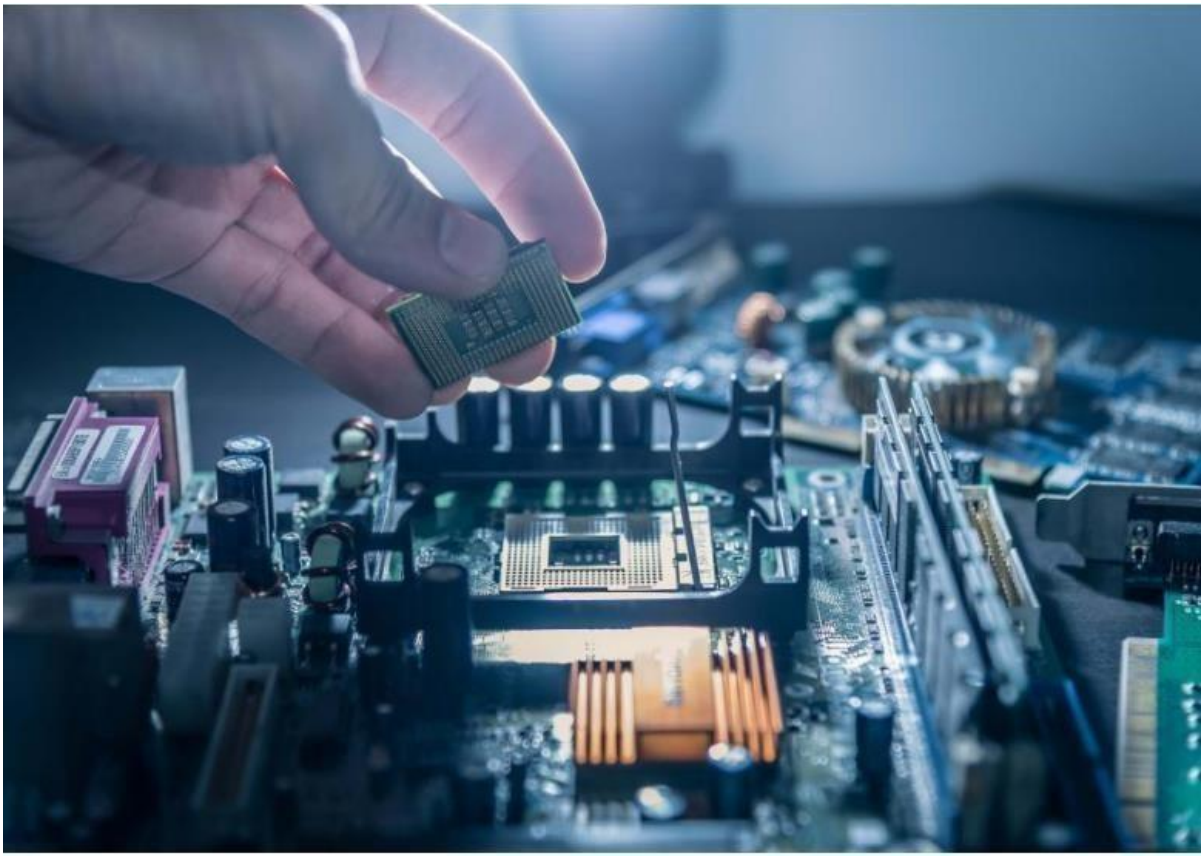
EMBEDDED SYSTEMS LAB

EXPERIMENT 2: Four-input Combinational Logic

CHANAKYA DUPPATLA: 21EC30017

RAHUL SAMINENI: 21EC30045



EMBEDDED SYSTEMS LABORATORY(EC39302)

DATE: 07/02/24

PART– A

Objective: To demonstrate the glowing of LEDs using both an external power supply with current-limiting resistors and through an IC 74245 with current-limiting resistors.

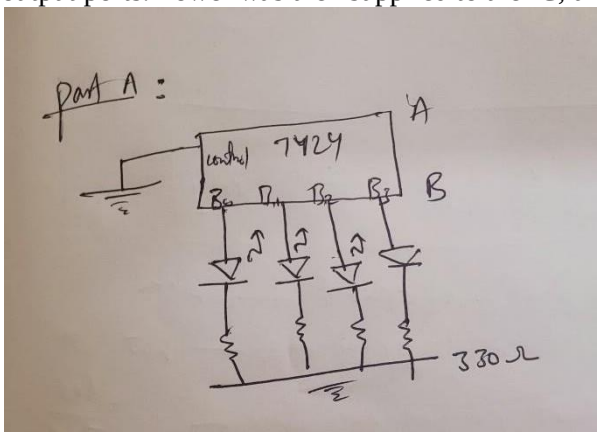
Procedure:

1) External Power Supply:

➔ The positive terminal (+) of the external power supply was linked to one end of the current-limiting resistor.
➔ The opposite end of the current-limiting resistor was joined to the anode (identified by the longer lead) of the LED.
➔ The cathode (identified by the shorter lead) of the LED was linked to the negative terminal (-) of the external power supply.
➔ Activate the power supply and regulate the voltage to match the suitable level for the LEDs in use.

2) Through IC 74245:

➔ The pins of the IC 74245 were identified: typically, it consists of 20 pins organized in two rows.
➔ The VCC pin of the IC was linked to the positive terminal of the power supply, while the GND pin of the IC was connected to the negative terminal of the power supply.
➔ One end of a current-limiting resistor was attached to each output port pin of the IC. The opposite end of the current-limiting resistor was then connected to the anode (identified by the longer lead) of each LED.
➔ Each LED was carefully connected through its corresponding current-limiting resistor to prevent any possibility of excessive current flow.
➔ The appropriate input signals were applied to the control pins of the IC to activate the desired output ports. Power was then supplied to the IC, and the LEDs were observed to illuminate according to the provided input signals.

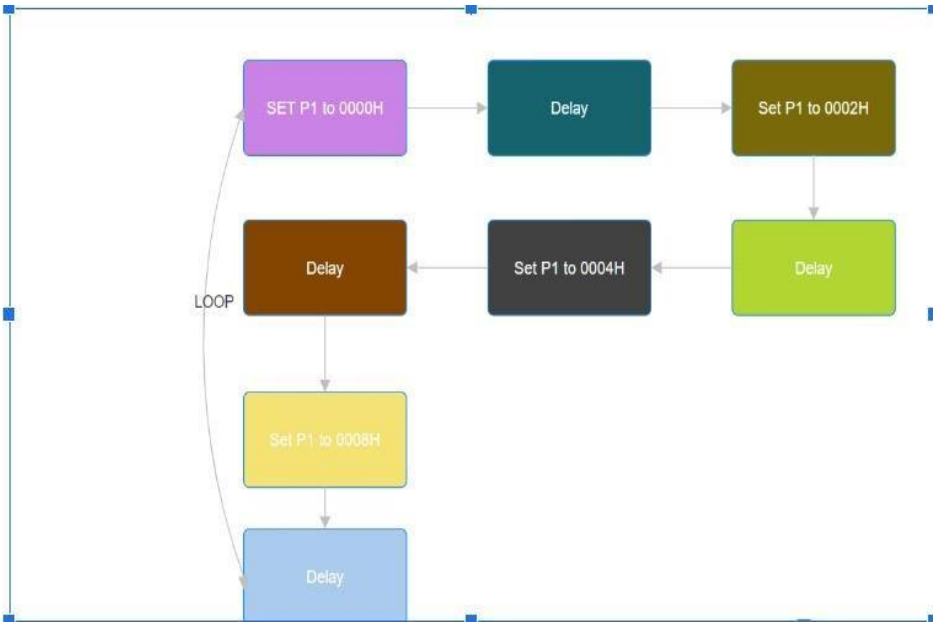


<<Circuit diagram

PART-B:

Objective: The objective of this experiment is to blink four LEDs sequentially with anON time and OFF time of 1 second each, using an 8051microcontroller kit. This involves programming the microcontroller to control the LEDs connected to its ports,ensuring sequential blinking and proper timing.

Flow Chart:



Pseudo code:

Set ORG to address 8100H

Clear P1.0, P1.1, P1.2, and P1.3

LIGHTS:

Set A to 1H

Move A to P1 (P1.0 will be set)

Call DELAY subroutine

Clear P1.0

Set A to 2H

Move A to P1 (P1.1 will be set)

Call DELAY subroutine

Clear P1.1

Set A to 4H

Move A to P1 (P1.2 will be set)

Call DELAY subroutine

Clear P1.2

Set A to 8H

Move A to P1 (P1.3 will be set)

Call DELAY subroutine

Clear P1.3

Clear A

Jump to LIGHTS

DELAY subroutine:

Set R7 to 10

DELAY2:

Set R1 to 240

WAITSTART:

Set R2 to 216

WAIT:

Decrement R2 and jump to WAIT until R2 becomes zero

Decrement R1 and jump to WAITSTART until R1 becomes zero

Decrement R7 and jump to DELAY2 until R7 becomes zero

Return from subroutine

END

Code:

```
ORG 8100H
```

```
CLR P1.0
CLR P1.1
CLR P1.2
CLR P1.3

;MOV DPTR,#0E801H
LIGHTS:
MOV A,#1H
MOV P1,A
ACALL DELAY
CLR P1.0
MOV A,#2H
MOV P1,A
ACALL DELAY
CLR P1.1
MOV A,#4H
MOV P1,A
ACALL DELAY
CLR P1.2
MOV A,#8H
MOV P1,A
ACALL DELAY
CLR P1.3
CLR A
SJMP LIGHTS
DELAY:
MOV R7,#10
```

```
DELAY2:

MOV R1,#240

WAITSTART:

MOV R2,#216

WAIT:

DJNZ R2,WAIT

DJNZ R1,WAITSTART

DJNZ R7,DELAY2

RET


END
```

CODE EXPLANATION:

This assembly code is designed to control output pins to create a light pattern. It consists of two main parts: the main loop labeled LIGHTS and a subroutine labeled DELAY. Here's a brief explanation:

1. Main Loop (LIGHTS):

- The main loop repeatedly cycles through turning on and off output pins connected to port 1, creating a light pattern.

- It sequentially turns on each output pin (P1.0 to P1.3) for a short duration and then turns it off before moving to the next pin.

- This loop runs indefinitely, creating a continuous light pattern.

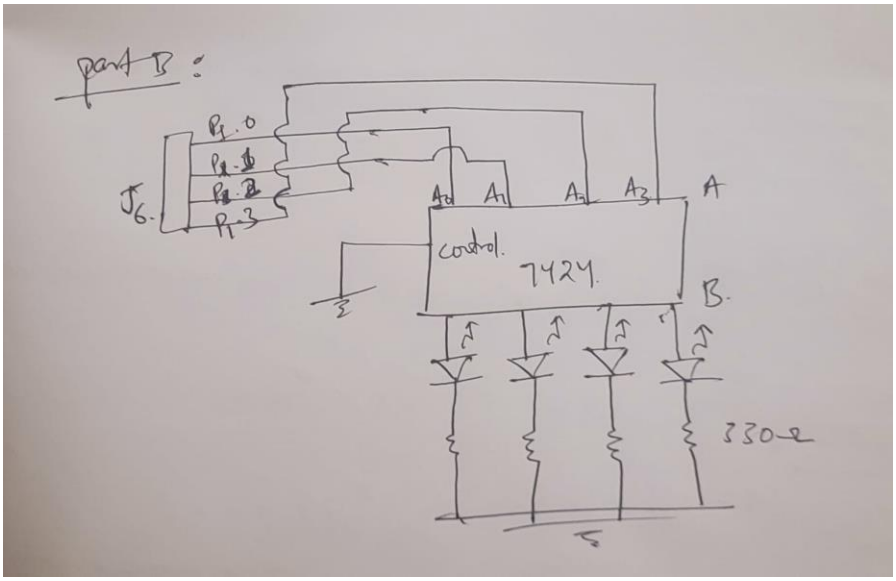
2. Subroutine (DELAY):

- The DELAY subroutine introduces a delay to control the timing of the light pattern.

- It utilizes nested loops to create a delay of a specific duration.

- The delay duration is determined by the number of iterations in the nested loops, providing a way to control the speed of the light pattern.

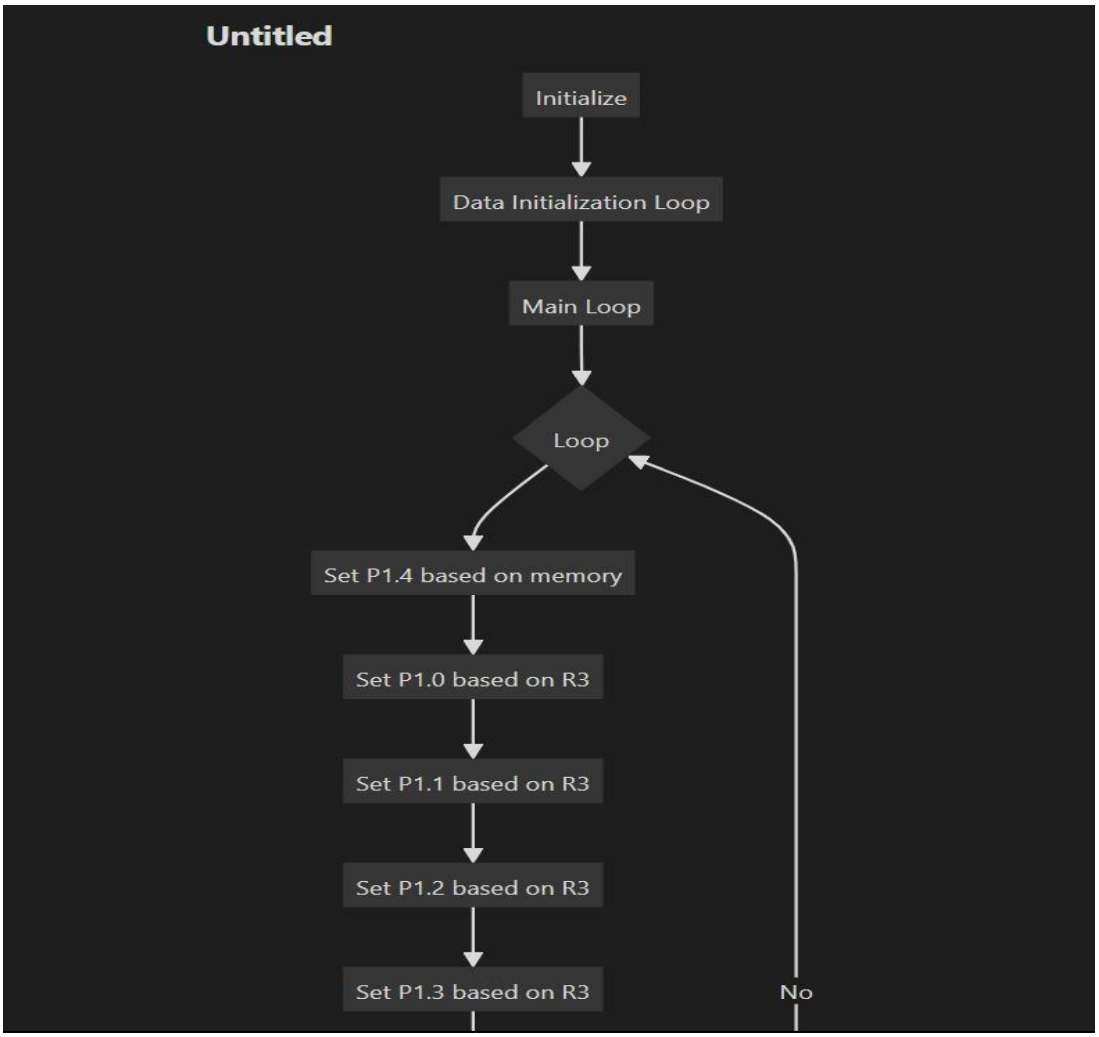
Circuit>>

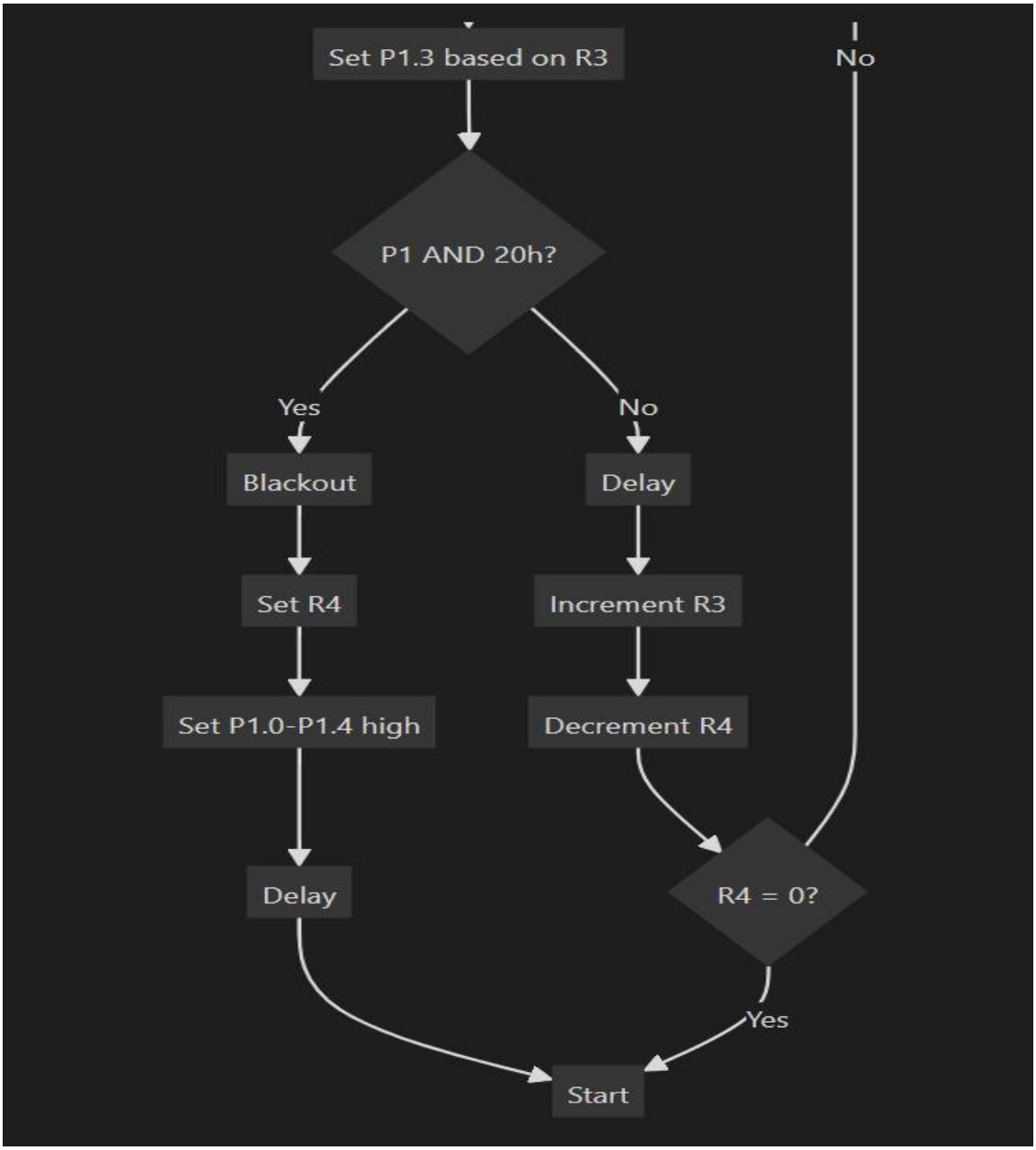


PART-C:

Objective: The objective of this experiment is to design a circuit that utilises five LEDs to represent four input combinations (A, B, C, D) and the corresponding output of a four-input combinational logic function. The circuit will cycle through all possible input combinations (0000 to 1111) and display the corresponding output on the fifth LED. Additionally, the experiment involves interfacing an LDR (Light Dependent Resistor) to the circuit. When the LDR is covered, all LEDs should blink, indicating a special mode. Upon uncovering the LDR, the circuit should return to its normal mode, starting from input combination 0000. The experiment aims to verify the proper functioning of the circuit and its ability to respond to changes in the truth-table content.

Flow Chart:




Set P1.3 based on R3
No
P1 AND 20h?
Yes
No
Blackout
Delay
Set R4
Increment R3
Set P1.0-P1.4 high
Decrement R4
Delay
R4 = 0?
Yes
Start

Pseudo Code:

Initialize:
- Set P1 to 00001000 (bit 3 high)
- Clear P1.0 to P1.4
- Set DPTR to 9000h
- Set R3 to 08h

Data Initialization Loop:
- Move 1 to memory location DPTR
- Increment DPTR
- Move 0 to memory location DPTR
- Increment DPTR
- Repeat 8 times

Main Loop:
- Set R4 to 16
- Set R3 to 0
- Set DPTR to 9000h

Loop:
- Move R3 to DPL
- Move memory contents at DPTR to A
- If A is 1, set P1.4 high, else clear P1.4
- If R3 AND 01h is 01h, set P1.0 high, else clear P1.0
- If R3 AND 02h is 02h, set P1.1 high, else clear P1.1
- If R3 AND 04h is 04h, set P1.2 high, else clear P1.2
- If R3 AND 08h is 08h, set P1.3 high, else clear P1.3

- If P1 AND 20h is 20h, go to Blackout
- Delay loop

- Increment R3
- Decrement R4
- If R4 not 0, repeat loop
- Else go to start

Blackout:
- Set R4 to 17
- Set P1.0 to P1.4 high
- Delay loop
- Go to start

Code:

```
ORG 8100H
MOV P1, #00100000B

strt:
CLR P1.0
```

```
CLR P1.1
CLR P1.2
CLR P1.3
CLR P1.4


MOV DPTR, #9000H
MOV R3, #08H

data_loop :
MOV A, #1
MOVX @DPTR,A
INC DPTR
MOV A,#0
MOVX @DPTR,A
INC DPTR
DJNZ R3, data_loop
MOV R4, #0x10
MOV R3,#0H
MOV DPTR,#9000H
loop :
MOV DPL, R3

MOVX A, @DPTR
CJNE A, #1, clearo
SETB P1.4

cont :

MOV A, R3
MOV DPL, A
ANL A, #0x01
CJNE A, #0x01, d_set
SETB P1.0

cont_c:
MOV A, R3
MOV DPL, A
ANL A, #0x02
CJNE A, #0x02, c_set
SETB P1.1

cont_b:
MOV A, R3
MOV DPL, A
ANL A, #0x04
CJNE A, #0x04, b_set
SETB P1.2
```

```
cont_a:
MOV A, R3
MOV DPL, A
ANL A, #0x08
CJNE A, #0x08, a_set
SETB P1.3



cont_d:
b2b :
MOV A, P1
ANL A, #0x20
CJNE A, #0x20, blackout



MOV R0, #255H

a1:
a2:
a3:
DJNZ R2, a3
MOV R2, #255H
DJNZ R1, a2
MOV R1, #255H
DJNZ R0, a1
INC R3
DJNZ R4, loop


SJMP strt

d_set :
CLR P1.0
SJMP cont_c

c_set :
CLR P1.1
SJMP cont_b

b_set :
CLR P1.2
SJMP cont_a

a_set :
CLR P1.3
SJMP cont_d
```

```
clearo:
CLR P1.4
SJMP cont

blackout:
MOV R4, #0x11

SETB P1.0
SETB P1.1
SETB P1.2
SETB P1.3
SETB P1.4
SJMP b2b
END
```

Code Explanation:

1. Initialization:
   - ORG 8100H: Specifies the origin address of the program.
   - MOV P1, #00100000B: Initializes Port 1 (P1) with the binary value 00100000B, setting only the 6th bit high and the rest low.

2. Main Program:
   - strt: Label marking the start of the program.
   - CLR P1.0, CLR P1.1, CLR P1.2, CLR P1.3, CLR P1.4: Clears all bits of Port 1, ensuring all LEDs are initially turned off.

3. Data Initialization Loop:
   - MOV DPTR, #9000H: Initializes the data pointer (DPTR) to memory address 9000H.
   - MOV R3, #08H: Initializes R3 to 08H, indicating the number of iterations for data initialization.
   - data_loop: Label marking the start of the data initialization loop.
   - Within the loop:
     - MOV A, #1: Loads accumulator A with the value 1.
     - MOVX @DPTR, A: Writes the value of A to the memory location pointed by DPTR.
     - INC DPTR: Increments the data pointer to point to the next memory location.
     - MOV A, #0: Loads accumulator A with the value 0.
     - MOVX @DPTR, A: Writes the value of A to the next memory location.
     - INC DPTR: Increments the data pointer.
     - DJNZ R3, data_loop: Decrements R3 and repeats the loop until R3 becomes zero.

4. Main Loop:
   - MOV R4, #0x10: Initializes R4 to the value 0x10.
   - MOV R3, #0H: Initializes R3 to 0.
   - MOV DPTR, #9000H: Resets the data pointer to the beginning of the data.
   - loop: Label marking the start of the main loop.
   - Within the loop:
     - MOV DPL, R3: Loads DPL with the value of R3.
     - MOVX A, @DPTR: Reads the value from the memory location pointed by DPTR into A.
     - CJNE A, #1, clearo: Compares A with 1. If not equal, jumps to the clearo label.
     - SETB P1.4: Sets bit 4 of Port 1, turning on the fifth LED.

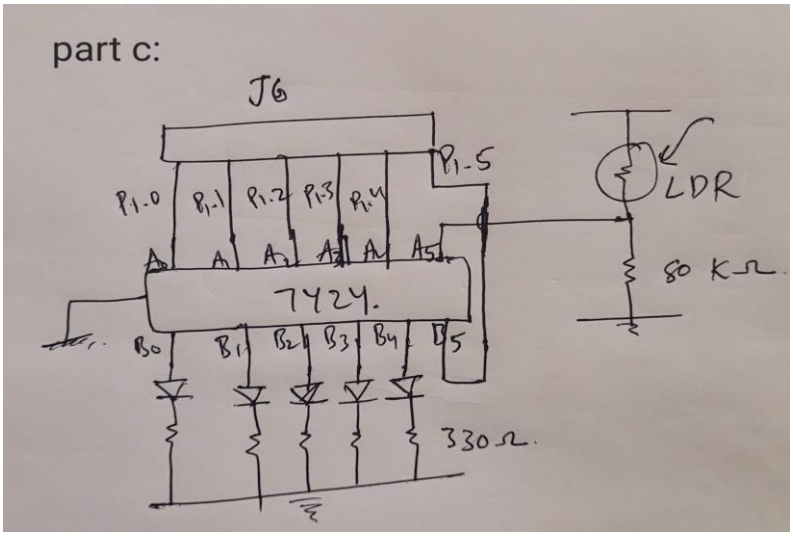
5. LED Control Sections:
   - Sections cont, cont_c, cont_b, cont_a, and cont_d control the other LEDs based on the value of R3.
   - Each section checks a specific bit of R3 and sets the corresponding LED on Port 1 accordingly.

6. Control Flow:
   - Various conditional jumps (SJMP) are used to control the flow of the program based on the LED states and loop counters.

7. Delay Loop (blackout):
   - This section sets all LEDs and creates a delay loop using nested loops to introduce a delay before resetting the LEDs.

8. End of Program:
   - END: Marks the end of the program.

Circuit:

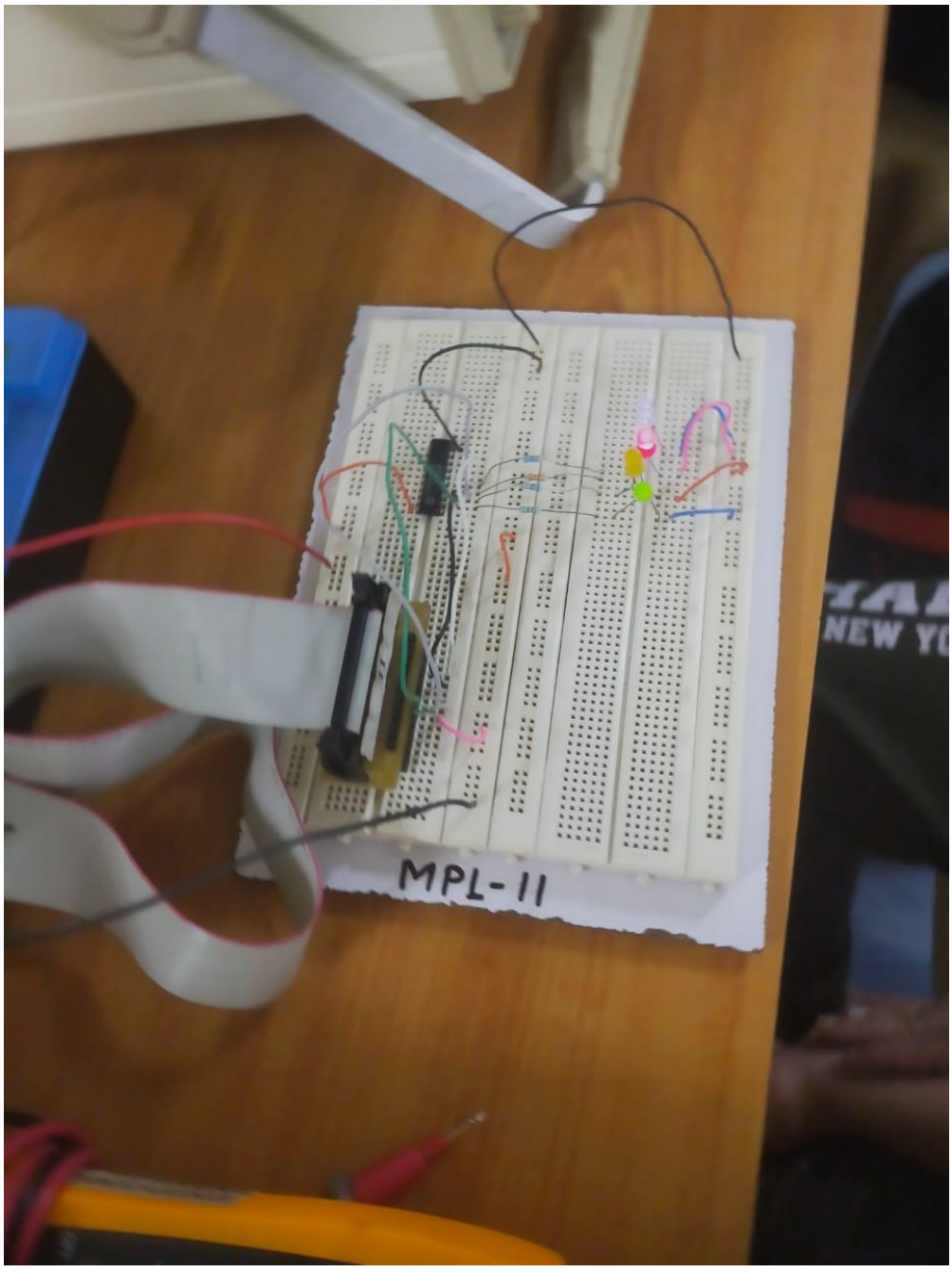


RESULTS:

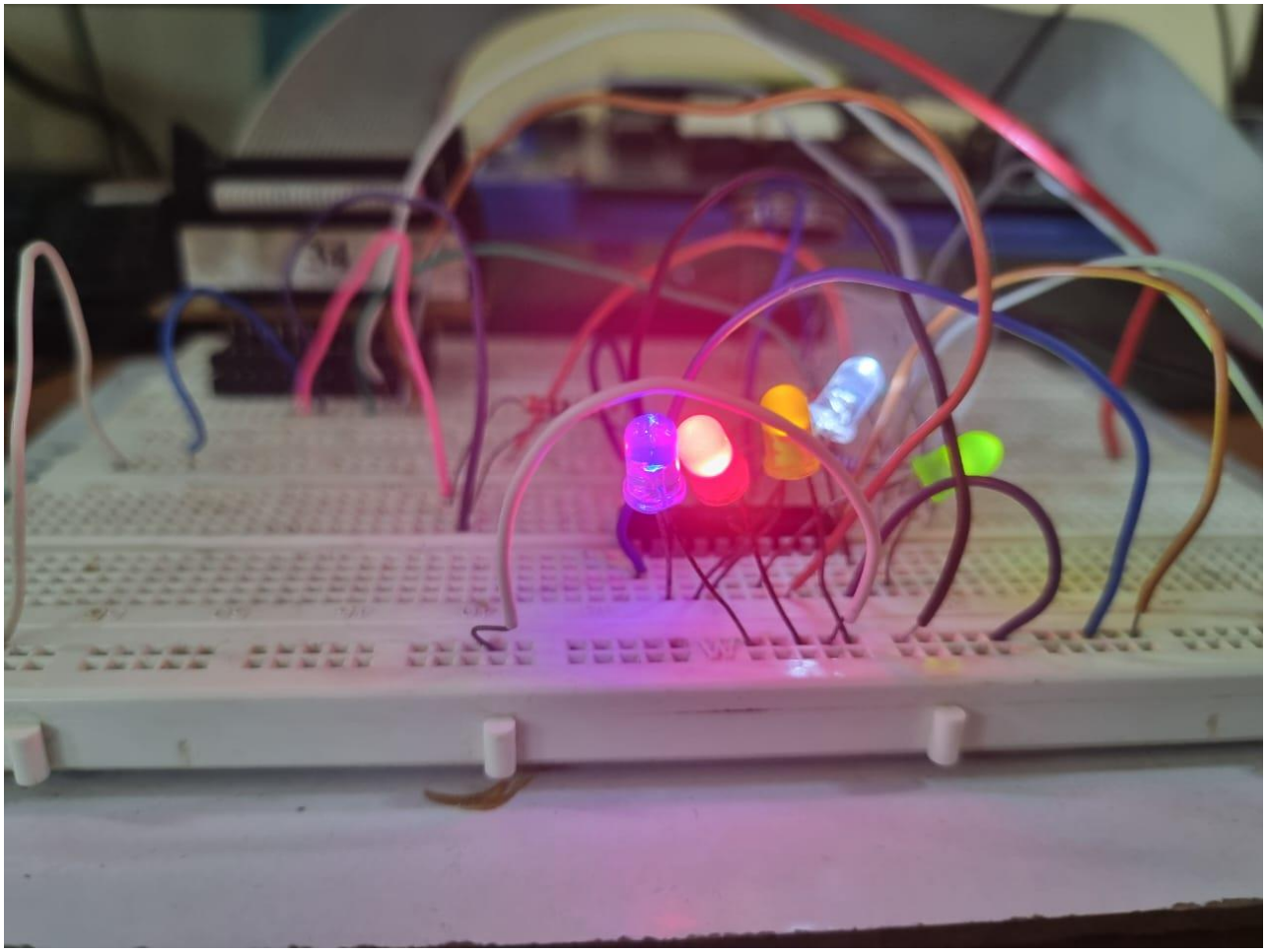
Part A:

Part B:

-



Part C:



Discussion:

Part A:

Illustrating LED illumination via an external power supply with current-limiting resistors and employing an IC 74245 reveals insights into two distinct LED driving methods:

1. External Power Supply:
   - LEDs are linked to an external power source via resistors.
   - Incorporation of resistors ensures LEDs are shielded from excessive current flow, thereby ensuring safe operation.
   - Adjusting LED brightness is achievable by altering resistor values.
   - This method provides a straightforward and adaptable approach to regulate brightness levels.

2. IC 74245:
   - The IC 74245 facilitates LED driving and voltage-level translation tasks.
   - Incorporating current-limiting resistors is essential to safeguard the LEDs.
   - It permits programmable LED manipulation by interfacing with microcontrollers.
   - Integration into intricate electronic systems is simplified, offering enhanced versatility and compatibility.

Part B**:**

- ➔ This assembly code is tailored for a 8051 microcontroller, programmed to systematically iterate through all potential four-input combinations (0000 to 1111) and visually represent them using LEDs.
- ➔ The program begins by resetting the output ports linked to the LEDs corresponding to inputs A, B, C, and D. Subsequently, it enters a primary loop where it sequentially cycles through each input combination.
- ➔ For each combination, it activates the relevant input LEDs, pauses briefly to showcase the combination, and subsequently deactivates the LEDs before progressing to the next combination.
- ➔ The program employs nested loops and decrement-and-jump instructions (DJNZ) to introduce delays for showcasing each combination. After displaying all combinations, the program enters an infinite loop to sustain

  the last displayed combination until it is either interrupted or reset.

➔ In essence, the code systematically iterates through every conceivable input combination and visually represents them using LEDs.

- ➔ The delay introduced between combinations guarantees that each combination is observable before transitioning to the next. After showcasing all combinations, the program enters an infinite loop, persisting with the display of the last shown combination until either interrupted or reset.

Part C:

- The following assembly code is designed for configuring LEDs connected to port P1 on an 8051 microcontroller. It orchestrates the LEDs to visually represent the various input combinations and outputs of a four-input combinational logic function.
- The provided assembly code sequentially iterates through all potential input combinations from 0000 to 1111. It utilizes an extra LED connected to port P1, specifically P1.4, to indicate the corresponding output for each combination. Upon detecting a valid sequence, it proceeds to display the subsequent combination in the sequence.
- In case the sequence is disrupted, indicating an error or interruption, the program switches to a blackout mode where it illuminates all LEDs until the continuity is reestablished. Subsequently, the program enters an indefinite loop, continuously cycling through the input combinations and adjusting the output LED accordingly. This mechanism ensures both proper functionality and effective error handling.
- This code orchestrates a sequence to exhibit the output of a four-input combinational logic function through LEDs. It guarantees orderly progression through the sequence and addresses continuity problems by engaging a blackout mode when necessary.

THE END