Microprocessor and Microcontroller Course

Course Overview

This course covers the fundamentals of the 8085 and 8086 microprocessors and the Arduino microcontroller, focusing on architecture, programming, and practical applications. The index below organizes the topics (1–18) from the student handout, mapping each to specific **Course Learning Outcomes (CLOs)** to clarify their role in achieving course objectives. Use this index to navigate topics and understand their educational purpose.

Course Learning Outcomes (CLOs)

1. **CLO1**: Understand the architecture and instruction set of the 8085 and 8086 microprocessors.
2. **CLO2**: Write and simulate assembly language programs for 8085 and 8086 to perform data transfer, arithmetic, and control flow operations.
3. **CLO3**: Manipulate memory and registers using appropriate instructions in 8085 and 8086.
4. **CLO4**: Apply bitwise operations and looping constructs to control program flow and data in microprocessors.
5. **CLO5**: Develop and debug Arduino programs for digital/analog I/O, interrupts, and sensor/actuator interfacing.
6. **CLO6**: Integrate multiple microcontroller concepts to design and implement a functional Arduino project.
7. **CLO7**: Use simulation tools to test and verify microprocessor and microcontroller programs.

Index with CLO Mapping

Topic	Title	Description	CLO Mapping
1	Introduction to 8085 and Data Transfer Commands	Learn 8085 architecture, registers (A, B, C, D, E, H, L), memory addressing, and data transfer instructions (`MVI`, `MOV`, `LXI`, `HLT`). Run programs in a simulator.	CLO1, CLO2, CLO3, CLO7
2	Memory Operations and Storage (8085)	Interact with memory using `STA`, `LDA`, `MOV M, A`, `MOV A, M`. Use HL pair as memory pointer and set memory values in a simulator.	CLO1, CLO2, CLO3, CLO7
3	Arithmetic Operations (8085)	Perform addition (`ADD`) and subtraction (`SUB`) with registers and memory. Understand flags (Zero, Carry, Sign) and store results using `STA`.	CLO1, CLO2, CLO3, CLO7
4	Comparison and Conditional Jumps (8085)	Use `CMP` for comparisons and `JZ`, `JNZ` for conditional jumps. Define labels to control program flow based on flags.	CLO1, CLO2, CLO4, CLO7
5	Looping and Increment/Decrement (8085)	Create loops using `INR`, `DCR`, and `JNZ` with a counter register. Combine with arithmetic/data transfer for iterative tasks.	CLO1, CLO2, CLO4, CLO7
6	Introduction to 8086 and Data Transfer Commands	Understand 8086 architecture, 16-bit registers (AX, BX, CX, DX), segment registers, and data transfer (`MOV`, `LEA`). Simulate in emu8086.	CLO1, CLO2, CLO3, CLO7
7	Memory Operations and Storage (8086)	Use direct (`[1000H]`) and indirect (`[BX]`) memory addressing with `MOV`. Load/store data in memory and verify in emulator.	CLO1, CLO2, CLO3, CLO7
8	Arithmetic Operations (8086)	Perform `ADD`, `SUB`, `INC`, `DEC` operations. Monitor Zero, Carry, Sign flags in emu8086.	CLO1, CLO2, CLO3, CLO7
9	Comparison and Conditional Jumps (8086)	Use `CMP` and jumps (`JZ`, `JNZ`, `JA`, `JG`) for decision-making. Define labels for program flow control.	CLO1, CLO2, CLO4, CLO7

10	Looping (8086)	Use `LOOP` instruction with CX counter for repetitive tasks. Simulate nested loops in emu8086.	CLO1, CLO2, CLO4, CLO7
11	Logical and Bitwise Operations (8086)	Apply `AND`, `OR`, `XOR`, `NOT` for bit manipulation. Use masking to set/clear/toggle bits.	CLO1, CLO2, CLO4, CLO7
12	Introduction to Arduino and Digital I/O	Learn Arduino Uno, ATmega328P, and digital I/O (`pinMode`, `digitalWrite`, `digitalRead`). Control LEDs and buttons.	CLO5, CLO7
13	Analog Input and PWM	Read analog inputs (`analogRead`) from A0–A5 and control PWM outputs (`analogWrite`) for LED brightness.	CLO5, CLO7
14	Serial Communication	Use `Serial.begin`, `Serial.print`, `Serial.read` for debugging and interaction via Serial Monitor.	CLO5, CLO7
15	Interrupts and Timers	Implement hardware interrupts (`attachInterrupt`) on pins 2/3 and non-blocking timing with `millis()`.	CLO5, CLO7
16	Sensors and Libraries	Interface with DHT11 and HC-SR04 sensors using `DHT` and `NewPing` libraries. Read and process sensor data.	CLO5, CLO7
17	Motors and Actuators	Control servo motors with `Servo.h` and PWM. Explore DC motor basics if time permits.	CLO5, CLO7
18	Final Project and Integration	Build a project integrating sensors (DHT11, photoresistor), actuators (servo, LED), and logic. Debug with Serial Monitor.	CLO5, CLO6, CLO7

CLO Mapping Summary

- **CLO1 (Architecture and Instruction Set)**: Topics 1–11 cover 8085/8086 architecture and instructions.
- **CLO2 (Assembly Programming)**: Topics 1–11 involve writing/simulating assembly programs.
- **CLO3 (Memory/Register Manipulation)**: Topics 1–3, 6–8 focus on memory/register operations.
- **CLO4 (Bitwise/Looping Constructs)**: Topics 4–5, 9–11 address comparisons, jumps, loops, and bitwise operations.
- **CLO5 (Arduino Programming)**: Topics 12–17 cover Arduino I/O, interrupts, sensors, and actuators.
- **CLO6 (Arduino Project Integration)**: Topic 18 integrates concepts for a complete project.
- **CLO7 (Simulation Tools)**: All topics use simulators (GNUSim8085, emu8086, Arduino IDE/Tinkercad).

Notes

- **Simulators**: Use GNUSim8085 (8085), emu8086 (8086), and Arduino IDE/Tinkercad (Arduino).
- **Hardware**: Arduino Uno, LEDs, 220Ω resistors, pushbuttons, potentiometer, DHT11, HC-SR04, servo.
- **Safety**: Use resistors with LEDs; verify servo power ratings.
- **Resources**: Arduino.cc, simulator manuals, library documentation.
- **Troubleshooting**: Check connections, monitor flags/registers (8085/8086), use Serial Monitor (Arduino).

8085 Microprocessor Topics

Topic 1: Introduction to 8085 and Data Transfer Commands

Objective: Understand 8085 architecture and basic data transfer instructions.
Key Concepts:

- **Registers**: A (accumulator), B, C, D, E, H, L.
- **Memory Addressing**: 16-bit addresses (e.g., 2000H).
- **Program Counter (PC)**: Tracks next instruction.
- **Stack Pointer (SP)**: Manages stack for subroutine calls.
- **Instructions**:
  - `MVI R, data`: Move immediate data to register (e.g., `MVI A, 25H`).
  - `MOV Rd, Rs`: Move data between registers (e.g., `MOV B, A`).
  - `LXI Rp, addr`: Load register pair (e.g., `LXI H, 2000H`).
  - `HLT`: Halt execution.
- **Program Execution**: Instructions fetched and executed via simulator.

Example Program:

```
MVI A, 25H  ; Load accumulator with 25H
MOV B, A    ; Copy A to B
HLT         ; Stop execution
```

Explanation: Loads 25H into A, copies to B, halts.

Exercises:

1. Write a program to load 10H into C and move it to D.
2. Modify the example to load 50H into A and move it to B and C.
3. Run in a simulator and verify register values.

Learning Outcome: Move data between registers and use a simulator.

Topic 2: Memory Operations and Storage

Objective: Interact with memory using load/store instructions.
Key Concepts:

- **Memory Addressing**: 16-bit addresses; HL pair as memory pointer.
- **Instructions**:
  - `STA addr`: Store A in memory (e.g., `STA 2001H`).
  - `LDA addr`: Load A from memory (e.g., `LDA 2000H`).
  - `MOV M, A`: Store A in memory pointed by HL.
  - `MOV A, M`: Load A from memory pointed by HL.
- **Simulator**: Set memory values (e.g., 03H at 2000H).

Example Program:

```
MVI A, 08H   ; Load A with 08H

LXI H, 2000H ; Set HL to 2000H

MOV M, A     ; Store A at 2000H

STA 2001H    ; Store A at 2001H

HLT          ; Stop execution
```

Explanation: Loads 08H into A, stores it at 2000H (via HL) and 2001H.

Exercises:

1. Load 15H into A and store at 3000H using STA.
2. Load value from 2000H into A using LDA, then move to B.
3. Set 07H at 2000H in simulator, load it, and store at 2001H.

Learning Outcome: Read/write to memory locations.

Topic 3: Arithmetic Operations

Objective: Perform addition and subtraction with flags.
Key Concepts:

- **Instructions**:
  - `ADD R`: Add register to A (e.g., `ADD B`).
  - `ADD M`: Add memory (via HL) to A.

    - `SUB R`: Subtract register from A.
    - `SUB M`: Subtract memory from A.
- **Flags**: Zero (ZF), Carry (CF), Sign (SF) affected by operations.
- **Storage**: Use `STA` for results.

Example Program:

```
LXI H, 2000H ; Set HL to 2000H

MVI A, 05H   ; Load A with 05H

ADD M        ; Add memory content to A

STA 2001H    ; Store result at 2001H

HLT          ; Stop execution
```

Explanation: Adds memory value (e.g., 03H at 2000H) to 05H in A, stores 08H at 2001H.

Exercises:

1. Add 10H (A) and 05H (B), store result at 2500H.
2. Subtract 03H (at 2000H) from 08H (A), store at 2002H.
3. Add values in C and D, store result in memory.

Learning Outcome: Perform arithmetic and understand flag changes.

Topic 4: Comparison and Conditional Jumps

Objective: Compare values and control program flow.
Key Concepts:

- **Comparison**:
    - `CMP R`: Compare register with A.
    - `CMP M`: Compare memory (via HL) with A.
    - Sets flags (ZF, CF) without changing A.
- **Jumps**:
    - `JZ label`: Jump if equal (ZF=1).
    - `JNZ label`: Jump if not equal (ZF=0).
- **Labels**: Define jump targets (e.g., `EQUAL:`).

Example Program:

```
MVI A, 0AH   ; Load A with 0AH
MVI B, 0BH   ; Load B with 0BH
CMP B        ; Compare B with A
JZ EQUAL     ; Jump if equal
HLT          ; Stop if not equal
EQUAL: MVI A, 01H ; Load A with 01H
STA 2000H    ; Store at 2000H
HLT          ; Stop execution
```

Explanation: Compares 0AH (A) with 0BH (B); no jump since unequal, halts. If equal, stores 01H at 2000H.

Exercises:

1. Compare 05H (A) with 05H (C), store 01H at 3000H if equal.
2. Modify example to jump if A > B (use `JC` for Carry flag).
3. Compare memory at 2000H with 07H in A, indicate equality.

Learning Outcome: Use comparisons and flags for decision-making.

Topic 5: Looping and Increment/Decrement

Objective: Create loops and use increment/decrement instructions.
Key Concepts:

- **Instructions**:
  - `INR R`: Increment register by 1 (e.g., `INR A`).
  - `DCR R`: Decrement register by 1 (e.g., `DCR C`).
- **Looping**: Use `JNZ` with counter register (e.g., C).
- **Program Flow**: Combine loops with arithmetic/data transfer.

Example Program:

```
MVI C, 05H   ; Set loop counter to 5
```

MVI A, 00H ; Initialize A to 00H

LOOP: INR A ; Increment A

DCR C ; Decrement C

JNZ LOOP ; Jump if C not zero

STA 2000H ; Store A at 2000H

HLT ; Stop execution

Explanation: Increments A five times (00H to 05H) using C as counter, stores at 2000H.

Exercises:

1. Increment B 10 times using a loop, store at 3000H.
2. Decrement A from 08H three times, store result.
3. Increment memory at 2000H five times using a loop.

Learning Outcome: Create iterative loops for data manipulation.

8086 Microprocessor Topics

Topic 6: Introduction to 8086 and Data Transfer Commands

Objective: Understand 8086 architecture and data transfer instructions.
Key Concepts:

- **Registers**: 16-bit (AX, BX, CX, DX); 8-bit halves (AH, AL, etc.).
- **Segment Registers**: CS, DS, SS, ES; Instruction Pointer (IP).
- **Instructions**:
  - `MOV dest, src`: Copy data (e.g., `MOV AX, 1234H`).
  - `LEA dest, src`: Load effective address.

Example Program:

MOV AX, 1234H ; Load AX with 1234H

MOV BX, AX ; Copy AX to BX

MOV CL, AH ; Copy high byte of AX (12H) to CL

Explanation: Loads 1234H into AX, copies to BX, and high byte (12H) to CL.

Simulator (emu8086):

- Emulate code, step through, observe AX=1234H, BX=1234H, CL=12H.

Exercises:

1. Load 55H into AL, AAH into AH.
2. Load FFFFH into DX, copy to AX.
3. Run in emu8086, verify register changes.

Learning Outcome: Move data between 8086 registers.

Topic 7: Memory Operations and Storage

Objective: Interact with 8086 memory using load/store instructions.
Key Concepts:

- **Memory Addressing**: Direct (e.g., `[1000H]`), indirect (e.g., `[BX]`).
- **Instructions**:
  - `MOV reg, [addr]`: Load from memory.
  - `MOV [addr], reg`: Store to memory.

Example Program:

```
MOV AX, 0005H   ; Load AX with 5

MOV [1000H], AX ; Store AX at 1000H

MOV BX, [1000H] ; Load from 1000H to BX
```

Explanation: Stores 5 at 1000H, loads it into BX.

Simulator (emu8086):

- Emulate, check memory at 1000H (05 00), verify BX=0005H.

Exercises:

1. Store CX at 1500H.
2. Load byte from [2000H] to AL, [2001H] to AH.

3. Set FFH at [1234H], load into DL.

Learning Outcome: Read/write to 8086 memory.

Topic 8: Arithmetic Operations

Objective: Perform arithmetic with flags.
Key Concepts:

- **Instructions**:
  - `ADD dest, src`: Add to destination.
  - `SUB dest, src`: Subtract from destination.
  - `INC dest`, `DEC dest`: Increment/decrement by 1.
- **Flags**: Zero (ZF), Carry (CF), Sign (SF).

Example Program:

```
MOV AL, 10H ; Load AL with 16

MOV BL, 08H ; Load BL with 8

SUB AL, BL  ; AL = 10H - 08H = 08H

INC AL      ; AL = 09H
```

Explanation: Subtracts 8 from 16, increments to 9.

Simulator (emu8086):

- Emulate, check AL=08H after SUB, AL=09H after INC, observe flags (ZF=0).

Exercises:

1. Add 1234H (AX) and 5678H (BX), check CF.
2. Load 01H into CL, subtract 01H, check ZF.
3. Load AX from [1000H], increment, store back.

Learning Outcome: Perform arithmetic and monitor flags.

Topic 9: Comparison and Conditional Jumps

Objective: Compare values and control flow.
Key Concepts:

- **Comparison**: `CMP dest, src` (sets flags like SUB).
- **Jumps**:
  - `JZ label`: Jump if equal (ZF=1).
  - `JNZ label`: Jump if not equal.
  - `JA label`: Jump if above (unsigned).
  - `JG label`: Jump if greater (signed).
- **Labels**: Define jump targets.

Example Program:

MOV AL, 05H ; Load AL with 05H

MOV BL, 05H ; Load BL with 05H

CMP AL, BL ; Compare

JZ EQUAL ; Jump if equal

MOV CX, 0FFFFH ; Skipped if equal

EQUAL: MOV CX, 0001H ; CX = 1 if equal

Explanation: Compares equal values, jumps to set CX=1.

Simulator (emu8086):

- Emulate, check ZF=1 after CMP, IP jumps to EQUAL.

Exercises:

1. Compare 10H and 20H, jump to label if not equal (JNZ).
2. Use JA to check if AX > BX.
3. Compare register with memory, jump if equal.

Learning Outcome: Implement decision-making with comparisons.

Topic 10: Looping

Objective: Create loops for repetitive tasks.
Key Concepts:

- **Counter**: CX register for loops.
- **Instruction**: `LOOP label` (decrements CX, jumps if CX≠0).

Example Program:

MOV CX, 5 ; Set counter to 5

MOV AX, 0 ; Initialize AX

AGAIN: ADD AX, 2 ; Add 2 to AX

LOOP AGAIN ; Decrement CX, jump if not zero

Explanation: Adds 2 to AX five times, AX=0AH.

Simulator (emu8086):

- Emulate, watch AX increase, CX decrease, loop stops when CX=0.

Exercises:

1. Loop 10 times, store BX counter at [3000H].
2. Subtract 5 from AX three times, starting AX=100.
3. Create nested loop with CX (outer) and DX (inner, manual decrement).

Learning Outcome: Create counter-controlled loops.

Topic 11: Logical and Bitwise Operations

Objective: Perform bitwise logical operations.
Key Concepts:

- **Instructions**:
  - `AND dest, src`: Clear bits.
  - `OR dest, src`: Set bits.
  - `XOR dest, src`: Toggle bits.
  - `NOT dest`: Flip all bits.
- **Bit Masking**: Isolate, set, or toggle specific bits.

Example Program:

MOV AL, 11010101b ; Load AL with binary

```
AND AL, 00001111b ; Clear upper 4 bits

MOV BL, 00110011b ; Load BL

OR BL, 11110000b  ; Set upper 4 bits
```

Explanation: AL becomes 00000101b, BL becomes 11110011b.

Simulator (emu8086):

- Emulate, view AL/BL in binary, check bit changes.

Exercises:

1. Check if bit 2 of DL is 1 (AND with 00000100b, check ZF).
2. Toggle bit 7 of AH using XOR.
3. Flip all bits in BX using NOT.

Learning Outcome: Manipulate data at the bit level.

Arduino Topics

Topic 12: Introduction to Arduino and Digital I/O

Objective: Learn Arduino basics and digital I/O.
Key Concepts:

- **Arduino Uno**: ATmega328P, digital pins, power.
- **Sketch**: `setup()`, `loop()`.
- **Functions**: `pinMode()`, `digitalWrite()`, `digitalRead()`.
- **Circuit**: LED (220Ω resistor), pushbutton.

Example Program:

```
void setup() {

  pinMode(13, OUTPUT); // Built-in LED

  pinMode(8, OUTPUT);  // External LED

}

void loop() {
```

```
  digitalWrite(13, HIGH); digitalWrite(8, HIGH);
  delay(1000);
  digitalWrite(13, LOW); digitalWrite(8, LOW);
  delay(1000);
}
```

Explanation: Blinks LEDs on pins 13 and 8.

Exercises:

1. Blink LEDs at 0.3-second intervals.
2. Use pushbutton on pin 2 to control LED on pin 8.
3. Blink built-in LED twice, then external LED once.

Learning Outcome: Set up Arduino and control digital I/O.

Topic 13: Analog Input and PWM

Objective: Read analog inputs, use PWM for outputs.
Key Concepts:

- **Analog Input**: A0–A5, `analogRead()` (0–1023).
- **PWM**: Pins ~3, ~5, ~6, ~9, ~10, ~11; `analogWrite()` (0–255).
- **Circuit**: Potentiometer, LED (220Ω).

Example Program:

```
int ledPin = 9;
int potPin = A0;
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  int potValue = analogRead(potPin);
```

```
  int brightness = potValue / 4;
  analogWrite(ledPin, brightness);
}
```

Explanation: Potentiometer controls LED brightness.

Exercises:

1. Control two LEDs (pins 9, 10) with one potentiometer.
2. Use photoresistor on A1 for LED brightness.
3. Invert brightness (high pot value = low brightness).

Learning Outcome: Read analog inputs, control PWM outputs.

Topic 14: Serial Communication

Objective: Use serial for debugging/interaction.
Key Concepts:

- **Functions**: `Serial.begin(9600)`, `Serial.print()`, `Serial.read()`.
- **Serial Monitor**: View output, send commands.
- **Circuit**: Reuse potentiometer.

Example Program:

```
int potPin = A0;
void setup() {
  Serial.begin(9600);
}
void loop() {
  int potValue = analogRead(potPin);
  Serial.print("Pot Value: ");
  Serial.println(potValue);
  delay(500);
```

}

Explanation: Prints potentiometer values to Serial Monitor.

Exercises:

1. Print "HIGH" if pot value > 512, else "LOW".
2. Control LED on pin 7 with '1' (on) or '0' (off) via Serial Monitor.
3. Log potentiometer (A0) and photoresistor (A1) values.

Learning Outcome: Use serial communication effectively.

Topic 15: Interrupts and Timers

Objective: Use interrupts and timers for responsiveness.
Key Concepts:

- **Interrupts**: Pins 2, 3; `attachInterrupt()`, modes (LOW, CHANGE, RISING, FALLING).
- **Timers**: `millis()` for non-blocking timing.
- **Circuit**: Pushbutton, LED.

Example Program:

```
volatile int state = LOW;

void setup() {

  pinMode(13, OUTPUT);

  pinMode(2, INPUT_PULLUP);

  attachInterrupt(digitalPinToInterrupt(2), toggle, FALLING);

}

void loop() {

  digitalWrite(13, state);

}

void toggle() {
```

```
  state = !state;
}
```

Explanation: Button on pin 2 toggles LED on pin 13.

Exercises:

1. Count button presses, print to Serial Monitor.
2. Blink LED every 2 seconds using `millis()`.
3. Turn off LED after three button presses.

Learning Outcome: Implement interrupts and timers.

Topic 16: Sensors and Libraries

Objective: Interface with sensors using libraries.
Key Concepts:

- **Libraries**: Install `DHT`, `NewPing`.
- **Sensors**: DHT11 (temperature/humidity), HC-SR04 (distance).
- **Circuit**: DHT11 with 10kΩ pull-up.

Example Program:

```
#include <DHT.h>

#define DHTPIN 7

#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  dht.begin();
}

void loop() {
  float temp = dht.readTemperature();
```

```
  float hum = dht.readHumidity();
  if (!isnan(temp) && !isnan(hum)) {
    Serial.print("Temp: "); Serial.print(temp);
    Serial.print("C, Humidity: "); Serial.println(hum);
  }
  delay(2000);
}
```

Explanation: Reads and displays DHT11 data.

Exercises:

1. Turn on LED if temperature > 25°C.
2. Measure distance with HC-SR04 using `NewPing`.
3. Log DHT11 and HC-SR04 data.

Learning Outcome: Use libraries for sensor interfacing.

Topic 17: Motors and Actuators

Objective: Control motors for physical outputs.
Key Concepts:

- **Actuators**: Servo motors, PWM control.
- **Library**: `Servo.h` for angle control.
- **Circuit**: Servo on pin 9.

Example Program:

```
#include <Servo.h>
Servo myServo;
void setup() {
  myServo.attach(9);
}
```

```
void loop() {
  myServo.write(0);   delay(1000);
  myServo.write(90);  delay(1000);
  myServo.write(180); delay(1000);
}
```

Explanation: Rotates servo between 0°, 90°, 180°.

Exercises:

1. Control servo angle with potentiometer.
2. Move servo to 90° on button press.
3. (Advanced) Control DC motor speed with PWM.

Learning Outcome: Control actuators for physical tasks.

Topic 18: Final Project and Integration

Objective: Build a project combining multiple concepts.
Key Concepts:

- **Project Design**: Integrate sensors, actuators, logic.
- **Debugging**: Use Serial Monitor.
- **Example Project**: Light/temperature controller (DHT11, photoresistor, servo, LED).

Example Program:

```
#include <DHT.h>
#define DHTPIN 7
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
Servo myServo;
void setup() {
  Serial.begin(9600);
```

```
  dht.begin();
  myServo.attach(9);
  pinMode(13, OUTPUT);
}
void loop() {
  float temp = dht.readTemperature();
  int light = analogRead(A0);
  int angle = map(light, 0, 1023, 0, 180);
  myServo.write(angle);
  digitalWrite(13, temp > 25 ? HIGH : LOW);
  Serial.print("Temp: "); Serial.print(temp);
  Serial.print("C, Light: "); Serial.println(light);
  delay(2000);
}
```

Explanation: Servo adjusts by light level; LED on if temp > 25°C.

Exercises:

1. Control servo with HC-SR04 distance.
2. Log temperature and button presses, activate servo if conditions met.
3. Document project (diagram, code, explanation).

Learning Outcome: Build integrated Arduino projects.

Notes

- **Simulators**: Use GNUSim8085 for 8085, emu8086 for 8086, Tinkercad/Arduino IDE for Arduino.
- **Safety**: Use 220Ω resistors with LEDs; check servo power ratings.
- **Resources**: Arduino.cc, emulator manuals, library docs.

- **Troubleshooting**: Verify pin connections, check Serial Monitor for Arduino, inspect flags/registers in simulators.