Embedded Circuit Simulator

OOPL MINI-PROJECT

Agraj Dubey
Department of Electronics Engineering
K. J. Somaiya School of Engineering
Roll No. : 16014023008
agraj.dubey@somaiya.edu

Aaditya Chavan
Department of Electronics Engineering
K. J. Somaiya School of Engineering
Roll No. : 16014023001
aaditya.chavan@somaiya.edu

***Abstract*—This project presents an embedded circuit simulator designed to emulate the fundamental functionalities of the 8051 micro-controller along with basic peripherals, including resistors, LEDs, NPN and PNP transistors, and a seven-segment display. The simulator was developed in Java, featuring a graphical user interface (GUI) for easy interaction, leveraging Java Swing and AWT. Core components of the simulator include a CPU model that supports essential 8051 operations, a memory module to manage ROM and RAM storage, and an instruction set implementation that interprets and executes opcodes according to the 8051 instruction set architecture (ISA). Users can initialize peripherals, monitor pin states, and execute 8051 instructions, allowing for a realistic embedded system simulation. The simulator aims to aid students and engineers in learning and testing embedded system concepts by providing an accessible platform for 8051 microcontroller experimentation.**

Key words: 8051 microcontroller, Embedded circuits, java swing , java awt

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

The Embedded Circuit Simulator is an educational Java-based tool specifically designed to simulate basic embedded circuits with a focus on the 8051 microcontroller, widely used in embedded systems education and development. The tool allows users to simulate the execution of 8051 assembly language instructions and observe the real-time impact of these instructions on virtual electronic components and registers. This simulator is ideal for beginners and enthusiasts looking to understand how 8051 instructions operate in a simulated hardware environment.

Project Objectives

The main objective of the Embedded Circuit Simulator is to provide an accessible, interactive way for users to gain practical experience with 8051 microcontroller programming and embedded circuit operation without needing physical hardware. By visualizing the effects of each instruction, users can better understand the fundamental concepts of micro-controller programming and circuit functionality, making the simulator an effective educational tool for learning the basics of embedded systems.

II. System Overview

1) **8051 Microcontroller Simulation**
The simulator includes a virtual 8051 microcontroller with support for a variety of assembly instructions that control the simulated components and registers. The tool focuses on core instructions commonly used in embedded applications, specifically instructions affecting the **Accumulator (ACC)**, **Registers** , and **flags**.
2) **Supported Instructions**
The simulator supports an essential set of instructions that allow users to perform arithmetic, logic, and data movement operations. These are categorized as follows:
   - **ACC Operations**:
     - CPL A: Complements the accumulator.
     - CLR A: Clears the accumulator.
     - DEC A: Decrements the accumulator.
     - MOV A, immediate: Loads an immediate value into the accumulator.
     - MOV A, Rn: Moves data from a register to the accumulator.
     - ADD A, immediate: Adds an immediate value to the accumulator.
     - ADD A, Rn: Adds the contents of a register to the accumulator.
     - SUB A, immediate: Subtracts an immediate value from the accumulator.
     - SUB A, Rn: Subtracts the contents of a register from the accumulator.
   - **Logic Instructions**:
     - ANL A, immediate: Performs bitwise AND between the accumulator and an immediate value.
     - ANL A, Rn: Performs bitwise AND between the accumulator and a register.
     - ORL A, immediate: Performs bitwise OR between the accumulator and an immediate value.
     - ORL A, Rn: Performs bitwise OR between the accumulator and a register.
     - XRL A, immediate: Performs bitwise XOR between the accumulator and an immediate value.
     - XRL A, Rn: Performs bitwise XOR between the accumulator and a register.

- **Register Operations**:
  - DEC Rn: Decrements a specified register.
  - MOV Rn, A: Moves the content of the accumulator to a specified register.
  - MOV Rn, immediate: Loads an immediate value into a specified register.

These instructions provide a well-rounded foundation for understanding data manipulation, register handling, and logical operations within the 8051 architectures

1) **Peripheral Components Simulation**
   The simulator includes various peripheral components that interact with the microcontroller, simulating the behaviour of components often found in embedded circuits. Users can experiment with:
   - **LEDs**: These respond to output values from the microcontroller, turning on or off based on the logic states driven by the instructions.
   - **Seven-Segment Displays**: The simulator provides control over seven-segment displays, allowing users to visualize numeric and hexadecimal outputs generated by the 8051.
   - **Resistors**: Virtual resistors are included for controlling current and voltage in the simulation, offering users a basic understanding of resistance in circuits.
   - **Transistors (NPN and PNP types)**: Simulated transistors allow users to learn about switching and amplification behaviour in circuits, a core concept in many embedded applications.
2) **Real-Time Visualization and Feedback**
   As users execute each assembly instruction, they can immediately observe changes to component states and register values. This real-time feedback allows users to experiment with different instruction sequences and understand their impact on the system, helping bridge the gap between theoretical knowledge and practical application.
3) **User-Friendly Interface**
   The simulator's interface is designed to be intuitive, making it accessible for beginners while still offering sufficient depth for more advanced users. The user can select instructions, provide values for immediate operations, and control component states from the interface. By displaying the microcontroller's internal registers, flags, and connected components, the interface provides comprehensive visibility into the operation of the 8051 and the simulated circuit.

III. Design and Implementation

The simulator is implemented in Java, utilizing object-oriented programming principles to model the 8051 microcontroller and the various components. Each component, such as LEDs, transistors, and registers, is represented by a class, encapsulating its behaviour and allowing easy extension and modification. The instruction set is implemented through a combination of methods that directly manipulate component states and register values, mimicking the operations of a real microcontroller.

The simulation of peripheral components is integrated within the tool's architecture, enabling direct interaction with the 8051's registers and instruction handling. The simulator's real-time feedback mechanism leverages Java's graphical libraries to visually update component states, allowing users to see the effects of their code in a near-instantaneous manner.

The project is namely divided into 5 files, cpu.java, instructionSet.java, MemorySet.java, Simulator.java and peripheral.java. Each file handles a different functionality of the 8051 MCU and are meant to emulate the different physical areas of the silicone die used in real 8051s .

A. cpu.java

This Java class simulates the core elements and operations of a CPU within an 8051 microcontroller simulation environment. The primary responsibilities of this class include defining and managing CPU registers, status flags, and basic operations like fetch, decode, and execute cycles. Here's a summary of its main components:

1) Key Components::

1) **CPU Registers and Flags**:
   - Core registers: Program Counter (`pc`), Data Pointer (`dptr`), Accumulator (`acc`), Register B (`b`), and Program Status Word (`psw`).
   - Flags within `psw`: These represent specific CPU states and include `carryFlag`, `auxiliaryCarryFlag`, `overflowFlag`, and `parityFlag`. The flags `regset0` and `regset1` determine register bank selection, while `fo` is a custom-defined flag.
2) **Memory and Instruction Set Interaction**:
   - The `CPU` class interacts with `Memory` and `InstructionSet` objects to fetch and execute instructions.
   - `fetch()`: Retrieves the next instruction (opcode) from memory and increments the program counter.
   - `execute()`: Passes the fetched opcode to the `InstructionSet` class for execution.
   - `cycle()`: Combines fetching and executing in one step, simulating a single CPU cycle.
3) **CPU Control Methods**:
   - `reset()`: Resets all CPU registers, flags, and relevant settings to their initial states.
   - Getter and Setter methods: Manage and access various registers and flags, supporting encapsulation and flexibility in the simulation.

This class is part of a larger simulation framework, using `InstructionSet` for instruction handling and `Memory` for data storage. The `Pins` class handles I/O and peripheral interactions. The `CPU` class centralizes all operations, enabling simulated CPU behavior for educational and testing purposes.

B. InstructionSet.java

This simulates the instruction set of the 8051.

Class and Objects:

InstructionSet: This class represents the instruction set of the processor. It takes three objects in its constructor:

- `CPU`: This object holds information about the processor's state, such as registers and flags.
- `Memory`: This object represents the memory of the processor where data and instructions are stored.
- `Pins`: This object might handle input/output operations with external devices.

Method:

executeInstruction(int opcode): This is the core method of the class. It takes an opcode (operation code) as input, which identifies a specific instruction. The method then uses a switch statement to execute the corresponding instruction based on the opcode. **Instructions:**

The switch statement handles various instructions, including:

- **MOV (Move):** These instructions move data between registers and memory locations. For example, `MOV A, #data` moves an immediate value to the accumulator register (A).
- **ADD:** These instructions add two values and update the accumulator and flags based on the result (carry, overflow, etc.).
- **SUB (Subtract):** Similar to ADD, these instructions subtract and update flags.
- **DEC (Decrement):** Decrements the value in a register or the accumulator.
- **CLR (Clear):** Clears the accumulator to zero.
- **CPL (Complement):** Performs a bitwise NOT operation on the accumulator.
- **SETB (Set Bit):** Sets a specific bit in memory, a register, or a flag.
- **Other:** There might be additional instructions not shown in the provided code snippet.

Helper Functions:

The class also includes several helper functions used by the instructions:

- **calculateParity(int value):** Calculates the parity flag based on the number of set bits in the value (even or odd).
- **calculateCarry(int operand1, int operand2, int result):** Calculates the carry flag for addition based on the operands and the result.
- **Other helper functions:** Similar functions exist for calculating auxiliary carry, overflow for addition/subtraction, carry/auxiliary carry/overflow for subtraction, setting a bit in a value, setting a bit in memory, setting a bit in a register, and setting a flag.

Overall, this code snippet demonstrates how a Java class can simulate the basic instruction set of a simple processor. It includes instruction handling, flag manipulation, and helper functions for various operations.

C. Memory.java

This defines the memory of the 8051.

Memory Types:

- **RAM (Random Access Memory):** Stores data that can be read from and written to during program execution. Defaults to 128 bytes.
- **ROM (Read-Only Memory):** Stores program instructions that can only be read. Defaults to 4096 bytes.

Constructors:

The class offers three constructors providing flexibility in memory size configuration:

- One sets user-defined ROM size while keeping RAM size at 128 bytes.
- Another sets user-defined RAM size while keeping ROM size at 4096 bytes.
- The last allows customization of both RAM and ROM size.

Memory Access Methods:

- **readByte(int address):** Reads a byte from the ROM at the specified address. Throws an exception for invalid addresses.
- **writeByte(int address, byte value):** Writes a byte to the ROM at the specified address. Throws an exception for invalid addresses.
- **readDataByte(int address):** Reads a byte from the RAM at the specified address. Throws an exception for invalid addresses.
- **writeDataByte(int address, byte value):** Writes a byte to the RAM at the specified address. Throws an exception for invalid addresses.

Display Methods:

- **ramDisplay(int address):** Prints the value of a specific byte in RAM along with its address. Throws an exception for invalid addresses.
- **romDisplay(int address):** Prints the value of a specific byte in ROM along with its address. Throws an exception for invalid addresses.

Masking with 0xFF:

Notice the `& 0xFF` operation used in read methods. This ensures the returned value is always an unsigned integer between 0 and 255, even if the underlying byte storage uses signed integers.

Overall, this class provides a basic memory model for a simulated computer system in Java, allowing programs to read from and write to both program instructions (ROM) and data (RAM).

D. peripheral.java

This code defines various classes for simulating electronic components and their interaction.

Interfaces and Classes:

- **Peripheral:** This interface defines a contract for components that can connect to the system using pins.

- **Pin:** This class represents a single pin with a name and state (HIGH or LOW). It provides methods to set/get state and retrieve the pin name.
- **Pins:** This class holds predefined pin objects for various ports (P0, P1, etc.) and special pins (RST, XTAL, etc.).
- **VoltageChecker:** This class implements the Peripheral interface and allows checking the voltage state of a connected pin.
- **LED:** This class implements Peripheral and models an LED. It requires two pins for the anode and cathode and offers methods to connect the pins and check if the LED is on or off. Optionally, color can be specified during creation.
- **seven_seg_display:** This class implements Peripheral and simulates a seven-segment display. It requires eight pins for connection and provides methods to connect, activate the display, and retrieve the current digit representation on the display segments.
- **Resistor:** This class implements Peripheral and models a resistor. It allows connecting to two pins and offers methods to check voltage drop across the resistor and access/set its resistance value.
- **npn, pnp:** These classes implement Peripheral and model NPN and PNP transistors, respectively. They require three pins for connection (base, collector, emitter) and offer a method to check if current flows through the transistor based on the voltage states of the pins.
- **Ground:** This class is a subclass of Pin specifically for representing the ground pin. It's always set to LOW voltage and cannot be set to HIGH.

Overall Functionality:

These components allow you to build a simulated circuit by connecting them using the `connect` method of each peripheral. You can then query their state or behavior using provided methods like `checkVoltage`, `checkVoltageDrop`, `checkState`, or `getDisplayOutput`. This enables basic simulation of digital circuits and their interactions.

E. simulator.java

This is the actual simulator for an 8051 microcontroller. Here's a breakdown of the functionality:

Components:

- **Memory:** Stores program instructions (ROM) and data (RAM).
- **CPU:** Executes instructions fetched from memory.
- **Pins:** Provides interface between the microcontroller and external components.
- **InstructionSet:** Defines how the CPU interprets each instruction.
- **LED:** Light-emitting diode (controlled by the program).
- **Resistor:** Electronic component used to limit current (simulated).
- **npn, pnp:** Transistor types (simulated).
- **seven_seg_display:** Seven-segment display (controlled by the program).

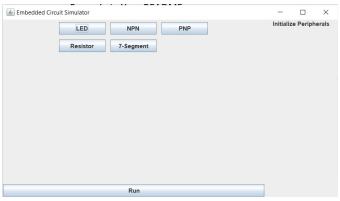

Fig. 1. Main Window

Functionality:

1) **Initialization:**
   - Creates a graphical user interface (GUI) with buttons to add components like LEDs, resistors, transistors, and seven-segment displays.
   - Each button opens a dialog where users can specify pin connections and other parameters for the chosen component.
   - The program keeps track of all added components in separate lists.
2) **Loading Program:**
   - User can define a text file containing opcodes (machine code instructions) for the 8051.
   - The program reads the file and loads the opcodes into the memory (ROM section).
3) **Simulation:**
   - Clicking the "Run" button starts the simulation.
   - The program fetches instructions from memory one by one.
   - Based on the instruction, the CPU performs various operations like setting pins, manipulating data, and jumping to different parts of the program.
   - The simulation loop continues until a specific "halt" instruction is encountered.
4) **Displaying Results:** After simulation, a new window displays various details like:
   - CPU register values (accumulator, program counter, etc.)
   - Flag values (carry, overflow, etc.) indicating the state of the CPU
   - Status of connected components (LED on/off, voltage drop across resistor, etc.) based on the program's execution

Additional Features:

- The memory panel allows users to inspect individual RAM and ROM locations after simulation.
- A "Reset" button allows users to restart the simulation with a clean state (clearing all components).

F. Final Execution

Here we have attached some screenshots from the program and the code running in the back end:

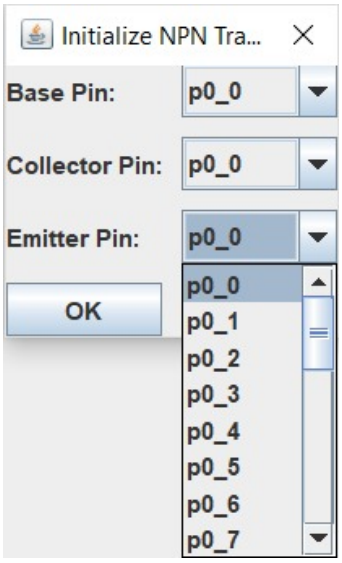

Fig. 2. Pin Initialization Window

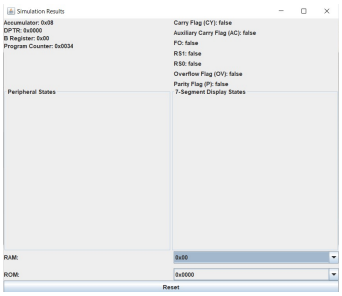

Fig. 3. Simulation Window

IV. Test Cases

A. Test Case 1

Initialization of an LED, 2 NPN transistors, 1 PNP transistor, 7 segment display. The program running in the instructions.txt file moves some data to the accumulator and from the accumulator to registers R0, R1. R2. It also sets pins of port 1 and port 2 high. NPN1 was initialized ti be ON, NPN2 was init to be OFF, the LED was initialized to be ON and set to red and the 7 segment display was set to display number 7.

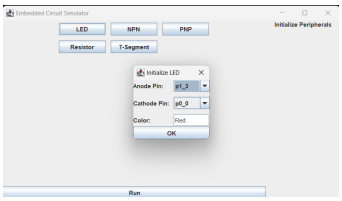

Fig. 4. Initializing Peripherals

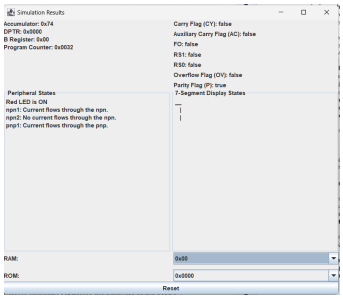

Fig. 5. Simulation result window

B. Test Case 2

In this test case we enter a wrong opcode inside the instructions.txt program. We also do not initialize any of the peripherals.

Fig. 6. Wrong opcode Error Message

V. Conclusion

The developed 8051 microcontroller emulator in Java provides a robust and flexible platform for understanding, learning, and experimenting with 8051 assembly language programming. It accurately emulates the core functionalities of the 8051, including its registers, memory, and instruction set. The provided peripheral classes further extend the emulator's capabilities, allowing for the simulation of real-world scenarios and the development of complex embedded systems.

Key features and benefits of this emulator include:

- **Accurate 8051 Emulation:** The emulator faithfully replicates the behavior of the 8051's hardware components, enabling users to execute 8051 assembly code and observe its effects on the system state.
- **Modular Design:** The modular design with well-defined classes and interfaces promotes code reusability, maintainability, and extensibility.
- **Peripheral Simulation:** The inclusion of peripheral classes like LEDs, seven-segment displays, resistors, and transistors allows for the simulation of various hardware configurations and interactions.
- **User-Friendly Interface:** A user-friendly interface (not explicitly shown in the provided code, but could be a GUI or command-line interface) can be implemented to provide a seamless experience for users to input assembly code, step through execution, and visualize the system's state.
- **Debugging Capabilities:** Implementing debugging features like breakpoints, single-stepping, and register/memory inspection can significantly aid in understanding and troubleshooting 8051 programs.

VI. Future Enhancements

While the current emulator provides a solid foundation, several areas can be explored for further enhancement:

- **Enhanced Peripheral Models:** Expanding the library of peripheral models to include more complex components like timers, serial communication modules, and analog-to-digital converters would increase the emulator's versatility.
- **Real-Time Simulation:** Implementing real-time simulation capabilities, where the emulator executes instructions at a specific clock speed, would allow for more accurate modeling of timing-critical applications.
- **Hardware Integration:** Integrating the emulator with real hardware devices through interfaces like USB or serial communication would enable users to test their code on physical systems.
- **Optimized Performance:** Optimizing the emulator's performance, particularly for large programs and complex simulations, can be achieved through techniques like instruction caching, parallel execution, and efficient data structures.
- **User-Friendly Interface:** Developing a user-friendly graphical user interface (GUI) would significantly improve the user experience. The GUI could provide features like code editing, assembly, debugging, and visualization of the system's state.
- **Integration with Development Tools:** Integrating the emulator with popular development tools like IDEs and debuggers would streamline the development process and provide a seamless experience for users.
- **Support for Different 8051 Variants:** Extending the emulator to support different 8051 variants with varying memory sizes and peripheral configurations would increase its applicability.

By addressing these areas, the 8051 emulator can become a powerful tool for education, research, and embedded systems development, empowering users to explore the intricacies of 8051 programming and design efficient and reliable embedded systems.

References

[1] Mazidi, M. A., & Mazidi, J. P. (2011). The 8051 Microcontroller and Embedded Systems: Using Assembly and C. Pearson Education India.

[2] Balagurusamy, E. (2021). Programming with Java: A Complete Reference (8th ed.). McGraw-Hill Education.

[3] https://www.iitk.ac.in/esc101/05Aug/tutorial/uiswing/index.html

[4] https://cse.iitkgp.ac.in/ dsamanta/java/index.htm