OBJECTIVE:

1. Technical Exploration: To the use of assembly language for programming an 8051 microcontroller-based auto-billing shopping cart system. This includes:

- Gaining a deeper understanding of the 8051-microcontroller architecture and its interaction with assembly language instructions.
- Developing proficiency in assembly language programming for future microcontroller applications.

2. Functional Development: To design and implement a functional auto-billing shopping cart system utilizing the chosen programming language (assembly language). This system should:

- Employ RFID tags and readers to identify items placed within the cart.
- Process product identifications using the 8051 microcontroller.
- Access a pre-loaded product database containing item names and prices.
- Dynamically calculate the total bill amount as items are added or removed.
- Display the real-time bill amount on a user interface (LCD screen) for customer convenience.

By achieving these objectives, the project aims to demonstrate the feasibility and benefits of assembly language programming for microcontroller-based systems while offering a practical solution for improved customer experience in retail environments.

INTRODUCTION:

This project delves into the development of an innovative auto-billing shopping cart system utilizing an 8051-based microcontroller unit. While the concept of auto-billing shopping carts is not entirely new, with various designs readily available online, the approach taken here differs significantly. Existing projects often rely on embedded C language, necessitating compilation before the code can interact with the microcontroller. This project, however, takes a more fundamental route by employing assembly language. The decision to utilize assembly language stems from a desire to explore the potential benefits it offers. Assembly language provides a level of granularity and direct control over the microcontroller's hardware that is often obscured by the higher-level abstractions present in C. This can potentially lead to a more efficient and streamlined codebase. While the resulting assembly code might be noticeably larger compared to its C counterpart, the anticipated payoff lies in achieving faster execution speeds due to the direct interaction with the underlying hardware.

The primary motivation for undertaking this project is not solely driven by the potential performance benefits of assembly language. It serves a more fundamental purpose – to assess and refine our own capabilities. By setting ambitious goals and targets, we aim to evaluate our ability to navigate the challenges associated with this project within the constraints of time and resources. Pushing ourselves to master this intricate programming language will not only enhance our understanding of microcontroller architecture but also equip us with a valuable skillset.

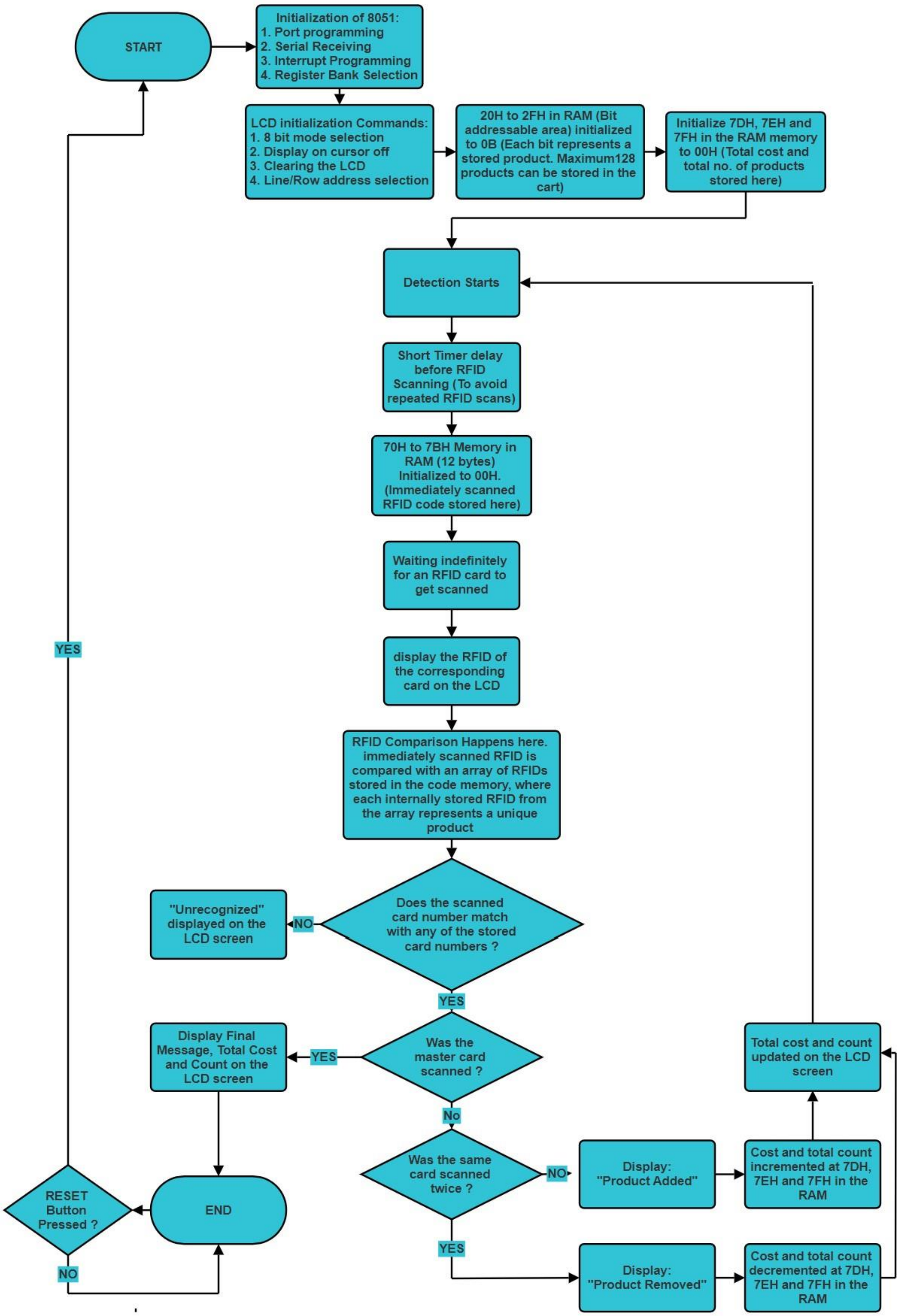
Developing an auto-billing shopping cart system offers a practical application of this technical exploration. This system envisions a scenario where shoppers can scan or identify items placed within the cart using RFID tags and readers. The microcontroller unit, programmed in assembly language, will be responsible for processing these product identifications. By referencing a pre-loaded product database containing item names and corresponding prices, the system will dynamically calculate the total bill amount as items are added or removed. This information can then be displayed on a user interface, such as an LCD screen, for real-time bill tracking.

The successful implementation of this project will not only demonstrate the feasibility of using assembly language for microcontroller-based systems but also provide a tangible benefit to the shopping experience. The auto-billing feature can significantly reduce checkout times, leading to improved customer satisfaction and operational efficiency within retail environments.

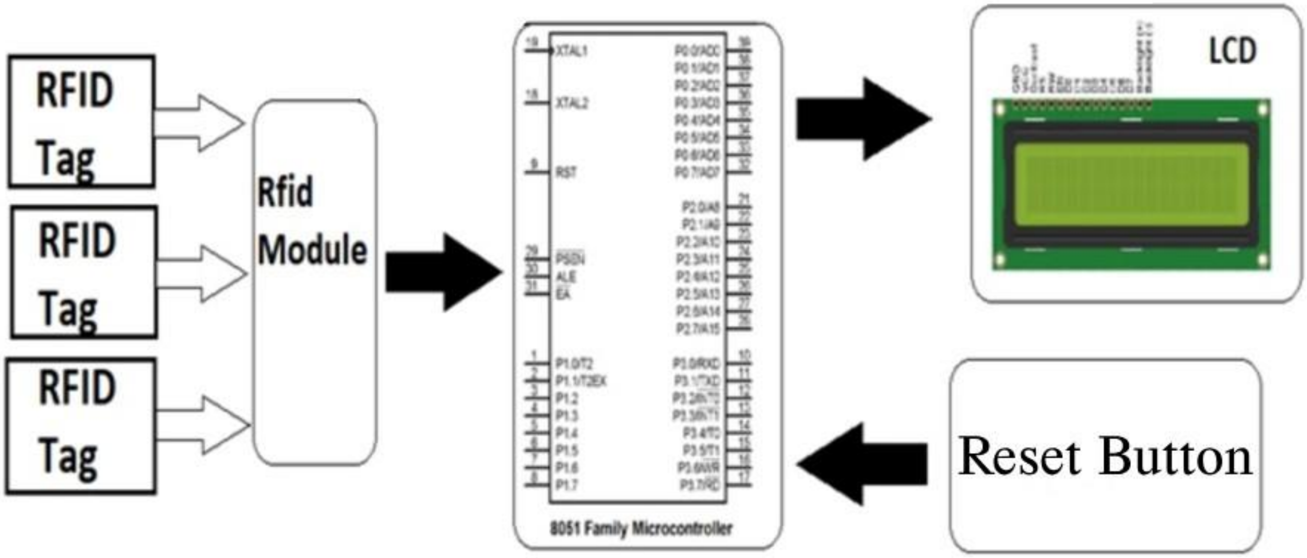
Furthermore, the knowledge gained throughout this project can be leveraged in future endeavours involving 8051 microcontrollers. By delving deeper into assembly language programming and its interaction with the underlying hardware, we can unlock new possibilities when designing and implementing microcontroller-based solutions for various applications.

To summarise, this project offers a unique opportunity to explore the potential benefits of assembly language programming for microcontroller-based systems. The primary goal is not merely to develop a faster auto-billing system, but rather to embark on a journey of self-discovery and skill development. Through successful execution, this project holds the potential to revolutionize the shopping experience while simultaneously enriching our understanding of microcontroller architecture and programming.

BLOCK DIAGRAM:



Hardware Block Diagram



COMPONENTS/ SOFTWARE REQUIRED:

1. Hardware Components Required:

Components:	Cost:
Microcontroller development board	₹849
USB Type A 2.0 to USB Type B 2.0 cable	₹199
9V transistor-radio batteries	₹200
9V snap connector with DC jack	₹49
EM18 125KHz RFID Reader Module	₹470
125KHz RFID Cards	₹199
16x4 LCD display	₹*350*
Jumpers MtM, MtF and FtF	₹179
Male Berg Strip	₹49
Total:	**₹2,544**

2. Software Components Required:

- Keil uVision: Software for assembly code development
- Proteus 8 Professional: Simulation software
- MCU 8051 IDE: backup software to check for errors in assembly code if Keil crashes
- Progisp: Software to flash binary code on to the microcontroller
- Drivers to allow communication between Development board and Computer through USB

PROJECT DESCRIPTION:

I. THE HARDWARE DESCRIPTION:

i. The AT89S52 Microcontroller Architecture

The Intel 8051 microcontroller (MCU) is an industry-standard 8-bit microcontroller core that serves as the foundation for a wide range of microcontrollers from various manufacturers. This project utilizes the Atmel AT89S52, a low-power, high-performance CMOS 8-bit microcontroller that is fully compatible with the 8051instruction set and pinout. The AT89S52 offers several advantages, including:

- **Industry Standard Instruction Set:** Compatibility with the established 8051 instruction set facilitates code reuse across diverse applications.
- **Low Unit Cost:** The affordability of the AT89S52 makes it an attractive choice for cost-sensitive projects.
- **DIP Package Availability:** The availability of these microcontrollers in DIP (dual in-line package) format simplifies integration into development boards.
- **In-System Programmable Flash Memory:** The 8KB of ISP flash memory allows for code updates without requiring removal of the microcontroller from the system.

Core Architecture:

While our project leverages the capabilities of the AT89S52, the core functionality relies on the traditional Intel 8051 architecture. This architecture offers the following key features:

- **8-Bit Processing:** The core processing unit revolves around an 8-bit arithmetic logic unit (ALU) and accumulator, making it particularly adept at handling 8-bit data manipulation.
- **Register Set:** The architecture provides a set of 8-bit registers, including one special 16-bit register accessible through dedicated move instructions. Additionally, four banks of eight registers each (memory-mapped) offer expanded data storage capabilities.
- **Memory Addressing:** The 8-bit data bus and two 16-bit address buses enable access to up to 64 KB of program memory (PMEM) and external RAM (XRAM) in a Harvard architecture configuration. While the traditional 8051 architecture does not possess on-chip ROM, the AT89S52 incorporates 8KB of ISP flash memory.
- **Program Counter and Data Pointer:** A dedicated program counter tracks the execution flow within program memory, while a data pointer facilitates memory manipulation tasks.
- **Instruction Set:** The 8051 instruction set encompasses operations for arithmetic, logic, data transfer, branching, and subroutine management, along with dedicated instructions for multiplication, division, and comparison.

- **Interrupt Handling:** The architecture supports fast interrupt handling with optional register bank switching, enabling the microcontroller to respond promptly to external events while preserving program execution context.
- **Input/Output (I/O):** Four bi-directional 8-bit I/O ports offer flexible control and data exchange with external peripherals.
- **Serial Communication:** A built-in UART (universal asynchronous receiver/transmitter) facilitates serial communication capabilities.
- **Timers/Counters:** Two on-chip 16-bit counter/timers provide timing and counting functionalities within the system.

Project-Specific Considerations:

For the purposes of this project, the code development adhered to the core instruction set and architecture of the traditional 8051 microcontroller. While the AT89S52 offers additional features beyond the baseline 8051 architecture, these functionalities were not utilized within the project scope.

ii. Radio-Frequency Identification (RFID) Technology

Core Functionality:

Radio-frequency identification (RFID) technology utilizes electromagnetic fields for automated identification and tracking of objects equipped with RFID tags. An RFID system comprises three key components:

- **RFID Tags:** Tiny radio transponders that store and transmit identifying data.
- **RFID Reader:** A radio receiver-transmitter unit that interrogates tags and retrieves their stored data.

Tag Types and Operating Principles:

- **Passive Tags:** These tags derive their operating power from the electromagnetic interrogation pulse transmitted by the RFID reader. This limits their effective reading range.
- **Active Tags:** These tags possess an internal battery, enabling them to operate at greater distances from the reader, potentially reaching hundreds of meters.

Applications of RFID Technology:

- **Inventory Management:** Tracking goods throughout the supply chain for efficient management.
- **Asset Tracking:** Monitoring the location and status of valuable assets in real-time.
- **Animal Identification:** Implanting RFID microchips in livestock and pets for positive identification.
- **Retail Checkout:** Expediting checkout processes and deterring theft.
- **Line-of-Sight Independence:** Unlike barcodes, RFID tags do not require a direct line of sight between the tag and the reader, facilitating integration within objects.

EM18 RFID Reader Module:

Our project employs the EM18 RFID reader module, which operates at a frequency of 125 kHz. This module functions by:

- **Tag Interrogation:** When an RFID tag is brought within close proximity, the tag, transmits its unique identification information back to the reader. EM18
- **Data Transmission:** The EM18 receives the tag's identification data and transmits it to the microcontroller unit (MCU) through a serial communication interface (usually RS232) at a baud rate of 9600 bps.

Key Specifications of the EM18 Module:

- Operating Voltage: +4.5V to +5.5V
- Current Consumption: 50mA
- Operating Temperature: 0ºC to +80ºC
- Operating Frequency: 125 KHz
- Communication Baud Rate: 9600 bps
- Reading Distance: Up to 10 cm (depending on tag)
- Integrated Antenna

EM18 Communication and Data Format:

- **Communication Mode Selection:** A dedicated pin (SEL) on the EM18 module allows configuration for either RS232 or Wiegand communication protocols. Our project utilizes the more common RS232 mode (SEL pin HIGH).
- **Data Transmission Format:** Upon successful tag identification, the EM18 transmits the tag's ID information to the MCU as a 12-character ASCII string. The first 10 characters represent the unique tag ID, and the final 2 characters represent the XOR checksum of the preceding 10 characters for error detection.

Project Integration:

The EM18 module functions as a sensor within the auto-billing shopping cart system. The microcontroller is programmed to receive the transmitted data through the RX pin, process the tag ID information, and perform actions based on the identified product.

iii. THE LCD DISPLAY

Display Functionality:

Liquid crystal displays (LCDs) are a form of flat-panel display technology that utilizes the light-modulating properties of liquid crystals in conjunction with polarizers. Unlike light-emitting displays, LCDs do not generate light themselves. Instead, they rely on a backlight or reflector to illuminate the display and modulate the light's passage through the liquid crystals to form visible images. LCDs can produce images in colour or monochrome.

Project-Specific LCD:

Our project incorporates a JHD 16x4 LCD display featuring a green backlight. This particular model is a 16x4 parallel LCD display, offering a cost-effective solution for integrating a 16-character by 4-line display with high contrast black text into the system.

Key Specifications:

- **Model:** JHD539 Y/YG
- **Display Size:** 16 characters x 4 lines
- **Outline Dimensions:** 87 mm x 60 mm x 14 mm
- **Viewing Area:** 61.8 mm x 25.2 mm
- **Display Controller:** SPLC780D
- **Character Size:** 2.95 mm x 4.75 mm
- **Duty Ratio:** 1/16

Project Integration:

The JHD 16x4 LCD display serves as an output device within the system. The microcontroller can transmit either text information or numerical values (such as sensor readings or program cycle counts) to the display for visualization.

II. THE SOFTWARE DESCRIPTION:

The software component of this auto-billing shopping cart system is designed to interact with the hardware elements and achieve the system's overall functionality. Written in assembly language for the 8051-microcontroller unit (MCU), the software performs the following tasks:

- **Data Acquisition and Storage:** Receives data from the RFID reader, stores the acquired RFID tag data, and performs calculations based on the scanned tag.
- **LCD Control:** Controls the LCD display based on system logic and pre-defined numerical data embedded within the code.

i. Initialization

The initialization process encompasses the following steps:

- **Memory Allocation:** A designated memory space between addresses 00H and 30H remains unallocated to facilitate future implementation of vectored interrupts.
- **Port Configuration:** Ports P0 and P2 are declared as output ports for data lines to the LCD, along with control lines RS, R/Wbar, and E.
- **Serial Communication Setup:** Serial communication is initialized as an 8-bit UART operating at a baud rate of 9600.
- **Interrupt Enablement:** The IE register is programmed to enable serial communication interrupts.
- **LCD Initialization:** The LCD screen is initialized using commands like 38H, 01H, 0EH, and 80H to configure 8-bit mode, clear the screen, display the cursor, and position it on the first line.
- **Memory Initialization:** Initialization of the bit-addressable area, total cost storage area, and product count storage area within RAM occurs here.
- **RFID Storage Initialization:** The area for storing the most recently scanned RFID tag is initialized within a loop, as the system is designed to handle multiple scans.

ii. Receiving Data from the RFID Reader

A vectored serial communication interrupt located at memory address 0023H facilitates receiving data from the RFID reader. This approach was chosen to achieve code modularity and separation from the main program, particularly during the initial development stage when the team was unfamiliar with RFID reader interface specifics. The interrupt allows the microcontroller to execute other program tasks while simultaneously reading data transmitted through the serial receiver port.

iii. Displaying the RFID Tag

Following initialization, the software awaits the complete reception of all 12 digits constituting the RFID tag. Once received, the RFID tag is displayed on the LCD screen using the R1 register as a pointer to the RAM memory location storing the tag data. The software displays the digits of any scanned tag, regardless of whether the tag information matches an entry within the internal database.

iv. Comparing Scanned RFID with Database

The DPTR register plays a crucial role in comparing the 12 digits of the most recently scanned and stored RFID tag against a database of pre-defined RFIDs stored within the code memory. If a match is found, the code proceeds to identify and display the corresponding product on the LCD screen, along with updating the total cost and product count. Scanning the master card signifies the end of the shopping session, disabling further card scans. An unrecognized card triggers the display of an "UNRECOGNIZED" message on the LCD screen.

v. Capacity for 128 Distinct Products

The 8051's RAM offers a 128-bit bit-addressable area. While product information is hardcoded into the software, each product is associated with a specific bit within this addressable range. Initially, the entire 16-byte range is cleared to 00H. When a product is added, the corresponding bit is set using the CPL instruction. The CPL instruction is employed strategically such that scanning the same product's RFID twice removes the product from the cart, as the corresponding bit is cleared back to 00H. This use of CPL extends to cost and count management within the code. When a product's bit is set, the cost and count are incremented; conversely, these values are decremented when the bit is cleared. This functionality is leveraged through the JNB instruction operating on the corresponding product bit within the bit-addressable area.

vi. Various Call Subroutines Used in Software

The software leverages a collection of subroutines to manage various functionalities:

- **LCD Display Subroutines:**
    - CMD(Command): Transmits commands to the LCD display.
    - DAT(Data): Sends data to be displayed on the LCD.
    - DLAY(Delay): Introduces a delay for timing purposes during LCD interactions.
    - DLAYSP(Delay-special): Provides a specialized delay function potentially tailored for specific use cases where other registers contain crucial data.
    - DATASP(Data-special): Transmits data characters to the LCD using DLAYSP.
    - RTLCD(Return to LCD): Prints strings on the LCD display.
- **RFID Comparison Subroutine:**
    - Compare: Compares the most recently scanned RFID tag with the database of pre-defined RFID entries stored within the code memory. This subroutine was implemented to address the need for modular code, particularly when the product list grew larger and replacing repetitive comparison logic became necessary.
- **Product Addition/Removal Subroutines:**
    - Each product is associated with two distinct subroutines:
        - One subroutine handles adding the product to the shopping cart.
        - Another subroutine manages removing the product from the cart.
    - These subroutines encompass further functionalities:
        - Displaying the added or removed product name.
        - Calculating and displaying the updated total cost and product count after scanning the corresponding product's RFID tag.
    - These subroutines serve as designated locations for storing individual product costs, facilitating their addition or subtraction from the total amount payable by the customer.
- **Master Card Subroutine:**

  A dedicated subroutine manages the master card scenario. This subroutine ensures that no further products can be added or removed from the cart by placing the code in an infinite loop of non-operation. While background card scan interrupts might still occur, the display remains unchanged due to the absence of provisions for handling such updates within this subroutine.
- **Unrecognized Card Subroutine:**

  The unrec subroutine is responsible for displaying an "unrecognized card" message on the LCD screen.

- **Cost and Count Calculation/Display Subroutines:**
  - Recdisp (display recognized product for add as well as remove product messages): Handles displaying messages related to adding or removing recognized products.
  - PRODADD: Increments the product count.
  - PRODSUBB: Decrements the product count.
  - no_disp: Displays the product count.
  - COSTADD: Increments the total cost.
  - COSTSUBB: Decrements the total cost.
  - cost_disp: Manages the display of the total cost on the LCD.
  - Two additional subroutines, HTD and Hex2BCD, are employed to convert the total product count and total cost from hexadecimal format to BCD (Binary-Coded Decimal) before displaying them on the LCD screen.

vii. Databases

The code memory incorporates two databases:

- **Product RFID Database:** This database stores the RFID tags of products for comparison purposes against scanned RFID tags.
- **Display String Database:** This database stores strings intended for display on the LCD screen at startup and during product addition/removal operations.

While the standard practice advocates for storing these databases separately from the code memory to maintain a clearer distinction, the current implementation keeps them within the code memory due to potential risks of code mix-up during future product additions. Consequently, these databases reside at the end of the code memory, followed by a small code segment at address 0A00H, which serves as the jump target for the serial communication interrupt.

III. DESCRIPTION OF CONNECTIONS BETWEEN HARDWARE COMPONENTS :

The following outlines the specific connections established between the various hardware components within the auto-billing shopping cart system:

- The LCD's Register Select (RS), Read/Write (R/Wbar), and Enable (E) signals are connected to pins P0.7, P0.6, and P0.5 of the 8051 microcontroller, respectively.
- The LCD's data lines are interfaced with port P2 of the microcontroller, utilizing all eight pins within this port.
- The LCD's power supply connections include Vdd (power supply), Vss (ground), LEDA (anode), and LEDK (cathode), which are tied to the corresponding supply and ground pins on the development board. The Vee pin (contrast control) is not employed in this implementation.
- The EM-18 module's Vcc (power supply) and GND (ground) pins are connected to the designated power supply pins on the development board.
- The EM-18 module's SEL (selection) pin is tied to Vcc, achieved through a direct connection to the relevant supply pin on the development board.
- The RS-232 transmit (Tx) pin of an external device is connected to the P3.0 (Rx or receive) pin of the microcontroller, facilitating serial communication.
- The development board receives its power supply from a DC jack connected to a standard 9V transistor radio battery.
- Code is uploaded onto the microcontroller using a USB cable that connects the development board to a computer.

SIMULATION:

1. Keil Software Configuration and Code Compilation:

- Within the Keil development environment, create a new project and select the appropriate 8051 microcontroller device (e.g., AT89S52).
- Configure the crystal oscillator frequency to 11.0592 MHz within the project settings.
- Ensure the option to create a HEX file is enabled under target options.
- Write and edit the assembly language code for the auto-billing shopping cart system functionality.
- Utilize the Keil software's functionalities to translate, build, and rebuild the code, ultimately saving the compiled code as a HEX file.

2. Proteus Simulation Environment Setup:

- Launch the Proteus software and initiate a new project using the "New Project Wizard."
- Specify the desired location for saving the project's simulation files.
- Select a suitable design template for the schematic diagram and proceed to the next step.
- Opt-out of generating a PCB layout by selecting "Do not create a PCB layout" and proceed.
- Activate the "Create firmware project" option and choose the 8051 family. Select the corresponding microcontroller chip (e.g., AT89S52) and specify "Keil for 8051" as the preferred compiler. Click "Next" and subsequently "Finish" to complete the project setup.

3. Schematic Design and Component Integration:

- Switch to the component mode within Proteus and proceed to add all necessary devices for the schematic. These components typically include a 16x2 LCD display, crystal oscillator, 22uF capacitors, and 10-ohm resistors.
- Navigate to the terminal mode and incorporate the ground and power supply components into the schematic.
- Navigate to the Virtual instruments mode and select the Virtual Terminal component.
- Utilize wires to establish appropriate connections between all components within the schematic.

4. Configuration and Simulation Settings:

- Right-click on the crystal oscillator and microcontroller components to access their properties. Modify the crystal oscillator frequency and microcontroller frequency to 11.0592 MHz within the respective property menus.
- To simulate the RFID reader functionality, access the virtual terminal component and modify its properties. Set the baud rate to 9600, data bits to 8, and stop bit to 1.

5. Code Flashing and Simulation Execution:

- To program the compiled code (HEX file) into the simulated microcontroller, left-click on the microcontroller component. Under the "Program File" section, navigate to the location where the HEX file generated by Keil is stored and select the HEX file under the "Objects" folder.
- Initiate the simulation process within Proteus.
- During simulation, input the RFID tag codes through the virtual terminal screen that appears. Optionally, right-click on the terminal screen and select "echo typed characters" to visually confirm the characters being transmitted to the microcontroller via the serial receiving pin.

By following these outlined procedures, developers can establish a functional simulation environment for testing and refining the auto-billing shopping cart system code written in assembly language.

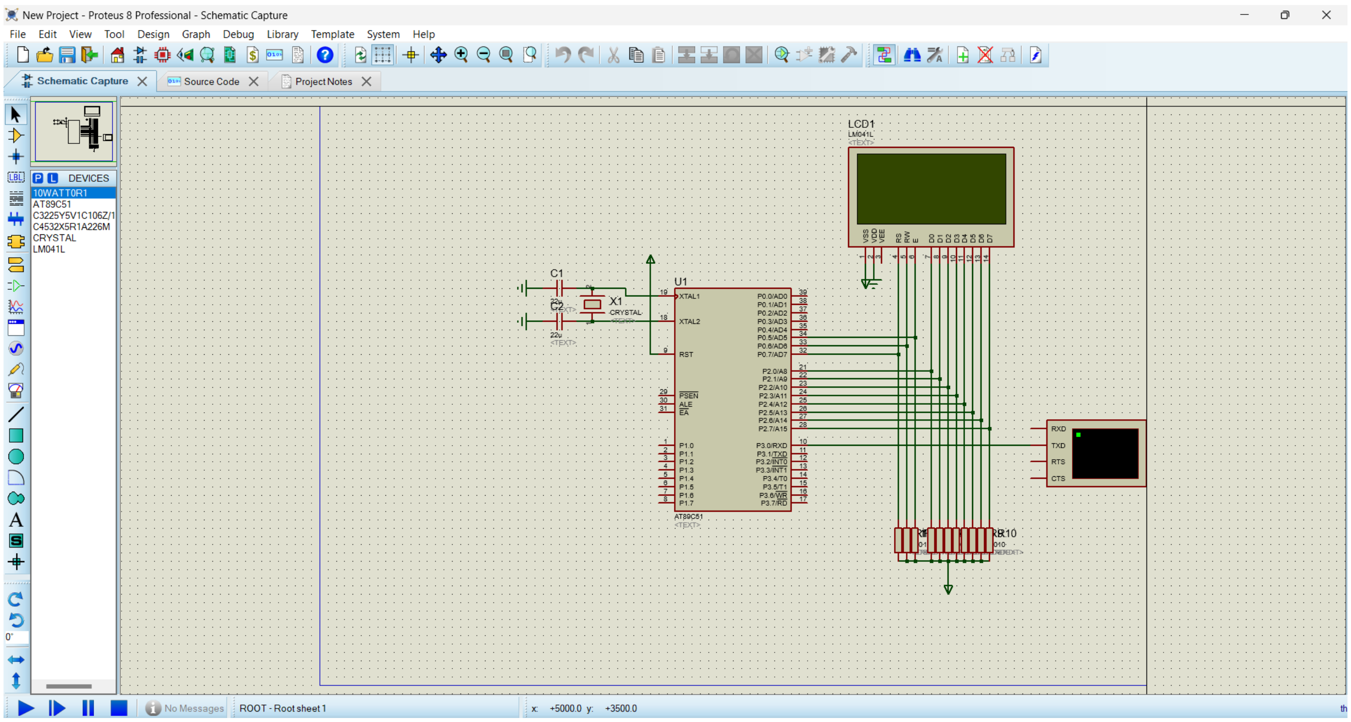


Figure 1: Setting up the Schematic for simulation in Proteus

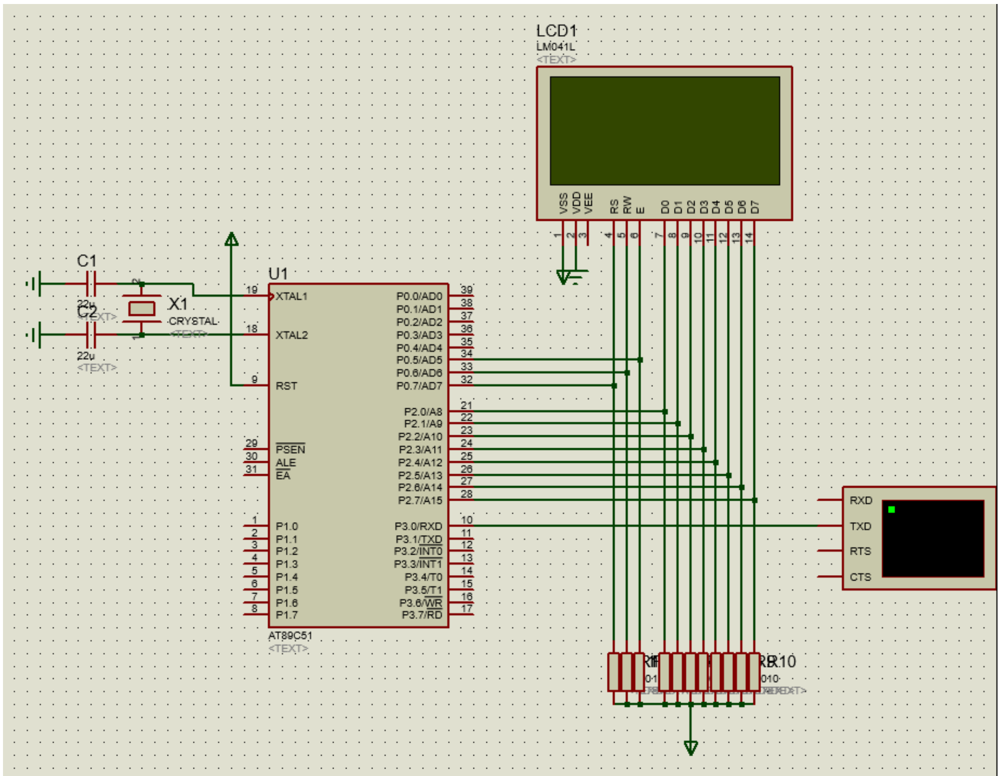


Figure 2: Schematic with all components assembled

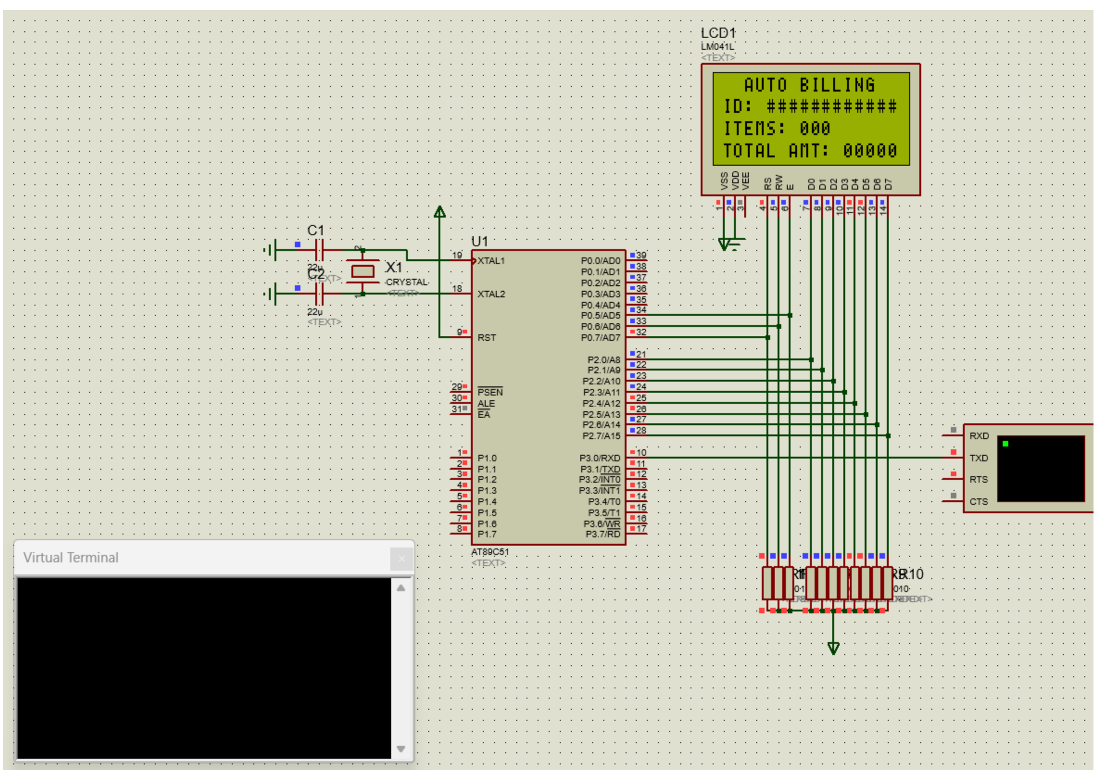


Figure 3: Schematic under a running simulation

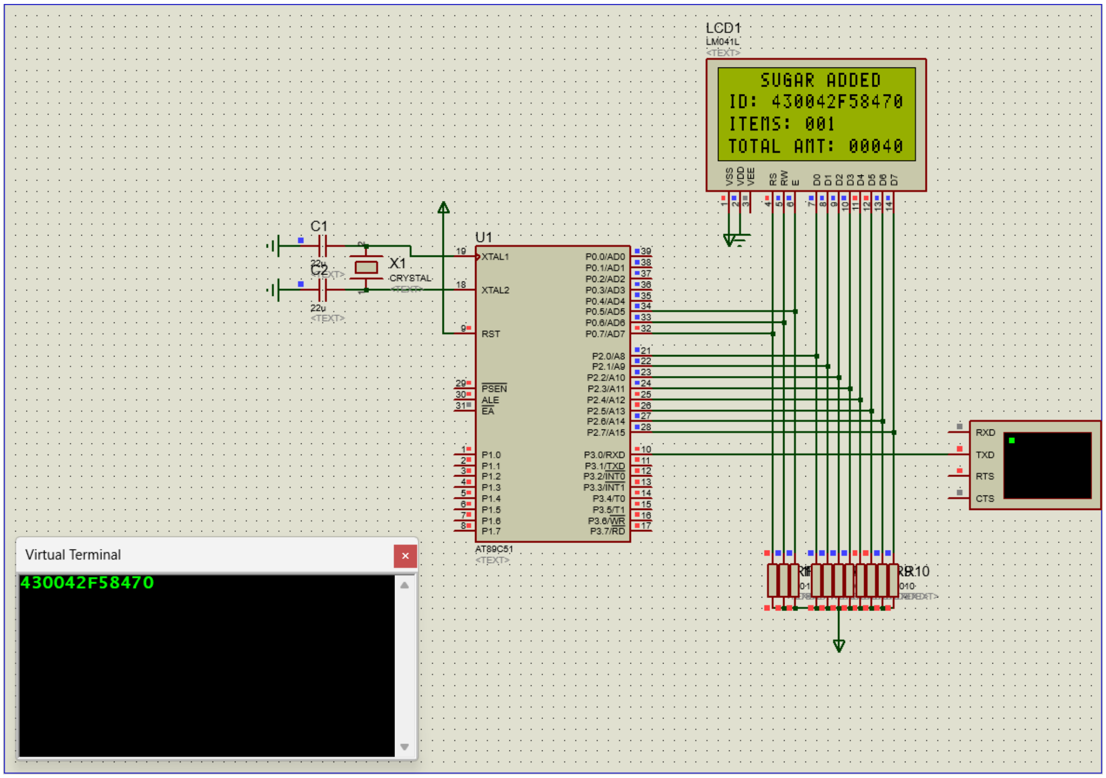


Figure 4: Simulating the scan of a card through a virtual terminal to demonstrate the working of the Auto Billing Cart

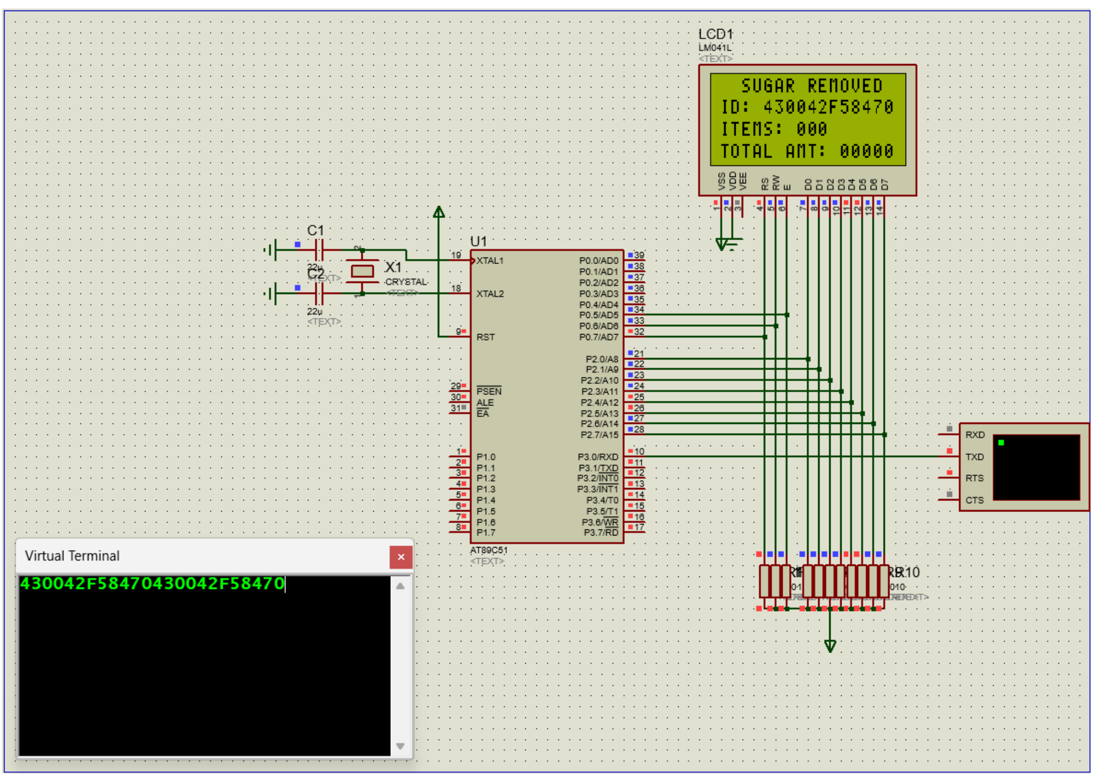


Figure 5: The same card scanned twice will result in the removal of the product from the cart

CONCEPTS LEARNED:

1. Programming the 8051 with the help of a development board:
    - **Driver Installation:** Establish communication between the computer and the 8051 development board by installing the appropriate device drivers. These drivers typically come with the development board manufacturer's software package.
    - Installing appropriate software on the computer to flash the assembly program onto the 8051.

2. Interfacing an RFID reader with an 8051 MCU. The process involves establishing a physical connection between the RFID reader and the 8051 development board according to the specific reader's communication protocol (Here, 8-bit UART)

3. Reading RFID card numbers:
    - Setting a delay to avoid quick repeated scans of the same card.
    - Comparing the RFID card number with a string of stored numbers.

4. Interfacing a 4x16 LCD with the 8051 MCU. The 8051 program will control the LCD display by:
    - **LCD Initialization:** Configure the LCD's control pins and establish communication parameters (e.g., data width, instruction set).
    - **Data and Command Transmission:** Send data and control commands to the LCD to display relevant information, such as the scanned RFID card number or authorization status.

5. To use the bit addressable range of the 8051 as 128 distinct single push ON/OFF switches. To observe the availability of 128 distinct items in the cart

6. To perform 16-bit hexadecimal addition and subtraction on 8051 MCU and to convert 8 bit and 16-bit hexadecimal numbers to BCD digits.

7. Timer control through TMOD register to set up delays when required.

8. Serial communication concepts Used to interface the RFID module with the 8051 MCU.

9. Interrupt Programming through IE register to make the 8051 capable of performing multi-tasking.

10. Use of Subroutines and subroutine calls to automate various repetitive tasks.

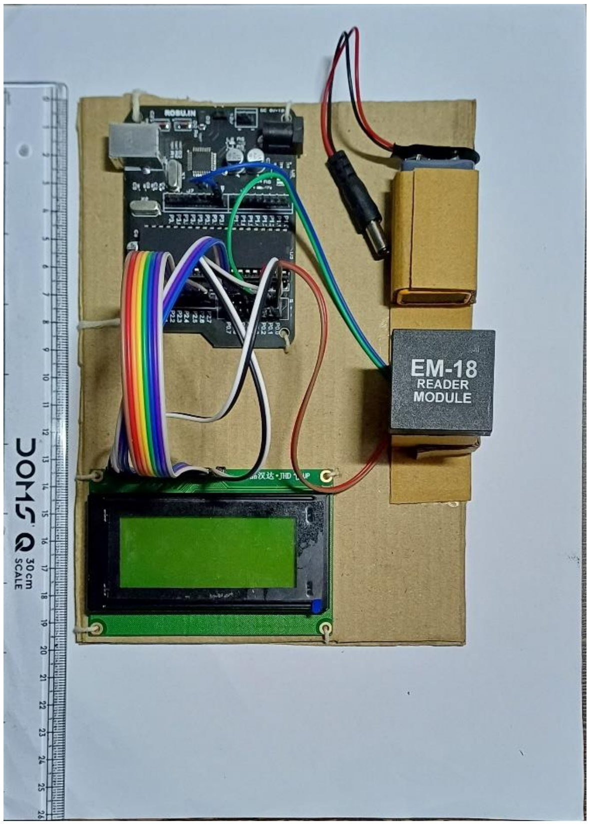
11. Use of softwares like keil and MCU 8051 IDE to program and debug the assembly level code and the use of proteus software to perform simulations and rectify mistakes in code or hardware.

IMPLEMENTATION:

1. YouTube link for your project demo:

https://youtu.be/Ds7VO0wFrH0

2. Images with Captions:



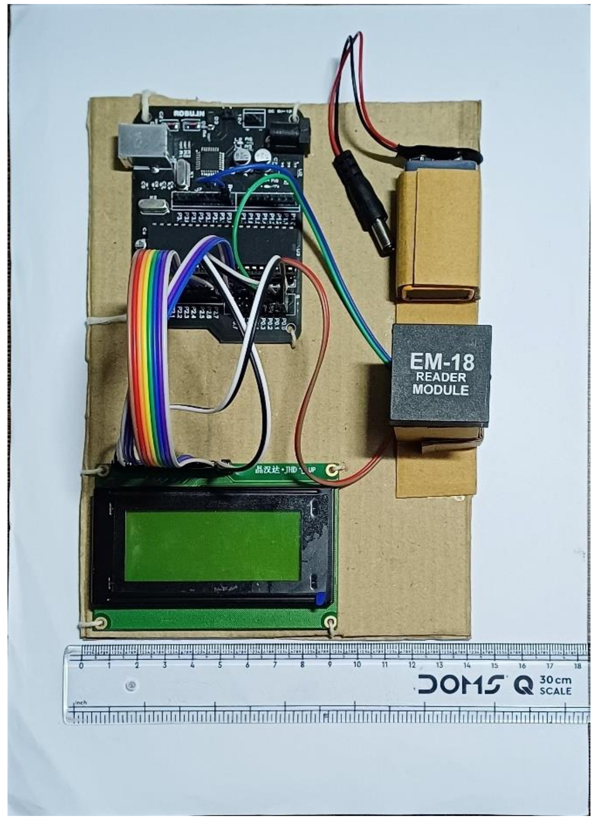


Figure 6 and 7: The Dimensions of the project: 19.8 cm in length and 14 cm in width

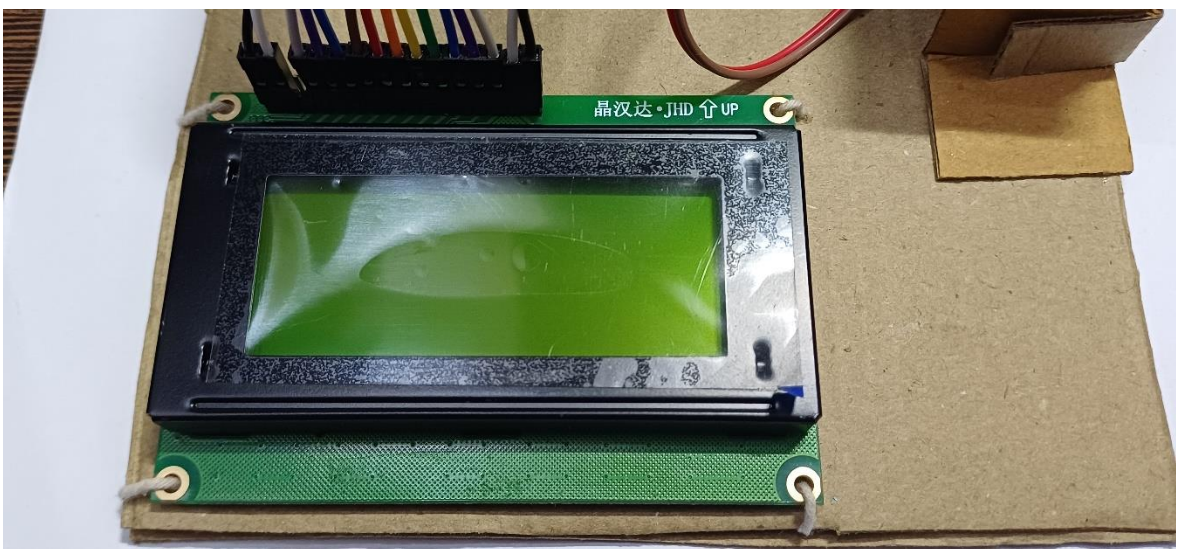


Figure 8: The LCD is tied onto the cardboard using strings and the jumpers connected to it are also connected to the AT89S52

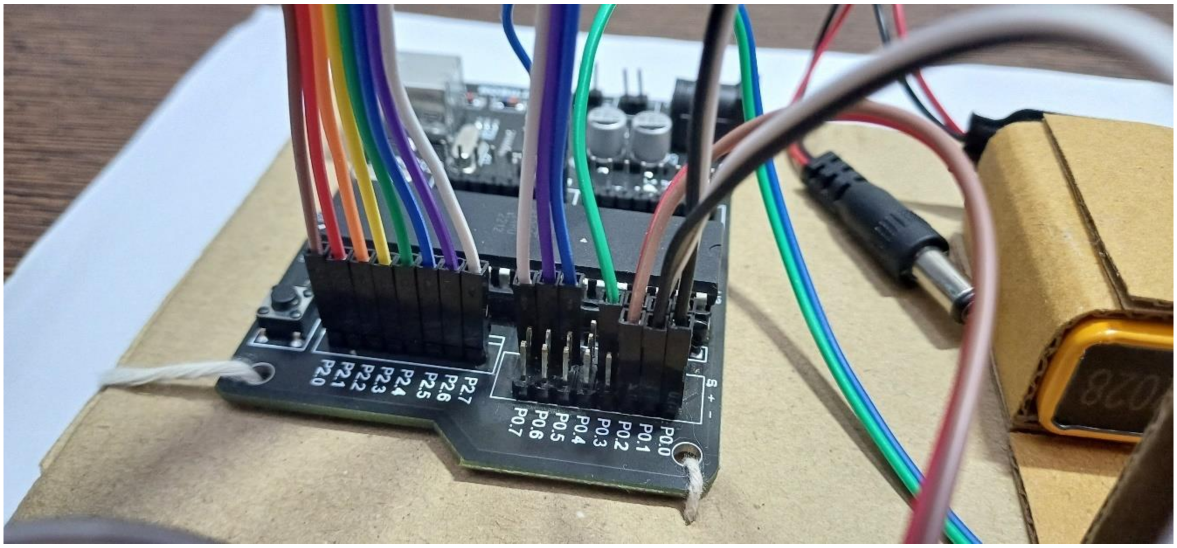


Figure 9: Port 0, Port 2 and Power Supply pins are visible in this image. The Data lines entirely consume P2 of the microcontroller, while P0.7, P0.6 and P0.5 are connected to the RS, R/Wbar and E of the LCD. The power supply is given to the Vss, Vdd, LEDA and LEDK of the LCD as well as to the Vcc and GND of the EM-18 RFID reader module

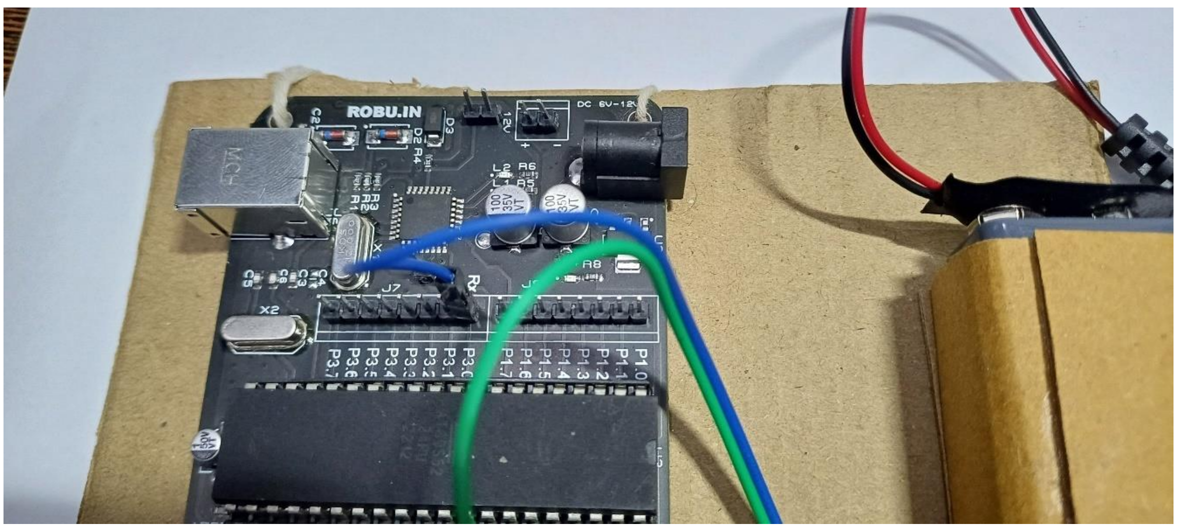


Figure 10: Port 1 and Port 3 of the Microcontroller are visible here. The Rx pin (P3.0) of the Microcontroller is given to the Tx pin of the EM-18 module

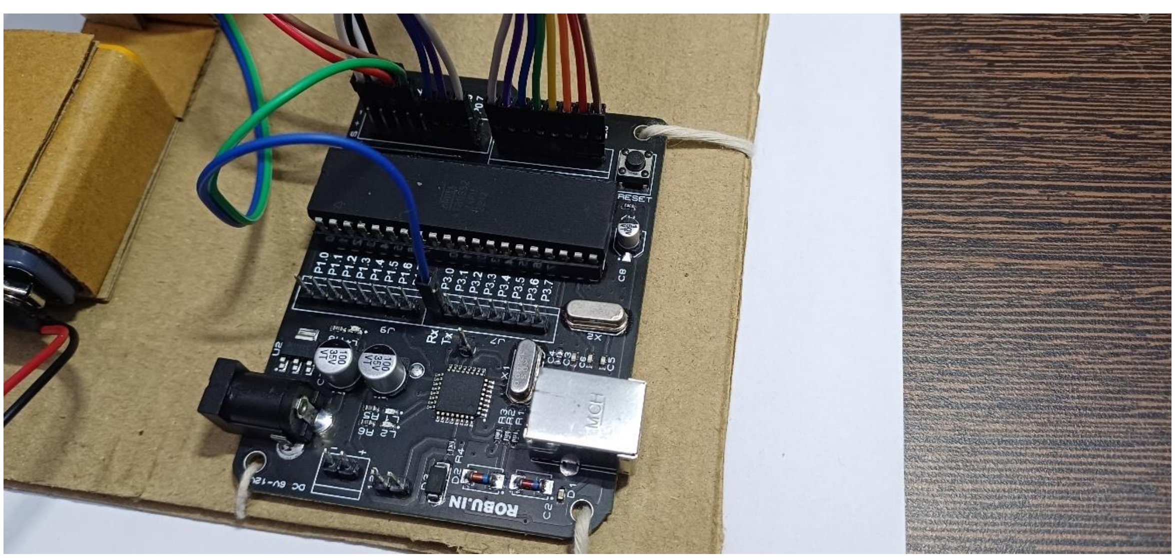


Figure 11: The AT89S52 Microcontroller is the Brain of this project. It is interfaced with all the other devices

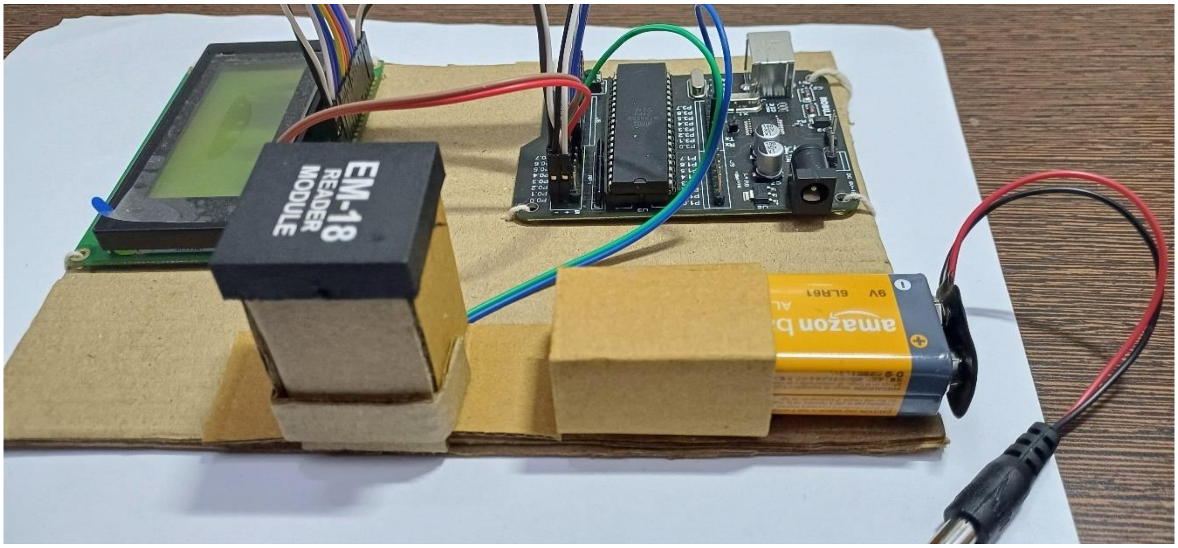


Figure 12: Here, the 9V Transistor Radio Battery is removed from its housing for displaying it. The housing of this component prevents it from moving during the working of the model

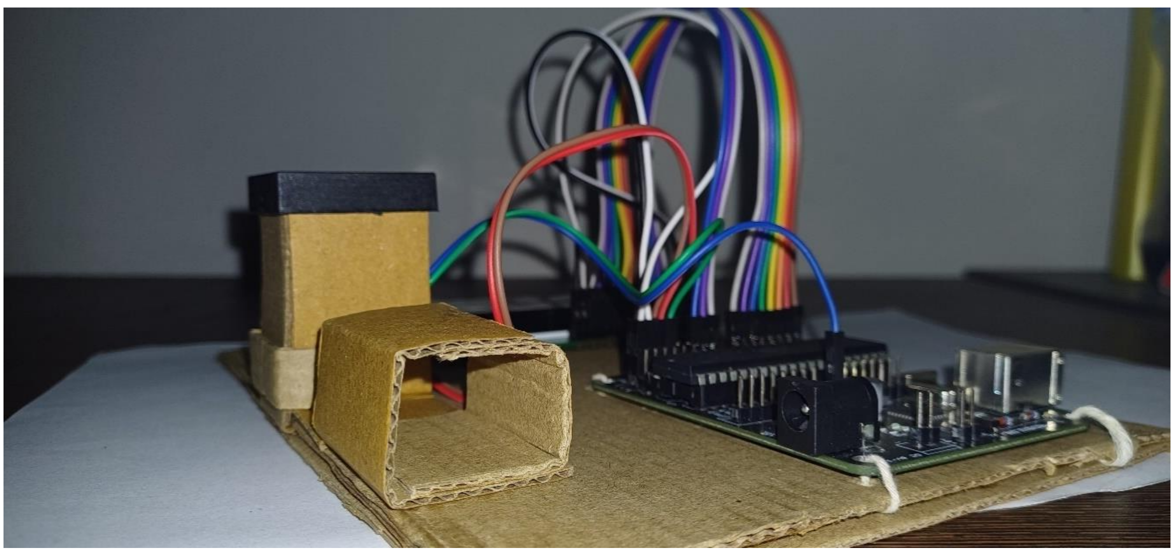


Figure 13: A side view of the battery housing with the battery removed

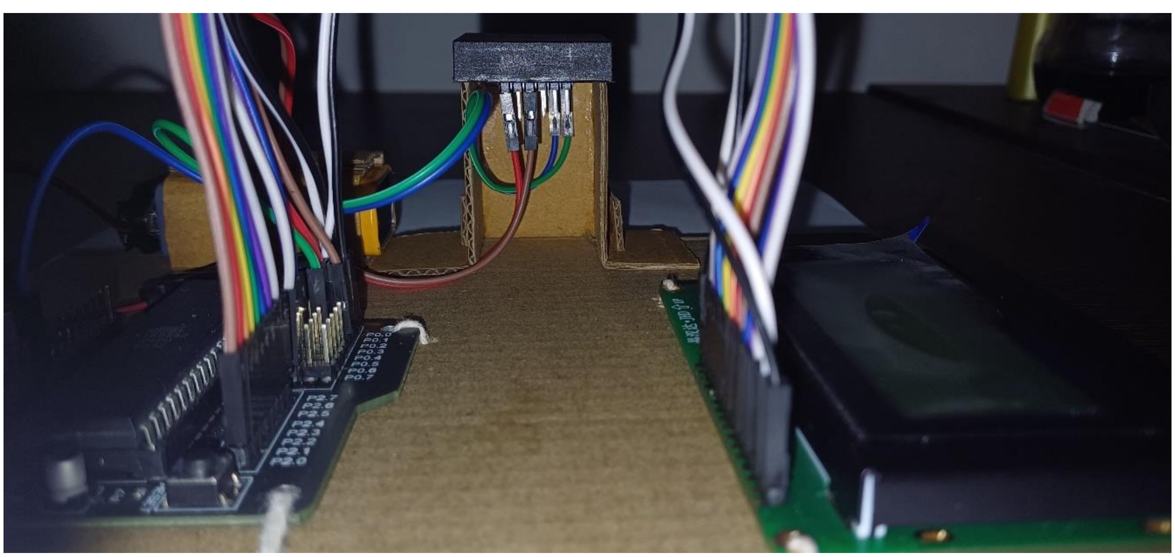


Figure 14: A side view of the structural support provided to the EM-18 module. Providing this support was necessary to ensure the component stays intact in place. 4 Jumper Wires are connected to the EM-18 for making it work

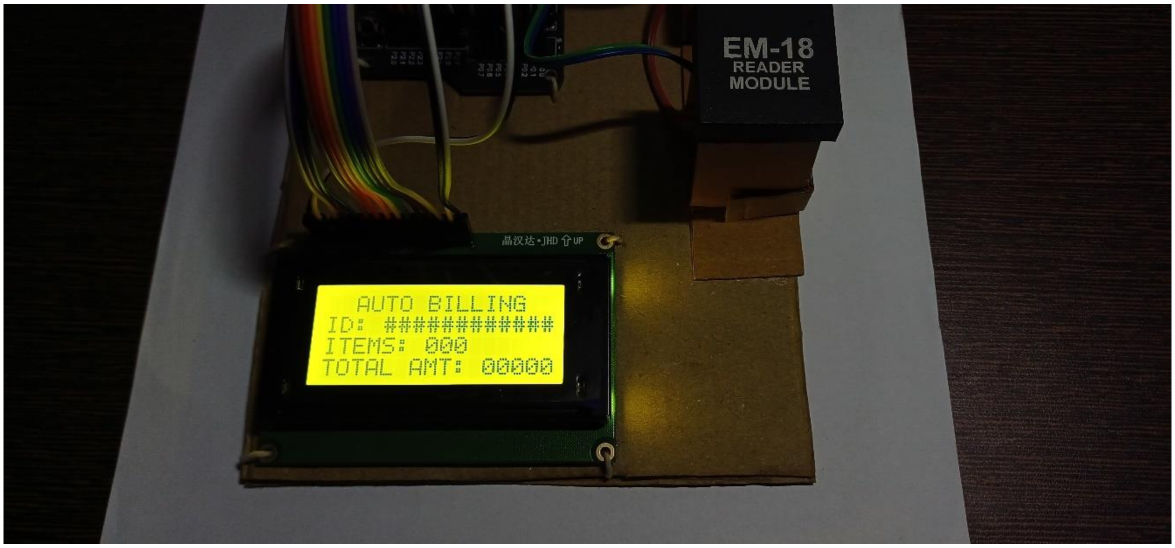


Figure 15: A low light image of the LCD under operation, for better visibility of the text printed on the LCD

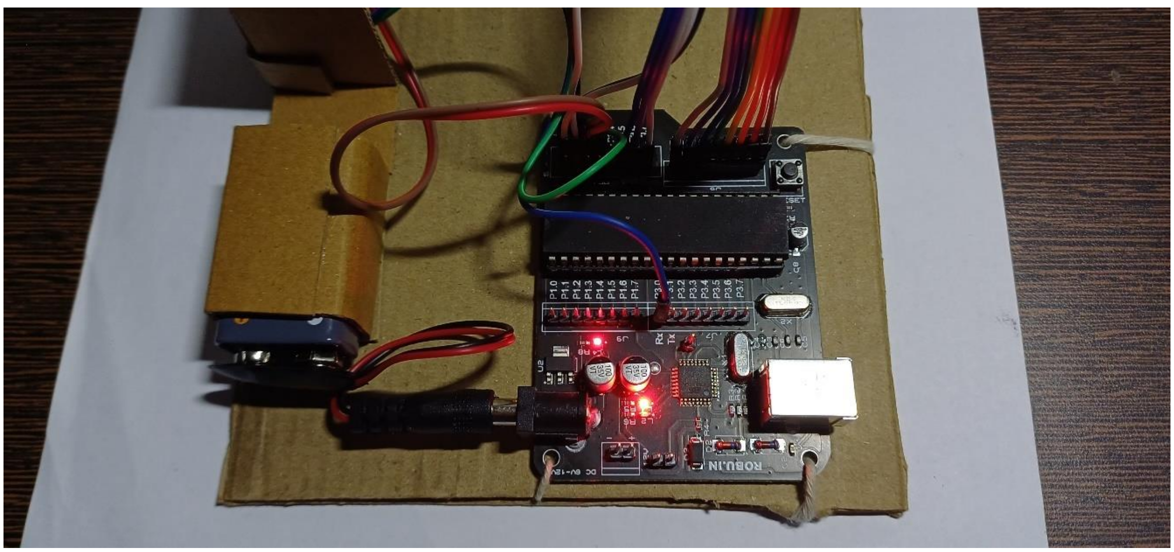


Figure 16: Microcontroller under operation when the DC jack is connected to it for supplying power

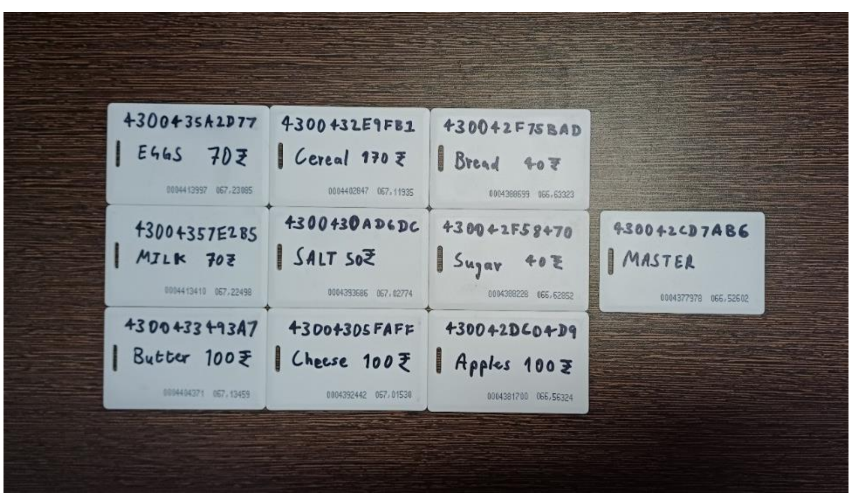


Figure 17: All the RFIDs which are associated with a product as well as the Master card

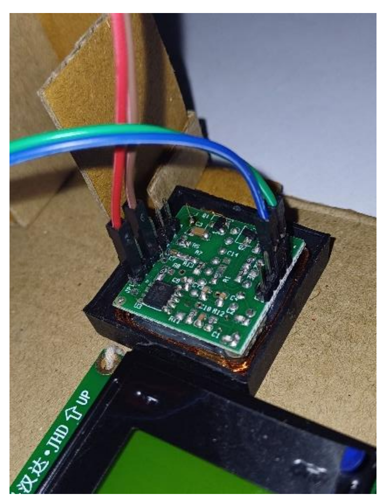


Figure 18: A view of the underside of the EM-18 module. The Red and Brown wires are Vcc and Ground respectively. The green is the selection line and the blue is the Tx

CHALLENGES FACED:

During the development of the auto-billing shopping cart system, our team encountered a number of hurdles that required innovative problem-solving and adaptation. Here, we will discuss the key challenges overcome throughout the project lifecycle.

1. LCD Simulation in Proteus Software:

While simulating the LCD in Proteus software, we discovered the requirement for pull-up resistors to be connected to the data and command/control lines for proper LCD function. Fortunately, this challenge was not encountered practically as the development board already featured internal pull-up resistors.

2. Lack of RFID Support in Proteus:

Proteus software did not offer built-in support for simulating RFID readers and cards. To overcome this limitation, we successfully interfaced a Virtual Terminal to the serial receiving port of the 8051 microcontroller, enabling simulated RFID tag scanning.

3. Understanding ASCII Codes:

The project involved challenges related to a thorough understanding of ASCII codes for various characters. We encountered initial setbacks due to this knowledge gap. However, through dedicated effort and exploration of different methods, we were able to successfully transmit characters ranging from 1 to F through the virtual terminal and display them accurately on the LCD screen. Ultimately, we achieved success in sending hexadecimal characters to the LCD screen via the serial communication port connected to the virtual terminal.

4. Displaying Cost and Product Count on LCD:

Calculating the total cost and total number of products within the code was a straightforward task. However, displaying this information on the LCD screen presented a hurdle. We were able to convert 8-bit hexadecimal values to their equivalent BCD (Binary-Coded Decimal) characters for LCD display. However, replicating this process for 16-bit hexadecimal values proved more challenging due to the 8051 microcontroller's 8-bit data architecture. This obstacle was ultimately overcome through the implementation of well-crafted code segments.

5. Software Misbehaviours:

Due to our initial inexperience with Keil development software and Proteus simulation software, we occasionally encountered software malfunctions. These included frequent crashes within Keil and undesirable outputs from Proteus. While our code might have contributed to some issues initially, running the same code on different software platforms produced successful results. To rectify this situation and eliminate potential software-related interference, we reinstalled both Keil and Proteus. Upgrading the Proteus software version also proved necessary to ensure proper functionality. Our inexperience, however, resulted in wasted time debugging and rewriting code unnecessarily.

6. Memory Limitations and Overflow Issues:

Initially, our product design allowed for a single RFID scan, and products could not be removed from the cart. Through further exploration, we learned to leverage the bit-addressable memory range of the 8051's RAM to implement product removal functionality through a second scan of the same RFID tag. However, this introduced a limitation of a maximum of 128 products within the cart due to memory constraints. Similarly, the maximum cost calculation was limited to 65536 (FFFF in hexadecimal) to avoid overflow errors. We learned this valuable lesson the hard way when progressive addition produced nonsensical results.

7. Development Board Drivers and Code Flashing Challenges:

On the hardware side, we encountered challenges related to installing drivers compatible with the development board. Keil software alone was insufficient for driving the development board, and flashing the code required special software. Through trial and error, and guidance from various sources on the internet we discovered this critical information. While the learning process was valuable, the lack of initial knowledge resulted in wasted time.

8. Interfacing the EM18 RFID Reader Module:

Interfacing the EM18 RFID reader module proved to be a significant challenge. We were unable to locate any source code specifically related to interfacing this component using assembly language programming. While C language source code was readily available, the level of abstraction from the hardware made it difficult to understand the interfacing process. The component's datasheet did mention the communication baud rate of 9600 and its use of an 8-bit UART, but it lacked details on how it transmitted 12-digit hexadecimal codes. We were only able to overcome this challenge through practical experimentation and repeated trial-and-error cycles. The lack of an EM18 reader module within Proteus further complicated the process, as we had to transition directly from the simulation software to real-world hardware interfacing.

9. Faulty RFID Reader Readings:

Despite correctly connecting the EM18 pins as specified in the datasheet, the reader malfunctioned during initial trials. The cause remained unclear, potentially stemming from loose jumper connections or faulty internal components within the device itself. Fortunately, after making minor adjustments, the reader began to function satisfactorily.

10. Spurious LCD Output During Repeated RFID Scans:

During testing, we observed that the LCD screen exhibited unexpected behaviour when the EM18 reader was subjected to rapid consecutive scans of the same RFID card. The screen would become filled with seemingly random characters. The cause of this phenomenon was not immediately apparent. However, we successfully addressed the issue by implementing a timer delay between each RFID card scan. This delay mitigated the spurious output and ensured proper LCD functionality.

11. Challenges in Reconciling Proteus Simulation with Physical Hardware:

Discrepancies were encountered between the physical hardware wiring and the corresponding schematic within the Proteus simulation software. Specifically, upon connecting the hardware components, it became evident that certain connections required twisting the jumper wires. These connections primarily involved the LCD's RS (Register Select), R/Wbar (Read/Write), and E (Enable) lines interfacing with the microcontroller. The Proteus schematic did not explicitly illustrate this requirement. To circumvent potential damage caused by stress and strain on the jumper wires, the assembly code was slightly modified to accommodate the necessary twisted connections. While this alteration represents a minor adjustment relative to the overall hardware-schematic comparison, it proved essential for ensuring the system's proper function and component integrity.

APPLICATIONS:

The approach of this auto-billing shopping cart system, cantered on an 8051-microcontroller programmed in assembly language, extends far beyond the realm of retail. Here's how the core functionalities and hardware setup can be adapted for diverse applications:

1. Streamlined Access Control:

- **Attendance Verification:** Educational institutions and workplaces can leverage the system's RFID tag scanning and verification capabilities for efficient attendance tracking. For example, students or employees simply tapping their ID cards with RFID tags for instant attendance registration. The system displays basic information like name or ID number on the LCD for confirmation.
- **Secure Entry Systems:** The same hardware setup can be adapted for secure entry control in restricted areas or buildings. Authorized personnel with encoded RFID tags can gain access by presenting their cards for verification by the system. The LCD screen can be used to display access granted or denied messages.

2. Inventory Management:

- **Warehouse Tracking:** Warehouses can utilize the system for efficient inventory tracking. Products equipped with RFID tags can be scanned upon arrival or shipment, updating a database within the microcontroller. The LCD display can be used to show real-time stock information for specific items.

3. Asset Tracking:

- **Equipment Management:** Companies with valuable equipment can leverage the system for asset tracking. RFID tags attached to equipment can be scanned periodically, allowing for real-time location tracking and status updates within the microcontroller's memory.

4. Beyond the List:

The potential applications extend even further:

- **Library Book Management:** Libraries can utilize the system for self-service book check-out and return. Borrowers with RFID-enabled library cards can scan books for automated check-out and return processes.

- **Smart Ticketing Systems:** Event organizers can create smart ticketing systems using the core functionalities. Tickets embedded with RFID tags can be scanned for quick and secure entry verification.

These are just a few examples of how the core functionalities of the auto-billing shopping cart system, powered by an 8051-microcontroller programmed in assembly language with RFID tags and an LCD display, can be adapted and repurposed for a wide range of applications. The flexibility of the microcontroller opens doors for innovation in various sectors, potentially streamlining processes, enhancing security, and improving user experiences.

CONCLUSION:

This project has successfully explored the development of a novel auto-billing shopping cart system cantered on an 8051-microcontroller programmed entirely in assembly language. The project delved into the finer control and potential performance benefits that assembly language offers for direct microcontroller interaction. The resulting system, guided by a comprehensive flowchart, demonstrates the feasibility of auto-billing through RFID tag scanning, code verification, and real-time cost updates. The project extends beyond functionality, fostering a deeper understanding of the 8051-microcontroller architecture and assembly language programming – valuable knowledge for future endeavours involving microcontroller-based systems.

The success of this project paves the way for a more efficient and streamlined auto-billing experience in retail environments. The system offers several potential benefits, including:

- **Reduced Checkout Times:** By eliminating the need for manual scanning and price checks, auto-billing can significantly shorten checkout lines, enhancing customer convenience.
- **Improved Accuracy:** The reliance on RFID tags and code verification minimizes the possibility of human error during the billing process.
- **Increased Efficiency:** The system automates a significant portion of the checkout process, freeing up employees for other tasks and potentially reducing operational costs.

This project serves as a stepping stone towards a future where auto-billing shopping carts become commonplace, revolutionizing the retail checkout experience. The exploration of assembly language programming not only offers performance advantages but also empowers a deeper understanding of microcontroller systems, paving the way for further innovation in this domain.

APPENDIX I

COMPLETE PROGRAM WITH COMMENTS

```
ORG 0000H
      SJMP 0030H            ;lEAVING SPACE FOR VECTORED INTERRUPTS
ORG 0030H
      MOV P0, #00H          ;Port 0.7, 0.6, 0.5 sends control signal output to
LCD RS, R/W and E respectively
      MOV P2, #00H          ;Port 2 is Data line output
      MOV TMOD,#21H         ;Timer1=Mode2<For serial comms> and Timer0=Mode1<For
delay>
      MOV TH1,#-3D          ;loads TH1 with 253D(9600 baud)
      MOV SCON,#50H         ;sets serial port to Mode1 <8 bit UART> and receiver
enabled
      MOV IE, #90H          ;Interrupt enabled but only serial communication
interrupt enabled
      SETB TR1              ;Timer set to start serial communication
      CLR PSW.3             ;Register bank 00 is used
      CLR PSW.4
;command for lcd initialization
      MOV A, #38H ; 8 bit mode
      ACALL CMD
      ACALL DLAY
;     MOV A, #0EH ; display on curson on
;     MOV A, #0FH ; Blinking block cursor
      MOV A, #0CH ; display on cursor off
      ACALL CMD
      ACALL DLAY
      MOV A, #01H ; clear LCD
      ACALL CMD
      ACALL DLAY
;Initial commands over, starting to move data to LCD
      MOV A, #080H ;1st line
      ACALL CMD
      ACALL DLAY
      MOV DPTR,#MESS01
      ACALL RTLCD
      MOV A, #0C0H ;2nd line
      ACALL CMD
      ACALL DLAY
      MOV DPTR,#MESS02
      ACALL RTLCD
      MOV A, #090H ;3nd line
      ACALL CMD
      ACALL DLAY
      MOV DPTR,#MESS03
      ACALL RTLCD
      MOV A, #0D0H ;4nd line
      ACALL CMD
      ACALL DLAY
      MOV DPTR,#MESS04
      ACALL RTLCD
; 20 to 2F Area Initialization <Bit addressable range cleared to 0, where each
bit works like a single button ON/OFF switch>
        MOV R2,#00H
        MOV R1,#20H
A0:     MOV @R1,#00
```

```
        INC R1
        INC R2
        CJNE R2,#010H,A0 ;<16 bits cleared to 0>

      MOV R3, #00H  ;Initializing registers to avoid data corruption
      MOV R4, #00H
      MOV R5, #00H
      MOV 7DH, #00H ;INITIALIZE LSB OF COST
      MOV 7EH, #00H ;INITIALIZE MSB OF COST
      MOV 7FH, #00H ;INITIALIZE TOTAL NO OF PRODUCTS

DETECTION:
;TIMER DELAY TO PREVENT REPEATED SCANS OF THE SAME CARD
      MOV R0, #2D
LL:   MOV TH0,#00H
      MOV TL0,#00H
      SETB TR0
HH: JNB TF0, HH
      CLR TF0
      CLR TR0
      DJNZ R0, LL
; 70 to 7B Area Initialization <12 Hexadecimal Digit RFID code is stored here>
        MOV R2,#00H
        MOV R1,#70H
A1:     MOV @R1,#00H
        INC R1
        INC R2
        CJNE R2,#0CH,A1 ;<12 bits cleared to 0 Leaving 7D, 7E and 7F untouched>

        MOV R1, #70H
        MOV R2, #12D
WAIT: CJNE R2,#00H, WAIT     ;Wait for R2==0 <Waiting indefinitely for an RFID
card to get scanned>
;RFID card detected
;Now display the RFID of the corresponding card on the LCD
        MOV A, #0C4H          ;2nd line 4th pos
        ACALL CMD
        ACALL DLAY
        MOV R1,#70H           ;R1 used as pointer to 70H
      MOV R2,#12D           ;loads R1 with 12D
BACK1:MOV A,@R1             ;loads A with data pointed by R1
        ACALL DAT             ;calls DAT <data display> subroutine
      ACALL DLAY
        INC R1                ;incremets R1
      DJNZ R2,BACK1


;RFID Comparison begins here. Compare Subroutine is used for this purpose
;If card numbers match, product is added, if the same card is scanned twice,
product is removed and total cost and count are updated and displayed
;If card numbers don't match, UNRECOGNIZED is displayed on the screen
;If Master card is scanned, shopping stops and final cost and count are
displayed. After this, no card can be further scanned
;User can update the cost of a product by storing LSB and MSB or cost in
Hexidecimal in registers R3 and R4 respectively
      ;egg
      MOV DPTR,#RF0        ;load the address of the first character of the
stored rfid into the dptr for the compare subroutine to compare 2 rfids
```

```
      ACALL Compare        ;Compare subroutine which used cjne to compare the
immediately scanned rfid with an rfid stored in the database
      JB PSW.7, n0         ;The compare subroutine sets the carry bit if the
rfids don't match, hence prompting the program to compare
                                   ;the immediately scanned rfid with the next
rfid in the database and hence check for the next product.
                                   ;If the cards match, the next line is
executed.
      CPL 00H              ;Complementing the bit corresponding to the product
in the bit addressable range
      JNB 00H, eggrem      ;jump to product removal subroutine if the bit
corresponding to the product is cleared
      ACALL eggadd         ;go to product adding subroutine if the bit
corresponding to the product is set
JMP DETECTION            ;after the product is added or removed, the space
where the scanned rfid is stored needs to be initialized again
eggadd:                  ; product adding subroutine
      MOV DPTR,#MESS06
      ACALL recdisp        ;display product added message
      ACALL PRODADD        ;increment product count
      ACALL no_disp        ;display the updated product count
      MOV R4, #00H         ;MSBytes of the cost of the product in hexadecimal
      MOV R3, #46H         ;LSBytes of the cost of the product in hexadecimal
<70 rupees>
      ACALL COSTADD        ;Add the cost of this product to the total cost
      ACALL cost_disp      ;display the updated total cost
      RET                  ;return from this subroutine
eggrem:                  ;product removing subroutine
      MOV DPTR,#MESS07     ;display product added message
      ACALL recdisp        ;display product added message
      ACALL PRODSUBB       ;decrement product count
      ACALL no_disp        ;display the updated product count
      MOV R4, #00H         ;MSBytes of the cost of the product in hexadecimal
      MOV R3, #46H         ;LSBytes of the cost of the product in hexadecimal
<70 rupees>
      ACALL COSTSUBB       ;Subtract the cost of this product to the total cost
      ACALL cost_disp      ;display the updated total cost
JMP DETECTION            ;jump for detecting the next scanned card

n0:                 ;milk     ;identification of the next product happens if the
rfid scanned erlier didn't match earlier
      MOV DPTR,#RF1
      ACALL Compare
      JB PSW.7, n1
      CPL 01H
      JNB 01H, milkrem
      ACALL milkadd
      JMP DETECTION
milkadd:
      MOV DPTR,#MESS08
      ACALL recdisp
      ACALL PRODADD
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #46H ;<70 rupees>
      ACALL COSTADD
      ACALL cost_disp
      RET
```

```
milkrem:
      MOV DPTR,#MESS09
      ACALL recdisp
      ACALL PRODSUBB
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #46H ;<70 rupees>
      ACALL COSTSUBB
      ACALL cost_disp
JMP DETECTION

n1:                           ;butter
      MOV DPTR,#RF2
      ACALL Compare
      JB PSW.7, n2
      CPL 02H
      JNB 02H, butterrem
      ACALL butteradd
      JMP DETECTION
butteradd:
      MOV DPTR,#MESS0A
      ACALL recdisp
      ACALL PRODADD
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #64H ;<100 rupees>
      ACALL COSTADD
      ACALL cost_disp
      RET
butterrem:
      MOV DPTR,#MESS0B
      ACALL recdisp
      ACALL PRODSUBB
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #64H ;<100 rupees>
      ACALL COSTSUBB
      ACALL cost_disp
JMP DETECTION

n2:                           ;cereal
      MOV DPTR,#RF3
      ACALL Compare
      JB PSW.7, n3
      CPL 03H
      JNB 03H, p4rem
      ACALL p4add
JMP DETECTION
p4add:
      MOV DPTR,#MESS0C
      ACALL recdisp
      ACALL PRODADD
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #0AAH ;<170 rupees>
      ACALL COSTADD
      ACALL cost_disp
      RET
```

```
p4rem:
      MOV DPTR,#MESS0D
      ACALL recdisp
      ACALL PRODSUBB
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #0AAH ;<170 rupees>
      ACALL COSTSUBB
      ACALL cost_disp
JMP DETECTION

n3:                           ;salt
      MOV DPTR,#RF4
      ACALL Compare
      JB PSW.7, n4
      CPL 04H
      JNB 04H, p5rem
      ACALL p5add
      JMP DETECTION
p5add:
      MOV DPTR,#MESS0E
      ACALL recdisp
      ACALL PRODADD
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #32H ;<50 rupees>
      ACALL COSTADD
      ACALL cost_disp
      RET
p5rem:
      MOV DPTR,#MESS0F
      ACALL recdisp
      ACALL PRODSUBB
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #32H ;<50 rupees>
      ACALL COSTSUBB
      ACALL cost_disp
JMP DETECTION

n4:                           ;cheese
      MOV DPTR,#RF5
      ACALL Compare
      JB PSW.7, n5
      CPL 05H
      JNB 05H, p6rem
      ACALL p6add
      JMP DETECTION
p6add:
      MOV DPTR,#MESS10
      ACALL recdisp
      ACALL PRODADD
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #64H ;<100 rupees>
      ACALL COSTADD
      ACALL cost_disp
      RET
```

```
p6rem:
      MOV DPTR,#MESS11
      ACALL recdisp
      ACALL PRODSUBB
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #64H ;<100 rupees>
      ACALL COSTSUBB
      ACALL cost_disp
JMP DETECTION

n5:                           ;bread
      MOV DPTR,#RF6
      ACALL Compare
      JB PSW.7, n6
      CPL 06H
      JNB 06H, p7rem
      ACALL p7add
JMP DETECTION
p7add:
      MOV DPTR,#MESS12
      ACALL recdisp
      ACALL PRODADD
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #28H ;<40 rupees>
      ACALL COSTADD
      ACALL cost_disp
      RET
p7rem:
      MOV DPTR,#MESS13
      ACALL recdisp
      ACALL PRODSUBB
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #28H ;<40 rupees>
      ACALL COSTSUBB
      ACALL cost_disp
JMP DETECTION

n6:                           ;sugar
      MOV DPTR,#RF7
      ACALL Compare
      JB PSW.7, n7
      CPL 07H
      JNB 07H, p8rem
      ACALL p8add
      JMP DETECTION
p8add:
      MOV DPTR,#MESS14
      ACALL recdisp
      ACALL PRODADD
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #28H ;<40 rupees>
      ACALL COSTADD
      ACALL cost_disp
      RET
```

```
p8rem:
      MOV DPTR,#MESS15
      ACALL recdisp
      ACALL PRODSUBB
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #28H ;<40 rupees>
      ACALL COSTSUBB
      ACALL cost_disp
JMP DETECTION

n7:                          ;apples
      MOV DPTR,#RF8
      ACALL Compare
      JB PSW.7, M
      CPL 08H
      JNB 08H, p9rem
      ACALL p9add
      JMP DETECTION
p9add:
      MOV DPTR,#MESS16
      ACALL recdisp
      ACALL PRODADD
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #64H ;<100 rupees>
      ACALL COSTADD
      ACALL cost_disp
      RET
p9rem:
      MOV DPTR,#MESS17
      ACALL recdisp
      ACALL PRODSUBB
      ACALL no_disp
      MOV R4, #00H
      MOV R3, #64H ;<100 rupees>
      ACALL COSTSUBB
      ACALL cost_disp
JMP DETECTION

M:    MOV DPTR,#RF9      ;RFID of the master card is loaded
      ACALL Compare      ;checking whethe the scanned card is the master card
      JB PSW.7, unrec    ;if it is  neither any card stored in the database nor
a master card, display "UNRECOGNIZED"
      ACALL MASTER       ;If it is the master card, go to its subroutine
MASTER:                ;The master card subroutine begins
      MOV DPTR,#MESSMm   ;Move the address of the final messages to the dptr
      ACALL recdisp      ;display the messages indicating that shopping has
ended on the 1st line of the LCD
      MOV A, #0C0H       ;2nd line
      ACALL CMD
      ACALL DLAY
      MOV DPTR,#MESSNn
      ACALL RTLCD        ;display the messages indicating that shopping has
ended on the 2nd line of the LCD

EXITT:SJMP EXITT       ;program gets stuck in this loop indefinitely until the
reset button is pressed
```

```
;IMPORTANT SUBROUTINES:
;CJNE used 12 times for RFID digit comparizon and recognization
Compare:
CLR PSW.7
CLR A
MOVC A, @A+DPTR
CJNE A,70H,Fail
INC DPTR
CLR A
MOVC A, @A+DPTR
CJNE A,71H,Fail
INC DPTR
CLR A
MOVC A, @A+DPTR
CJNE A,72H,Fail
INC DPTR
CLR A
MOVC A, @A+DPTR
CJNE A,73H,Fail
INC DPTR
CLR A
MOVC A, @A+DPTR
CJNE A,74H,Fail
INC DPTR
CLR A
MOVC A, @A+DPTR
CJNE A,75H,Fail
INC DPTR
CLR A
MOVC A, @A+DPTR
CJNE A,76H,Fail
INC DPTR
CLR A
MOVC A, @A+DPTR
CJNE A,77H,Fail
INC DPTR
CLR A
MOVC A, @A+DPTR
CJNE A,78H,Fail
INC DPTR
CLR A
MOVC A, @A+DPTR
CJNE A,79H,Fail
INC DPTR
CLR A
MOVC A, @A+DPTR
CJNE A,7AH,Fail
INC DPTR
CLR A
MOVC A, @A+DPTR
CJNE A,7BH,Fail
SJMP Success
Fail:
SETB PSW.7
SJMP Fin
Success:
```

```
CLR PSW.7
SJMP Fin
Fin: RET

;Display for Unrecognized Card
unrec: NOP
      MOV A, #080H ;1st line
      ACALL CMD
      ACALL DLAY
      MOV DPTR,#MESS05
      ACALL RTLCD
      JMP DETECTION
;Display Recognized Product
recdisp:MOV A, #080H ;1st line
      ACALL CMD
      ACALL DLAY
      ACALL RTLCD
      RET
;ADD A PRODUCT
PRODADD:
      MOV A, 7FH
      CLR C
      ADD A, #01H
      MOV 7FH, A
      RET
;REMOVE A PRODUCT
PRODSUBB:
      MOV A, 7FH
      CLR C
      SUBB A, #01H
      MOV 7FH, A
      RET
;Display total number of Products
no_disp:
      MOV A, #097H ;3rd line 7th position
      ACALL CMD
      ACALL DLAY
;HEX TO DECIMAL CONVERSION OF TOTAL NUMBER OF PRODUCTS
HTD:
      MOV A, 7FH
      MOV B, #100D
      DIV AB
      MOV R1, A
      MOV A, B
      MOV B, #10D
      DIV AB
      MOV R2, A
      MOV R3, B
      MOV A, R1
      ADD A, #30H
      ACALL DAT
      ACALL DLAY
      MOV A, R2
      ADD A, #30H
      ACALL DAT
      ACALL DLAY
      MOV A, R3
      ADD A, #30H
```

```
      ACALL DAT
      ACALL DLAY
      RET

;COST AFTER ADDING A PRODUCT
 COSTADD:
  ;Step 1 of the process
  MOV A,7DH     ;Move the low-byte into the accumulator
  ADD A,R3      ;Add the second low-byte to the accumulator
  MOV R3,A      ;Move the answer to the low-byte of the result
  MOV 7DH,A
  ;Step 2 of the process
  MOV A,7EH     ;Move the high-byte into the accumulator
  ADDC A,R4     ;Add the second high-byte to the accumulator, plus carry.
  MOV R4,A      ;Move the answer to the high-byte of the result
  MOV 7EH,A
  RET           ;Return
;COST AFTER REMOVING A PRODUCT
COSTSUBB:
  ;Step 1 of the process
  MOV A,7DH     ;Move the low-byte into the accumulator
  CLR C         ;Always clear carry before first subtraction
  SUBB A,R3     ;Subtract the second low-byte from the accumulator
  MOV R3,A      ;Move the answer to the low-byte of the result
  MOV 7DH,A
  ;Step 2 of the process
  MOV A,7EH     ;Move the high-byte into the accumulator
  SUBB A,R4     ;Subtract the second high-byte from the accumulator
  MOV R4,A      ;Move the answer to the high-byte of the result
  MOV 7EH,A
  RET           ;Return

;Display total cost of Products
cost_disp:
      MOV A, #0DBH ;1st line
      ACALL CMD
      ACALL DLAY
      MOV A, R4
      MOV R1, A ;R1 HAS MSByte
      MOV A, R3
      MOV R2, A ;R2 HAS LSByte
;HEX TO DECIMAL CONVERSION OF TOTAL COST OF PRODUCTS
Hex2BCD:
           MOV R3,#00D
           MOV R4,#00D
           MOV R5,#00D
           MOV R6,#00D
           MOV R7,#00D

           MOV B,#10D
           MOV A,R2
           DIV AB
           MOV R3,B ;
           MOV B,#10 ; R7,R6,R5,R4,R3
           DIV AB
           MOV R4,B
           MOV R5,A
           CJNE R1,#0H,HIGH_BYTE ; CHECK FOR HIGH BYTE
```

```
            SJMP ENDD
HIGH_BYTE:
            MOV A,#6
            ADD A,R3
            MOV B,#10
            DIV AB
            MOV R3,B
            ADD A,#5
            ADD A,R4
            MOV B,#10
            DIV AB
            MOV R4,B
            ADD A,#2
            ADD A,R5
            MOV B,#10
            DIV AB
            MOV R5,B
            CJNE R6,#00D, ADD_IT
            SJMP CONTINUE
ADD_IT:
            ADD A,R6
CONTINUE:
            MOV R6,A
            DJNZ R1, HIGH_BYTE
            MOV B, #10D
            MOV A,R6
            DIV AB
            MOV R6,B
            MOV R7,A
ENDD:NOP
      MOV A, R7
      ADD A, #30H  ;Adding 30H to each decimal digit converts it to the
corresponding ascii value of the same digit so that it can be displayed on the
LCD
      ACALL DATSP
      ACALL DLAYSP
      MOV A, R6
      ADD A, #30H
      ACALL DATSP
      ACALL DLAYSP
      MOV A, R5
      ADD A, #30H
      ACALL DATSP
      ACALL DLAYSP
      MOV A, R4
      ADD A, #30H
      ACALL DATSP
      ACALL DLAYSP
      MOV A, R3
      ADD A, #30H
      ACALL DATSP
      ACALL DLAYSP
    RET

;Command Subroutine
CMD: CLR P0.7
       CLR P0.6
       SETB P0.5
```

```
       MOV P2, A
       ACALL DLAY
       CLR P0.5
       RET
;Data Subroutine
DAT: SETB P0.7
       CLR P0.6
       SETB P0.5
       MOV P2, A
       ACALL DLAY
       CLR P0.5
       RET
;Delay Subroutine
DLAY: MOV R7, #0FFH
Back: MOV R6, #0AH
Here: DJNZ R6, Here
        DJNZ R7, Back
        RET
;Return to LCD Subroutine
RTLCD:NOP
L1: CLR A
      MOVC A,@A+DPTR
      JZ OVER
      ACALL DAT
      ACALL DLAY
      INC DPTR
      SJMP L1
OVER:RET
;SPECIAL DELAY RESERVED IF R7 AND R6 ARE UNDER USE
DLAYSP:MOV R1, #0FFH
Back2: MOV R2, #0AH
Here1: DJNZ R2, Here1
        DJNZ R1, Back2
        RET
;SPECIAL Data Subroutine WHICH USES SPECIAL DELAY
DATSP: SETB P0.7
       CLR P0.6
       SETB P0.5
       MOV P2, A
       ACALL DLAYSP
       CLR P0.5
       RET


;DATABASE
;Initial Display Messages
MESS01:DB '  AUTO BILLING  ',0
MESS02:DB 'ID: ############',0
MESS03:DB 'ITEMS: 000      ',0
MESS04:DB 'TOTAL AMT: 00000',0
MESS05:DB '  UNRECOGNIZED  ',0
MESS06:DB '   EGGS ADDED   ',0
MESS07:DB '  EGGS REMOVED  ',0
MESS08:DB '   MILK ADDED   ',0
MESS09:DB '  MILK REMOVED  ',0
MESS0A:DB '  BUTTER ADDED  ',0
MESS0B:DB ' BUTTER REMOVED ',0
MESS0C:DB '  CEREAL ADDED  ',0
```

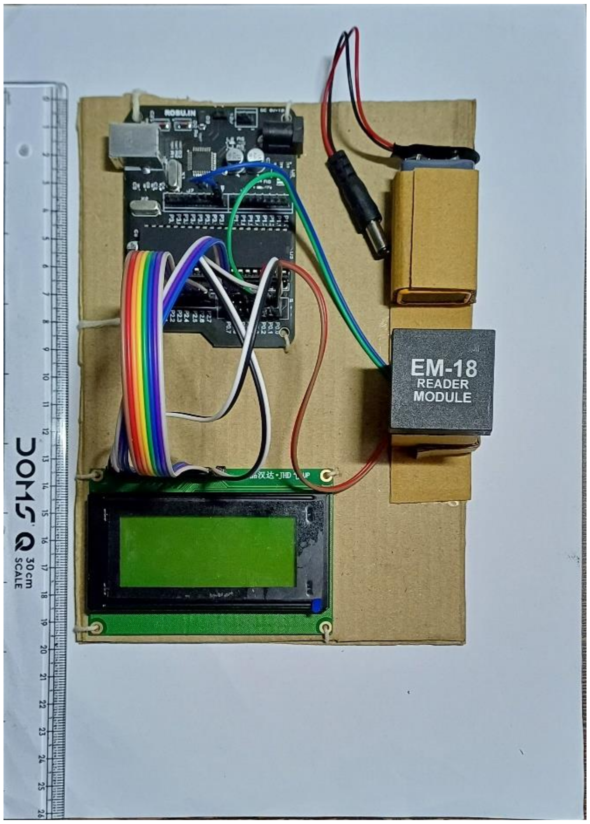
```
MESS0D:DB ' CEREAL REMOVED ',0
MESS0E:DB '   SALT ADDED   ',0
MESS0F:DB '  SALT REMOVED  ',0
MESS10:DB '  CHEESE ADDED  ',0
MESS11:DB ' CHEESE REMOVED ',0
MESS12:DB '   BREAD ADDED  ',0
MESS13:DB '  BREAD REMOVED ',0
MESS14:DB '   SUGAR ADDED  ',0
MESS15:DB '  SUGAR REMOVED ',0
MESS16:DB '  APPLES ADDED  ',0
MESS17:DB ' APPLES REMOVED ',0
MESSMm:DB '   THANK YOU!   ',0
MESSNn:DB ' DO VISIT AGAIN ',0

;RFIDs Stored as strings corresponding to the product
RF0:DB '4300435A2D77',0
RF1:DB '43004357E2B5',0
RF2:DB '4300433493A7',0
RF3:DB '4300432E9FB1',0
RF4:DB '4300430AD6DC',0
RF5:DB '43004305FAFF',0
RF6:DB '430042F75BAD',0
RF7:DB '430042F58470',0
RF8:DB '430042DC04D9',0
RF9:DB '430042CD7AB6',0

;Serial Communication Interrupt
ORG 0023H
      JMP 0A00H
ORG 0A00H
      CLR RI
      MOV A, SBUF
      MOV @R1,A
      INC R1
      DEC R2
      RETI
END
```

APPENDIX II

PHOTO OF HARDWARE



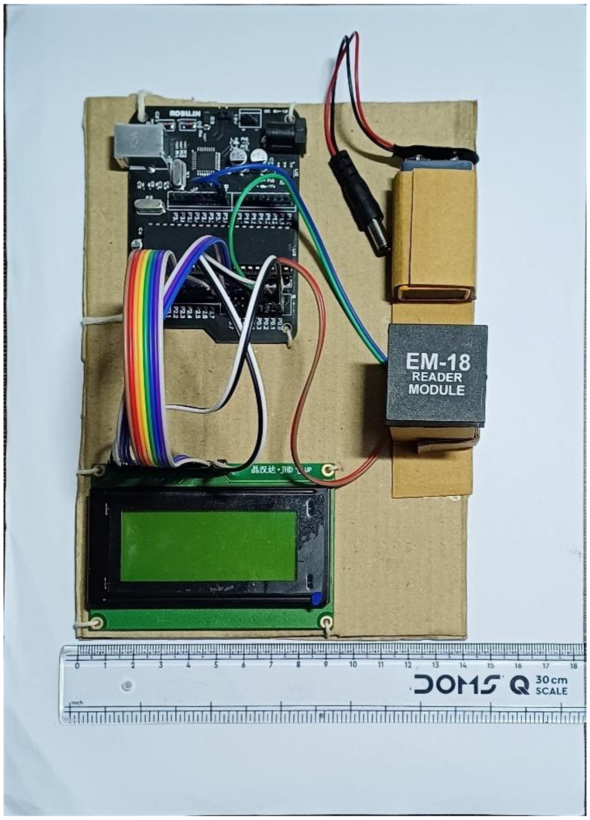


Figure 6 and 7: The Dimensions of the project: 19.8 cm in length and 14 cm in width

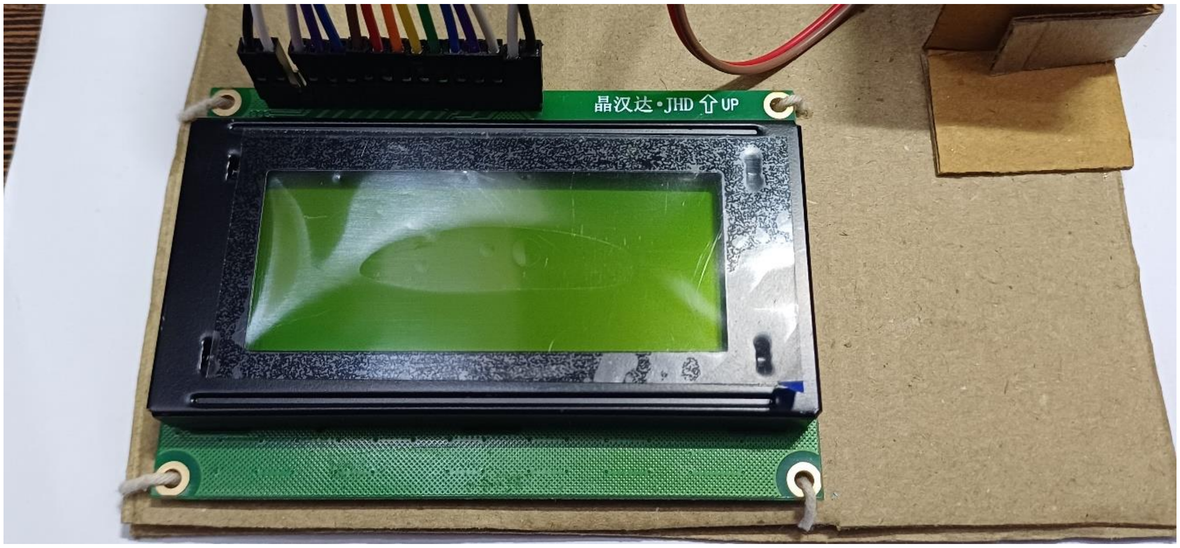


Figure 8: The LCD is tied onto the cardboard using strings and the jumpers connected to it are also connected to the AT89S52

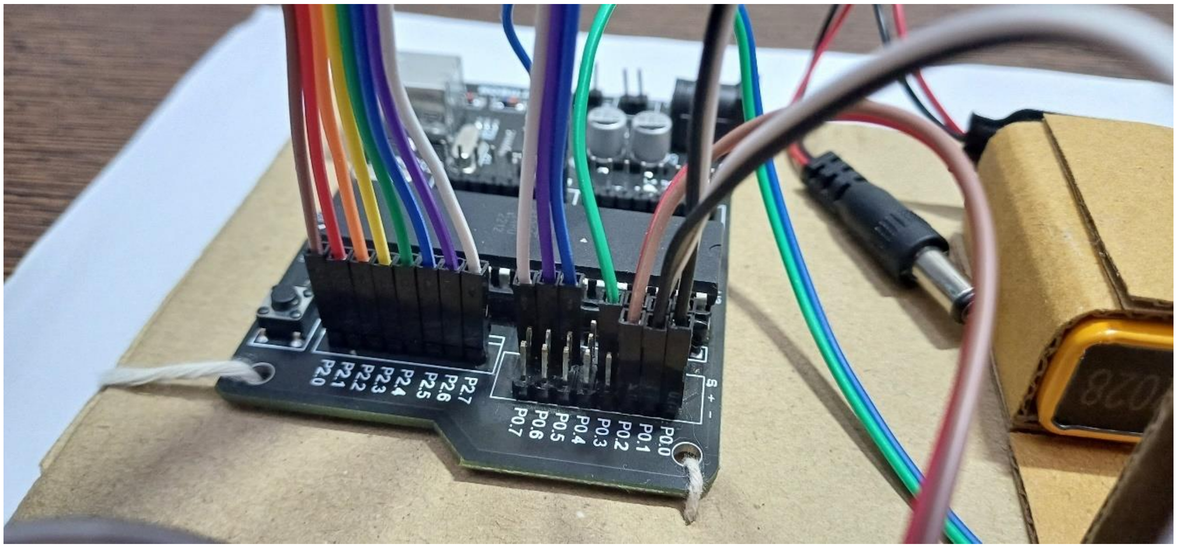


Figure 9: Port 0, Port 2 and Power Supply pins are visible in this image. The Data lines entirely consume P2 of the microcontroller, while P0.7, P0.6 and P0.5 are connected to the RS, R/Wbar and E of the LCD. The power supply is given to the Vss, Vdd, LEDA and LEDK of the LCD as well as to the Vcc and GND of the EM-18 RFID reader module

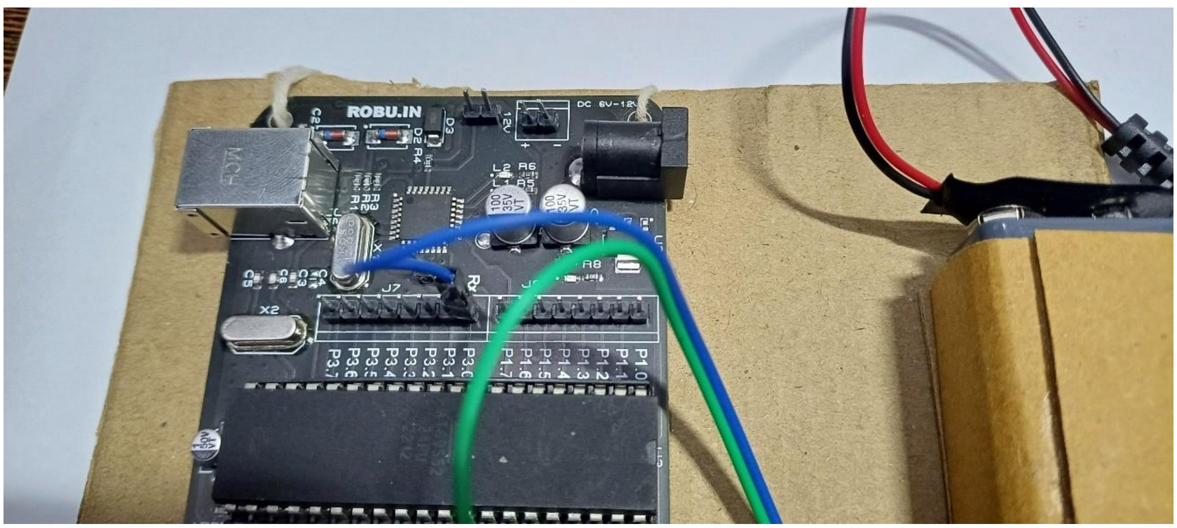


Figure 10: Port 1 and Port 3 of the Microcontroller are visible here. The Rx pin (P3.0) of the Microcontroller is given to the Tx pin of the EM-18 module

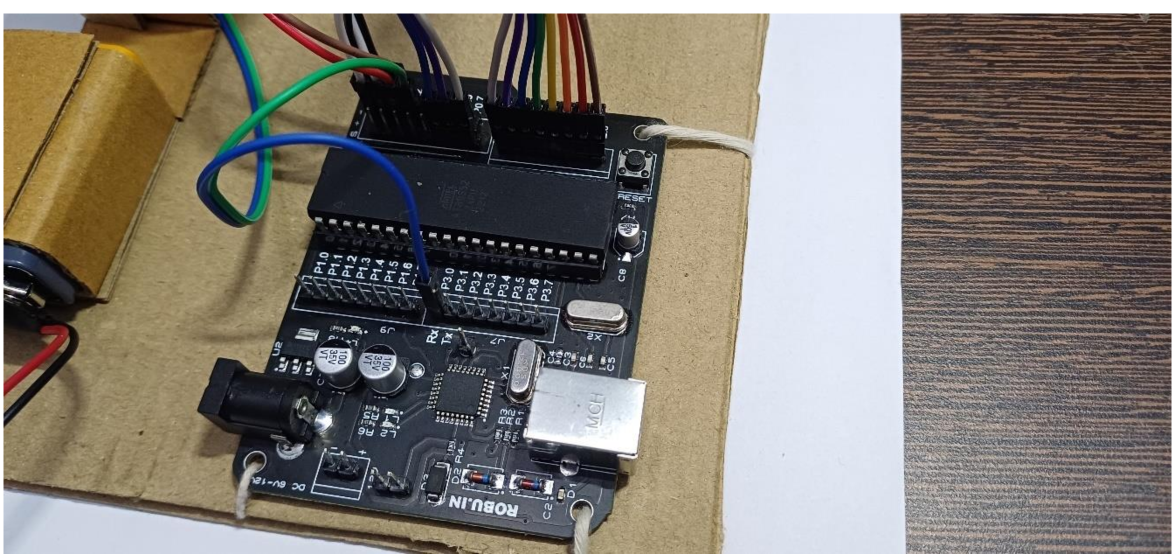


Figure 11: The AT89S52 Microcontroller is the Brain of this project. It is interfaced with all the other devices

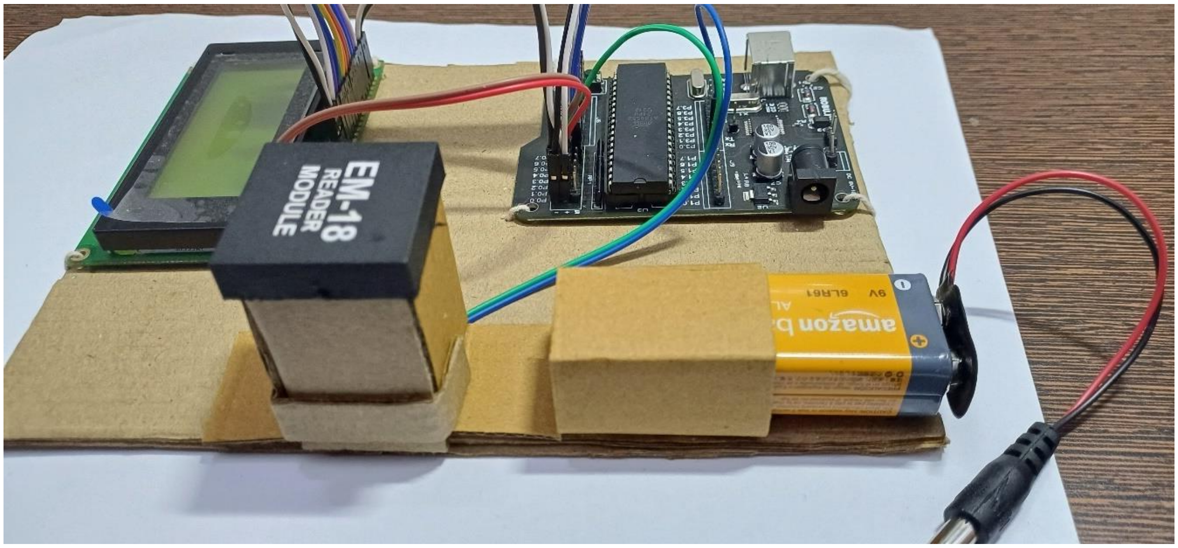


Figure 12: Here, the 9V Transistor Radio Battery is removed from its housing for displaying it. The housing of this component prevents it from moving during the working of the model

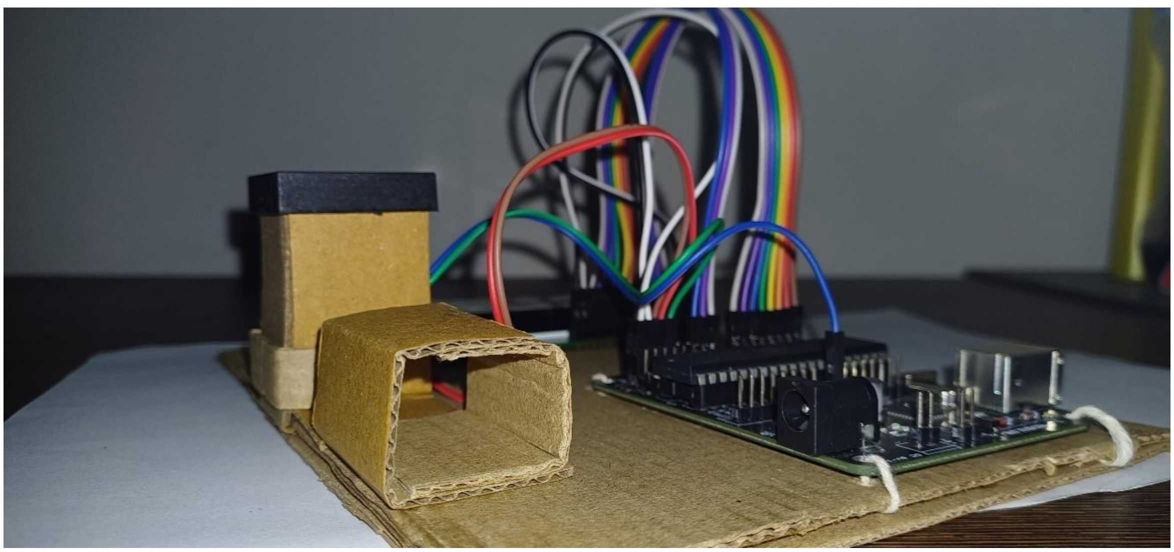


Figure 16: A side view of the battery housing with the battery removed

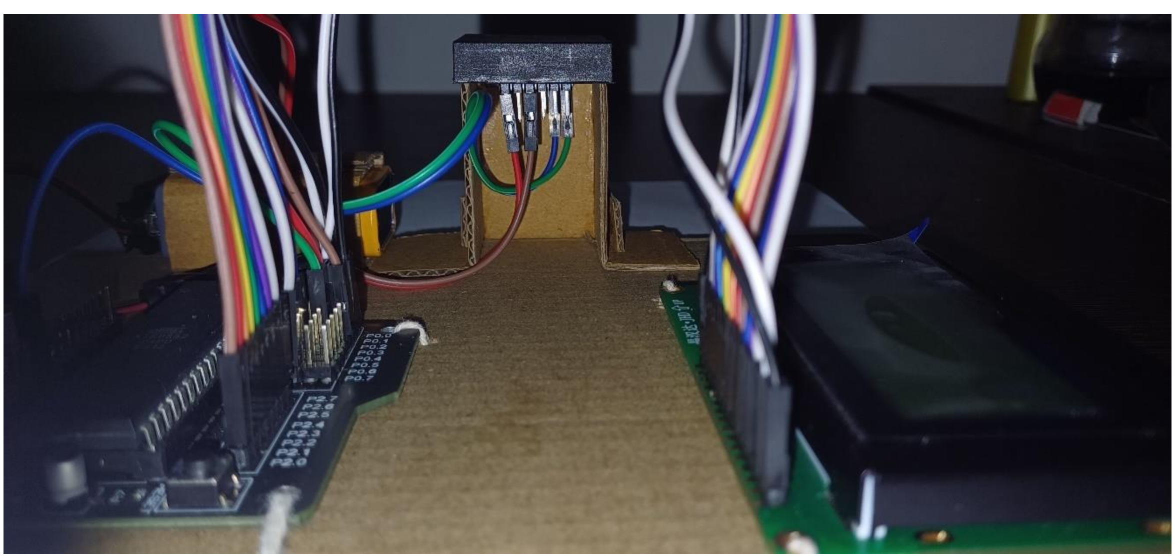


Figure 17: A side view of the structural support provided to the EM-18 module. Providing this support was necessary to ensure the component stays intact in place. 4 Jumper Wires are connected to the EM-18 for making it work

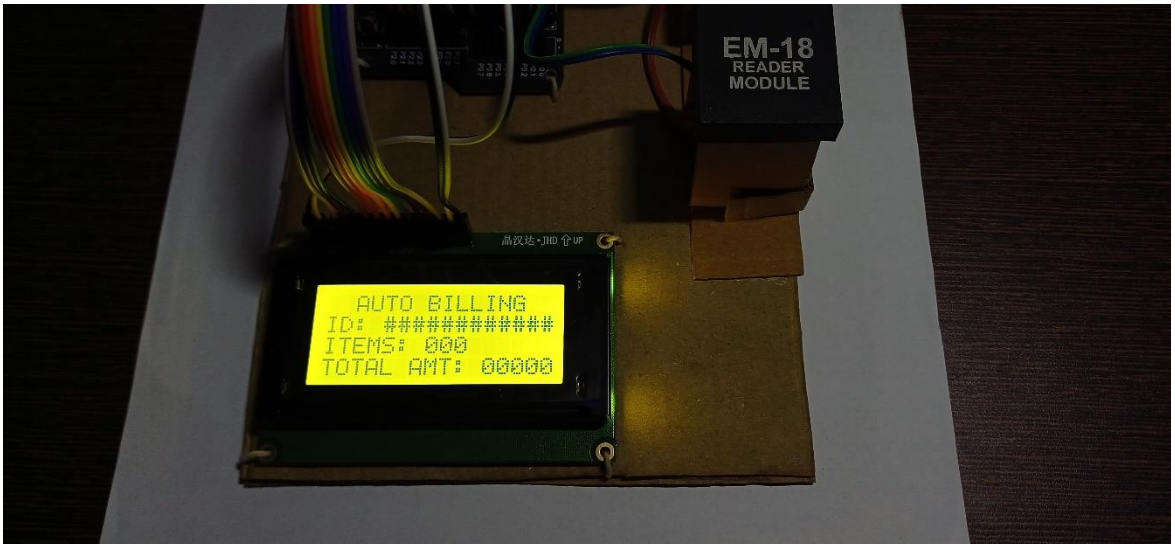


Figure 18: A low light image of the LCD under operation, for better visibility of the text printed on the LCD

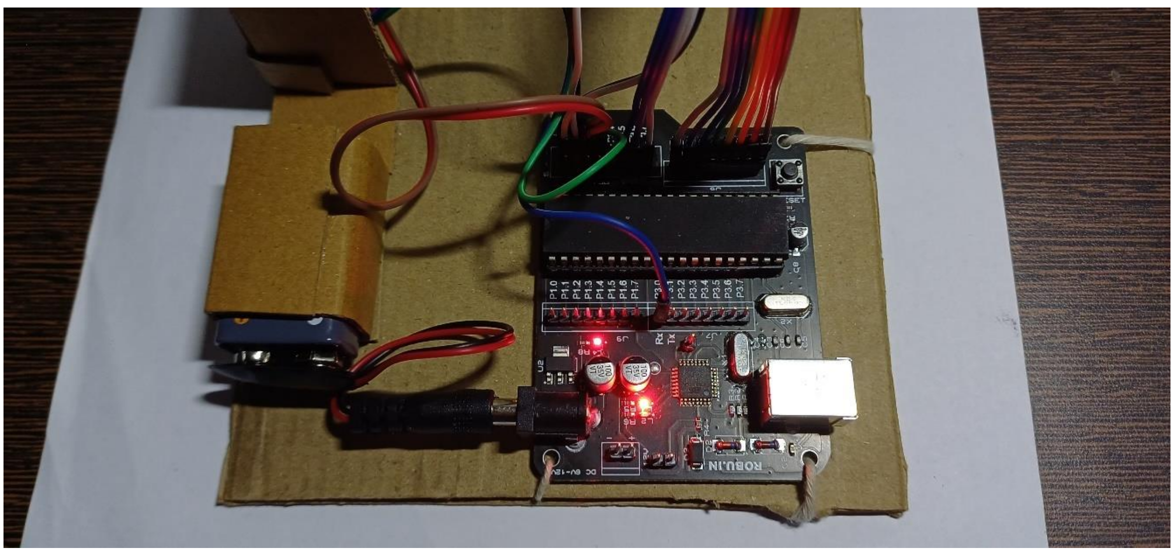


Figure 16: Microcontroller under operation when the DC jack is connected to it for supplying power

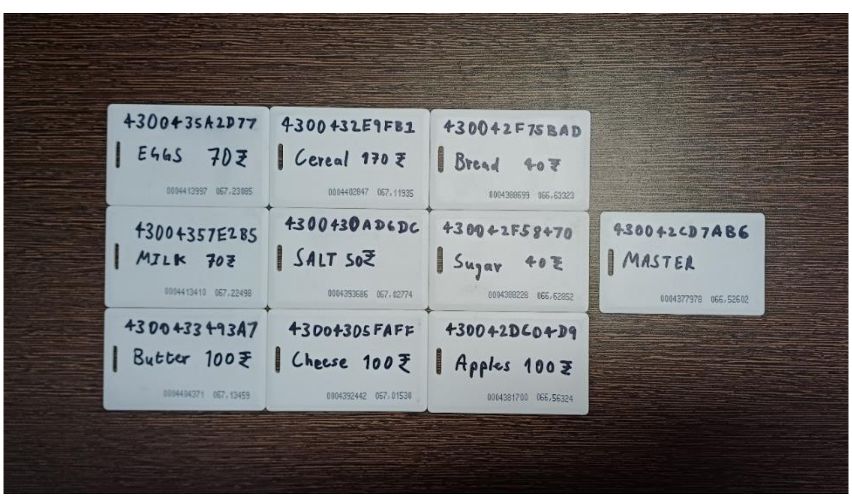


Figure 17: All the RFIDs which are associated with a product as well as the Master card

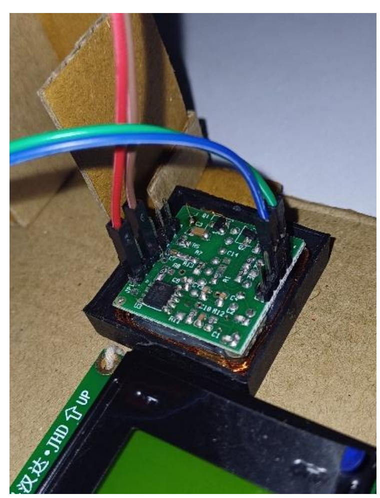


Figure 18: A view of the underside of the EM-18 module. The Red and Brown wires are Vcc and Ground respectively. The green is the selection line and the blue is the Tx

APPENDIX III

USEFUL REFERENCES AND LINKS

AT89S52 Datasheet:
https://ww1.microchip.com/downloads/en/DeviceDoc/doc1919.pdf

EM18 description:
https://components101.com/modules/em18-rfid-reader-module

LCD Datasheet:
https://pdf1.alldatasheet.com/datasheetpdf/view/226539/ETC2/JHD164A.html

Interfacing of EM18 RFID reader in C language:
https://www.electronicwings.com/8051/rfid-reader-em18-interface-with-8051

Webpage which claimed to interface an RFID reader, but their source code doesn't seem to work in proteus simulation. However, this code gave us an insight into how to proceed with the RFID reader interfacing:
https://www.circuitstoday.com/interfacing-rfid-module-to-8051

Webpage which gave us an insight to work with bit addressable range in the RAM:
https://what-when-how.com/8051-microcontroller/bit-addresses-for-io-and-ram/

Basic LCD Interfacing with 8051:
https://www.youtube.com/watch?v=pihAdSek7oM

Efficient way of Passing Strings to LCD:
https://www.youtube.com/watch?v=LtevacTk7Ww

Webpage which aided in conversion of 16 bit Hexadecimal number to BCD:
https://www.refreshnotes.com/2016/05/8051-program-16-bit-hex-to-bcd.html

Tutorial on flashing code into the AT89S52 on our specific development board:
https://robu.in/smartelex-aryabhatta-8051-development-board-interfacing-with-led-tutorial/

8051 Development board was purchased from:
https://robu.in/product/smartelex-aryabhatta-8051-microcontroller-development-board-at89s52-with-onboard-usb-programmer/

16x4 LCD display was purchased from:
https://robu.in/product/jhd-16x4-character-lcd-display-with-yellow-backlight/

9V Transistor Radio Battery was purchased from:
https://amzn.in/d/4Upz4tg

Data Transfer Cable was purchased from:
https://amzn.in/d/bD2i0KO

Jumper wires were purchased from:
https://www.amazon.in/Aptechdeals-Jumper-Wires-Female-Breadboard/dp/B074JB6SX8/ref=pd_day0_d_sccl_2_2/260-0563295-8781152?pd_rd_w=nOsF7&content-id=amzn1.sym.49a78501-94d9-417e-bc7b-53394962d54d&pf_rd_p=49a78501-94d9-417e-bc7b-53394962d54d&pf_rd_r=44F7RB0K606MKQJXBYHQ&pd_rd_wg=089z9&pd_rd_r=c22cfd00-bddf-4539-ac2f-7014e50d4365&pd_rd_i=B074JB6SX8&psc=1

EM-18 RFID Reader Module was purchased from:
https://www.amazon.in/dp/B09PZ3NB4Q?ref=ppx_pop_mob_ap_share

RFID Cards were purchased from:
https://www.amazon.in/dp/B07M6H9G91?ref=ppx_pop_mob_ap_share

Male Berg strip (Breakaway header pins) were purchased from:
https://www.amazon.in/40%C3%971-Male-Strip-Break-Header-Straight/dp/B09JV21G2H/ref=sr_1_3?crid=3VL50FA1TD2SW&dib=eyJ2IjoiMSJ9.G_H7pZGTIWcFiOFWkjL6CxxxpeuQSjaLX4iG-DBQj4iOP8z066IUo-RNPdE4i8GXeNSwTlskYU0kl77wuXiRq6KEpptzKn3cUujgCprzqpOFBl72ePHHN_uRhR8aOYxXyjtA_v7l3mUqDTIh-VnieVUwp4gI58jlceyvqrVzmYRvHWJbU8_laQCrJ7fbgAf8U6Qr-GAk3SQf2p_wf7FsDwtXS04N4-DHwMgRgemsYHHsMsftN_Zysq1BqmoSYcAVVtVjbEjKVjOuGQSKL0QoLQqoTcBad-ctR5CLhMxHa4I.eBBximKh0uGPo3zD0YGU91ZSh6s2AEqFfziLhXcpXbQ&dib_tag=se&keywords=breakaway+header+pins&qid=1714313345&s=industrial&sprefix=breakaway%2Cindustrial%2C3891&sr=1-3