23CSE201 - Punch Card Based Instruction Set Architecture Using Arduino

Term Project

Adithya Ranjith
CB.SC.U4CSE24703

Adwit Singh
CB.SC.U4CSE24704

Akilan S S
CB.SC.U4CSE24707

Naveen Velan S N
CB.SC.U4CSE24734

August 2025

Contents

Abstract

This project, as part of the course 23CSE201 - Procedural Programming Using C, attempts to use Arduino® to develop a punch-card-based instruction set architecture that has minimal instructions. The instructions are encoded in a punch card strip containing 4 holes and 1 auxiliary hole per row. These instructions are inspired by the x86 instruction listings. Despite the limitation in the number of holes per row, the auxiliary hole is used to combine the holes of adjacent rows for a single instruction. Simulated registers are used and displayed using an LED matrix, where the size of each register will be 256 units. C programming language will be used to decode and perform the instructions of the punch-card strip. At the end of the project, we will attempt to develop a punch card strip to generate n^{th} Fibonacci number and other algorithms.

1 Introduction

1.1 Project Overview

Inspired from historic background going way back to Jacquard's Loom, which uses punch-card based systems to control weaving patterns, which was the first forms of automation and programmability, we try to integrate modern ideas of computer architecture for this project. The project involves the standard **fetch-decode-execute cycle**, where there is a long strip of card containing 4 columns separated into rows with additional columns, which can be punched or not. This card encodes the instruction to be performed. This card is then fed into a reader, which consists of highly luminous LED strip at the bottom of the feeder, and four LDR sensors as close as possible to detect holes in the strip. This information is sent to Arduino for decoding and execution. The feeder is also additionally controlled by a stepper motor, which can also perform JMP instruction, allowing our punch-card strip to be flexible. We also use a buzzer to provide a auditory feedback for every instruction performed, and use an LED matrix to display our "simulated" register. At the end of the project, we try to develop punch-card strips for some well known algorithms.

2 Problem Definition and Requirement

2.1 The Problem Statement

We attempt to develop a physical ISA with punch-card strips using Arduino.

2.2 Conceptual Requirements

2.2.1 Instruction Set Definition

The ISA is currently proposed to have 4 holes in a row containing instruction, and an additional auxiliary holes to instruct the feeder to wait for data/ more instruction in the next row. We plan on implementing basic opcodes inspired from x86 like MOV, INC, DEC, CMP, JMP, ADD, CPY, MULT, NOP and HALT. We also propose virtual minimal registers for storing data and flags.

2.2.2 Punch-Card Format

The dimensions of the punch-card are not finalized yet, but the strip would contain at least 4 holes for instruction and an auxiliary hole containing instructions to the feeder.

2.2.3 Reading Mechanism

We propose to use bright LED strips under the feeder to illuminate the holes, and use LDR sensors effectively to read the punch-card. Since the feeder is a physical system, we will also develop fault and error tolerance measure for the strip. The holes will be placed such that the light from one hole doesn't leak into the the sensor of the other hole. We will be using stepper motors to feed the strip into the feeder. As mentioned earlier, this will also be used to implement JMP instruction.

2.2.4 Execution Cycle

We develop state-machines to interpret the state of the feeder and the instruction. We will use memory-onboard the microcontroller to simulate the register, and use LED matrix to display the results and data stored in the registers.

3 Architecture and Design

3.1 High Level System Architecture

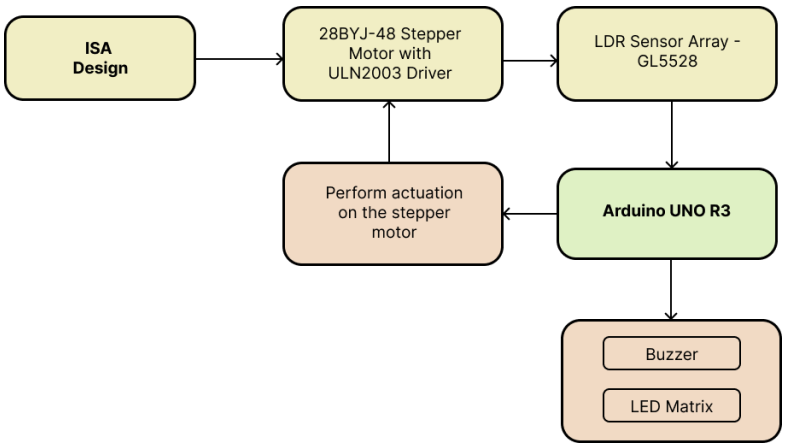

Figure 1: Architecture Diagram

3.2 Flow Diagram

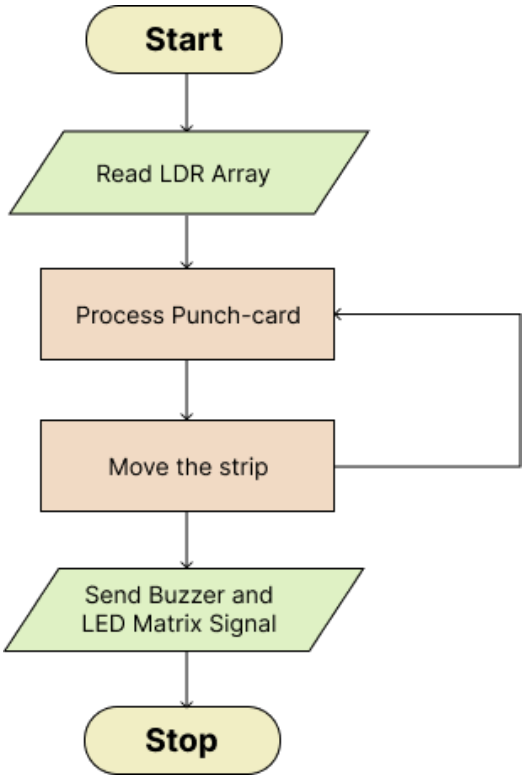

Figure 2: Flow Diagram

3.3 Instruction Set Architecture (ISA) Design

Opcode	Definition	Punched_card_rep
NOP	No Operation	1111
HALT	Terminate Program	0000
MOV	Move the contents of one register to another register	1000
CPY	Copy the contents of one register to another register	0100
ADD	Add content of a register or a value to another register	1100
INC	Increment the content of register by 1	0110
DEC	Decrement the content of the register by 1	1001
MULT	Multiply content of a register or a value to another register	0101
JMP	Move to the specified instruction row	1011
CMP	Compares the contents of two register	1101

3.4 Physical Punch-Card Design

Ins. No.	I_1	I_2	I_3	I_4	Aux
1		○	○	○	
	○	○	○	○	●
	○	○	○		
2	○	○	○	○	
	○	○	○	○	●
	○	○	○		
3		○	○	○	
	○	○	○	○	●
	○	○	○	○	
			○	○	

4 Hardware Specification

4.1 Software Requirements

We shall use the Arduino IDE to develop the project. A working computer with 8GB of RAM and a processor above 5th generation is required for development.

4.2 Arduino UNO R3

We use Arduino® UNO R3 board to control and process the sensors required for the Punch-card strip. The 14 Digital IO Pins along with the 6 analog IO pins will be used to detect and send appropriate feedback

4.3 LDR Sensors - GL5528

We will be using GL5528 CdS Photoconductive cells to detect the presence of hole.

4.4 28BYJ-48 Stepper Motor with ULN2003 Driver

The 5 wired stepper motor will be used along with ULN2003 driver which reduces the number of pins to 4, which will be connected to the the digital IO of the arduino board.

4.5 Buzzer

Buzzer is used to provide real time feedback about the status of the readings.

4.6 LED Matrix

An LED Matrix will be used to display and view the results of the programs and the values of the registers in real time.

5 Testing and Validation

Below is the sample opcode-pseudocode for the program to print fibonacci number.

```
1 MOV 0 A
2 MOV 0 B
3 MOV 0 I
4 MOV 5 N
5 CMP I N
6 GOTO 12
7 MOV A C
8 ADD C B
9 MOV B A
10 MOV C B
11 GOTO 5
12 HALT
```

6 C Concepts Used

6.1 Data Structures

Structs are used to create custom data types that logically group related information. For example, a single punch-card instruction can be represented by a struct containing its opcode and any operands. This encapsulation simplifies program logic by allowing entire instructions to be passed between functions.

6.2 Dynamic Memory

This allows the program to request and release memory from the heap at runtime, which would be crucial if the number of instructions in a punch-card program were not known in advance.

6.3 State Machines

The project's logic is structured as a state machine. This architectural pattern dictates the program's behaviour based on its current state and transitions between states in response to specific inputs. For example, the instruction decoder can transition from a FETCH_OPCODE state to an FETCH_OPERAND state when the auxiliary bit of an instruction indicates more data is needed.

6.4 Functions and Modularity

The code is broken down into small, reusable functions, promoting modularity. Instead of a single monolithic loop, separate functions are created for specific tasks like readSensors(), decodeInstruction(), and executeInstruction(). This approach makes the code easier to read, debug, and maintain.

6.5 Pointers

Pointers are a fundamental concept in C that allows for direct memory manipulation. They are essential for advanced tasks like passing large data structures to functions by reference, thereby avoiding the performance overhead of copying the entire structure. Pointers are also implicitly used when working with arrays and are crucial for low-level interaction with hardware registers.

References

[1] Arduino. *Home*. `https://www.arduino.cc`. Accessed: August 8, 2025.

[2] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual*. `https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html`. Publication Year: 2024.

[3] The RISC-V Foundation. *The RISC-V Instruction Set Manual*. `https://riscv.org/technical/specifications/`. Publication Year: 2019.

[4] International Organization for Standardization and International Electrotechnical Commission. *ISO/IEC 9899: Programming Languages – C*. Year: 2018.

[5] cppreference.com. *C language*. `https://en.cppreference.com/w/c/language`. Accessed: August 8, 2025.