BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEE 416 (July 2023) **A1**
Microprocessor and Embedded Systems Laboratory

Final Project Report

A 415 Art: Single Cycle Processor

Evaluation Form:

STEP	DESCRIPTION	MAX	SCORE
1	Report (Format, Reference)	10	
2	Design Method and Complete Design (Hardware Implementation)	20	
3	Video Demonstration	10	
4	Novelty of Design	20	
5	Project Management and Cost Analysis	15	
6	Considerations to Public Health and Safety, Environment and Cultural and Societal Needs	10	
7	Assessment of Societal, Health, Safety, Legal and Cultural issues relevant to the solution	10	
8	Evaluation of the sustainability and impact of designed solution in societal and environmental contexts	10	
9	Individual Contribution	20	
10	Team work	10	
11	Diversity of Team	05	
	TOTAL	150	

Signature of Evaluator: ______________________________

Academic Honesty Statement:

IMPORTANT! Please carefully read and sign the Academic Honesty Statement, below. Type the student ID and Write your name in your own handwriting. *You will not receive credit for this project experiment unless this statement is signed in the presence of your lab instructor.*

"In signing this statement, We hereby certify that the work on this project is our own and that we have not copied the work of any other students (past or present), and cited all relevant sources while completing this project. We understand that if we fail to honor this agreement, We will each receive a score of ZERO for this project and be subject to failure of this course."

Full Name: B.M. Emraj Hossain Towfik
Student ID: 1906013

Full Name: Shadman Saquib
Student ID: 1906020

Full Name: Himel Kundu
Student ID: 1906014

Full Name: Chinmoy Biswas
Student ID: 1906029

Table of Contents

1 Abstract

This project focuses on the implementation of a single-cycle processor on a hardware board which typically includes LED Matrix Panel and ESP32 for implementing the processor's logic gates and registers. The main objective here is to design a compelling wall art that incorporates the Single Cycle Processor in an LED Matrix panel, where the datapaths and control signals for a particular set of instructions is continuously looped.

2 Introduction

In the realm of computer architecture, the implementation of a single-cycle processor on a board project marks a significant milestone. A single-cycle processor is a fundamental design where each instruction is executed in a single clock cycle. The essence of this project lies in the integration of various components onto a single board to create a fully functioning single-cycle processor.
The endeavor involves designing the processor architecture, including the instruction set architecture (ISA), datapath, memory organization and control unit. The ISA defines the set of instructions supported by the processor, while the datapath consists of the interconnection of various functional units such as registers, ALU (Arithmetic Logic Unit) and memory. The control unit generates control signals to coordinate the operations of these units based on the instruction being executed. After implementation, the processor is tested using software programs or testbenches to verify its functionality and performance.

3 Design

3.1 Design Method

The project utilized ESP32 microcontrollers in conjunction with three 64x64 LED Matrix Panels. To facilitate the task, the team employed the AdafruitGFX library obtained from Waveshare Electronics. A systematic approach was adopted, with each datapath being encapsulated within an object-oriented class, enabling enhanced control and automation capabilities.

For the demonstration, two boards were dedicated to showcasing the datapath functionality, while a separate board was allocated for displaying instructions and register values. This partitioning allowed for clear and focused demonstrations, ensuring efficient communication of the project's capabilities.

The ESP32 microcontrollers served as the backbone of the project, orchestrating the interactions between the LED Matrix Panels and facilitating the seamless operation of the demonstration. Through meticulous design and strategic utilization of resources, the project achieved its objectives effectively, showcasing the versatility and potential of the ESP32 platform in implementing complex functionalities with LED displays.

The entire setup was carefully mounted onto sturdy PVC boards, providing a stable foundation for the intricate arrangement of components. Each element was meticulously positioned and secured, ensuring durability and resilience during operation. Additionally, the PVC boards served as an ideal surface for designing and organizing the setup with clear, informative labels. These labels were strategically placed to provide quick and easy reference points for identifying various components, connections, and functionalities. This thoughtful design not only enhanced the aesthetic appeal of the setup but also improved accessibility and usability. With the aid of these well-placed labels, operators could swiftly navigate the system, troubleshoot issues effectively, and optimize performance with confidence. Overall, the combination of PVC mounting and detailed labeling contributed to the professionalism and functionality of the project, facilitating efficient operation and maintenance.

3.2 Circuit Diagram

The entire datapath board was meticulously mapped onto a 2D grid, and a corresponding diagram was drawn. This strategic approach significantly expedited the process of translating the diagram onto the physical board. By aligning the components of the datapath with the grid, the mapping process became more intuitive and efficient. This systematic method ensured accuracy and saved valuable time during the implementation phase. Overall, the utilization of the 2D grid not only facilitated a quicker mapping process but also enhanced the precision and clarity of the final representation on the board.

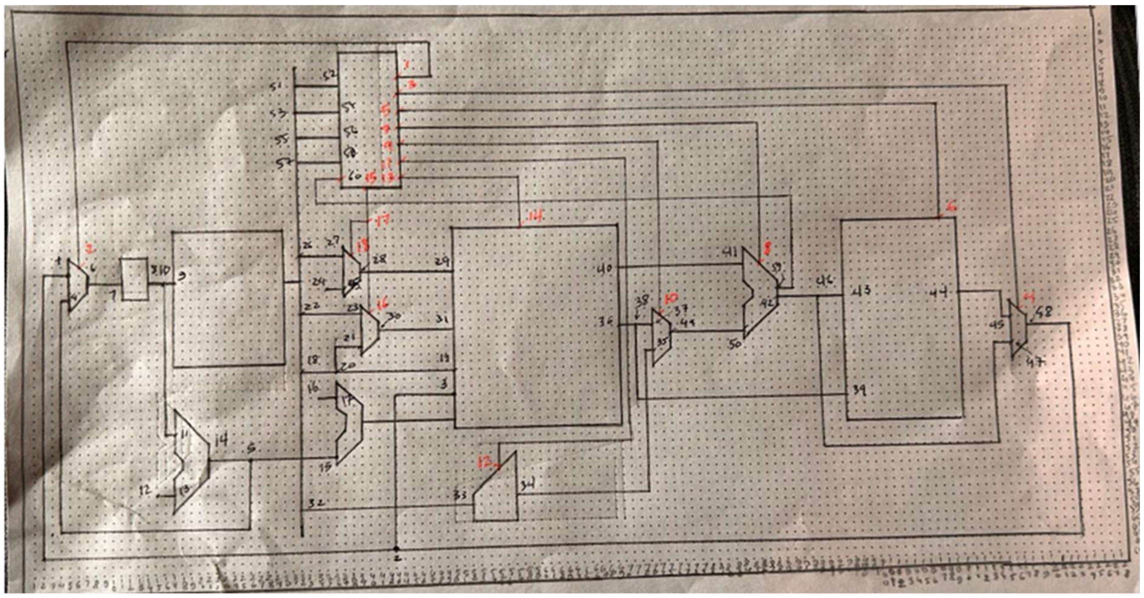
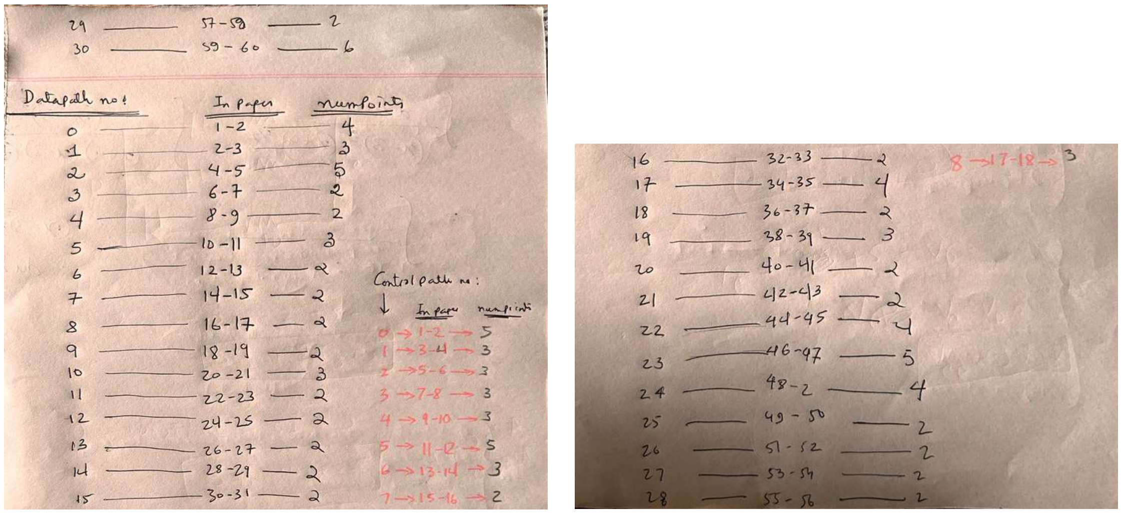


Figure 1: Hand-drawn schematic of the implemented art

To ensure precise management and enhanced control, every datapath and control signal received a unique identification (ID). This individualized labeling system enabled seamless tracking and manipulation of each component within the system. By assigning distinct IDs, the project team could easily reference and address specific datapaths and control signals during operation and troubleshooting phases. This approach streamlined the management process, reducing the risk of confusion or errors that may arise when dealing with multiple interconnected elements. Ultimately, the implementation of unique IDs bolstered the project's efficiency and reliability, facilitating smoother execution and maintenance of the system.



29	57-58	2
30	59-60	6

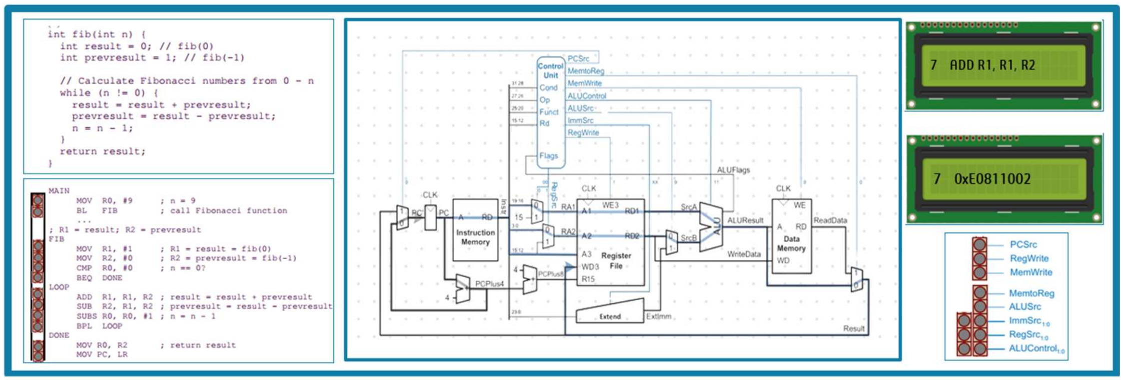
Datapath no:	In paper	numPoints
0	1-2	4
1	2-3	3
2	4-5	5
3	6-7	2
4	8-9	2
5	10-11	3
6	12-13	2
7	14-15	2
8	16-17	2
9	18-19	2
10	20-21	3
11	22-23	2
12	24-25	2
13	26-27	2
14	28-29	2
15	30-31	2
16	32-33	2
17	34-35	4
18	36-37	2
19	38-39	3
20	40-41	2
21	42-43	2
22	44-45	4
23	46-47	5
24	48-2	4
25	49-50	2
26	51-52	2
27	53-54	2
28	55-56	2

Control path no:

↓	In paper	numpoints
0	1-2	5
1	3-4	3
2	5-6	3
3	7-8	3
4	9-10	3
5	11-12	5
6	13-14	3
7	15-16	2
8	17-18	3

Figure 2: IDs of the datapaths and control signals

We embarked on a thorough ideation phase to conceptualize the project's appearance and functionality. Through brainstorming sessions and collaborative discussions, we visualized various aspects, including the overall layout, design elements, and user interface. Sketches, diagrams, and mock-ups were drafted to depict different iterations and explore diverse possibilities. We considered factors such as ergonomics, aesthetics, and practicality to ensure the final design aligned with our objectives and user requirements.



```
int fib(int n) {
  int result = 0; // fib(0)
  int prevresult = 1; // fib(-1)

  // Calculate Fibonacci numbers from 0 - n
  while (n != 0) {
    result = result + prevresult;
    prevresult = result - prevresult;
    n = n - 1;
  }
  return result;
}
```

```
MAIN
      MOV  R0, #9      ; n = 9
      BL   FIB         ; call Fibonacci function
      ...
; R1 = result; R2 = prevresult
FIB
      MOV  R1, #1      ; R1 = result = fib(0)
      MOV  R2, #0      ; R2 = prevresult = fib(-1)
      CMP  R0, #0      ; n == 0?
      BEQ  DONE
LOOP
      ADD  R1, R1, R2  ; result = result + prevresult
      SUB  R2, R1, R2  ; prevresult = result - prevresult
      SUBS R0, R0, #1  ; n = n - 1
      BPL  LOOP
DONE
      MOV R0, R2       ; return result
      MOV PC, LR
```

```
7  ADD R1, R1, R2
```

```
7  0xE0811002
```

Figure 3: Initial Visualized layout of the project

This ideation process enabled us to refine our ideas, identify potential challenges, and devise innovative solutions. By envisioning the project from different perspectives, we gained valuable insights that informed the development process and contributed to the creation of a compelling and user-friendly final product.

3.3 Full Source Code of Firmware

```
#include "ESP32-HUB75-MatrixPanel-I2S-DMA.h"
#include "Picopixel.h"
#include "Font_5x7_practical8pt7b.h"

/*--------------------- MATRIX PANEL CONFIG -------------------------*/
#define PANEL_RES_X 64      // Number of pixels wide of each INDIVIDUAL panel module.
#define PANEL_RES_Y 64     // Number of pixels tall of each INDIVIDUAL panel module.
#define PANEL_CHAIN 3      // Total number of panels chained one to another

MatrixPanel_I2S_DMA *display = nullptr;

// Module configuration
HUB75_I2S_CFG mxconfig(
  PANEL_RES_X,   // module width
  PANEL_RES_Y,   // module height
  PANEL_CHAIN    // Chain length
);

uint16_t myDARK = display->color565(64, 64, 64);
uint16_t myWHITE = display->color565(192, 192, 192);
uint16_t myRED = display->color565(255, 0, 0);
uint16_t myGREEN = display->color565(0, 255, 0);
uint16_t myBLUE = display->color565(0, 0, 255);
uint16_t myYELLOW = display->color565(196, 180, 84);
uint16_t myVIOLET = display->color565(127, 0, 255);

uint16_t colours[5] = { myDARK, myWHITE, myRED, myGREEN, myBLUE };
int DELAY = 1500;
int DELAY1 = 2000;
int n = 0;
void setup() {
  // put your setup code here, to run once:
  delay(1000);
  Serial.begin(115200);
  delay(200);

  Serial.println("...Starting Display");
  //mxconfig.double_buff = true; // Turn of double buffer
  mxconfig.clkphase = false;
  mxconfig.gpio.e = 32;
```

Table: Source Code for the main program

```
// OK, now we can create our matrix object
  display = new MatrixPanel_I2S_DMA(mxconfig);
  display->begin();  // setup display with pins as pre-defined in the library
  display->setFont(&Font_5x7_practical8pt7b);

}

// volatile int R0 = 0, R1 = 0, R2 = 0;

void loop() {
  display->flipDMABuffer(); // not used if double buffering isn't enabled
  delay(25);
  display->clearScreen();

  drawLayout(); // Draw the blocks
  drawDefaultDataFlowPaths(); //Default White Signals
  drawDefaultControlSignals(); //Defaukt Yellow Control Signals

  printVariableNames();

  lightUpInstructions();

  display->setCursor(130, 2);
  display->setTextColor(display->color565(15,15,15));
  display->println("RGB");
}

void printVariableNames() {

  // PC
  display->setCursor(130, 35);
  display->setTextColor(display->color565(0,255,0));
  display->print("PC:0x");

  display->setCursor(130, 45);
  display->print("R0:");

  display->setCursor(160, 45);
  display->print("R1:");

  display->setCursor(145, 55);
  display->print("R2:");
}

// DRAW LAYOUT
void drawLayout() {
  display->drawRect(11,27,4,6,myRED); // PC
  display->drawRect(17,24,14,17,myRED); // IM
  display->drawRect(37,3,8,17,myBLUE); // CU
  display->drawRect(50,24,21,25,myRED); // RF
  display->drawRect(97,24,15,25,myRED); // DM
  drawMux(5,27,5,33,7,31,7,29); // PCSrc
  drawMux(37,26,37,32,39,30,39,28); // RegSrc[0]
  drawMux(39,33,39,39,41,37,41,35); // RegSrc[1]
  drawMux(74,34,74,41,76,39,76,36); // ALUSrc
  drawMux(117,34,117,41,119,39,119,36); // MemToReg
  drawMux(52,56,52,59,57,59,57,51); // ExtImm
  drawALU(17,45,17,50,18,51,18,52,17,53,17,57,21,53,21,49); // PC + 4
  drawALU(36,42,36,45,37,46,37,48,36,49,36,52,39,49,39,45); // PC + 8
  drawALU(85,27,85,31,86,32,86,33,85,34,85,38,89,34,89,31); // ALUControl
  display->drawLine(32,5,32,60,myRED); // Instruction Bus
  drawMux(92,32,92,38,94,36,94,34); //SrcB
}

void drawMux(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) {
  display->drawLine(x1, y1, x2, y2, myRED);
  display->drawLine(x2, y2, x3, y3, myRED);
  display->drawLine(x3, y3, x4, y4, myRED);
  display->drawLine(x4, y4, x1, y1, myRED);
}

void drawALU(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4, int x5, int y5, int x6, int y6, int
x7, int y7, int x8, int y8) {
  display->drawLine(x1, y1, x2, y2, myRED);
  display->drawLine(x2, y2, x3, y3, myRED);
  display->drawLine(x3, y3, x4, y4, myRED);
  display->drawLine(x4, y4, x5, y5, myRED);
  display->drawLine(x5, y5, x6, y6, myRED);
  display->drawLine(x6, y6, x7, y7, myRED);
  display->drawLine(x7, y7, x8, y8, myRED);
  display->drawLine(x8, y8, x1, y1, myRED);
}

// DRAW DATAPATH
struct Point {
  int x, y;
};

struct Line {
  Point p1, p2;
};

class DataPath {
    //
  public:
    Line* lines = nullptr;
    int numPoints;
    uint16_t COLOR;

    DataPath(){

    };
    DataPath(int numPoints) {
      this->numPoints = numPoints;
      this->lines = new Line[numPoints-1];
      this->COLOR = myWHITE;
    }
    ~DataPath() { delete [] lines; }

    // Copy Constructor
    DataPath(DataPath const& copy) {
        numPoints  = copy.numPoints;
        COLOR = copy.COLOR;
        lines = new Line[copy.numPoints-1];
        std::copy(&copy.lines[0],&copy.lines[copy.numPoints-1],lines);
    }

    // "=" Operator overloading
    DataPath& operator=(DataPath rhs) {
        rhs.swap(*this);
        return *this;
    }

    void swap(DataPath& s) noexcept {
        using std::swap;
        swap(this->lines,s.lines);
        swap(this->numPoints,s.numPoints);
        swap(this->COLOR,s.COLOR);
    }

    void makeDataPath(Point* pointsForDataPath){
      for(int i = 0; i < this->numPoints-1; i++) {
        if(pointsForDataPath[i].x == -1) break;
        this->lines[i].p1 = pointsForDataPath[i];
        this->lines[i].p2 = pointsForDataPath[i+1];
        display->drawLine(this->lines[i].p1.x, this->lines[i].p1.y, this->lines[i].p2.x, this->lines[i].p2.y, 
this->COLOR);
      }
    }

    void setColor(uint16_t COLOR){
      this->COLOR = COLOR;
      for(int i = 0; i < numPoints-1; i++) {
        display->drawLine(this->lines[i].p1.x, this->lines[i].p1.y, this->lines[i].p2.x, this->lines[i].p2.y, 
COLOR);
      }
    }
};

const int MAX_POINTS = 7;
const int totalDataPaths = 35;
DataPath* dataPaths = new DataPath[totalDataPaths];

// Data for DataPaths
int numpoints[totalDataPaths] = {4,3,5,2,2,3,2,2,2,2,3,2,2,2,2,2,2,4,2,3,2,2,4,5,4,2,2,2,2,2,6,2,5,2,2};
Point pointsForDataPaths[totalDataPaths][MAX_POINTS] = {
  {{.x=5,.y=29}, {.x=2,.y=29}, {.x=2,.y=62}, {.x=43,.y=62}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 0
  {{.x=43,.y=62}, {.x=43,.y=44}, {.x=50,.y=44}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 1
  {{.x=5,.y=32}, {.x=4,.y=32}, {.x=4,.y=59}, {.x=26,.y=59}, {.x=26,.y=51}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 2
  {{.x=7,.y=30}, {.x=11,.y=30}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 3
  {{.x=14,.y=30}, {.x=17,.y=30}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 4
  {{.x=16,.y=30}, {.x=16,.y=48}, {.x=17,.y=48}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 5
  {{.x=15,.y=55}, {.x=17,.y=55}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 6
  {{.x=21,.y=51}, {.x=36,.y=51}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 7
  {{.x=34,.y=44}, {.x=36,.y=44}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 8
  {{.x=32,.y=41}, {.x=50,.y=41}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 9
  {{.x=36,.y=41}, {.x=36,.y=38}, {.x=39,.y=38}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 10
  {{.x=32,.y=34}, {.x=39,.y=34}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 11
  {{.x=35,.y=31}, {.x=37,.y=31}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 12
  {{.x=32,.y=27}, {.x=37,.y=27}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 13
  {{.x=39,.y=29}, {.x=50,.y=29}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 14
  {{.x=41,.y=36}, {.x=50,.y=36}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 15
  {{.x=32,.y=57}, {.x=52,.y=57}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 16
  {{.x=57,.y=56}, {.x=73,.y=56}, {.x=73,.y=39}, {.x=74,.y=39}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 17
  {{.x=70,.y=36}, {.x=74,.y=36}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 18
  {{.x=72,.y=36}, {.x=72,.y=45}, {.x=97,.y=45}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 19
  {{.x=70,.y=29}, {.x=85,.y=29}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 20
  {{.x=89,.y=33}, {.x=92,.y=33}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 21
**
  {{.x=111,.y=33}, {.x=116,.y=33}, {.x=116,.y=36}, {.x=117,.y=36}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, //
22
  {{.x=96,.y=35}, {.x=96,.y=51}, {.x=115,.y=51}, {.x=115,.y=39}, {.x=117,.y=39}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, //
23
  {{.x=119,.y=37}, {.x=124,.y=37}, {.x=124,.y=62}, {.x=43,.y=62}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, //
24
  {{.x=76,.y=37}, {.x=85,.y=37}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 25
  {{.x=32,.y=7}, {.x=37,.y=7}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 26
  {{.x=32,.y=10}, {.x=37,.y=10}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 27
  {{.x=32,.y=13}, {.x=37,.y=13}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 28
  {{.x=32,.y=16}, {.x=37,.y=16}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 29
  {{.x=89,.y=32}, {.x=91,.y=32}, {.x=91,.y=21}, {.x=34,.y=21}, {.x=34,.y=18}, {.x=37,.y=18}, {.x=-1,.y=-1}}, // 30
  {{.x=94,.y=35}, {.x=97,.y=35}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 31
** MuxToMem
  {{.x=92,.y=37}, {.x=89,.y=37},  {.x=89,.y=41}, {.x=81,.y=41}, {.x=81,.y=37}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 32
** SrcB
  {{.x=39,.y=47}, {.x=50,.y=47}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 33
** R15
  {{.x=30,.y=30}, {.x=32,.y=30}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 34
** InstMem2Bus
};

void drawDefaultDataFlowPaths() {
  for(int i = 0; i < totalDataPaths; i++) {
    DataPath currDataPath = DataPath(numpoints[i]);
    dataPaths[i] = currDataPath;
    dataPaths[i].makeDataPath(pointsForDataPaths[i]);
  }
}

// DRAW CONTROL SIGNALS

const int totalControlSignals = 12;
DataPath* control_signals = new DataPath[totalControlSignals];

int numpoints2[totalControlSignals] = {5,3,3,3,3,5,3,2,3,3,5,5};
Point pointsForControlSignals[totalControlSignals][MAX_POINTS] = {
  {{.x=44,.y=4}, {.x=48,.y=4}, {.x=48,.y=2}, {.x=6,.y=2}, {.x=6,.y=28}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 0
  {{.x=44,.y=6}, {.x=118,.y=6}, {.x=118,.y=35}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 1
  {{.x=44,.y=8}, {.x=109,.y=8}, {.x=109,.y=24}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 2
  {{.x=44,.y=12}, {.x=87,.y=12}, {.x=87,.y=29}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 3
  {{.x=44,.y=14}, {.x=75,.y=14}, {.x=75,.y=35}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 4
  {{.x=44,.y=16}, {.x=71,.y=16}, {.x=71,.y=50}, {.x=55,.y=50}, {.x=55,.y=53}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 5
  {{.x=44,.y=18}, {.x=58,.y=18}, {.x=58,.y=24}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 6
  {{.x=40,.y=19}, {.x=40,.y=34}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 7
  {{.x=40,.y=23}, {.x=38,.y=23}, {.x=38,.y=27}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 8
  {{.x=44,.y=10}, {.x=94,.y=10}, {.x=94,.y=34}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 9
  {{.x=44,.y=16}, {.x=71,.y=16}, {.x=71,.y=50}, {.x=53,.y=50}, {.x=53,.y=55}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 10
  {{.x=44,.y=12}, {.x=87,.y=12}, {.x=87,.y=26}, {.x=89,.y=26}, {.x=89,.y=31}, {.x=-1,.y=-1}, {.x=-1,.y=-1}}, // 11
};

void drawDefaultControlSignals() {
  for(int i = 0; i < totalControlSignals; i++) {
    control_signals[i] = DataPath(numpoints2[i]);
    control_signals[i].makeDataPath(pointsForControlSignals[i]);
    control_signals[i].setColor(myYELLOW);
  }
}

// DRAW DATAPATH FOR EACH INSTRUCTIONS

const int MAX_DATAPATHS = totalDataPaths;
const int numIns = 6;
int dataPathForInstruction[numIns][MAX_DATAPATHS] = {
  {3,4,5,6,7,2,16,17,25,32,23,24,1,-1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,}, // MOV
  {3,4,5,6,7,2,16,17,25,32,23,24,1,-1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,}, // MOV
  {3,4,5,6,7,2,13,14,11,15,9,20,21,31,18,25,23,24,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,}, //ADD
  {3,4,5,6,7,2,13,14,11,15,9,20,21,31,18,25,23,24,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,}, // CMP 1
  {3,4,5,6,7,2,13,14,11,9,10,15,18,19,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,}, // STR
  {3,4,5,6,7,12,16,14,6,17,20,25,21,23,24,0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,},// B

  // {3,4,5,7,2,13,14,11,15,9,20,21,31,18,25,23,24,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,}, // SUB
  // {3,4,5,7,2,13,14,11,15,9,20,21,31,18,25,23,24,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,}, // SUB
  // {12,16,14,17,20,25,21,23,24,0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,},// B
  // {3,4,5,7,2,13,14,11,15,9,20,21,31,18,25,23,24,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,}, // CMP 2
  // {12,16,14,17,20,25,21,23,24,0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,},// BEQ
  // {3,4,5,7,2,13,14,11,15,9,20,21,31,18,25,23,24,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,}, //ADD
  // {3,4,5,7,2,13,14,11,15,9,20,21,31,18,25,23,24,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,}, // SUB
  // {3,4,5,7,2,13,14,11,15,9,20,21,31,18,25,23,24,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,}, // SUB
  // {12,16,14,17,20,25,21,23,24,0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,},// B
  // {3,4,5,7,2,13,14,11,15,9,20,21,31,18,25,23,24,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,}, // CMP 3
  // {12,16,14,17,20,25,21,23,24,0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,},// BEQ

};
const char* INS[numIns] = {"MOV  ", "MOV  ", "ADD  ", "CMP  ", "STRGT", "B    "};
const char* REG[numIns] = {"R0,#4      ", "R1,#7      ", "R2,#0      ", "R3,#0      ", "R2,[R1,#8]",
"DONE       " };

const int MAX_CONTROL_SIGNALS = totalControlSignals;
int controlSignalsForInstruction[numIns][MAX_CONTROL_SIGNALS] = {
  {4,6,9,-1,-1,-1,-1,-1,-1,-1,-1}, // MOV R0, #4;
  {4,6,9,-1,-1,-1,-1,-1,-1,-1,-1}, // MOV R1, #7;
  {6,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1}, // ADD R2, R1, R0;
  {11,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1}, // CMP R2, #0
  {2,10,7,-1,-1,-1,-1,-1,-1,-1,-1}, // STRGT R2, [R1, #8];
  {0,5,8,-1,-1,-1,-1,-1,-1,-1,-1}, // B DONE
};

int R[3][numIns] = {
  {4,4,4,4,4,4},
  {0,7,7,7,7,7},
  {0,0,11,11,11,11}
};

void lightUpInstructions() {
  for(int i = 0; i < numIns; i++) {
    display->setCursor(160, 35);
    display->setTextColor(display->color565(255,0,0));
    display->print(4*i, HEX);

    display->setCursor(145, 45);
    display->print(R[0][i]);

    display->setCursor(175, 45);
    display->print(R[1][i]);

    display->setCursor(160, 55);
    display->print(R[2][i]);

    // display->setCursor(145, 30);

    for(int j = 0; j < MAX_CONTROL_SIGNALS; j++) {
      if(controlSignalsForInstruction[i][j] == -1) break;
      control_signals[controlSignalsForInstruction[i][j]].setColor(myVIOLET);
    }

    for(int j = 0; j < MAX_DATAPATHS; j++) {
      if(dataPathForInstruction[i][j] == -1) break;
      dataPaths[dataPathForInstruction[i][j]].setColor(myGREEN);

      display->setCursor(130, 10);
      display->setTextColor(display->color565(255,0,0));
      display->print(INS[i]);

      display->setCursor(130, 20);
      display->print(REG[i]);
      display->setCursor(130, 30);


      // display->println(R1);

      delay(DELAY);
    }

    // // Clearing
    delay(DELAY1);
    display->setCursor(160, 35);
    display->setTextColor(display->color565(0,0,0));
    display->print(4*i, HEX);
    display->setCursor(145, 45);
    display->print(R[0][i]);

    display->setCursor(175, 45);
    display->print(R[1][i]);

    display->setCursor(160, 55);
    display->print(R[2][i]);

    for(int j = 0; j < MAX_CONTROL_SIGNALS; j++) {
      if(controlSignalsForInstruction[i][j] == -1) break;
      control_signals[controlSignalsForInstruction[i][j]].setColor(myYELLOW);
    }

    for(int j = 0; j < MAX_DATAPATHS; j++) {
      if(dataPathForInstruction[i][j] == -1) break;
      dataPaths[dataPathForInstruction[i][j]].setColor(myWHITE);

      display->setCursor(130, 10);
      display->setTextColor(display->color565(0,0,0));
      display->print(INS[i]);

      display->setCursor(130, 20);
      display->print(REG[i]);
      display->setCursor(130, 30);

      // display->println(R1);

      // delay(200);
    }
    delay(DELAY); //inter-ins
    // n = n + 1;
    // if (n > 1)
    // DELAY = 300;
  }

}

////////////////// CODE ////////////////////////////

/*

MOV R0, #4; {4, 0, 0}
MOV R1, #7; {4, 7, 0}
ADD R2, R1, R0; {4, 7, 11}
CMP R2, #0 {4, 7, 11}
STRGT R2, [R1, #8]; {4,7,11}
B DONE {4,7,11}

*/
```

4 Implementation

4.1 Description

The project implementation started with wiring the LED matrix panel with esp32 microcontroller. We have used total of three panels in our project. The LED panels have onboard two HUB75 headers, one for controller data input, one for output, and support daisy chaining multiple panels. The HUB75 headers contained 16 pins which are labelled as select lines (A, B, C, D, E), RGB data pins (R1, R2, G1, G2, B1, B2), latch pin (LAT), clock pin (CLK) and ground pins (GND). We wired the esp32 with the first LED panel following the connection diagram shown in figure 4.

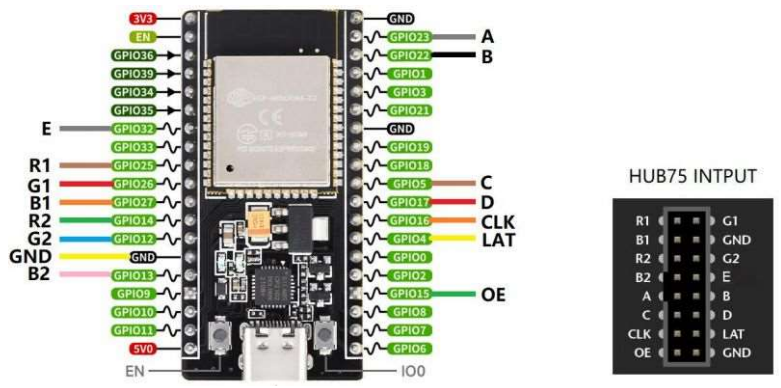


Figure 4: ESP32 to LED Matrix Input Connection Scheme

Next, using the HUB75 connectors provided with the panels, we daisy chained the three panels by connecting first panel's output to the next panel's input and so forth. The 5V 6A DC adapter was used to power up the matrix panels, which provided sufficient power when the panels were lit at their highest brightness. This basically concluded our hardware wiring segment.

Instead of writing code for from scratch, it is more efficient to use existing resources that are publicly available and build upon them. We searched for public esp32 LED matrix interfacing libraries and found *ESP32-HUB75-MatrixPanel-DMA* (author: *mrfaptastic*) worked well with our hardware setup. This library 'out of the box' (mostly) supports HUB75 panels where simple two rows/lines are updated in parallel and, it is referred to as 'two scan' panels. The library uses the DMA functionality provided by the ESP32's 'LCD Mode' for fast data output.

After successfully calibrating the displays using the examples from the ESP32-HUB75-MatrixPanel-DMA library, we began our primary task: constructing the framework for the entire single-cycle architecture. This constituted 70% of our coding task. The single cycle diagram is demonstrated using two panels. In the third panel, we coded to display the currently executing assembly instruction along with values of the program counter and some registers. Figure 3 depicts the project's state upon the completion of the architecture.

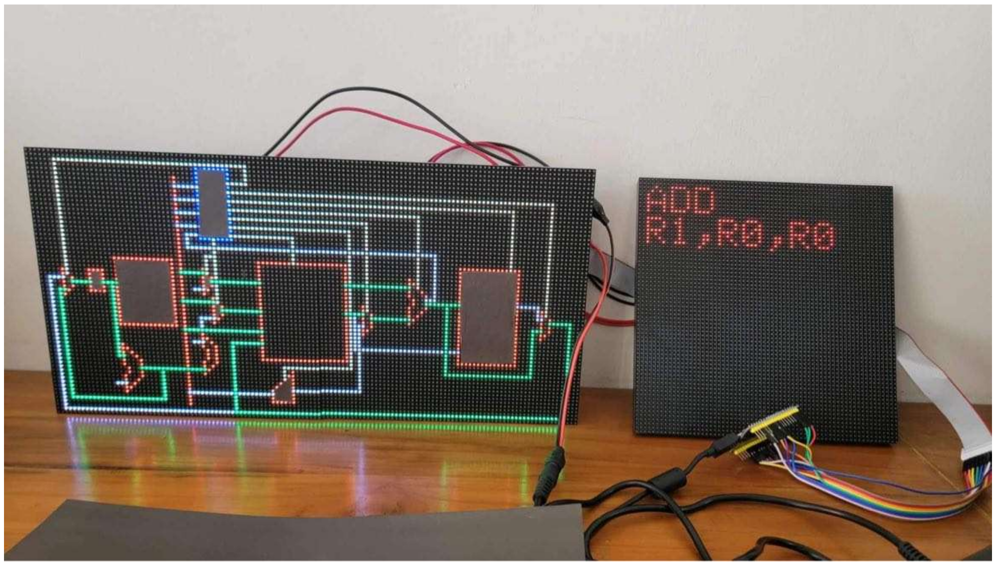


Figure 5: Initial Layout of our project

It is essential that we present our project in a visually captivating manner. We bought a good quality PVC board on top of which the LED matrix panels are laid out. Because of the HUG75 and power connecters sticking out at the back of the panels, we required laser cutting sections of PVC board.

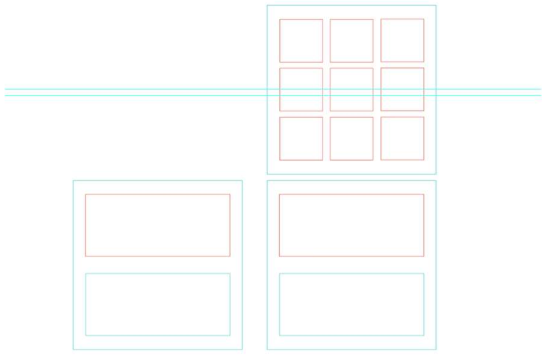


Figure 6: Cutting Schematic of the PVC Board

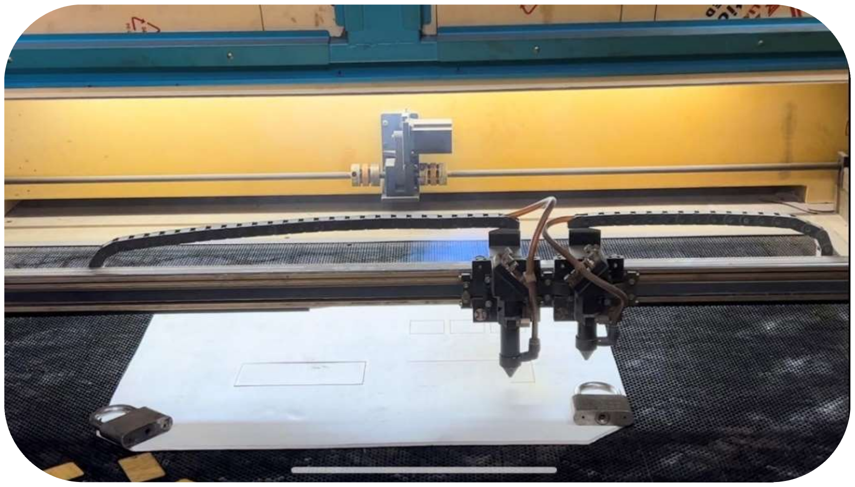


Figure 7: Laser cutting PVC board for accommodating space for panel power connectors

We organized the three panels in a 2x2 grid layout with upper left area empty. In this section, we would include comprehensive assembly instructions and supplementary details such as the color conventions employed to distinguish between active and inactive data paths and control signal paths. Upper right panel is programmed to display register values of interest and the assembly instructions being executed one by one. The lower two panels display the single cycle framework and animates the data paths and control signal paths as per the assembly instructions.

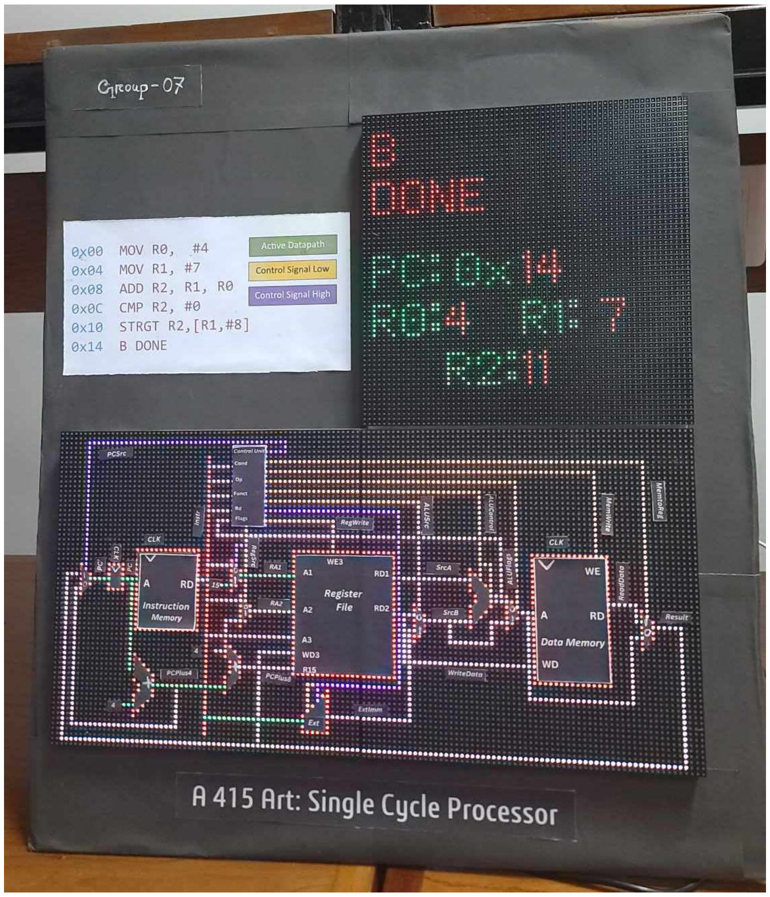

```
0x00  MOV R0,  #4
0x04  MOV R1, #7
0x08  ADD R2, R1, R0
0x0C  CMP R2, #0
0x10  STRGT R2,[R1,#8]
0x14  B DONE
```

Figure 8: Final Layout of our project

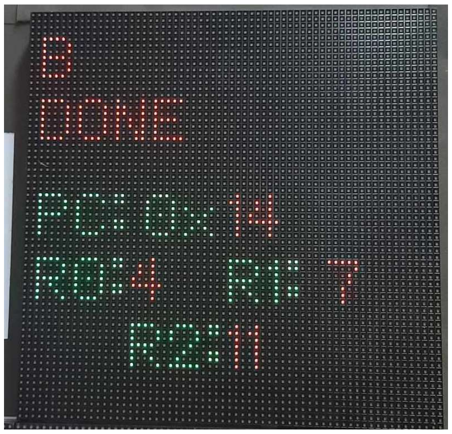
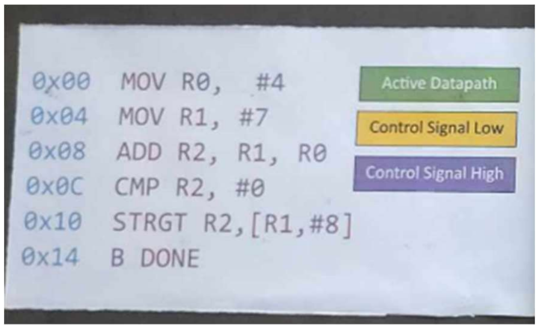


Figure 9: Instruction Board



```
0x00  MOV R0,  #4
0x04  MOV R1, #7
0x08  ADD R2, R1, R0
0x0C  CMP R2, #0
0x10  STRGT R2,[R1,#8]
0x14  B DONE
```

Active Datapath
Control Signal Low
Control Signal High
Figure 10: Instructions

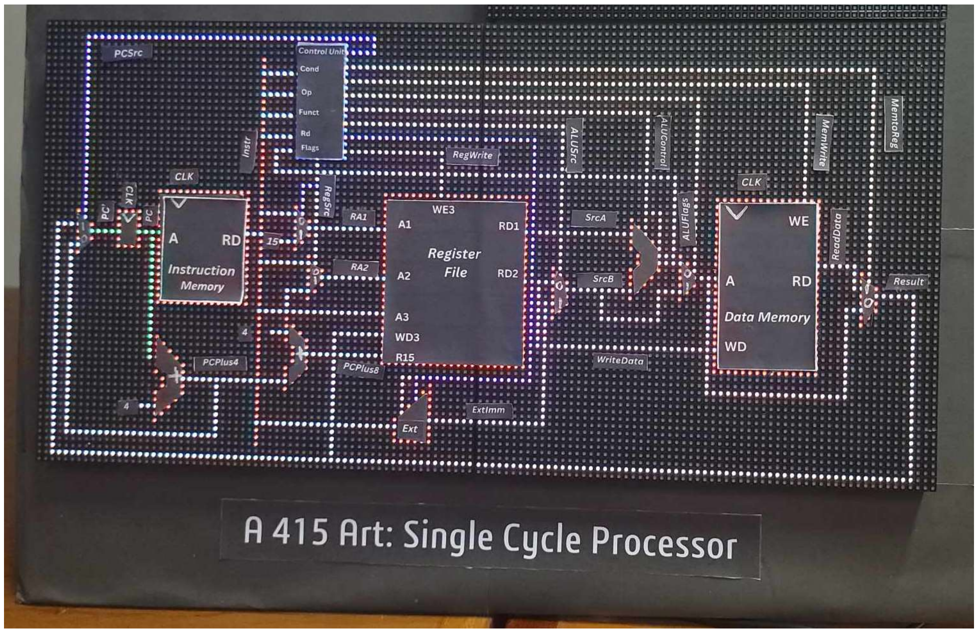


Figure 11: Main board illustrating the single cycle processor

4.2 Results

4.3 Github Link

The following link contains all the necessary source code and documentation of our project . https://github.com/chinime/Single-Cycle-Processor-Wall-Art

4.4 YouTube Link

The following link provides a video demonstration of the working of our project. https://youtu.be/VUKxlMp3Fqs?feature=shared

5 Design Analysis and Evaluation

5.1 Novelty

- **Simultaneous Instruction Processing**: In a single-cycle processor, different stages of instruction execution (such as instruction fetch, decode, execute, and write-back) occur concurrently within a single clock cycle. This simultaneous processing is visualized by LED matrix in our project.

- **Visualized Control Logic**: Single-cycle processors typically have less complex control logic compared to multi-cycle processors since each instruction follows the same fixed execution path. This simplification facilitates easier verification, debugging, and maintenance of the processor design. Even it can display datapath for the MOV command, which need an extra multiplexer.

- **Customization and Optimization**: Single-cycle processor designs offer flexibility for customization and optimization based on specific application requirements or constraints. Designers can extend to multiple instructions or tailor the processor's architecture, instruction set, and functionality to suit particular use cases, making it a versatile platform for various embedded systems and small-scale computing applications.

5.2 Project Management and Cost Analysis

5.2.1 Bill of Materials

Components	Price
RGB Panels (3)	2120*3
ESP32	715
Power Adapter	2230
Stationaries	400
Frame	500
Total	10205

5.3 Practical Considerations of the Design to Address Public Health and Safety, Environment, Cultural, and Societal Needs

- We initially thought of using an Arduino Mega as it had more pins, but we couldn’t as the speed of the Arduino Mega was not compatible with the refresh rate of the LED matrix.
- As we had to switch to ESP32, the pin number was reduced, and so we could not interface the LCD and the control LEDs.
- To remedy this, we used another LED Matrix to display the instructions, and decided to light up the control signals in the Main Board.

- Once the single-cycle processor project reaches the end of its lifecycle, proper disposal or recycling is essential to prevent environmental contamination.
- We used paper coating and eco-friendly materials for packaging our project.

5.4 Assessment of the Impact of the Project on Societal, Health, Safety, Legal and Cultural Issues

5.4.1 Assessment of Societal Issues

This project of single-cycle processors often require knowledge of computer architecture, digital logic design, and programming. By engaging in such projects, individuals can enhance their technical skills, contributing to their personal development and potentially opening up opportunities for employment in the technology sector. This can have a positive impact on the cultural and social fabric by empowering individuals with relevant skills for the digital age.

5.4.2 Assessment of Health and Safety Issues

The impact of a our project on health and safety is generally minimal, as single cycle processor is low-risk electronic device that does not pose significant health or safety hazards.

Before using or demonstrating this project, we thoroughly tested it to ensure it functions correctly and safely. This includes checking for any potential software bugs or hardware malfunctions that could pose a risk to users. Clear and concise user manuals or instructions to be provided for this project, including safety information and guidelines for safe use.

5.4.3 Assessment of Legal Issues

Assessing the legal issues of a single cycle processor project involves considering various aspects related to intellectual property, contracts, liability, and compliance with relevant laws and regulations.

It was ensured that our design do not infringe on existing patents. Cautious actions were taken of using any trademarks or branding that might confuse consumers or infringe on the rights of others.

5.4.4 Assessment of Cultural Issues

Given the interconnected nature of the modern world, single-cycle processor project facilitate cultural exchange and global collaboration. We can collaborate on shared projects, exchanging ideas and perspectives. This cross-cultural interaction promotes mutual understanding and appreciation, contributing to a more interconnected and culturally rich global community.

5.5 Evaluation of the Sustainability the and Impact of the Designed Solution in the Societal and Environmental Contexts

5.5.1 Evaluation of Sustainability

This project is sustainable in any learning environment.As the whole board is mapped, new instruction blocks can be added.The LED Matrices are interfaced using HUB75, and so a lot of them can be chained together for future exploration.The long-term effects of the single-cycle processor on the environment, society, and the economy was considered and it has potential to contribute to a sustainable future or exacerbate existing challenges.

5.5.2 Evaluation of Impact of Design in Societal Context

Creating a sustainable and environmentally friendly SCP project involves considering various aspects of the product's lifecycle, from raw material extraction to manufacturing, use and disposal.

1. **Materials Selection:**
    - o Choosing materials with a lower environmental footprint, no plastics or biodegradable materials, for the project's components.
    - o Prioritization materials that are non-toxic and easily recyclable at the end of the product's life.

2. **Energy Efficiency:**
    - Designing the single cycle processor to be energy-efficient by using low-power components and optimizing the power management system.
    - Using minimal and eco-friendly packaging to reduce waste.

3. **Environmental Certification:**
    - Considering investing in carbon offset projects to compensate for the emissions generated during the project's production and use.
    - Used Arduino source to avoid lead-acid battery that causes toxicity to environment after wasting it.

5.5.3 Evaluation of Impact of Design in Environmental Context

6 Reflection on Individual and Team work

6.1 Individual Contribution of Each Member

Member ID	Contribution
1906013	Hooking everything up
1906014	Components selection and research
1906020	Front end Design
1906029	Back End Implementation

6.2 Mode of TeamWork

We began by gathering our team members and collaboratively plan the project,defined the project scope, objectives, and timeline. Then assigned roles and responsibilities based on each team member's strengths and skills.
The project was divided into smaller tasks or user stories and allocate them to team members based on their expertise. Regular team meetings were held to discuss progress, challenges, and solutions. We maintained clear and up-to-date documentation for your calculator project. Utilize communication tools like Whatsapp, Microsoft Teams to facilitate discussions, share updates and track progress.

6.3 Log Book of Project Impelementation

Week	**Milestone achieved**
6	Component Selection and Research
7	Component Buying
8	Hooking everything up in the Arduino Mega
9	Failing and debugging
10	Further component buy
11	Coding
12	Demonstrating
13	Decorating and panel fixing

7 References

*[1] Sarah Harris, David Harris; Digital Design and Computer Architecture, **ARM Edition***
[2] https://www.waveshare.com/wiki/RGB-Matrix-P3-64x64?fbclid=IwAR26jVtsXXGJiy-9WncQMiu-R42p7zqwcKAQrFhLlblqjcTRpCjMmaHK8_Q
[3] https://github.com/topics/single-cycle-processor?o=asc&s=forks