Chittagong University of Engineering and Technology

SAP-1 Microprocessor Logisim Implementation

Documentation

Submitted By:

Avishek Chowdhury
ID: 2008019
Course Code: ETE 404
Date: 06-10-2025

Department of Electronics and Telecommunication Engineering

Submitted To:

Arif Istiaque
Course Instructor
VLSI Technology Sessional

Abstract

This report documents the design, implementation, and testing of a fully functional SAP-1 (Simple-As-Possible) microprocessor using Logisim Evolution. The project includes a hardwired control unit, ROM-based boot-loader, custom assembler, and implements the complete fetch-decode-execute cycle. A unique JMP instruction was implemented as the required distinctive feature. The system successfully executes assembly programs including arithmetic operations and control flow instructions.

Contents

1 Introduction

1.1 Project Overview

The SAP-1 microprocessor is an educational 8-bit CPU architecture designed to demonstrate fundamental computer organization principles. This implementation enhances the basic SAP-1 design with automatic program loading, a web-based assembler, and extended instruction set capabilities.

1.2 Objectives

- Design and implement a fully operational SAP-1 microprocessor in Logisim Evolution
- Develop a hardwired control unit for automated instruction execution
- Implement a ROM-based bootloader for automatic program loading
- Create a unique instruction (JMP) for program control flow
- Build a web-based assembler for program development
- Demonstrate complete fetch-decode-execute cycle operation

1.3 Project Scope

The project encompasses:

- Complete SAP-1 architecture with all core components
- Dual-mode operation (Manual and Automatic)
- 16-byte memory system with auto-loading capability
- Custom instruction set with JMP implementation
- Web-based development tools

1.4 Project Links

- **GitHub Repository:**
  `https://github.com/AvishekChy/sap-1-cpu-microprocessor-logisim-evolution`
- **Assembler Web App:** `https://avishek-sap1-compiler-app.streamlit.app/`
- **Video Demonstration:** `https://youtu.be/wIqGsQ7BMSo`

2 SAP-1 Architecture Design

2.1 System Overview

The SAP-1 microprocessor follows the classic von Neumann architecture with a unified bus structure. The system operates on 8-bit data with 4-bit addressing, supporting 16 memory locations.

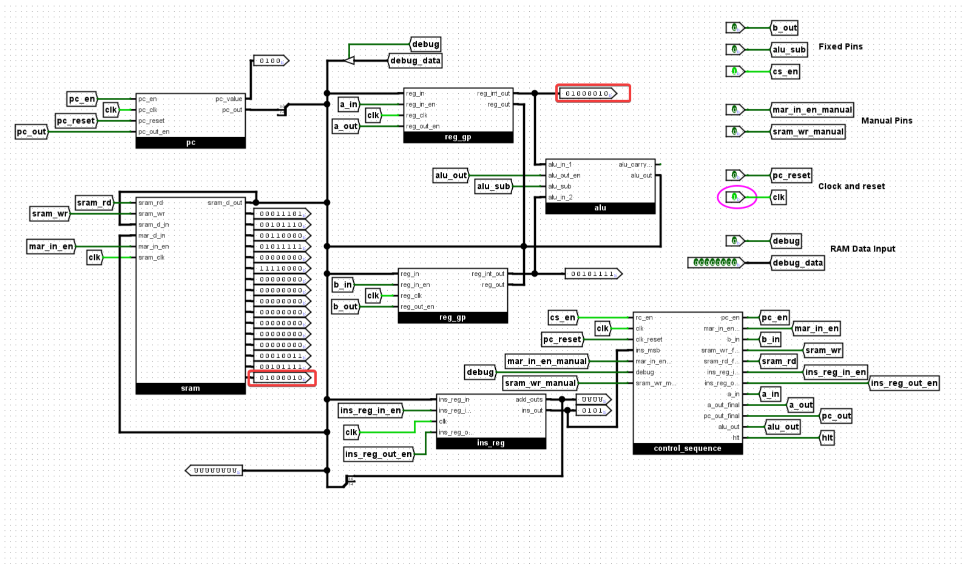


Figure 1: Complete SAP-1 Microprocessor System

2.2 Core Components

2.2.1 Program Counter (PC)

The 4-bit program counter tracks the memory address of the next instruction to execute. It increments automatically after each instruction fetch.

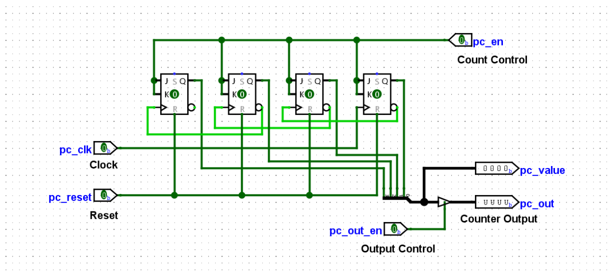


Figure 2: 4-bit Program Counter Circuit

2.2.2 Memory Address Register (MAR)

The 4-bit MAR holds the current memory address being accessed for read or write operations.

2.2.3 Random Access Memory (RAM)

The system includes 16 bytes of static RAM, organized as 8-bit words with 4-bit addressing.

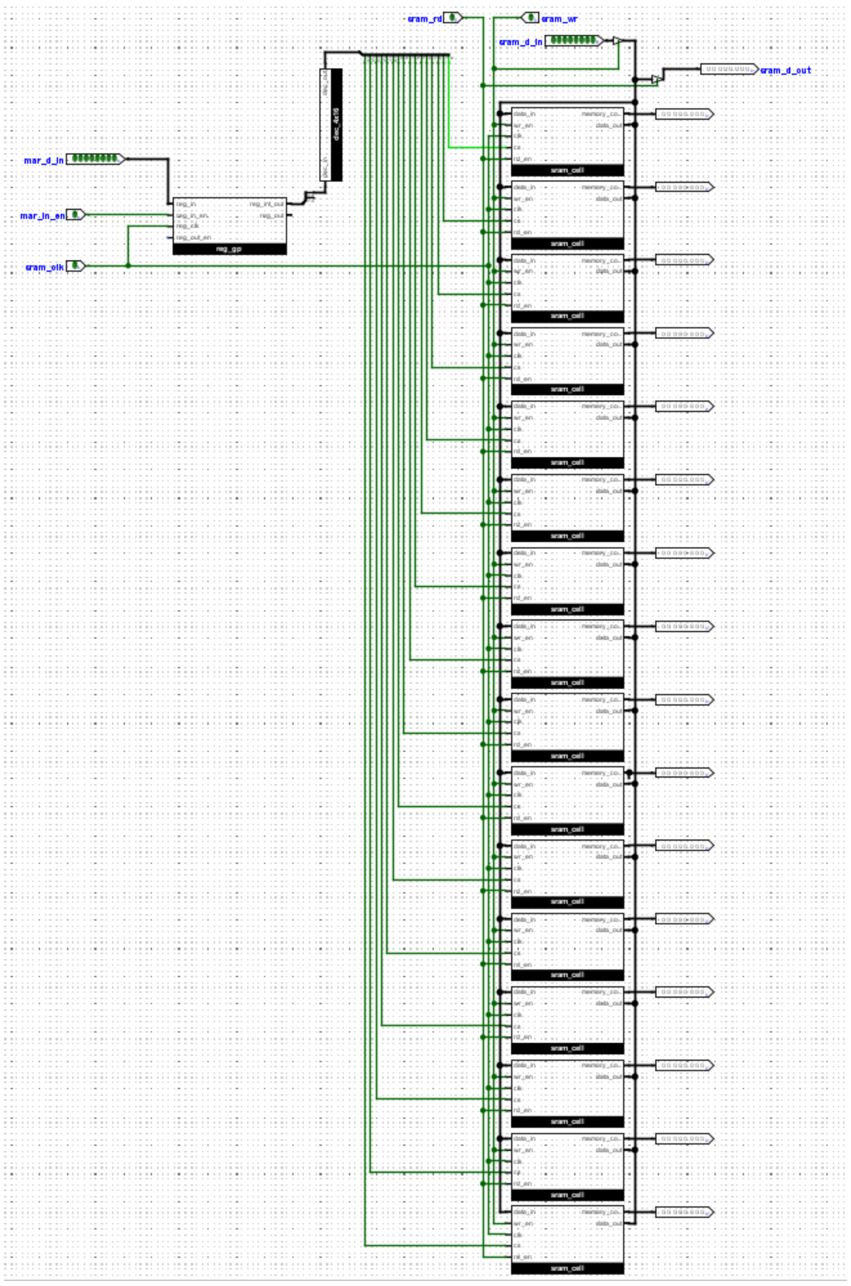


Figure 3: 16-byte RAM Module

2.2.4 Instruction Register (IR)

The 8-bit instruction register stores the current instruction, divided into 4-bit opcode and 4-bit operand fields.

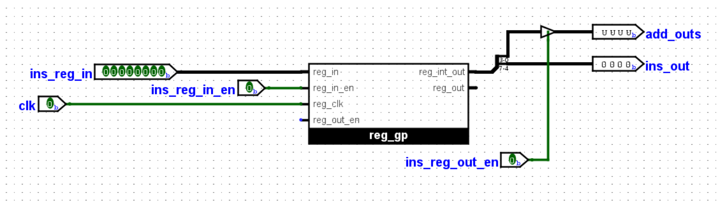


Figure 4: Instruction Register

2.2.5 General Purpose Registers

- **Register A (Accumulator):** Primary register for arithmetic operations and data manipulation
- **Register B:** Secondary register for holding operands

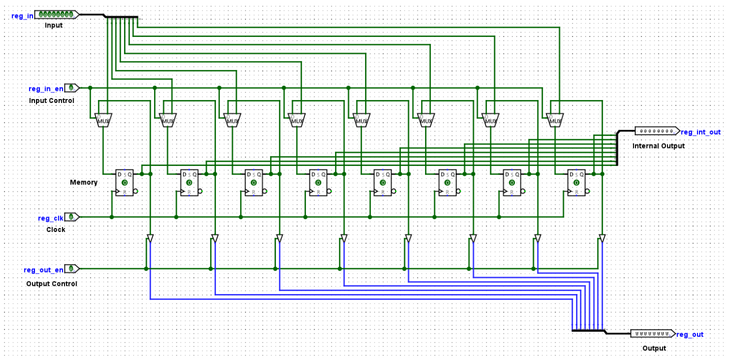


Figure 5: General Purpose Registers A and B

2.2.6 Arithmetic Logic Unit (ALU)

The 8-bit ALU performs addition and subtraction operations.

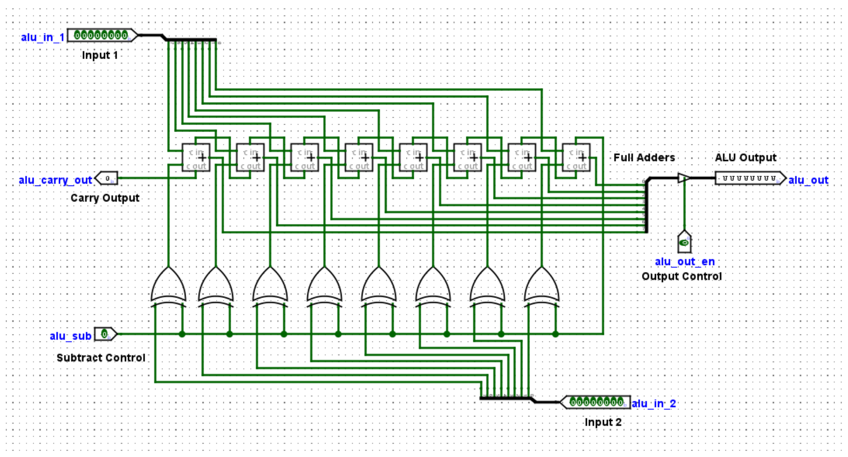


Figure 6: 8-bit Arithmetic Logic Unit

3 Control Sequencer Implementation

3.1 Hardwired Control Unit

The control unit uses combinational logic to generate control signals based on the current instruction and timing state.

3.2 Ring Counter (RC)

A 3-bit ring counter generates six timing states (T1-T6) that coordinate the fetch-decode-execute cycle.

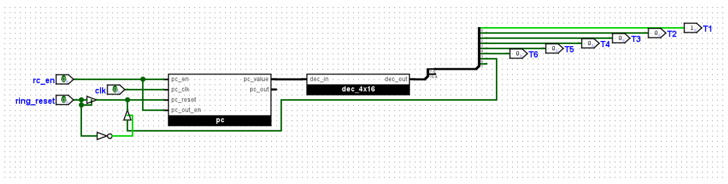


Figure 7: 3-bit Ring Counter

3.3 Instruction Decoder

A 4-to-16 decoder processes the opcode to activate specific instruction lines.

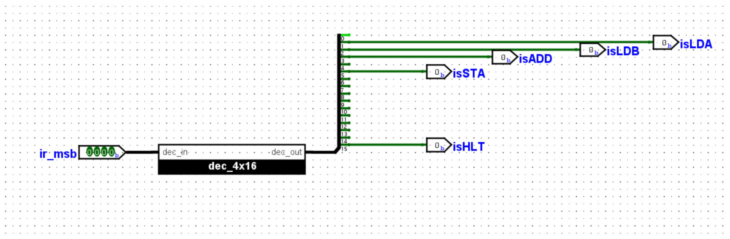


Figure 8: Instruction Decoder

3.4 Control Signal Generation

The control matrix combines timing states and decoded instructions to produce control signals using Boolean logic:

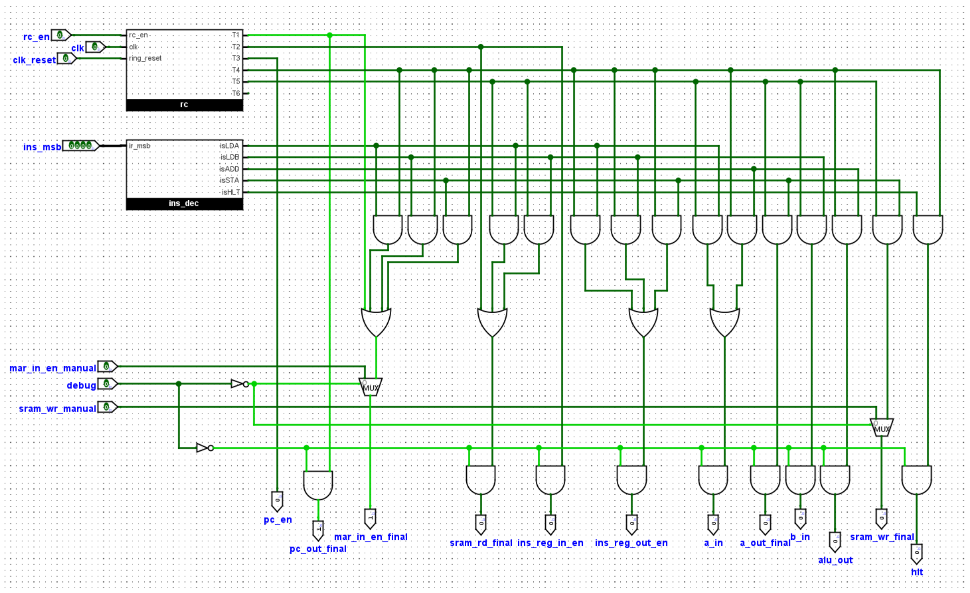


Figure 9: Control Signal Generation Logic

3.5 Fetch-Decode-Execute Cycle

3.5.1 Fetch Phase (T1-T3)

1. **T1:** PC → MAR (Program Counter outputs address to Memory Address Register)
2. **T2:** RAM → IR (Memory reads instruction to Instruction Register)
3. **T3:** PC++ (Program Counter increments)

3.5.2 Decode Phase (T3-T4)

1. Instruction Register opcode is decoded
2. Appropriate instruction line is activated

3.5.3 Execute Phase (T4-T6)

1. Control signals execute the specific instruction
2. ALU operations, data transfers, or memory accesses occur

4 RAM Auto-Loader System

4.1 ROM-Based Bootloader

The bootloader system automatically transfers program code from ROM to RAM, eliminating manual data entry.

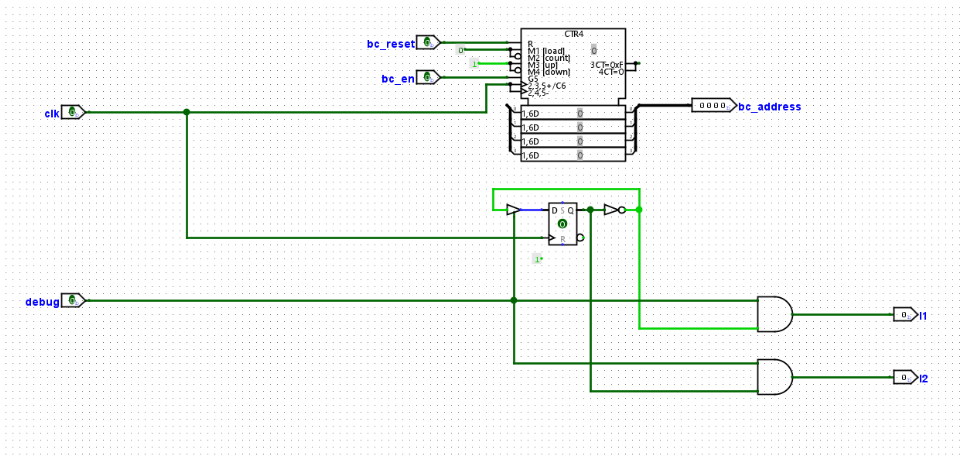


Figure 10: Instruction Loader Circuit

4.2 Operation Modes

- **Debug Mode:** Manual control for testing and initialization
- **Auto Mode:** Automatic program execution with bootloader

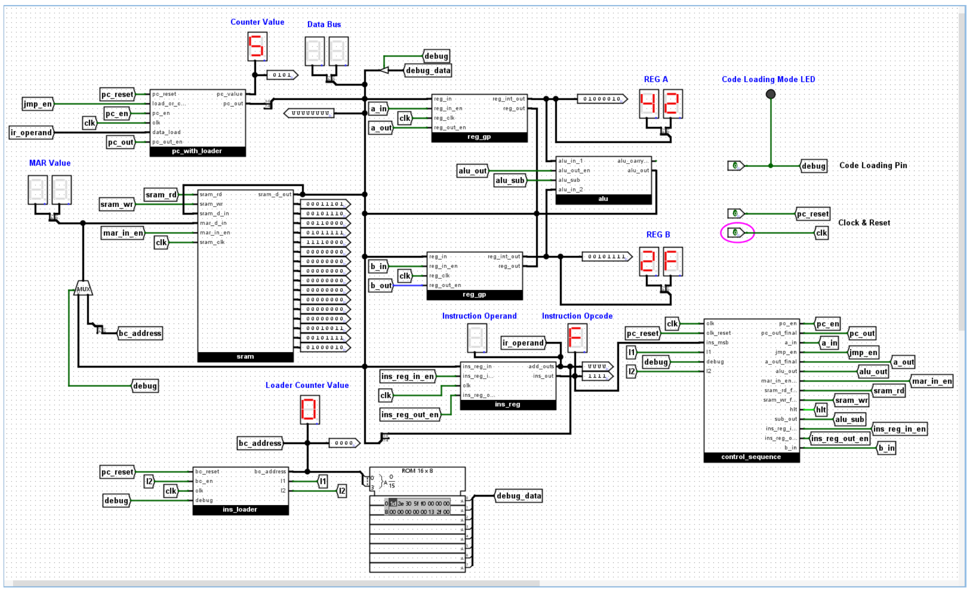


Figure 11: Auto Mode Control Unit Overview

5 Unique Feature: JMP Instruction

5.1 Implementation

The JMP (Jump) instruction was implemented as the unique feature, enabling program control flow and looping capabilities.

5.2 Instruction Format

- **Opcode:** 0110
- **Operand:** 4-bit target address
- **Binary:** 0110 XXXX (where XXXX is the jump address)
- **Hex:** 6X

5.3 Operation

When executed, the JMP instruction loads the specified address into the Program Counter, causing the program to continue execution from the new location.

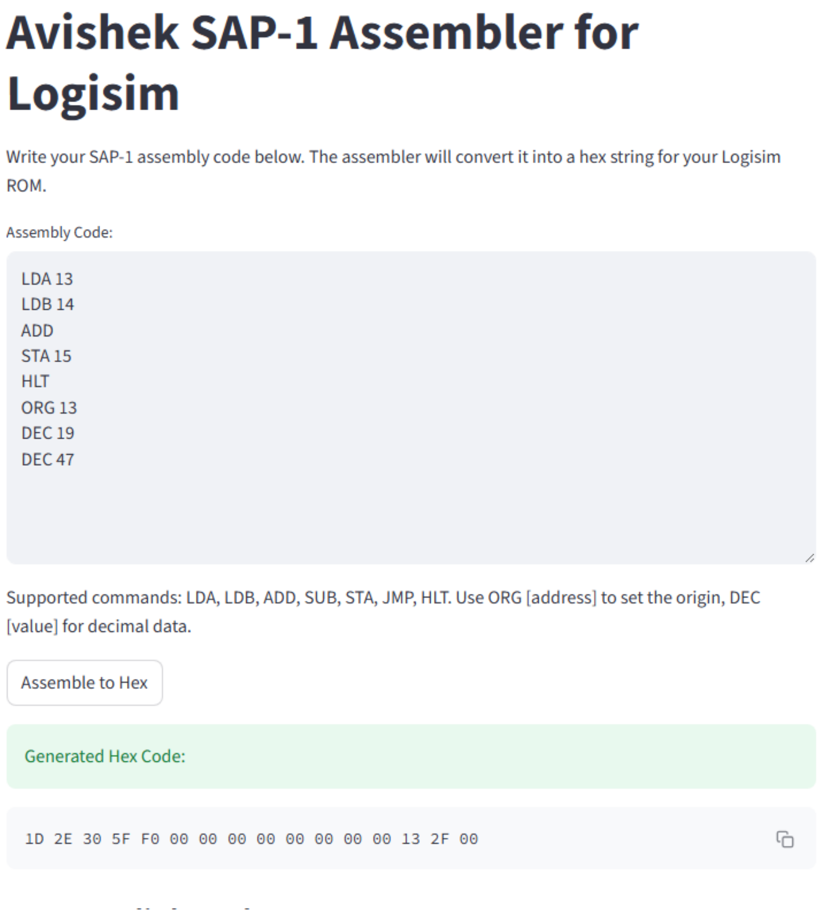
5.4 Control Signals

```
jmp_en = (T3 AND isJMP) AND cpu_mode AND (NOT I2)
```

6 Compiler/Assembler Implementation

6.1 Web-Based Assembler

A Streamlit web application was developed to convert assembly code to machine-readable hex format.



Avishek SAP-1 Assembler for Logisim

Write your SAP-1 assembly code below. The assembler will convert it into a hex string for your Logisim ROM.

Assembly Code:

```
LDA 13
LDB 14
ADD
STA 15
HLT
ORG 13
DEC 19
DEC 47
```

Supported commands: LDA, LDB, ADD, SUB, STA, JMP, HLT. Use ORG [address] to set the origin, DEC [value] for decimal data.

Generated Hex Code:

```
1D 2E 30 5F F0 00 00 00 00 00 00 00 00 13 2F 00
```

Figure 12: Web Assembler - ADD Program

6.2 Supported Instructions

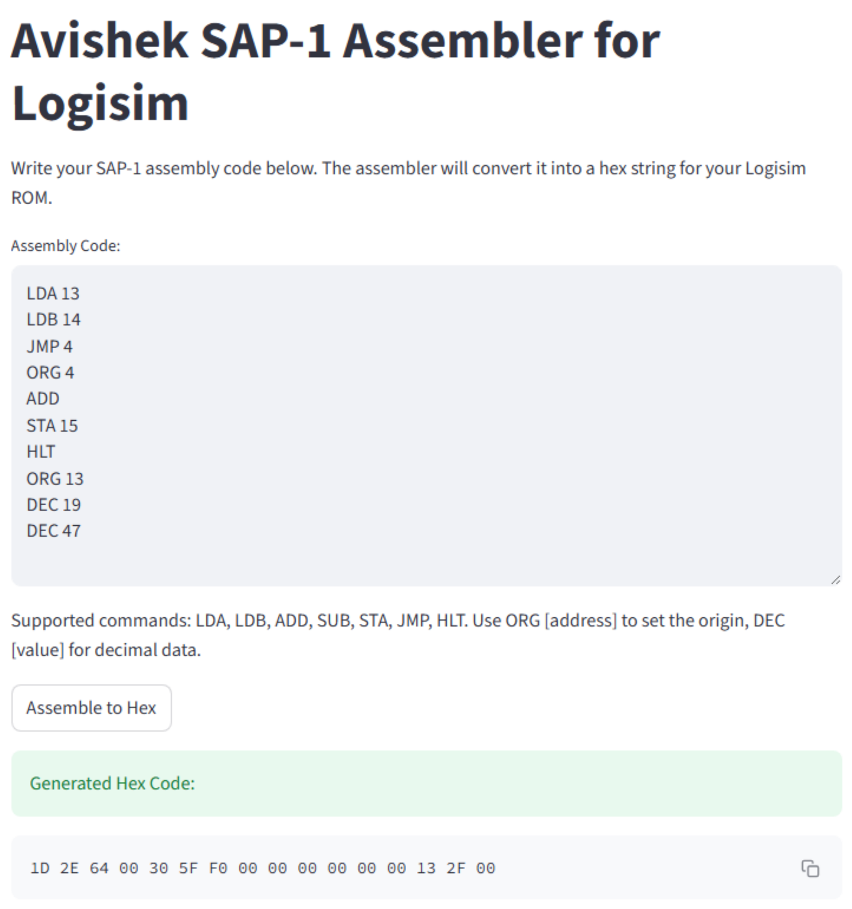
The assembler supports the complete SAP-1 instruction set:

Mnemonic	Opcode	Hex	Description
LDA	0001	1X	Load Register A from memory
LDB	0010	2X	Load Register B from memory
ADD	0011	3X	Add B to A
SUB	0100	4X	Subtract B from A
STA	0101	5X	Store A to memory
JMP	0110	6X	Jump to address
HLT	1111	FX	Halt execution

Table 1: SAP-1 Instruction Set

6.3 Assembly Language Features

- Label support for memory addresses
- Decimal and hexadecimal number support
- ORG directive for address specification
- DEC directive for data declaration



Avishek SAP-1 Assembler for Logisim

Write your SAP-1 assembly code below. The assembler will convert it into a hex string for your Logisim ROM.

Assembly Code:

```
LDA 13
LDB 14
JMP 4
ORG 4
ADD
STA 15
HLT
ORG 13
DEC 19
DEC 47
```

Supported commands: LDA, LDB, ADD, SUB, STA, JMP, HLT. Use ORG [address] to set the origin, DEC [value] for decimal data.

Generated Hex Code:

```
1D 2E 64 00 30 5F F0 00 00 00 00 00 00 13 2F 00
```

Figure 13: Web Assembler - JMP and ADD Program

7 Sample Programs and Testing

7.1 Addition Program

```
LDA 13       ; Load first number from address 13
LDB 14       ; Load second number from address 14
ADD          ; Add B to A
STA 15       ; Store result to address 15
HLT          ; Halt execution
ORG 13       ; Data section starting at address 13
DEC 19       ; First number (Student ID: 19)
DEC 47       ; Second number (Random: 47)
```

Listing 1: Addition Program Assembly Code

7.2 Machine Code Output

```
1D 2E 30 5F F0 00 00 00 00 00 00 00 00 13 2F 00
```

Listing 2: Hex Machine Code for Addition Program

7.3 Program with JMP Instruction

```
LDA 13      ; Load first number
LDB 14      ; Load second number
JMP 4       ; Jump to address 4 (ADD instruction)
ORG 4       ; Code at address 4
ADD         ; Add numbers
STA 15      ; Store result
HLT         ; Halt
ORG 13      ; Data section
DEC 19      ; First number
DEC 47      ; Second number
```

Listing 3: Program with JMP Instruction

7.4 Execution Results

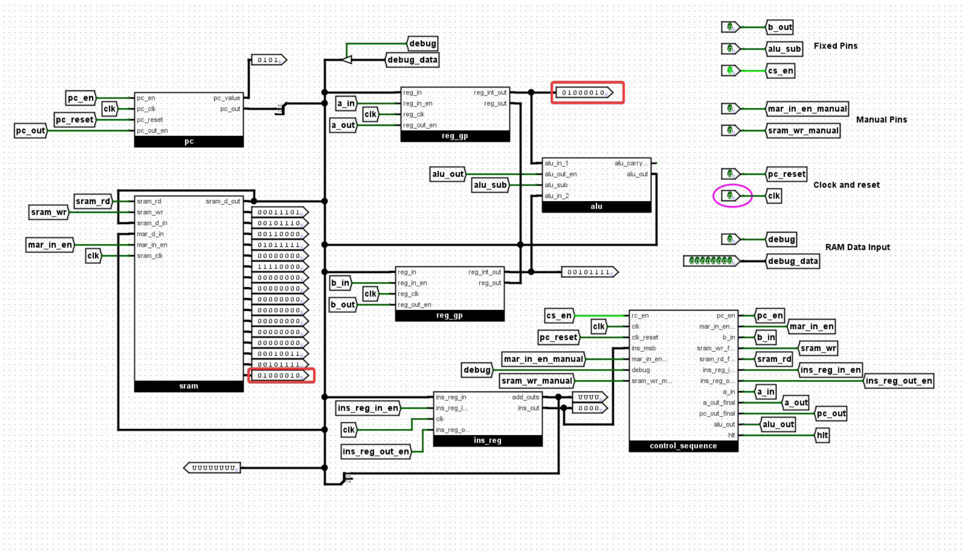


Figure 14: Final Result Display (66 = 19 + 47)

8 Setup and Operation Guide

8.1 Prerequisites

- Logisim Evolution installed
- Web browser for assembler access

- Project circuit files

8.2 Auto Mode Operation

1. Open `sap1_auto_avishek.circ` in Logisim Evolution
2. Set `debug` to OFF (LOW)
3. Pulse `pc_reset` to initialize Program Counter
4. Load hex code into ROM using Edit Contents
5. Turn ON `debug` (HIGH) for bootloader mode
6. Pulse clock to load program from ROM to RAM
7. Turn OFF `debug` (LOW)
8. Start clock to execute program

8.3 Manual Mode Operation

1. Open `sap1_manual_avishek.circ` in Logisim Evolution
2. Set `debug` to ON (HIGH)
3. Manually enter addresses and data using control pins
4. Use manual clock stepping for debugging

9 Challenges and Solutions

9.1 Technical Challenges

- **Control Signal Timing:** Implemented precise state-based control using ring counter
- **Memory Loading:** Developed ROM-based bootloader with mode switching
- **Instruction Decoding:** Used 4-to-16 decoder with combinatorial logic
- **Bus Conflicts:** Implemented tri-state buffers for proper bus management

9.2 Design Solutions

- Dual-mode operation for flexibility (Auto/Manual)
- Modular design approach for easy testing
- Web-based assembler for rapid development
- Comprehensive documentation for reproducibility

10 Conclusion and Future Work

10.1 Project Achievements

- Successfully implemented complete SAP-1 microprocessor
- Developed functional hardwired control unit
- Created automated bootloader system
- Implemented unique JMP instruction
- Built web-based development tools
- Demonstrated full program execution capability

10.2 Future Enhancements

- Expand instruction set with additional operations
- Implement microprogrammed control unit
- Add input/output device support
- Develop advanced debugging features
- Create graphical simulation interface

10.3 Educational Value

This project provides comprehensive understanding of:

- Computer architecture fundamentals
- CPU design principles

- Digital logic implementation
- Control system design
- Software-hardware co-design

References

1. ”Digital Computer Electronics” by Albert Paul Malvino
2. Ben Eater’s SAP-1 implementation guides
3. Logisim Evolution documentation
4. Computer Organization and Design textbooks

Appendix: Complete Circuit Diagrams

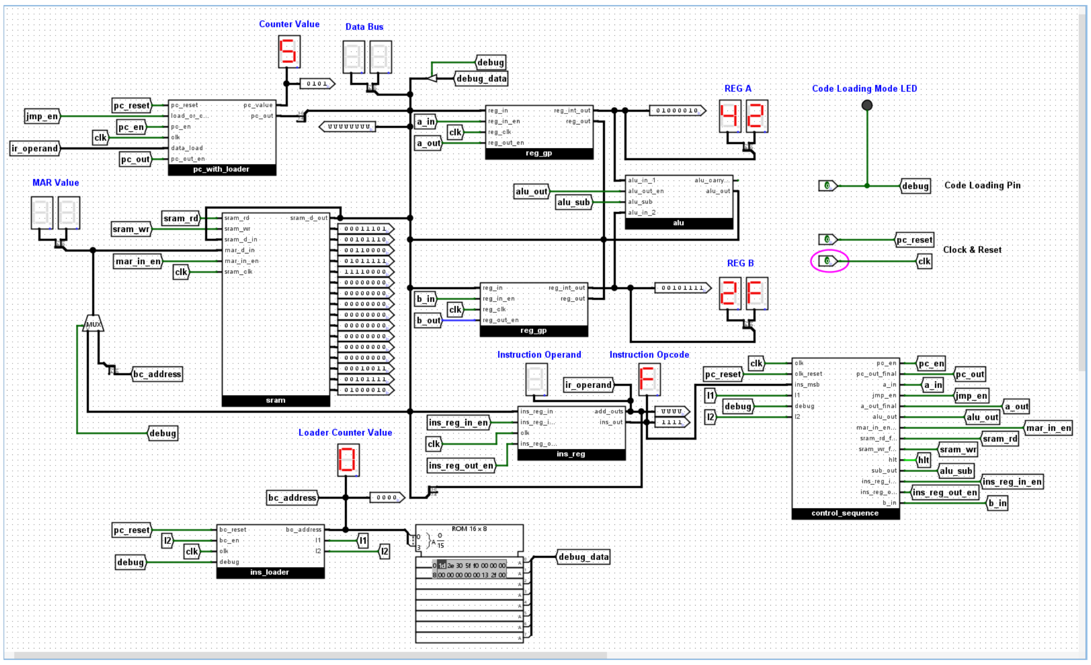


Figure 15: Complete Auto Mode Circuit

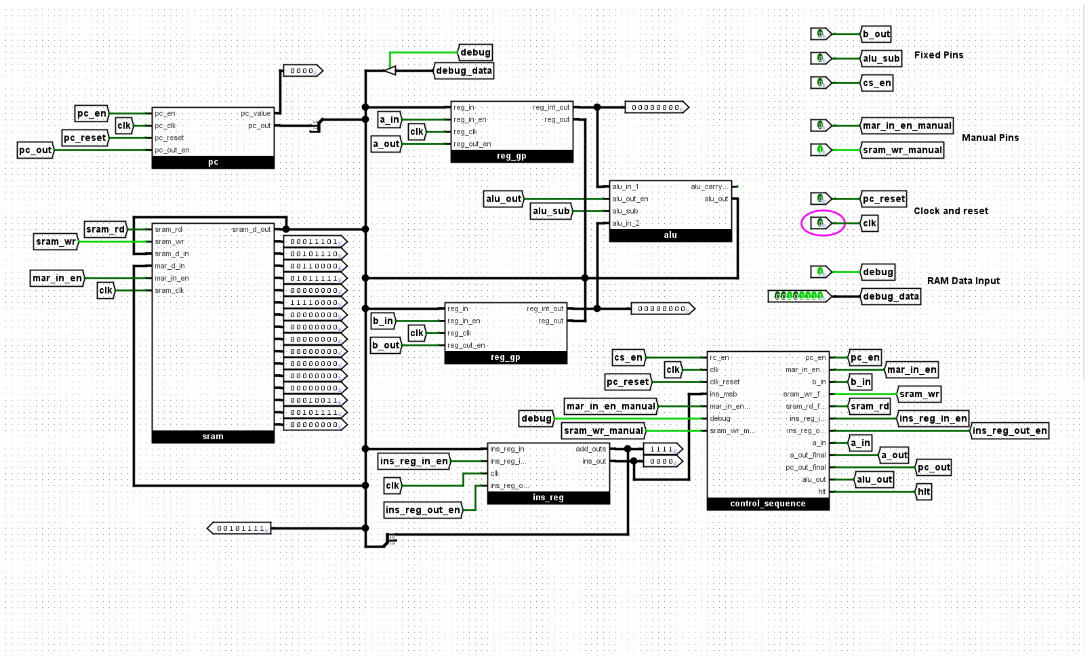


Figure 16: Complete Manual Mode Circuit