MT-256L Digital Logic Design Lab

End Project Report

GROUP MEMBERS:	
Waleed Asif **(242052)** Syed Anis Shah Gillani **(242073)** Wasim Haider **(242…)**	
SESSION: Fall 25	
DATE OF REPORT SUBMITTED: 25 December 2025	**GRADE/POINTS:**

Table of Contents

List of Figure

List of Tables

Chapter 1

Introduction to SAP-1 and Project Overview

1.1 Introduction

Digital systems form the foundation of all modern computers. From simple calculators to advanced processors, every computing system is built using basic digital logic components such as gates, registers, counters, and memory units. Understanding how these components work together is an essential part of learning **Digital Logic Design (DLD)** and **Computer Organization**.

In academic courses, it is often difficult for students to understand how software instructions are actually executed inside hardware. To bridge this gap between theory and practical understanding, simplified computer architectures are used. One of the most famous educational models is the **SAP-1 (Simple-As-Possible-1) computer**.

This project is based on the **SAP-1 architecture**, which is a very small and simple 4-bit computer designed only for learning purposes. Although SAP-1 is not used in real commercial systems, it clearly demonstrates the internal working of a CPU, including instruction execution, data transfer, and control sequencing.

1.2 What is SAP-1?

SAP-1 stands for **Simple-As-Possible-1**. It is an educational computer architecture introduced to help students understand how a basic computer works at the hardware level.

The main characteristics of SAP-1 are:

- Very simple architecture
- Limited instruction set
- Uses a small number of registers
- Operates on a shared system bus
- Uses a hardwired control unit
- Designed for learning, not performance

SAP-1 typically works with:

- A **4-bit or 8-bit data path**
- A small **RAM**
- A basic **ALU**
- A **program counter**
- A **memory address register**
- A **control unit with timing states**

Because of its simplicity, SAP-1 allows students to trace every signal and understand exactly what happens during instruction execution.

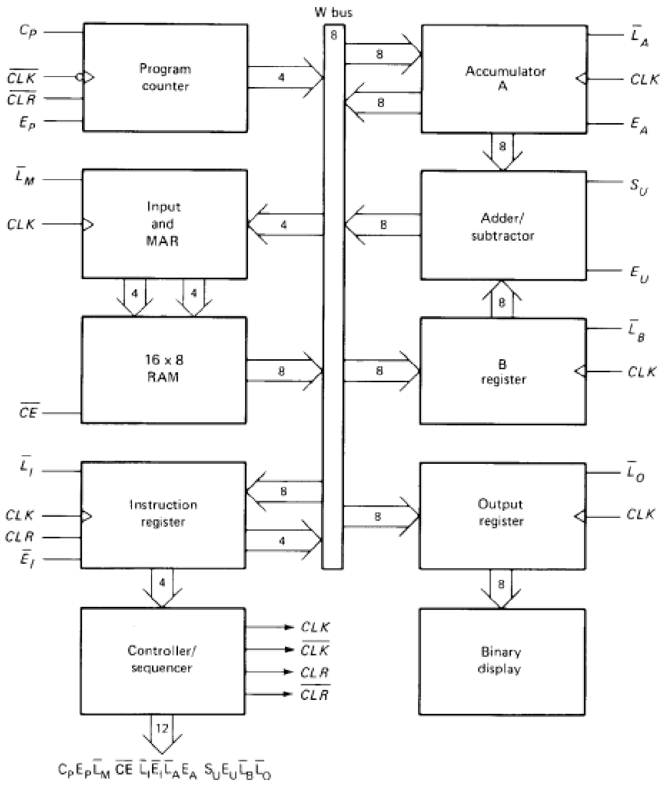


Figure 1. 1 Basic block diagram of SAP-1 computer architecture.

1.3 Purpose of This Project

The main purpose of this project is to **design and simulate a SAP-1 based computer system** that can perform basic arithmetic operations using digital logic components.

The objectives of this project are:

- To understand the internal structure of a basic CPU
- To implement registers using D flip-flops
- To design and use an ALU for arithmetic operations
- To understand the role of a system bus and tri-state buffers
- To study the function of RAM and MAR in instruction execution
- To design a simple control unit and program sequencer
- To simulate the complete system using **Proteus**

This project focuses mainly on implementing and demonstrating **ADD and SUB instructions**, which are sufficient to show how SAP-1 executes instructions step by step.

1.4 Scope of the Project

The scope of this project is limited to the **SAP-1 architecture** only. The design is intentionally kept simple so that each block can be clearly understood and analyzed.

The project includes:

- 4-bit data processing

- Manual and clock-controlled operation
- Manual opcode loading (for testing and learning purposes)
- Arithmetic operations (Addition and Subtraction)
- Output display using 7-segment display

The project does **not** include:

- Advanced instructions
- Microprogrammed control
- Pipelining
- Interrupt handling
- Complex memory management

These limitations are intentional and suitable for an academic DLD project.

1.5 Why SAP-1 is Important for Learning

SAP-1 is important because it clearly shows:

- How instructions are executed in hardware
- How data moves through registers and buses
- How control signals coordinate different components
- How timing states control sequential operations

Instead of treating the CPU as a "black box," SAP-1 allows students to observe and understand each internal operation. This makes it an ideal architecture for laboratory-based learning.

1.6 Tools and Technologies Used

The following tools and components are used in this project:

- **Proteus is**for simulation
- Standard TTL and CMOS ICs
- D flip-flops for register implementation
- 74HC181 for ALU operations
- 6116 Static RAM
- 74LS138 decoder for opcode decoding
- Tri-state buffers for bus control
- 7-segment display for output

Proteus provides a realistic environment to test and verify the design before any hardware implementation.

1.7 Project Organization

This report is organized chapter-wise to explain the project in a clear and structured manner. Each chapter focuses on a specific part of the SAP-1 system, starting from basic theory and moving towards detailed design and implementation.

The next chapter discusses the **background theory**, including digital logic concepts, registers, ALU, and bus systems, which are necessary to understand the SAP-1 architecture.

Chapter 2

Background Theory

2.1 Introduction

Before understanding the complete SAP-1 architecture, it is necessary to review some basic concepts of **Digital Logic Design (DLD)**. SAP-1 is built using fundamental digital components such as logic gates, flip-flops, registers, counters, memory, and buses. Each of these blocks plays a specific role in making the system work as a simple computer.

This chapter explains the theoretical background required to understand the design and operation of the SAP-1 system implemented in this project.

2.2 Digital Logic and Binary System

Digital systems operate using the **binary number system**, which consists of only two values:

- **0 (LOW)**
- **1 (HIGH)**

These two values are represented physically using voltage levels in electronic circuits. All data, instructions, and control signals inside a computer are represented in binary form.

In this project, a **4-bit binary system** is used. A 4-bit binary number can represent values from:

0000 (0) to 1111 (15)

This limited range is sufficient for understanding how arithmetic operations and instruction execution work in a simple CPU.

2.3 Logic Gates

Logic gates are the basic building blocks of digital circuits. They perform logical operations on binary inputs and produce a binary output.

The main logic gates used in this project are:

- **AND Gate** – output is 1 only if all inputs are 1
- **OR Gate** – output is 1 if any input is 1
- **NOT Gate** – inverts the input
- **NAND / NOR Gates** – combinations of AND/OR with NOT

Logic gates are mainly used in the **control unit** to generate control signals based on opcode and timing states.

2.4 Flip-Flops and Sequential Logic

Unlike combinational logic, **sequential logic** depends on both the current inputs and previous state. The basic storage element used in sequential circuits is the **flip-flop**.

In this project, **D flip-flops** are used extensively.

D Flip-Flop Characteristics:

- Stores one bit of data
- Data is captured on the active clock edge
- Has clear/reset capability

D flip-flops are used to build:

- Registers
- Opcode register
- Output register
- Memory Address Register (MAR)

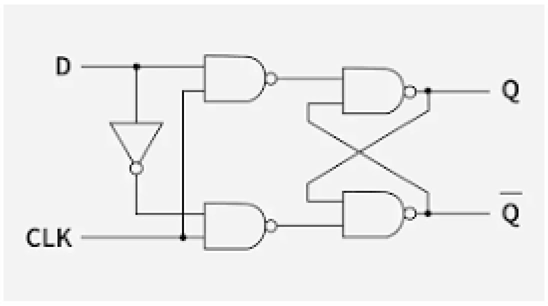


Figure 2. 1 Logic symbol and operation of a D flip-flop.

2.5 Registers

A **register** is a group of flip-flops used to store multiple bits of data. Registers temporarily store data inside the CPU during execution.

In the SAP-1 project, different registers are used:

- **Register A (Accumulator)** – stores primary operand and ALU result
- **Register B** – stores second operand
- **Opcode Register (Instruction Register)** – stores the instruction opcode
- **Output Register** – stores the final result for display
- **MAR (Memory Address Register)** – stores memory addresses

Registers allow data to be transferred and processed in an organized manner.

2.6 Counters and Program Counter

A **counter** is a sequential circuit that increments or decrements its value with each clock pulse.

In SAP-1, a counter is used as the **Program Counter (PC)**.

Functions of the Program Counter:

- Stores the address of the next instruction
- Increments automatically after each instruction fetch
- Works together with MAR to access memory

The program counter ensures that instructions are executed in the correct sequence.

2.7 Arithmetic Logic Unit (ALU)

The **Arithmetic Logic Unit (ALU)** is the core processing unit of the CPU. It performs arithmetic and logical operations on binary data.

In this project:

- A **4-bit ALU (74HC181)** is used
- Supported operations include **ADD and SUB**
- Operations are selected using control signals

The ALU receives inputs from Register A and Register B and produces a result that is stored back into a register.

2.8 System Bus

A **bus** is a shared set of wires used to transfer data between different components of a system.

In SAP-1:

- A **4-bit data bus** is used
- Only one device is allowed to drive the bus at a time
- **Tri-state buffers** are used to avoid bus conflicts

The bus simplifies wiring and allows flexible data transfer between registers, memory, and the ALU.

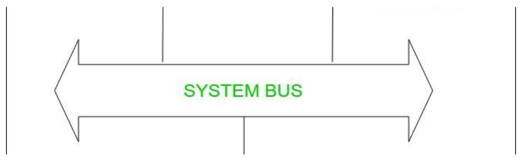


Figure 2. 2 Concept of shared system bus in digital systems.

2.9 Tri-State Buffers

Tri-state buffers have three output states:

- Logic 0
- Logic 1
- High impedance (disconnected)

Purpose of tri-state buffers:

- Control which device places data on the bus
- Prevent short circuits and data conflicts
- Enable shared bus architecture

Tri-state buffers are essential for proper SAP-1 operation.

2.10 Memory System (RAM)

Memory is required to store:

- Instructions
- Data values

In this project:

- **6116 Static RAM** is used
- Although it is a 2K × 8 RAM, only a small portion is utilized
- Effectively used as **16 × 4 RAM**

Static RAM is preferred because it is fast and does not require refresh circuitry.

2.11 Control Unit Basics

The **Control Unit** generates control signals that coordinate the operation of all components.

It is responsible for:

- Register loading
- ALU operation selection
- Bus enable signals
- Memory read/write control

In SAP-1, the control unit is **hardwired**, meaning it is implemented using logic gates and decoders instead of software.

2.12 Importance of Timing (T-States)

Instruction execution is divided into **timing states** (T-states):

- T1 – Fetch address
- T2 – Fetch instruction
- T3 – Execute operation
- T4 – Store result

These T-states ensure that operations occur in the correct order and at the correct time.

2.13 Summary

This chapter provided the theoretical foundation required to understand the SAP-1 architecture. Concepts such as logic gates, flip-flops, registers, ALU, bus systems, memory, and control units form the basis of the SAP-1 design.

In the next chapter, the **complete SAP-1 architecture** and the role of each block will be explained in detail.

Chapter 3

SAP-1 Architecture

3.1 Introduction

The SAP-1 (Simple-As-Possible-1) architecture is a small and educational computer architecture designed to demonstrate how a basic CPU works internally. It is not meant for high-speed or real-world applications; instead, it focuses on clarity and simplicity so that students can understand how different hardware blocks interact during instruction execution.

This chapter explains the **overall architecture of SAP-1**, describes each major block used in the project, and explains how these blocks are interconnected to form a working computer system.

3.2 Overview of SAP-1 Architecture

The SAP-1 architecture is based on a **single shared data bus** and a **hardwired control unit**. All data transfers between components take place through this common bus, and control signals determine which component is active at a given time.

The main blocks of SAP-1 architecture are:

- Program Counter (PC)
- Memory Address Register (MAR)
- Static RAM
- Instruction Register / Opcode Register
- Arithmetic Logic Unit (ALU)
- Register A (Accumulator)
- Register B
- Output Register
- System Bus
- Control Unit and Program Sequencer
- Clock and Reset Circuit

Each block performs a specific task, and together they complete the instruction execution cycle.

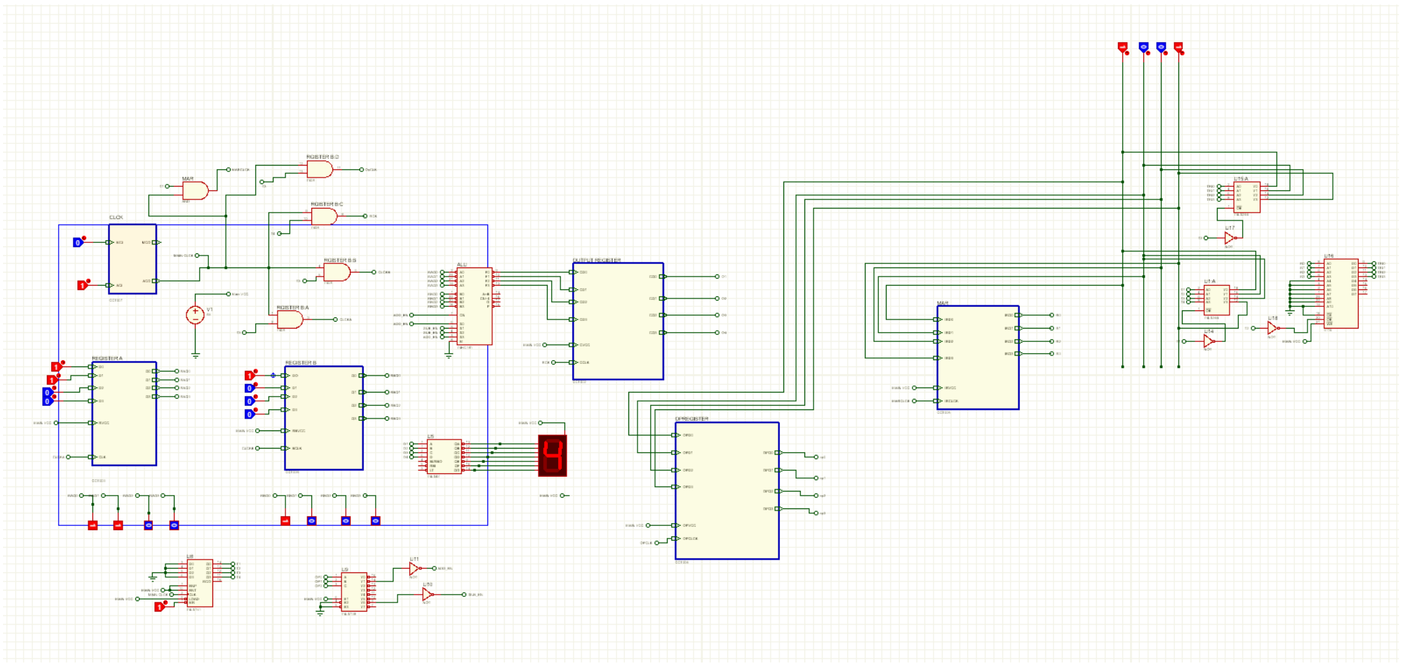


Figure 3. 1 Complete SAP-1 architecture showing datapath and control unit.

3.3 Program Counter (PC)

The **Program Counter** is a register that holds the address of the next instruction to be executed.

Functions of the PC:

- Stores the current instruction address
- Increments automatically after each instruction fetch
- Sends its value to the bus when enabled

In this project, the PC is implemented using a counter IC. The PC works closely with the MAR to access memory in an orderly sequence.

3.4 Memory Address Register (MAR)

The **Memory Address Register (MAR)** stores the address of the memory location that is currently being accessed.

Purpose of MAR:

- Holds a stable memory address
- Prevents address changes during memory access
- Separates address handling from data handling

During the fetch cycle, the PC places an address on the bus, and MAR captures this address. The RAM then uses the MAR output to select the correct memory location.

3.5 Static RAM

Static RAM is used to store both **instructions and data**.

In this project:

- A **6116 Static RAM** IC is used
- Only a limited number of address and data lines are utilized
- Effectively operates as a **16 × 4 memory**

The RAM provides stored data to the system bus during read operations and accepts data from the bus during write operations.

3.6 Instruction Register (Opcode Register)

The **Instruction Register (IR)**, also referred to as the **Opcode Register**, stores the instruction currently being executed.

Functions of the IR:

- Stores opcode fetched from memory or bus
- Feeds opcode to the control unit
- Determines which operation the ALU will perform

In the current implementation, the opcode can be loaded manually through the bus for testing and learning purposes.

3.7 Arithmetic Logic Unit (ALU)

The **ALU** is the computational core of the SAP-1 system.

In this project:

- A **4-bit ALU (74HC181)** is used
- Supports arithmetic operations such as **ADD and SUB**
- Receives operands from Register A and Register B
- Produces a result that is stored back into a register

The ALU operation is controlled by signals generated from the control unit based on the opcode.

OPERATION	M	S3	S2	S1	S0	CN
ADD	0	1	0	0	1	1
SUB	0	0	1	1	0	0

TABLE 1: OPCODE of ADD and SUB

3.8 Register A (Accumulator)

Register A acts as the **accumulator**.

Functions:

- Stores one operand for ALU operations

- Receives ALU results
- Sends data to the bus when enabled

Register A plays a central role in arithmetic operations.

3.9 Register B

Register B stores the second operand required for arithmetic operations.

Functions:

- Holds data loaded from the bus
- Supplies operand to the ALU
- Works together with Register A

Register B does not directly receive ALU results; it is mainly used as an input register.

3.10 Output Register

The **Output Register** stores the final result of an operation so that it can be displayed.

Functions:

- Receives data from the bus
- Holds result stable for display
- Connects to 7-segment display circuitry

This register ensures that output does not change unexpectedly during execution.

3.11 System Bus

The system bus is a **4-bit shared data bus** used to transfer data between all major components.

Characteristics:

- Only one device drives the bus at a time
- Uses tri-state buffers to avoid conflicts
- Simplifies wiring and data transfer

The bus is a key element of SAP-1 architecture.

3.12 Control Unit and Program Sequencer

The **Control Unit** generates control signals required to operate the SAP-1 system.

It consists of:

- A **Program Sequencer** (step counter)
- Opcode decoding logic

- Logic gates for control signal generation

The Program Sequencer controls the timing (T-states), while the control logic determines which operation to perform based on the opcode.

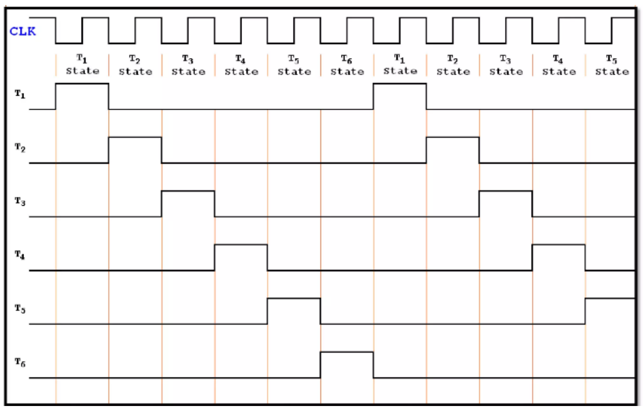


Figure 3. 2 Control unit and program sequencer structure in SAP-1.

3.13 Clock and Reset Circuit

The clock provides synchronization to all sequential components.

Functions:

- Drives counters and registers
- Controls timing of operations

The reset circuit:

- Clears registers and counters
- Initializes the system to a known state

Together, clock and reset ensure reliable operation.

3.14 Instruction Execution Flow

In SAP-1, instruction execution follows a simple sequence:

1. Fetch instruction address from PC
2. Load address into MAR
3. Read instruction from RAM
4. Store opcode in IR
5. Decode opcode
6. Execute operation using ALU
7. Store and display result

This flow repeats for each instruction.

3.15 Summary

This chapter described the complete SAP-1 architecture and explained the role of each block used in the project. The architecture shows how simple digital components can be combined to create a functioning computer system.

The next chapter will discuss the **Instruction Set Architecture (ISA)** and explain how ADD and SUB instructions are represented and executed.

Chapter 4

Instruction Set Architecture (ISA)

4.1 Introduction

The **Instruction Set Architecture (ISA)** defines the set of instructions that a computer can understand and execute. In simple terms, the ISA acts as a bridge between hardware and programming. It tells the hardware **what operations are possible** and **how those operations are encoded**.

In the SAP-1 architecture, the ISA is intentionally kept very small. This simplicity allows students to clearly understand how each instruction is fetched, decoded, and executed at the hardware level.

This chapter explains the **instruction format**, **opcode structure**, and **execution behavior** of the instructions implemented in this SAP-1 project.

4.2 Instruction Format in SAP-1

SAP-1 uses a **fixed-length instruction format**. Each instruction consists of a small number of bits so that decoding is simple.

In this project:

- Instruction width is **4 bits** (opcode only)
- The opcode directly selects the operation to be performed
- No complex addressing modes are used

The instruction format can be represented as:

[OP3 OP2 OP1 OP0]

Where:

- OP3–OP0 represent the **opcode**
- Each unique opcode corresponds to one instruction

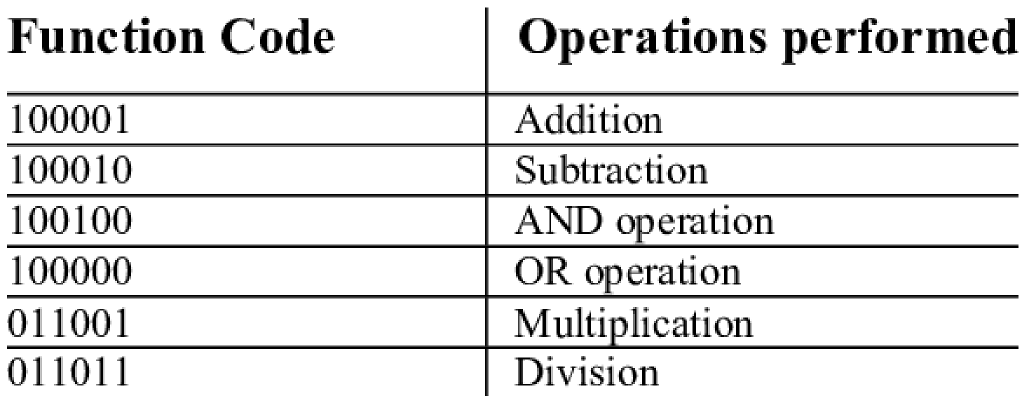
4.3 Opcode Concept

An **opcode (operation code)** is a binary pattern that tells the control unit which operation to execute.

In SAP-1:

- The opcode is stored in the **Opcode Register (Instruction Register)**
- The opcode is decoded using a **decoder IC (74LS138)**
- The decoded output enables the appropriate control signals

The opcode does not directly control hardware blocks. Instead, it is first decoded and then combined with timing signals (T-states) to generate proper control signals.



Function Code	Operations performed
100001	Addition
100010	Subtraction
100100	AND operation
100000	OR operation
011001	Multiplication
011011	Division

Figure 1 OPCODE

4.4 Implemented Instructions

This project focuses on implementing two fundamental arithmetic instructions:

1. **ADD (Addition)**
2. **SUB (Subtraction)**

These instructions are sufficient to demonstrate how arithmetic operations are performed inside a CPU.

4.5 ADD Instruction

4.5.1 Description

The **ADD instruction** performs addition between the contents of Register A and Register B.

Mathematically:

$A \leftarrow A + B$

Where:

- A is the accumulator (Register A)
- B is the operand stored in Register B

4.5.2 Opcode for ADD

In this project, a specific 4-bit opcode is assigned to the ADD instruction. For example:

ADD opcode = 1001

(This assignment is project-specific and chosen for simplicity.)

4.5.3 Execution of ADD

The execution of ADD involves the following steps:

1. Opcode is loaded into the Opcode Register
2. Control unit decodes the opcode
3. ALU is configured for addition

4. Register A and Register B provide inputs to the ALU
5. ALU produces the sum
6. Result is stored back into Register A or Output Register

The entire process is controlled by timing states generated by the program sequencer.

4.6 SUB Instruction

4.6.1 Description

The **SUB instruction** performs subtraction between the contents of Register A and Register B.

Mathematically:

$A \leftarrow A - B$

This operation allows the system to compute differences between two numbers.

4.6.2 Opcode for SUB

A separate opcode is assigned to the SUB instruction, for example:

SUB opcode = 0110

4.6.3 Execution of SUB

The execution process for SUB is similar to ADD, except that the ALU is configured for subtraction:

1. Opcode is loaded into the Opcode Register
2. Control unit decodes the opcode
3. ALU control signals select subtraction mode
4. Register A and Register B supply operands
5. ALU produces the difference
6. Result is stored and displayed

4.7 Role of ALU Control Signals

The ALU does not understand opcodes directly. Instead, it operates based on **control signals** generated by the control unit.

For the 74HC181 ALU, important control signals include:

- Mode select (M)
- Function select inputs (S3–S0)
- Carry input (CN)

Different combinations of these signals result in different arithmetic operations.

4.8 Manual Opcode Loading

In the current implementation:

- Opcode is loaded **manually through the system bus**
- This approach simplifies testing and debugging
- It allows direct verification of ALU and control logic

Manual opcode loading is acceptable in educational projects and helps isolate and test individual components of the SAP-1 system.

4.9 Instruction Execution Timing

Each instruction is executed over multiple clock cycles using timing states:

- T1 – Instruction fetch or setup
- T2 – Opcode decoding
- T3 – Arithmetic operation
- T4 – Result storage

These timing states ensure that data transfer and computation occur in the correct order.

T-State	Operation
T1	Address fetch
T2	Opcode fetch
T3	Execute instruction
T4	Store result

TABLE 2: Timing State

4.10 Advantages of a Small Instruction Set

Using a limited instruction set has several benefits:

- Simplifies hardware design
- Reduces decoding complexity
- Improves understanding of instruction execution
- Makes debugging easier

SAP-1 demonstrates that even a small instruction set can perform meaningful computation.

4.11 Summary

This chapter discussed the Instruction Set Architecture of the SAP-1 project. The instruction format, opcode concept, and execution of ADD and SUB instructions were explained in detail. Understanding the ISA is essential to appreciate how the control unit and ALU work together to execute instructions.

The next chapter will explain the **Program Sequencer and Control Unit Design**, focusing on timing states and control signal generation.

Chapter 5

Program Sequencer and Control Unit Design

5.1 Introduction

In any computer system, performing arithmetic operations alone is not sufficient. A proper mechanism is required to **control the sequence of operations** and ensure that each component acts at the correct time. This responsibility is handled by the **Program Sequencer** and the **Control Unit**.

In the SAP-1 architecture, the control system is **hardwired**, meaning it is implemented using counters, decoders, and logic gates instead of software or microcode. This chapter explains how the program sequencer and control unit are designed and how they work together to execute instructions step by step.

5.2 Program Sequencer

5.2.1 Definition

The **Program Sequencer** is the part of the control system that generates **timing signals**, also known as **T-states**. These timing signals define the order in which different micro-operations are performed during instruction execution.

In simple words:

- The program sequencer decides **when** each action occurs.

5.2.2 Implementation of Program Sequencer

In this project, the program sequencer is implemented using:

- A **clock signal**
- A **counter IC (74LS161)**

The counter increments with each clock pulse and generates a sequence of timing states such as:

- T1
- T2
- T3
- T4

Each timing state corresponds to a specific phase of instruction execution.

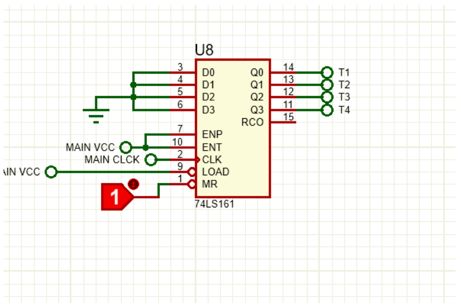


Figure 2 PC

5.2.3 Role of Clock Signal

The clock signal provides synchronization for the entire system.

Functions of the clock:

- Advances the program sequencer
- Synchronizes register loading
- Ensures predictable operation timing

Each rising edge of the clock causes the sequencer to move to the next timing state.

5.3 Timing States (T-States)

The execution of an instruction in SAP-1 is divided into multiple timing states to avoid conflicts on the bus and ensure correct data transfer.

Typical timing states used in this project are:

- **T1** – Address setup or preparation
- **T2** – Opcode decoding
- **T3** – Arithmetic execution
- **T4** – Result storage

These states repeat for every instruction.

5.4 Control Unit

5.4.1 Definition

The **Control Unit** is responsible for generating control signals that enable or disable various components of the SAP-1 system.

In simple terms:

- The control unit decides **what** operation is performed.

5.4.2 Components of Control Unit

The control unit in this project consists of:

- Opcode Register (Instruction Register)
- Opcode Decoder (74LS138)
- Logic gates (AND, OR, NOT)
- Timing signals from the program sequencer

These components work together to produce the required control signals.

5.5 Opcode Decoding

The opcode stored in the Opcode Register is decoded using a **3-to-8 line decoder (74LS138)**.

Purpose of decoding:

- Convert binary opcode into individual control lines
- Ensure only one instruction is active at a time

For example:

- ADD opcode activates ADD control line
- SUB opcode activates SUB control line

The decoder outputs are then combined with timing signals.

5.6 Generation of Control Signals

Control signals are generated by logically combining:

- Opcode decoder outputs
- Timing state signals

Example:

ADD_ENABLE = ADD_OPCODE AND T3

SUB_ENABLE = SUB_OPCODE AND T3

This ensures that:

- The ALU performs ADD or SUB
- Only during the correct timing state

5.7 ALU Control Signal Generation

The ALU requires specific control inputs such as:

- Mode select (M)
- Function select (S3–S0)
- Carry input (CN)

These signals are derived from the decoded opcode and timing signals. This allows the ALU to perform the correct arithmetic operation at the correct time.

5.8 Register Control Signals

Registers are controlled using load enable signals.

Examples:

- **LOAD_A** – loads data into Register A
- **LOAD_B** – loads data into Register B

- **LOAD_OUT** – loads data into Output Register

These signals ensure that registers capture data only when required.

5.9 Bus Control Signals

Since SAP-1 uses a shared bus, bus access must be carefully controlled.

Control unit responsibilities include:

- Enabling tri-state buffers
- Ensuring only one device drives the bus at a time
- Preventing bus contention

Bus enable signals are synchronized with timing states.

5.10 Hardwired Control vs Microprogrammed Control

SAP-1 uses **hardwired control**, which has the following characteristics:

Advantages:

- Simple implementation
- Fast operation
- Easy to understand

Limitations:

- Difficult to modify instruction set
- Requires hardware changes for new instructions

Despite these limitations, hardwired control is ideal for educational purposes.

5.11 Summary

This chapter explained the design and working of the **Program Sequencer** and **Control Unit** in the SAP-1 architecture. The sequencer generates timing states, while the control unit uses these states along with opcode decoding to generate control signals. Together, they ensure that instructions are executed in an orderly and correct manner.

The next chapter will focus on the **Datapath Design**, including bus structure, tri-state buffers, and register interconnections.

Chapter 6

Datapath Design

6.1 Introduction

The datapath is the physical structure through which data moves and is processed inside the SAP-1 computer. While the control unit decides **when** and **what** operation to perform, the datapath is responsible for **actually carrying out those operations**. It consists of registers, the ALU, the system bus, and tri-state buffers that connect all components together.

This chapter explains the datapath design of the SAP-1 project in detail, focusing on how data flows between different blocks during instruction execution.

6.2 Concept of Datapath

In a simple computer system, the datapath includes:

- Storage elements (registers)
- Processing elements (ALU)
- Data transfer medium (bus)

The datapath must be carefully designed so that:

- Data moves only when required
- No two components drive the bus at the same time
- Correct values reach the ALU and registers

In SAP-1, the datapath is intentionally kept simple to clearly demonstrate these concepts.

6.3 System Bus Structure

The SAP-1 datapath is based on a **single shared system bus**.

Characteristics of the bus:

- 4-bit wide data bus
- Used for all data transfers
- Connects registers, RAM, ALU, and output unit
- Controlled using tri-state buffers

The bus reduces wiring complexity and allows flexible communication between components.

6.4 Need for Tri-State Buffers

A major challenge in bus-based systems is **bus contention**, which occurs when more than one device tries to place data on the bus simultaneously.

To prevent this problem, **tri-state buffers** are used.

Tri-state buffer states:

- Logic 0
- Logic 1
- High impedance (disconnected)

When a buffer is disabled, its output is electrically disconnected from the bus.

6.5 Bus Driving Devices

In this SAP-1 design, the following blocks can place data on the bus:

- Register A (Accumulator output)
- Register B (when required)
- ALU output
- RAM output
- Manual input switches (for testing)

Each of these devices is connected to the bus through a tri-state buffer, and only one buffer is enabled at any given time.

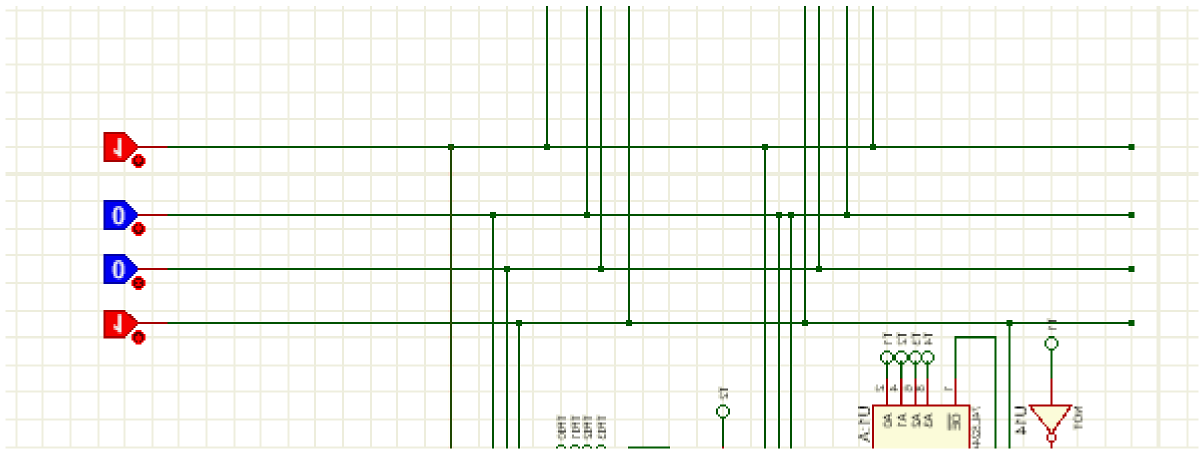


Figure 3 BUS DATA PATH

6.6 Register Connections in Datapath

Registers play a key role in the datapath.

Register A (Accumulator)

- Receives data from the bus
- Sends data to the ALU
- Stores ALU results

Register B

- Receives data from the bus
- Supplies second operand to the ALU

Output Register

- Receives data from the bus
- Holds result for display

All register inputs are connected to the bus, and their loading is controlled by load enable signals.

6.7 ALU Integration in Datapath

The ALU is connected directly to:

- Output of Register A
- Output of Register B

The ALU output is connected to the bus using a tri-state buffer.

Datapath flow for arithmetic operation:

1. Register A and Register B supply inputs to ALU
2. ALU performs ADD or SUB
3. ALU output is enabled onto the bus
4. Result is stored back into Register A or Output Register

This design allows reuse of the same bus for multiple purposes.

6.8 Data Flow During ADD Instruction

For an ADD instruction, the datapath operates as follows:

1. Data is loaded into Register A
2. Data is loaded into Register B
3. Control unit enables ALU in ADD mode
4. ALU output is placed on the bus
5. Result is captured by Register A or Output Register

This step-by-step flow ensures correct and conflict-free data movement.

6.9 Data Flow During SUB Instruction

The datapath behavior for SUB is similar to ADD, except the ALU is configured for subtraction.

Steps:

1. Register A provides minuend
2. Register B provides subtrahend
3. ALU performs subtraction
4. Output is sent to the bus

5. Result is stored and displayed

Only ALU control signals differ between ADD and SUB.

6.10 Importance of Load and Enable Signals

Load and enable signals ensure correct datapath operation.

- **Load signals** determine when registers capture bus data
- **Enable signals** determine which device drives the bus

These signals are generated by the control unit and synchronized with timing states.

6.11 Avoiding Bus Contention

To avoid bus contention:

- Only one tri-state buffer is enabled at a time
- All other buffers remain in high-impedance state
- Control logic ensures proper sequencing

This is a fundamental rule followed throughout the datapath design.

6.12 Summary

This chapter discussed the datapath design of the SAP-1 system. The use of a shared bus, tri-state buffers, registers, and the ALU allows efficient and controlled data movement. The datapath works closely with the control unit to ensure correct execution of instructions such as ADD and SUB.

The next chapter will focus on the **Memory System**, including RAM and MAR design and their role in instruction execution.

Chapter 7

Memory System (RAM, MAR, and Bus Interaction)

7.1 Introduction

In the SAP-1 computer, memory is used to **store instructions and data** that the processor needs to execute operations automatically. Without memory, all operations must be controlled manually, which limits the system.
This chapter explains the **exact purpose, design, and connection** of:

- RAM
- Memory Address Register (MAR)
- Their interaction with the bus

These blocks are the **missing link** between a calculator-like circuit and a real microprocessor.

7.2 Why Memory Is Required in SAP-1

In the current stage of the project, opcodes are entered manually through the bus. This works for testing, but it is **not how a real CPU operates**.

Memory is required to:

- Store a sequence of instructions (program)
- Allow automatic execution without manual opcode input
- Enable sequential operation using the Program Counter

Without RAM:

- SAP-1 behaves like a manual arithmetic unit
- Instruction execution is not automatic

With RAM:

- Instructions are fetched automatically
- SAP-1 behaves like a real computer

7.3 Role of RAM in SAP-1

The **RAM (Random Access Memory)** stores:

- Instructions (opcode + operand)
- Data values used by instructions

In your project:

- A **16 × 8 RAM (e.g., 6116)** is used conceptually
- Only the required lower bits are connected (4-bit datapath)

Functions of RAM:

- Stores instruction bytes
- Outputs data when enabled
- Works under control of MAR and control unit

7.4 RAM Pins and Their Purpose

Typical RAM pins used in SAP-1:

- **A0–A3** → Address lines (from MAR)
- **D0–D7** → Data lines (to/from bus)
- **CE (Chip Enable)** → Enables RAM operation
- **OE (Output Enable)** → Allows RAM to place data on bus
- **WE (Write Enable)** → Allows writing data into RAM

In SAP-1:

- CE is usually tied active
- OE is controlled by the control unit
- WE is used during programming mode

7.5 Memory Address Register (MAR)

7.5.1 Purpose of MAR

The MAR holds the **address of the memory location** to be accessed.

It solves a key problem:

The Program Counter cannot stay connected to RAM directly because the bus is shared.

MAR provides:

- Stable memory address
- Isolation between PC and RAM
- Correct timing during instruction fetch

7.5.2 MAR Implementation

MAR is implemented using:

- **4 D-flip-flops**
- Common clock
- Load enable signal

Connections:

- **MAR inputs** ← Bus

- **MAR outputs** → RAM address lines

Once loaded, MAR keeps the address stable even if the bus changes.

7.6 Data Flow: How RAM and MAR Work Together

Step-by-step flow:

1. Program Counter places address on bus
2. MAR loads this address
3. MAR outputs address to RAM
4. RAM places instruction data on bus
5. Instruction Register loads opcode

This sequence happens during the **fetch cycle**.

7.7 Why Tri-State Buffers Are Required

RAM outputs must not always drive the bus.

Tri-state buffers are used to:

- Enable RAM output only when required
- Prevent bus contention
- Allow other components to use the bus

When RAM is not selected:

- Output is in high-impedance state
- Bus remains free

7.8 RAM Interaction with Bus

RAM ↔ Bus connection rules:

- RAM output connects to bus via tri-state buffer
- OE controls when RAM can send data
- Bus carries instruction/data to registers

This design allows:

- Multiple devices to share the same bus
- Safe and controlled data transfer

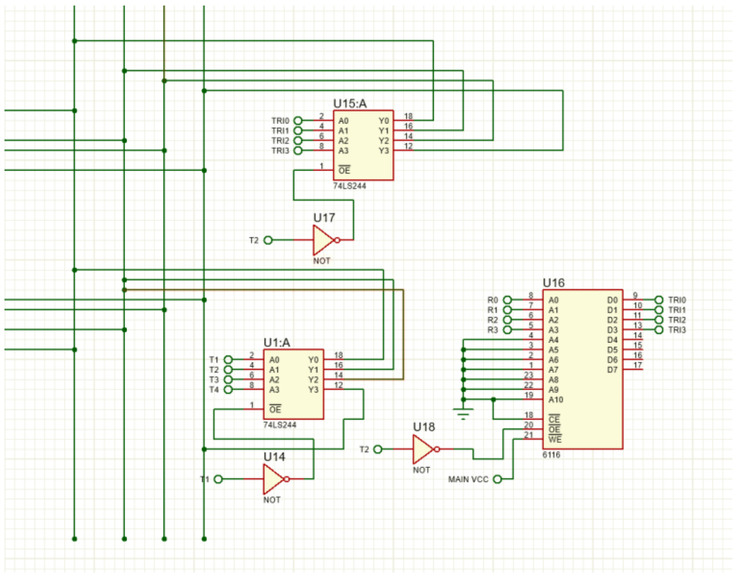


Figure 4 Ram integration with bus and tri state buffer

7.9 Why Opcode Is Still Entered Manually (Current Stage)

In My current project:

- RAM and MAR are added
- Opcode register still receives manual input

Reason:

- Full fetch–execute automation is not yet enabled
- Control sequencing is partially manual

This is **acceptable for SAP-1 implementation stages**.

Once RAM is fully integrated:

- Opcode will come from RAM automatically
- Manual input will no longer be required

7.10 Difference Between Manual Opcode and Memory-Based Opcode

Manual Opcode	**RAM-Based Opcode**
User controlled	Automatic
Used for testing	Used for programs
Simple design	True SAP-1 behavior
No fetch cycle	Full fetch cycle

TABLE 3: Difference Between Manual Opcode and Memory-Based Opcode

My project is currently **between these two stages**, which is normal.

7.11 Summary

This chapter explained the **memory system of SAP-1**, focusing only on what is required for your project. RAM stores instructions, MAR holds address, and tri-state buffers ensure safe bus operation. Together, these blocks allow SAP-1 to move from manual control to automatic instruction execution.

The next chapter will focus on **Instruction Execution Flow (Fetch–Execute Cycle)** and show how all completed blocks work together as one system.

Chapter 8

Instruction Fetch and Execute Cycle (SAP-1)

8.1 Introduction

The **instruction cycle** defines how SAP-1 executes an instruction step by step.
Each instruction is not executed in one action; instead, it is divided into smaller steps called **micro-operations**.

In this project, the instruction cycle is divided into two main phases:

1. **Fetch Cycle**
2. **Execute Cycle**

These phases are controlled by the **program sequencer**, **control unit**, and **bus system**.

8.2 Fetch Cycle

The fetch cycle is **common for all instructions**.

Its purpose is to:

- Get the instruction from RAM
- Store it in the Instruction (Opcode) Register

T1 – Address Transfer

Operations:

- Program Counter places address on the bus
- MAR loads this address

Control actions:

- PC → Bus (enabled)
- MAR Load = 1

Result:

- MAR now holds the memory address of the instruction

T2 – Program Counter Increment

Operations:

- Program Counter increments by 1

Purpose:

- Prepare address for next instruction

Result:

- PC points to the next memory location

T3 – Instruction Fetch

Operations:

- RAM outputs instruction
- Instruction Register loads opcode

Control actions:

- RAM Output Enable = 1
- IR Load = 1

Result:

- Opcode register now holds the instruction

8.3 Execute Cycle (ADD Instruction)

Once the opcode is decoded as **ADD**, the execute cycle begins.

T4 – Operand Fetch

Operations:

- Register B loads operand (from RAM or bus)

Control actions:

- RAM → Bus
- Load B = 1

T5 – ALU Operation

Operations:

- ALU performs addition
- Result placed on bus

ALU control:

M = 0

S3 S2 S1 S0 = ADD pattern

CN = 1

T6 – Result Storage

Operations:

- Register A loads ALU result

Control actions:

- ALU → Bus
- Load A = 1

Result:

- Accumulator updated with sum

8.4 Execute Cycle (SUB Instruction)

Execution steps are **similar to ADD**, only ALU control changes.

ALU control for SUB:

M = 0

S3 S2 S1 S0 = SUB pattern

CN = 0

Result:

- Register A stores subtraction result

8.5 Role of Control Unit During Instruction Cycle

The control unit:

- Decodes opcode
- Combines opcode with timing signals
- Enables correct registers and buffers

Example:

ADD_ENABLE = ADD_OPCODE AND T5

This ensures:

- Correct operation
- Correct timing
- No bus conflict

8.6 Why This Is a Real CPU Cycle

Unlike FSM calculators:

- Operations are sequenced
- Instructions are fetched from memory
- Same hardware executes different instructions

This confirms that your project behaves as a **true SAP-1 microprocessor**, not a hardwired calculator.

8.7 Summary

This chapter explained how SAP-1 executes instructions using the fetch–execute cycle. The combination of RAM, MAR, bus, registers, ALU, and control unit enables correct and automatic instruction execution.

Chapter 9

Simulations and Final Integration

9.1 Introduction

Simulation and integration are essential phases of any digital system design. Before physical implementation, it is necessary to verify that each individual module works correctly and that all modules operate together as a complete system.

In this project, **Proteus ISIS** is used to simulate and test the complete **SAP-1 architecture**. The simulation phase ensures correct functionality of registers, ALU, control unit, memory system, and bus before moving towards hardware implementation.

9.2 Module-Wise Simulation

To ensure reliable operation, each block of the SAP-1 system was simulated separately before final integration.

9.2.1 Clock and Reset Circuit Simulation

- Both **manual and automatic clock modes** were tested
- Clock pulses correctly advanced the program sequencer
- Reset signal cleared:
  - Program Counter
  - Registers
  - Opcode Register
  - Output Register

This confirmed stable initialization of the system.

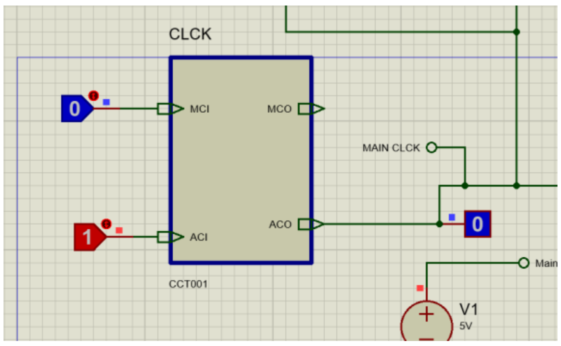


Figure 5. 1 CLOCK CONTROL

9.2.2 Register Simulation

Registers A, B, Opcode Register, and Output Register were simulated using D flip-flops.

Verification included:

- Correct data loading on clock edge

- Proper clearing using reset signal
- Stable output during idle states

Registers loaded data correctly from the bus without unintended changes.

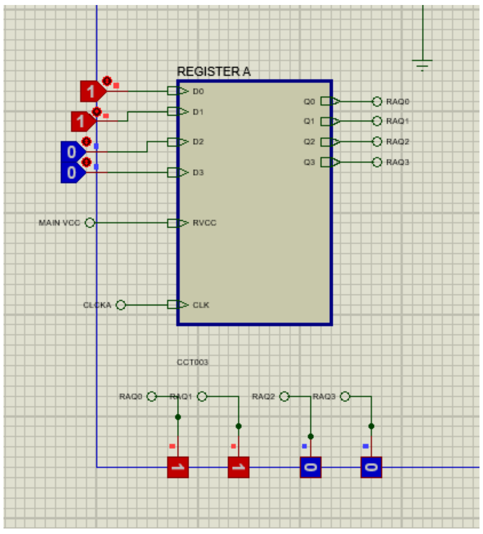


Figure 5. 2 Register Simulation

9.2.3 ALU Simulation

The **74HC181 ALU** was simulated for:

- ADD operation
- SUB operation

Control signals were applied through the opcode decoder and logic gates.
The ALU produced correct outputs for all tested input combinations.

9.2.4 Bus and Tri-State Buffer Simulation

The shared system bus was tested carefully to avoid contention.

- Only one tri-state buffer was enabled at a time
- High-impedance state correctly disconnected inactive devices
- No bus conflicts were observed during execution

This verified proper bus discipline.

9.2.5 RAM and MAR Simulation

The **6116 Static RAM** and **MAR** were simulated to confirm:

- Correct address loading into MAR
- Stable memory addressing
- Proper RAM read operation
- Correct data transfer between RAM and bus

RAM was effectively used as **16×4 memory** for SAP-1.

9.3 System Integration

After individual testing, all modules were integrated into a complete SAP-1 system.

Integrated blocks include:

- Program Counter
- MAR
- RAM
- Opcode Register
- Control Unit
- ALU
- Registers
- Bus system
- Output display

The integrated system successfully executed ADD and SUB instructions.

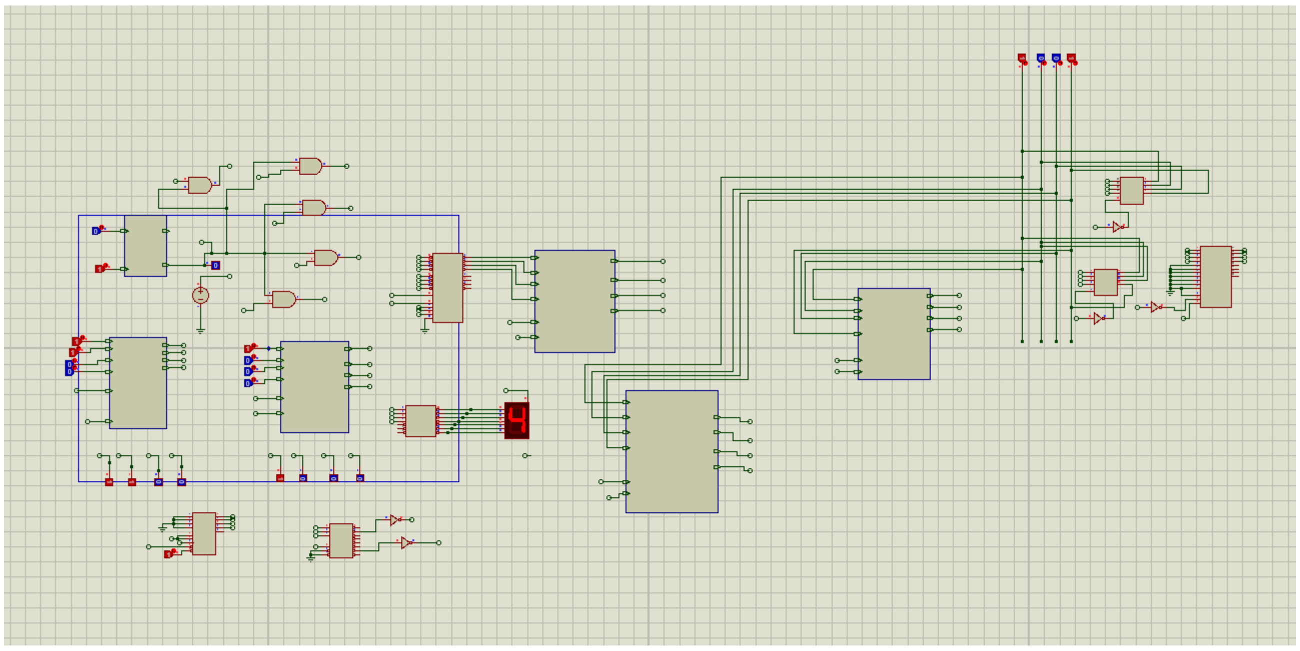


Figure 5. 3: Integrated Block

9.4 Proteus Simulation Results

Observations:

- Instructions executed in correct sequence
- Timing states operated as expected
- ALU produced accurate arithmetic results
- Output register displayed stable values

Multiple test cases were applied using different operand values, and results matched theoretical expectations.

9.4.1 Cost Analysis

Sr. No.	Component	IC / Type	Quantity	Approx. Cost (PKR)
1	Arithmetic Logic Unit	74HC181	1	800
2	Static RAM	6116	1	900
3	Opcode Decoder	74LS138	1	250
4	Program Counter	74LS161	1	300
5	D Flip-Flops (Registers)	74LS74	6	900
6	Tri-State Buffers	74LS244	4	800
7	Logic Gates	74LS08 / 74LS04	4	600
8	7-Segment Displays	Common Anode	2	300
9	BCD to 7-Segment Decoder	7447 / CD4511	2	500
10	Clock Components	IC + Passive	1	300
11	Push Buttons & Switches	Manual Input	—	200
12	Breadboard	Standard	1	500
13	Power Supply	5V Regulated	1	600
Total Estimated Cost				**≈ 6,950 PKR**

TABLE 4: Cost Analysis

9.4.2 Component Specification

Arithmetic Logic Unit (ALU) – 74HC181

Component Name: 74HC181

Function: 4-bit Arithmetic Logic Unit

Key Specifications:

Parameter	**Description**
Data width	4-bit
Operations	Arithmetic and logic
Supply voltage	2V – 6V (5V typical)
Control inputs	M, S0–S3, CN
Output	F0–F3, Carry out
Technology	CMOS

TABLE 5: 74HC181

Role in SAP-1:
Performs **ADD and SUB operations** based on control signals generated by the control unit.

Figure 5 ALU IC

Static RAM – 6116

Component Name: 6116 SRAM

Function: Data and instruction storage

Key Specifications:

Parameter	**Description**
Memory size	2048 × 8 bits
Access type	Random
Supply voltage	5V
Address lines	A0–A10
Data lines	D0–D7
Control pins	CE, OE, WE
Memory type	Static RAM

TABLE 6: Static RAM – 6116

Role in SAP-1:
Used as **16 × 4 memory** by grounding unused address and data lines. Stores instructions and operands.

Figure 6 Static RAM

Program Counter – 74LS161

Component Name: 74LS161

Function: 4-bit binary counter

Key Specifications:

Parameter	Description
Counter size	4-bit
Counting	Synchronous
Reset	Asynchronous clear
Clock	Positive edge triggered
Supply voltage	5V

TABLE 7: 74LS161

Role in SAP-1:
Generates sequential memory addresses for instruction execution.

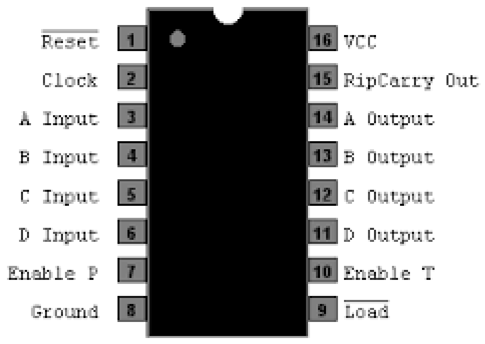


Figure 7 Program Counter

Opcode Decoder – 74LS138

Component Name: 74LS138

Function: 3-to-8 line decoder

Key Specifications:

Parameter	Description
Inputs	3
Outputs	8 (active LOW)
Enable pins	3
Technology	TTL
Supply voltage	5V

TABLE 8: 74LS138

Role in SAP-1:
Decodes opcode bits to generate **instruction-specific control signals**.

D Flip-Flop – 74LS74

Component Name: 74LS74

Function: Dual D-type flip-flop

Key Specifications:

Parameter	**Description**
Flip-flops per IC	2
Triggering	Positive edge
Reset	Asynchronous clear
Preset	Asynchronous set
Supply voltage	5V

TABLE 9: 74LS74

Role in SAP-1:
Used to construct:

- Register A
- Register B
- Opcode Register
- Output Register
- MAR

Tri-State Buffer – 74LS244

Component Name: 74LS244

Function: Octal buffer / line driver

Key Specifications:

Parameter	**Description**
Channels	8
Output states	0, 1, High-Z
Enable pins	2
Supply voltage	5V

TABLE 10: 74LS244

Role in SAP-1:
Controls access to the **shared system bus** and prevents bus contention.

Logic Gates (74LS08, 74LS04)

Component Names:

- 74LS08 (AND gate)
- 74LS04 (NOT gate)

Key Specifications:

IC	**Function**
74LS08	Quad 2-input AND
74LS04	Hex inverter
Voltage	5V

TABLE 11: 74LS08, 74LS04)

Role in SAP-1:
Used in:

- Control signal generation
- Opcode and timing combination
- Enable logic

Seven-Segment Display

Component Type: Common Anode 7-Segment Display

Key Specifications:

Parameter	**Description**
Display type	LED
Segments	a–g + dp
Operating voltage	5V
Configuration	Common Anode

TABLE 12: 74LS08, 74LS04)

Role in SAP-1:
Displays output value stored in the Output Register.

BCD to 7-Segment Decoder – 7447 / CD4511

Component Name: 7447 / CD4511

Key Specifications:

Parameter	Description
Input	BCD (4-bit)
Output	7-segment
Display type	Common Anode (7447)
Supply voltage	5V

TABLE 13: 7447

Role in SAP-1:
Converts binary/BCD data into a format suitable for display.

Clock Circuit Components

Components Used:

- IC-based clock generator / timer
- Resistors
- Capacitors
- Push buttons (manual clock)

Role in SAP-1:
Provides synchronized timing signals for the program sequencer and registers.

Power Supply

Specification:

Parameter	Value
Voltage	+5V DC
Regulation	Stable
Source	External / lab supply

Role in SAP-1:
Powers all TTL and CMOS ICs reliably.

9.5 Challenges During Simulation

Some challenges faced during simulation included:

- Bus contention due to incorrect enable signals
- Floating inputs causing undefined ALU behavior
- Incorrect opcode decoding during early stages

- Timing mismatches between control signals

These issues were resolved by:

- Proper tri-state buffer control
- Forcing unused inputs to defined logic levels
- Refining opcode decoding logic
- Synchronizing control signals with T-states

9.6 Hardware Implementation

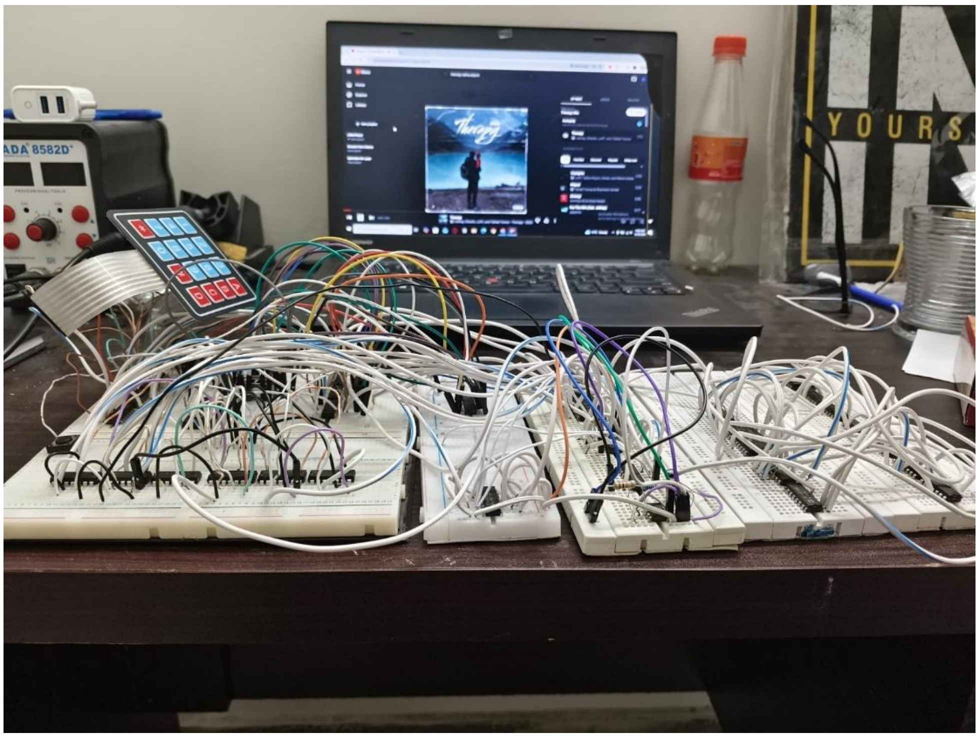


Figure 5. 4: Hardware Implementation

9.7 Discussion

Simulation confirmed that the SAP-1 design functions correctly and behaves as a true sequential digital system. Manual opcode loading was useful for debugging and validating datapath and control logic.

The simulation phase significantly reduced the risk of hardware errors and improved overall system reliability.

9.8 Summary

This chapter presented the simulation and integration results of the SAP-1 project. All modules were verified individually and then integrated successfully. The Proteus simulation validated correct execution of ADD and SUB instructions and demonstrated proper coordination between datapath and control unit.

Chapter 10

Conclusion, Limitations, and Future Work

10.1 Introduction

This chapter concludes the SAP-1 project by summarizing the overall work carried out, highlighting the achievements of the design, discussing the limitations of the current implementation, and suggesting possible improvements and extensions for future work. The objective is to evaluate the project outcome against the original goals and reflect on the learning achieved through this implementation.

10.2 Project Summary

In this project, a **SAP-1 (Simple-As-Possible-1) computer architecture** was successfully designed and simulated using digital logic components. The system was implemented with a **4-bit datapath**, focusing on clarity and educational value rather than performance.

The key achievements of the project include:

- Design and implementation of core SAP-1 blocks such as:
  - Program Counter (PC)
  - Memory Address Register (MAR)
  - Static RAM (6116)
  - Instruction (Opcode) Register
  - Register A (Accumulator) and Register B
  - Arithmetic Logic Unit (74HC181)
  - Output Register
  - Shared system bus with tri-state buffers
- Implementation of a **hardwired control unit** with a **program sequencer**
- Successful execution of **ADD and SUB instructions**
- Verification of the complete system using **Proteus simulation**
- Proper handling of bus contention and timing through control signals

The project demonstrates how simple digital components can be combined to form a working microprocessor.

10.3 Achievement of Objectives

The objectives defined at the start of the project were achieved as follows:

- **Understanding CPU architecture:**
  The project provided a clear understanding of how a basic CPU is structured and how its components interact.

- **Datapath and control separation:**
  The separation between datapath (registers, ALU, bus) and control unit (sequencer and decoding logic) was successfully implemented.
- **Instruction execution:**
  ADD and SUB instructions were executed correctly using opcode decoding and ALU control signals.
- **Simulation-based verification:**
  All modules were tested individually and then as a complete integrated system in Proteus.

These outcomes confirm that the project meets the academic requirements of a SAP-1 implementation.

10.4 Limitations of the Project

Although the SAP-1 system was successfully implemented, some limitations exist due to the educational scope of the project:

1. **Limited Instruction Set**
   Only ADD and SUB instructions were implemented. More complex operations were not included to keep the design simple.
2. **Manual Opcode Loading**
   In the current stage, opcodes are entered manually through the bus for testing. Full automatic instruction fetch from RAM can be added as an extension.
3. **Limited Data Width**
   The system uses a 4-bit datapath, which restricts the numerical range and precision.
4. **No Conditional or Branch Instructions**
   Instructions such as jump, branch, or comparison are not implemented.
5. **Simulation-Based Implementation**
   The project is verified using Proteus simulation only. Physical hardware implementation may introduce additional challenges such as noise and timing delays.

These limitations are acceptable and expected for a basic SAP-1 educational project.

10.5 Learning Outcomes

This project provided significant learning outcomes, including:

- Practical understanding of digital logic design concepts
- Experience with registers, counters, ALU, and memory
- Knowledge of bus-based architecture and tri-state control
- Understanding of instruction sequencing and control logic
- Hands-on experience with Proteus simulation tools

The project helped bridge the gap between theoretical concepts and practical hardware realization.

10.6 Future Work and Enhancements

Several improvements and extensions can be made to enhance the current SAP-1 design:

- **Automatic Instruction Fetch:**
  Enable full PC → MAR → RAM → IR instruction fetch cycle to eliminate manual opcode entry.
- **Additional Instructions:**
  Implement INC, DEC, AND, OR, and other basic instructions.
- **Expanded Datapath:**
  Increase datapath width from 4-bit to 8-bit for larger data handling.
- **Program ROM:**
  Use ROM instead of RAM to store fixed programs.
- **Hardware Implementation:**
  Implement the design on breadboard or PCB for real-world testing.
- **Improved Output Display:**
  Use multiple 7-segment displays or LCD for better result visualization.

These enhancements would move the design closer to a more complete CPU architecture.

10.7 Final Conclusion

The SAP-1 project successfully demonstrates the internal working of a basic computer system using digital logic components. Through careful design of datapath, control unit, memory system, and bus architecture, the project achieved correct execution of arithmetic instructions.

Despite its simplicity, SAP-1 provides deep insight into how real processors operate at the hardware level. This project fulfills the academic objectives of understanding CPU fundamentals and serves as a strong foundation for studying more advanced computer architectures in the future.
