

PROJECT REPORT – SAP-1

DIGITAL LOGIC DESIGN
BSCS – CLASS OF 2026



DECLARATION

We, the undersigned members of the group "Boolean Builders," comprising Farah Inayat (26912), Muhammad Hammad Khan (26917), Zuha Aqib (26106), Rafsha (26639), Ibrahim Iqbal (27085), Abdul Rafay (25151), and Muddasir Javed (24463), hereby assert that the project report titled "SAP-1" reflects the collective effort and contributions of all group members. Each member has played a crucial role in the development and completion of the SAP-1 project, designed to provide a simplified representation for understanding how computers process data and execute instructions at a fundamental level. We confirm that the content of this project report is accurate, true, and original. Furthermore, we affirm that the project report is free from plagiarism, and all external sources of information are duly acknowledged in the bibliography provided at the end of this document. This declaration is made in accordance with ethical standards and underscores our commitment to maintaining academic integrity in our work.

TABLE OF CONTENTS

01

OVERVIEW

02

MORE ABOUT SAP

03

ACKNOWLEDGMENT AND
CONCLUSION

OVERVIEW OF OUR PROJECT

UNVEILING THE DIGITAL FRONTIER:

INTRODUCTION AND OVERVIEW

OVERVIEW

Introduction: SAP-1, a simplified representation designed to elucidate the fundamental processes of computer data processing and instruction execution, forms the core of our project. Our implementation involves a 4-bit architecture mirroring the SAP-1 model, featuring a Program Counter capable of holding up to 16 instructions. The primary focus of our SAP-1 model centers around executing addition operations. It's noteworthy that the project's implementation may evolve as we progress, with due notification to our instructors regarding any innovations.

Components:

Program Counter: The Program Counter (PC) plays a pivotal role in tracking program execution by storing the memory address of the next instruction to be fetched. Following the completion of a given instruction, the PC points to the address in main memory for the subsequent fetch operation.

Base Register: The Base Register (B-register) fetches the current instruction from the Program Counter, incrementing the Program Counter by 1. A single instruction comprises an opcode and a memory address for data retrieval.

Arithmetic Logic Unit: The Arithmetic Logic Unit (ALU) serves to execute arithmetic operations on provided data and instructions. Initially focused on addition, the ALU's functionality may adapt as the project progresses.

Accumulator: The Accumulator stores the output of the ALU. Future modifications to the accumulator's function are anticipated as the project advances.

Instructions to Perform: Our project involves the implementation of specific instructions, including:

Load: Fetches data from memory.

Add: Executes addition operations.

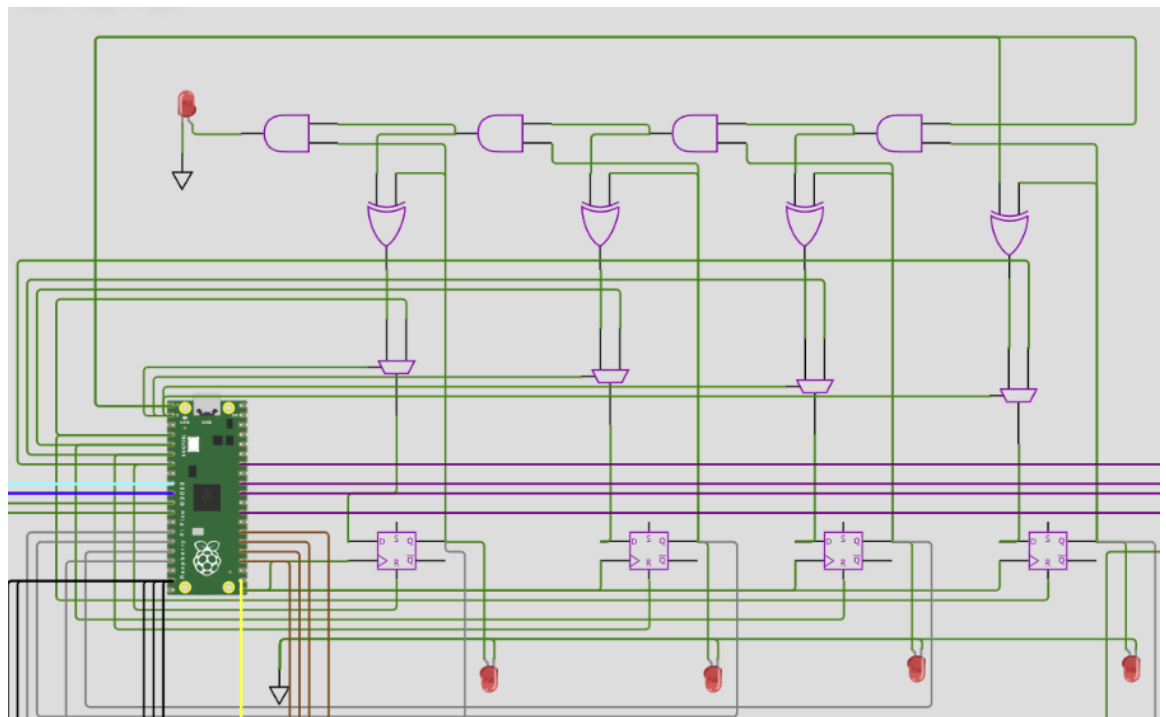
Output: Displays the output of the ALU, stored in the accumulator.

Conclusion: As we embark on this journey into SAP-1 architecture implementation, the flexibility of our design allows for adaptability and refinement. We commit to keeping our instructors informed of any developments and innovations that may arise throughout the course of the project.

MORE ABOUT SAP-1

”IN THE REALM OF
COMPUTING, SAP-1 STANDS
AS A TESTAMENT TO THE
VISIONARY WORDS OF ALAN
TURING: ’WE CAN ONLY SEE
A SHORT DISTANCE AHEAD,
BUT WE CAN SEE PLENTY
THERE THAT NEEDS TO BE
DONE.’”

Program Counter: The program counter is the component responsible for tracking the memory address of the next instruction to be fetched and executed. It functions as a register that stores the address of an instruction. In our project, we have implemented a 4-bit program counter utilizing 4 D flip-flops, 4 AND gates, 4 NAND gates, and 4 MUX. The enable signal, the reset input of each D-flipflop, and one of the inputs of each MUX are directly fed from the Raspberry Pi chip. The other MUX input is calculated through logic, involving the outputs of the D flip-flops, an AND gate, and a NAND gate, marking the initialization stage of the program counter. The outputs of the D-flipflops are then inputted into the Raspberry Pi chip, poised to be fetched by the B-registers.

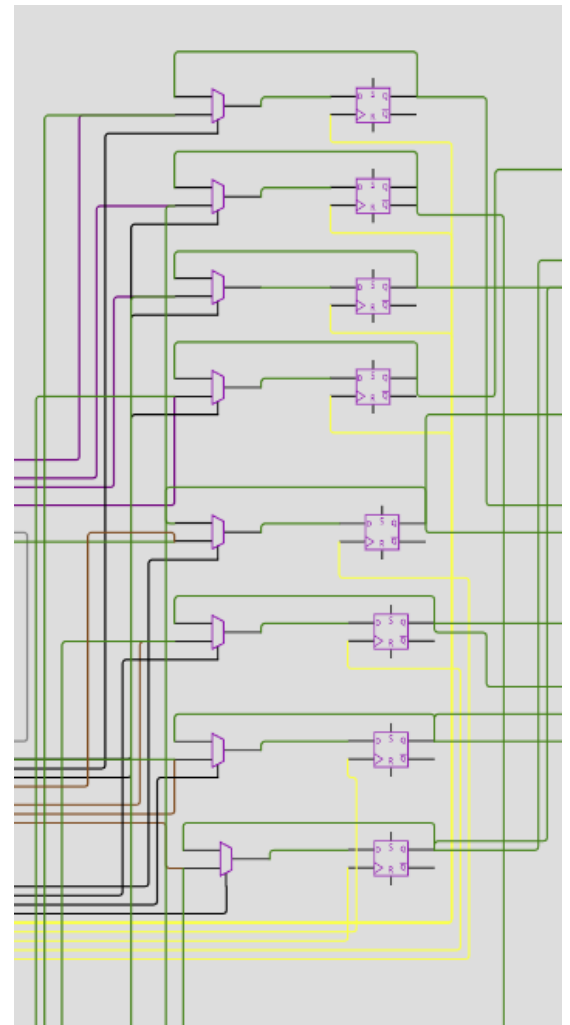


Base Register: The Base Register(B-register) is a register that temporarily holds the current instruction being executed. Our project incorporates two B-registers, essentially two 4-bit shift registers with parallel load capabilities. Initially, the 4 bits within the program counter are deciphered to generate the opcode. We have utilized if statements to generate the opcode, and the tables illustrate the data corresponding to each opcode and its significance.

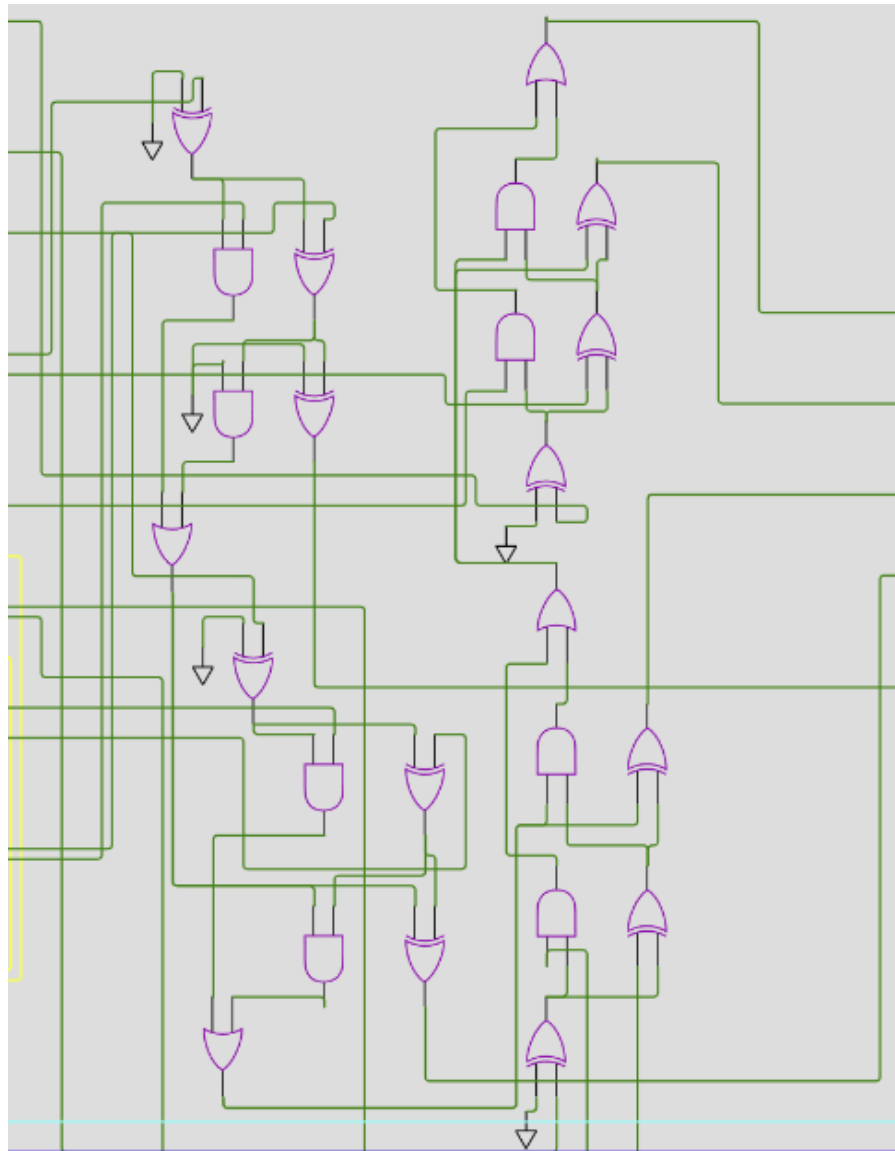
Program Counter Data	Opcode
0000	00
0001	00
0010	00
0011	00
0100	10
0101	10
0110	10
0111	10
1000	01
1001	01
1010	01
1011	01
1100	11
1101	11
1110	11
1111	11

Opcode	Instruction
00	Hold
01	data loaded into the top instruction register
10	data loaded into the bottom instruction register
11	Data in the two registers are added

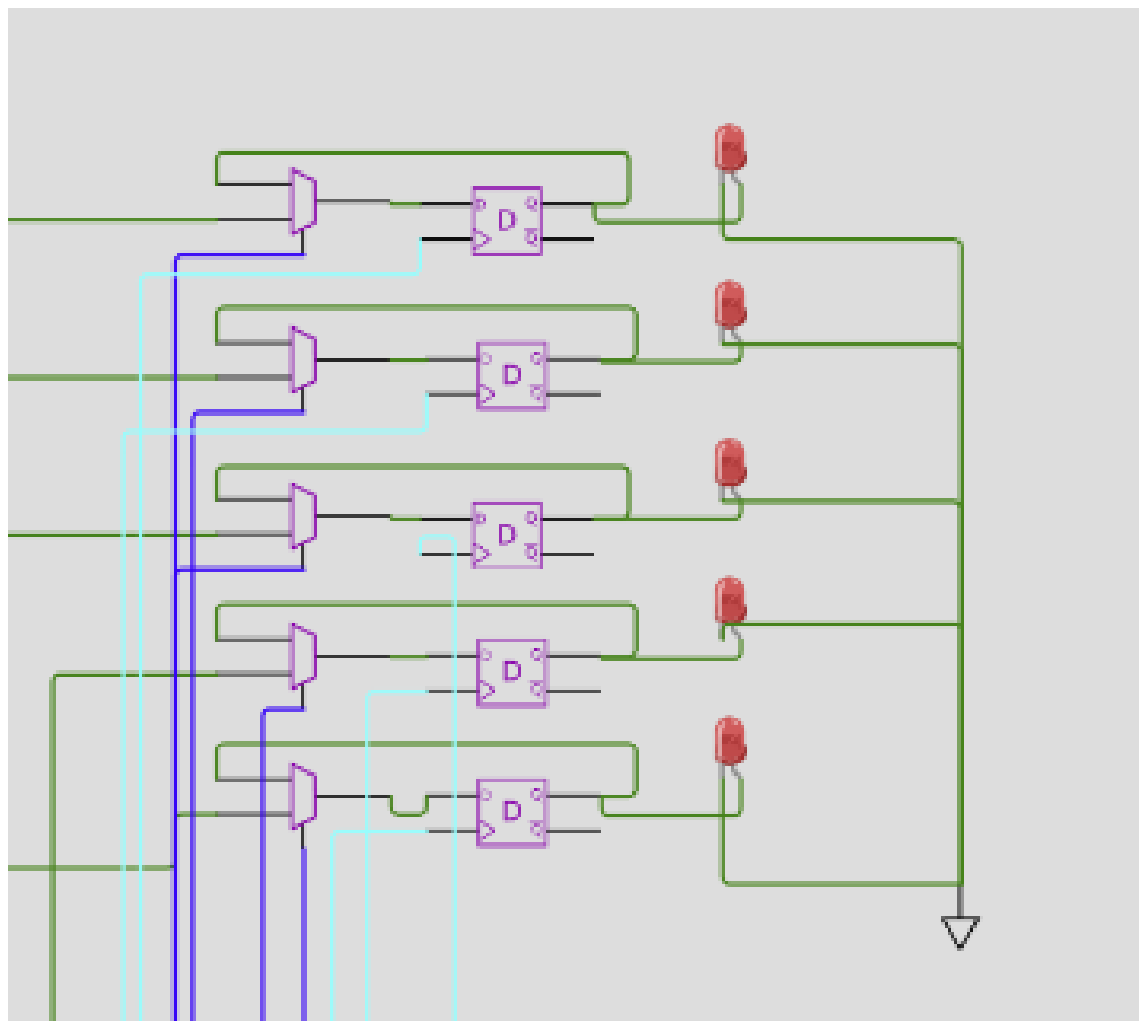
We have implemented bulbs to display the values of the opcodes, respectively. As explained earlier, our B-registers are 4-bit shift registers with parallel load functionality. Once the opcode is deciphered, the memory address is simulated and stored in a B-register based on the opcode. This is achieved through if-statements, with the opcodes serving as comparators. All the mux inputs share the same load input, a common feature in parallel load operations. The inputs of the mux include the outputs of the D-flip-flops and the simulated data from the memory. These registers hold the data, allowing the circuit to progress further.



ALU: The Arithmetic Logic Unit (ALU) is tasked with adding the data stored in the two B-registers. Our ALU operates by adding the data within the two B-registers and forwarding the output to the accumulator. The logic employed utilizes the outputs of the instruction registers and a 4-bit adder to calculate the output, subsequently routed to the accumulator. The circuit follows this structure:



Accumulator: The accumulator is a straightforward register designed to store the output of the ALU. In our implementation, we've employed a 5-bit shift register with parallel load to construct the accumulator. When the opcode is 11, the mux load input—directly sourced from the Raspberry Pi chip—is modified to 1, allowing the output of the ALU to be loaded into the accumulator. To visually represent the output of the SAP-1 logic, we've integrated five bulbs, each connected to the output of a respective D-flipflop. The circuit is depicted as follows:



WHAT WORKS AND HOW IT BEHAVES:

Simulated Circuit Behavior:

During simulation, the load and enable pins of the Program Mux are set to 1, initiating the program counter. In our circuit, the initial value of the program counter is 1000.

Following the initialization of the program counter pins, they are utilized to generate opcodes. In this scenario, the opcode is 01, as referenced in the table provided in the Base Register section.

Given the opcode of 01, arbitrary data from the simulated memory array is stored in the top Base register. In our simulation, the initial data is 1000, which is stored in the top BR.

Subsequently, the next instruction is initialized in the program counter, and steps 2 and 3 are reiterated, with the program counter value now set to 0100. As a result, the second arbitrary simulated memory value of 1110 is stored in the bottom Base register since the opcode this time is 10.

Once again, steps 2 and 3 are repeated. In the third run, the output of the program counter is 1100. The generated opcode is 11, prompting the addition of data in both Base registers via the ALU. The result is then stored in the accumulator. In this instance, the sum's value is 10110, visible in the bulbs connected to the D-flipflops.

Console Output: The console output is designed to display the contents of the Program Counter (PC) and the instruction after each iteration. In our simulated memory, we have represented it as follows: one out of the sixteen values are taken as input for the instruction registers.

```
OPCODE RECEIVED: 01 - data loaded into the top instruction register -  
1 0 0 0  
OPCODE RECEIVED: 10 - data loaded into the bottom instruction register -  
1 1 1 0  
OPCODE RECEIVED: 11 - DATA IS BEING ADDED  
Hold
```

```
#Memory simulation  
Memory = [  
    [0, 0, 0, 0], #0  
    [0, 0, 0, 1], #1  
    [0, 0, 1, 0], #2  
    [0, 0, 1, 1], #3  
    [0, 1, 0, 0], #4  
    [0, 1, 0, 1], #5  
    [0, 1, 1, 0], #6  
    [0, 1, 1, 1], #7  
    [1, 0, 0, 0], #8  
    [1, 0, 0, 1], #9  
    [1, 0, 1, 0], #10  
    [1, 0, 1, 1], #11  
    [1, 1, 0, 0], #12  
    [1, 1, 0, 1], #13  
    [1, 1, 1, 0], #14  
    [1, 1, 1, 1] #15  
]
```

[CLICK ON THIS TO VIEW OUR PROJECT](#)

ACKNOWLEDGEMENT, AND CONCLUSION

ACKNOWLEDGMENT

We begin by expressing our gratitude to **Allah Almighty**, whose guidance and blessings have been the cornerstone of our endeavors. We extend heartfelt thanks to our esteemed teachers, Sir **Zain Uddin**, Dr. **Faisal Iradat**, and Miss **Asma Touqir**, whose mentorship and guidance have been invaluable throughout this project. Their unwavering support and expertise have been instrumental in shaping our understanding and refining our skills.

We also extend our appreciation to our group members for their dedicated contributions, collaborative spirit, and hard work. Each member played a crucial role in the success of this project, and their commitment has been commendable.

Furthermore, we acknowledge the opportunity provided by **IBA**, which has been instrumental in our academic and professional growth. The conducive learning environment and support from the institution have been pivotal in fostering our intellectual development.

CONCLUSION

In conclusion, this project has been a journey of exploration and learning. The implementation of the SAP-1 architecture, the meticulous design of components such as the Program Counter, Instruction Register, ALU, and Accumulator, and the simulation of circuit behavior have deepened our understanding of computer systems and digital logic.

As we simulated the circuit behavior and observed the console output, we witnessed the tangible results of our efforts. This project has not only enriched our technical skills but also enhanced our problem-solving abilities and teamwork.

The experiences gained and lessons learned during this project will undoubtedly contribute to our future academic and professional pursuits. We are grateful for the opportunity to undertake this endeavor, and we look forward to applying the knowledge and skills acquired in future projects and endeavors.