Pamantasan ng Lungsod ng Maynila

SAP-1 Defense

Simple as Possible Computer

A Computer Architecture Presented to the
Faculty of Computer Science Department
College of Information System and Technology Management
Pamantasan ng Lungsod ng Maynila

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science in Computer Science

By

GROUP LEADER
Mark Daniel A. Apelledo / Shawn Kieffer E. Goyena

MEMBERS

Jonalene Ryza B. Abundo
Angel Lyn R. Camus
Hershey Anne P. Dalangin
Xienen Lei M. Gernale
Carla R. Mabutas
Mariebethel Roussamiere E. Miranda
Mariel Kim R. Vaflor
Venelyn Mae C. Cordova
Rita Angeli M. De Mesa
Shawn Kieffer E. Goyena
Lorelie Joy A. Musni
Sofia Alexi P. Navarro
Christian Andrei V. Santiago
Aliyah Loise C. Segovia
Maverick Isaiah A. Verdida

Prof. Elsa S. Pascual

ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to **Mr. Mark Daniel A. Apelledo**, our Head Consultant, for his invaluable guidance, and for overseeing the entire progress of this project. His expertise and have been instrumental in every step of our journey.

Our heartfelt thanks also go to **every member of the group**. Your commitment, hard work, and teamwork have made this project not only possible, but truly meaningful. Every late night, brainstorming session, and shared effort brought us closer to our goal.

We are sincerely thankful to **Ms. Mariebethel Roussamiere E. Miranda** for graciously providing accommodation throughout this project. Your kindness and generosity gave us a comfortable space to work and grow together as a team.

Special thanks to **Sir Froilan Cantillon**, whose YouTube videos served as a valuable guide in helping us better understand the concepts and steps necessary to complete our project. Your content made a real difference in our learning experience.

We also extend our gratitude to the **school library**, which became our second home during this project. Thank you for providing us an accommodating space where we could focus, collaborate, and bring our ideas to life.

To our **families**, thank you for your unwavering support—especially in covering necessary expenses and continuously motivating us throughout the process. Your love and sacrifices have meant everything to us.

To **everyone** who contributed in their own way—thank you. This project was made better because of you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

MEMBERS AND CONTRIBUTIONS

Member	Contributions
Jonalene Ryza B. Abundo	• Designed and assembled the Clock Module using the 555 timer, ensuring stable timing signals throughout the build. • Took the initiative to lead team meetings, manage progress, and assign tasks effectively to maintain smooth workflow and collaboration.
Angel Lyn R. Camus	• Played a key role in testing and debugging various components to ensure accurate execution. • Provided ongoing consultation and support during development stages to refine the functionality of each module.
Mark Daniel Apelledo	• Served as the Lead Debugger, meticulously testing and troubleshooting the entire SAP-1 system to ensure it functioned as expected. • Led the overall hardware integration of the SAP-1, overseeing connections and logic flow between components.
Hershey Anne P. Dalangin	• Focused on precise solid wiring of critical modules such as the ALU and control units. • Worked closely with the debugging team to validate circuit continuity and eliminate short connections.

Xienen Lei M. Gernale	• Assisted in solid wiring across major components including RAM, Program Counter, and Output Register. • Took responsibility for sourcing and purchasing electronic components essential for the project's completion.
Carla R. Mabutas	• Played a key part in the design and implementation of the Program Counter, contributing to the sequencing of instructions. • Helped verify and debug the RAM and MAR connections, ensuring correct data storage and retrieval.
Mariebethel Roussamiere E. Miranda	• Acted as the Lead for Physical Design and Packaging, ensuring that all modules were securely and neatly housed. • Designed and assembled the Instruction Decoder, making sure it interpreted machine code correctly into control signals.
Mariel Kim R. Vaflor	• Supported the development of the Program Counter by assisting with control signal wiring and testing. • Took part in the assembly of the main bus wiring and helped organize components logically on the breadboard.
Venelyn Mae C. Cordova	• Contributed to the consistent wiring of logic gates and connections in the Control Unit. • Helped validate outputs through manual checking and multimeter testing.
Rita Angeli M. De Mesa	• Played a key role in the layout and physical organization of wiring to minimize confusion during testing. • Assisted in troubleshooting connection mismatches during final integration.

Shawn Kieffer E. Goyena	• Participated in debugging the ALU and Control Matrix, focusing on signal accuracy and logical correctness. • Supported component testing by comparing simulated and actual outputs during early stages
Lorelie Joy A. Musni	• Helped build and test the Control Matrix, verifying the logic needed for instruction decoding. • Contributed significantly to wiring, particularly in connecting control signals to data paths.
Sofia Alexi P. Navarro	• Worked on designing and wiring the Control Matrix with a focus on opcode decoding. • Maintained documentation of wiring diagrams for better understanding and future debugging.
Christian Andrei V. Santiago	• Assisted in the testing and debugging of modules like RAM and Output Register. • Verified consistency of outputs during the RUN mode to ensure proper CPU behavior.
Aliyah Loise C. Segovia	• Built and tested the Accumulator, ensuring it could correctly store and pass data during computations. • Assisted with stable wiring across the bus lines, especially during final integration.
Maverick Isaiah A. Verdida	• Assembled the B Register and ensured it interfaced properly with the ALU for arithmetic operations. • Helped organize final testing and system walkthrough, ensuring each module worked under clocked conditions.

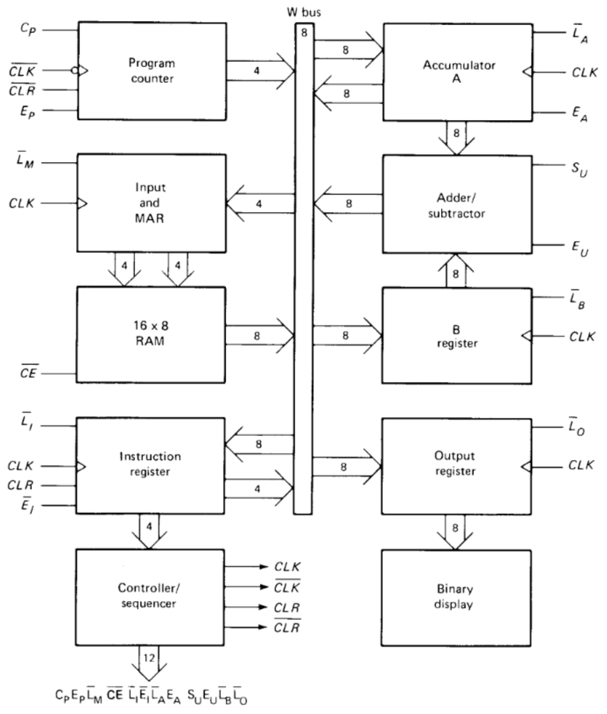
CHAPTER ONE

INTRODUCTION

1.1 Introduction

According to Albert Paul Malvino, the Simple-As-Possible (SAP)-1 computer is a very simple microprocessor model. The SAP-1 design includes the fundamental requirements for a working microprocessor. Its main goal is to give a fundamental grasp of how a microprocessor functions and communicates with memory as well as other system components like input and output. The instruction set is relatively limited and straightforward.

1.2 SAP-1 Introduction

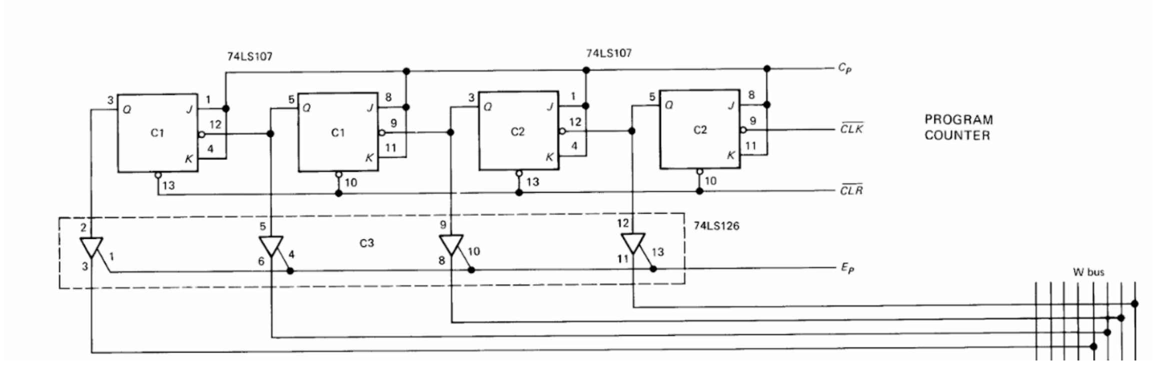


The SAP (Simple-As-Possible) computer has been designed for a beginner. The main purpose of SAP is to introduce all the crucial ideas behind computer operation without burying you in unnecessary detail. But even a simple computer like SAP covers many advanced concepts. To avoid bombarding you with too much all at once, we will examine three different generations of the SAP computer.

SAP-1 is the first stage in the evolution toward modern computers. Although primitive, SAP-1 is a big step for a beginner. So, dig into this chapter; master SAP-1, its architecture, its programming, and its circuits. Then you will be ready for SAP-2.

Program Counter

Figure 1.2: Program Counter



The Program Counter (PC) is a CPU register that stores the memory address for the next instruction to be retrieved and executed. It ensures that the CPU executes the instructions in the correct order by automatically incrementing after each fetch and pointing to the next instruction in memory. At the start of execution, the PC is set to the location of the first instruction, which is then stored in the Memory Address Register (MAR) for retrieval. As each instruction is fetched, the PC updates, allowing the CPU to run programs smoothly and orderly.

MUX and MAR

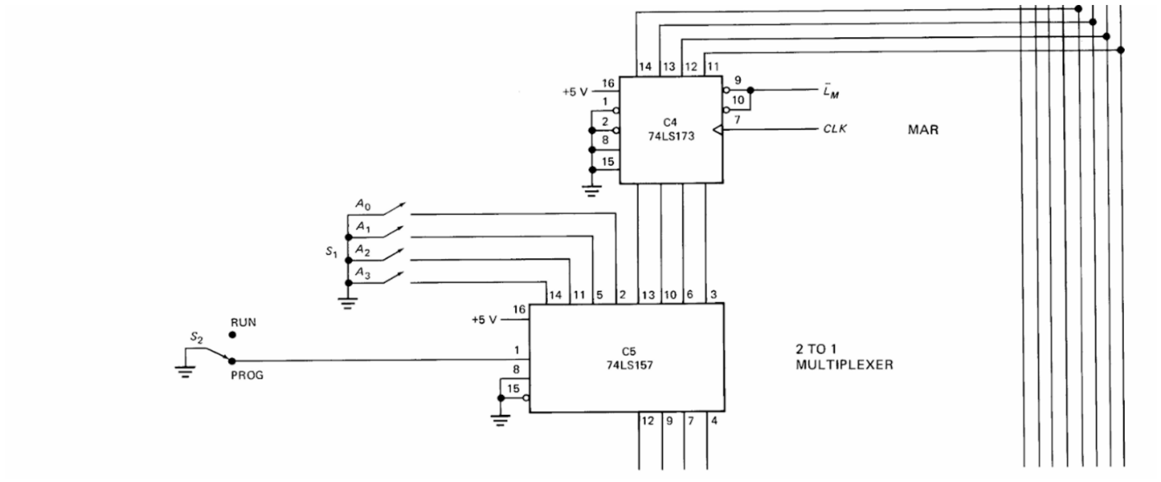


Figure 1.3: Multiplexer (MUX) and Memory Address Register (MAR)

The multiplexer (MUX) and memory address register (MAR) operate together to control the flow of memory addresses within the CPU. The MUX functions as a selector switch, selecting between the address provided by the program counter (PC) and a manual input, which is especially helpful for tasks such as debugging or manually loading programs. Once the address has been selected, it is passed to the MAR, which stores the memory address of the instruction or data to be accessed. This configuration ensures that the right memory location is reached when retrieving or storing data. During normal execution, the PC delivers the next instruction's address, which is selected by the MUX and stored by the MAR, allowing the CPU to operate smoothly and accurately.

RAM

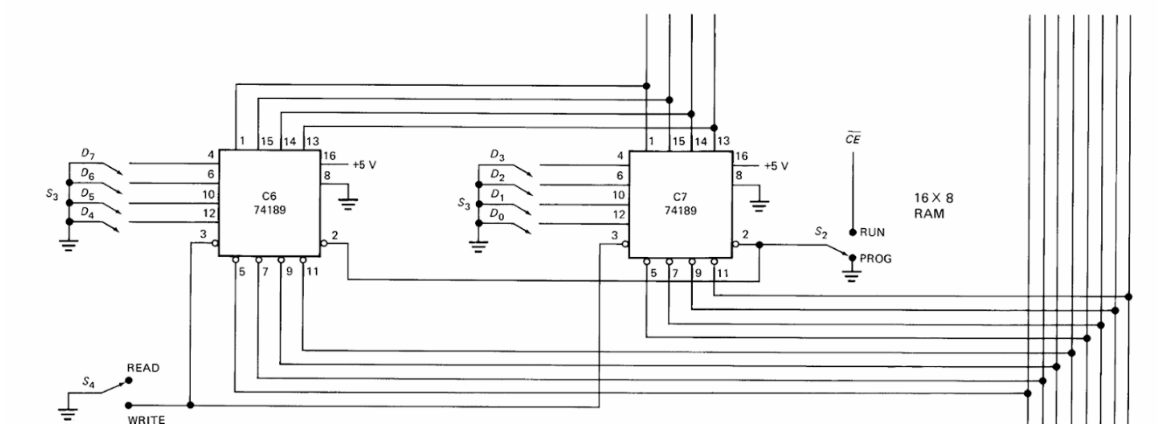


Figure 1.4: Random Access Memory (RAM)

Random Access Memory (RAM) is a component that stores both program instructions and the data required for execution. It normally has 16 memory locations, each of which may store an 8-bit instruction or piece of data. RAM serves as the computer's short-term memory, storing whatever the CPU requires throughout operation. The process begins with the Memory Address Register (MAR) providing the address of the required instruction or data. RAM then returns the value stored at that location, which is transferred to the CPU via the W Bus for processing, allowing for efficient and continuous program execution.

Instruction Decoder

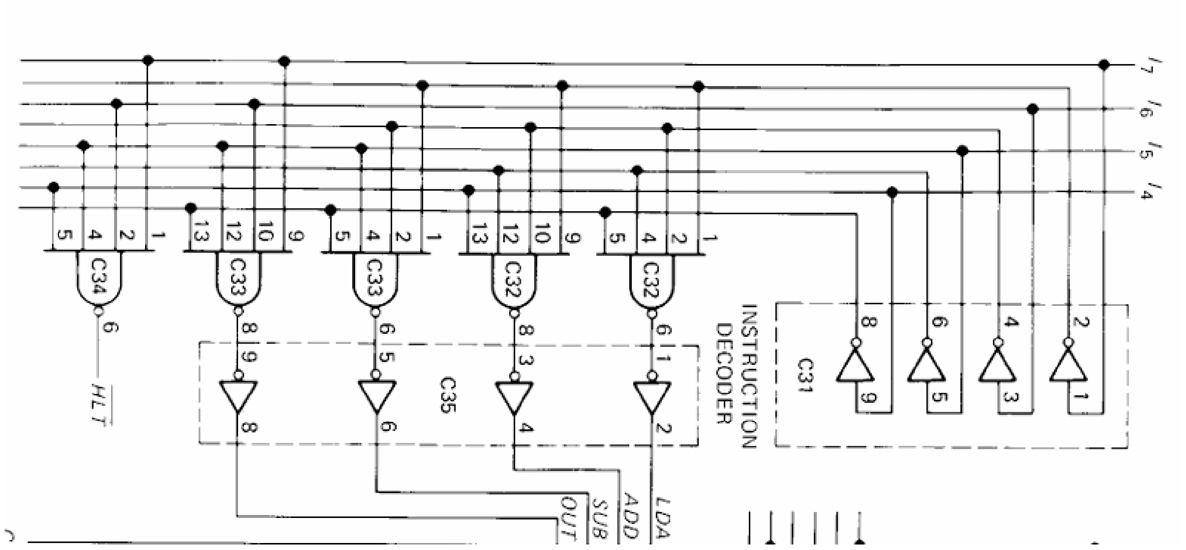


Figure 1.5: Instruction Decoder

The Decoder is a circuit that translates the opcode, or operation code, of an instruction to determine which action the CPU has to execute. It is required for converting binary instructions into control signals that activate the necessary CPU components. The procedure begins with the current instruction being stored in the Instruction Register (IR); the decoder then extracts the opcode, which is normally the first four bits. Based on this opcode, it generates the control signals required to do the connected action, such as loading data, adding, or saving a value, ensuring that instructions are executed accurately.

Instruction Register

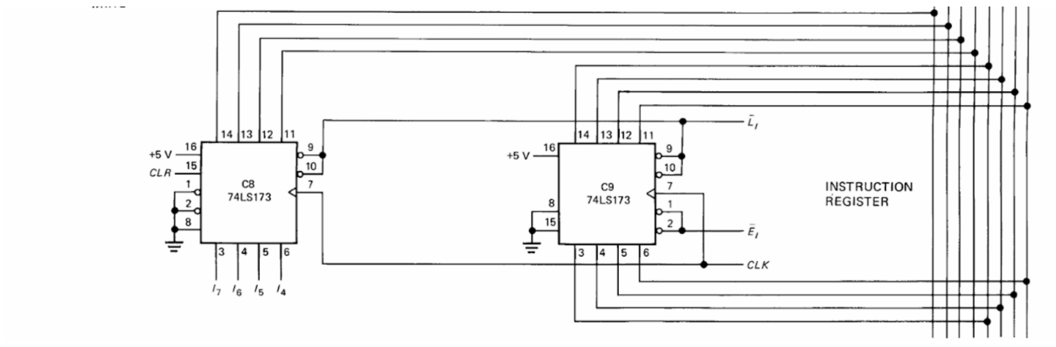


Figure 1.6: Instruction Register

The Instruction Register (IR) is a register that records the current instruction, including both the opcode and the operand. It is required to keep the fetched instruction available for processing, allowing the Instruction Decoder to interpret and execute it. The procedure starts with the instruction being fetched from RAM and stored in the IR. Once saved, the decoder reads the opcode from the IR and transmits the necessary control signals to carry out the relevant operation, ensuring that the CPU processes instructions correctly.

Ring Counter

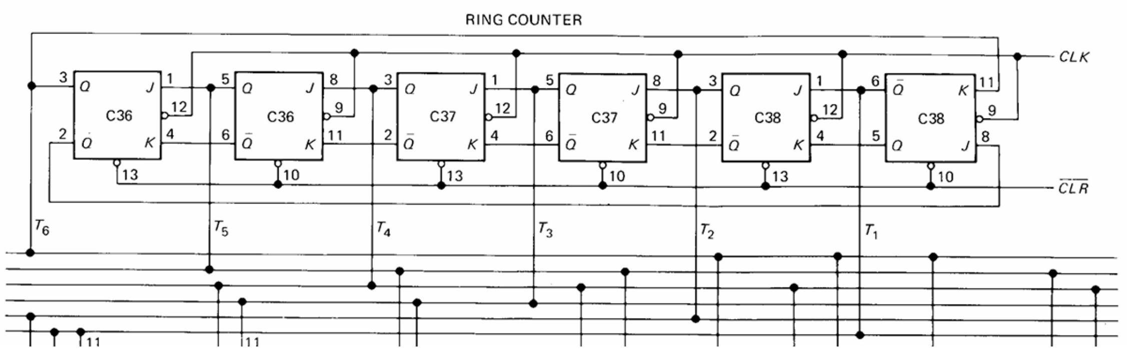


Figure 1.7: Ring Counter

A Ring Counter is a type of counter that cycles through a predetermined number of states, generating timing signals for each step of instruction execution. It assures the correct order of activities, such as fetch, decode, and execute, by acting as a step-by-step guide for the computer. The counter progresses through states T1, T2, T3, T4, and so on, with each state representing a certain portion of the instruction cycle. This cycle mechanism ensures that each action is carried out in the correct sequence, preserving the flow of instructions within the CPU.

Control Matrix

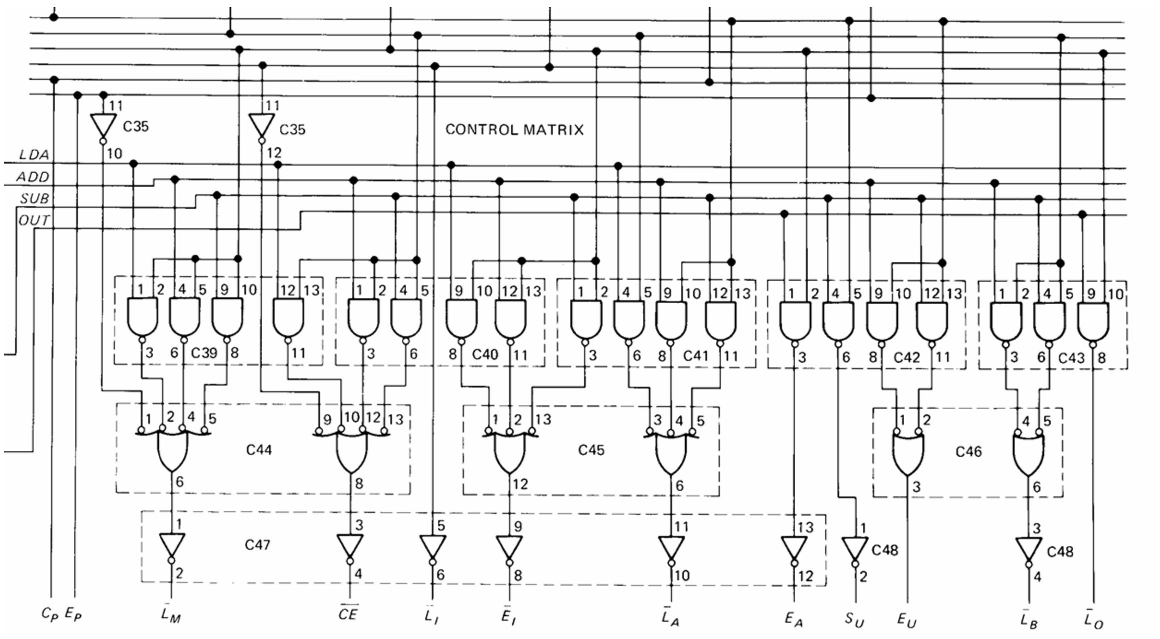


Figure 1.8: Control Matrix

The Control Matrix is a circuit that generates customized control signals using the opcode and ring counter. It is necessary to activate the correct components at the appropriate time, much like a conductor guiding an orchestra. The Control Matrix receives input from the Instruction Decoder and Ring Counter, and then sends control signals to registers, the ALU, and memory, ensuring that the CPU operates correctly at each stage of instruction execution.

W Bus (Data Bus)

The W Bus is the primary highway that connects all CPU components, transporting data and instructions between registers, RAM, and the ALU. It is necessary because without it, components would be unable to communicate with one another. The W Bus operates by allowing the active component to write data to the bus, which the destination component can then receive and use, consequently facilitating communication and data exchange within the CPU.

Accumulator (A Register)

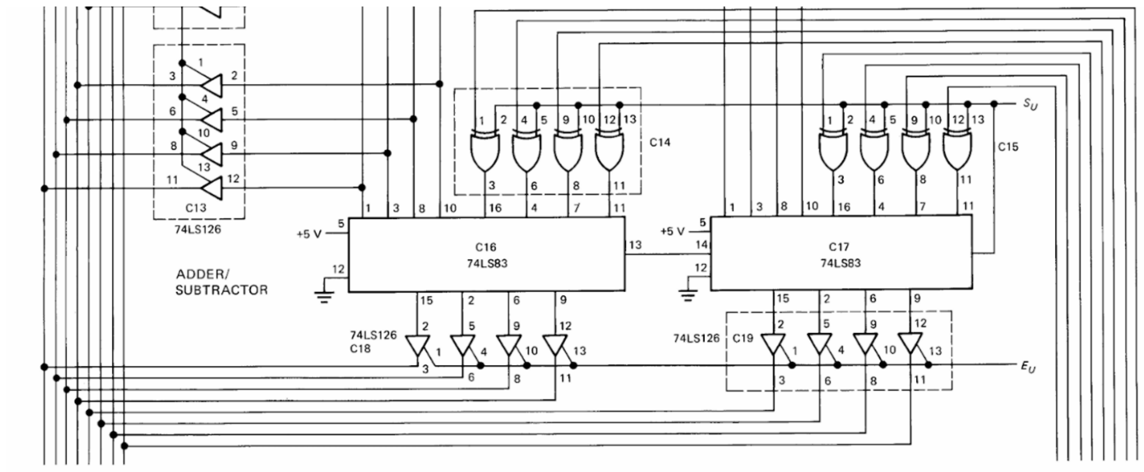


Figure 1.9: Accumulator (A Register)

The Accumulator (A Register) is a specialized register that stores the results of arithmetic and logic operations. It serves as temporary storage for ongoing calculations and is required for multi-step computations. After the ALU completes an operation, the result is stored in the Accumulator for later use, allowing the CPU to perform other actions depending on that result.

Adder/Subtractor (ALU - Arithmetic Logic Unit)

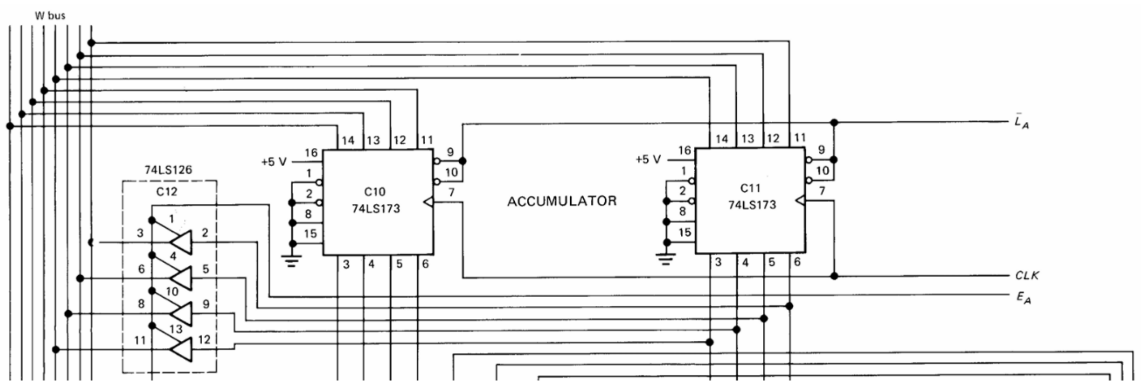


Figure 1.10: Adder/Subtractor

The **Adder/Subtractor (ALU - Arithmetic Logic Unit)** is the computational engine that performs addition and subtraction. It is required to do mathematical calculations within the CPU. The ALU accepts inputs from the Accumulator (A Register) and B Register, executes the addition or subtraction based on the instruction, and returns the result to the Accumulator for use in the next operation.

B Register

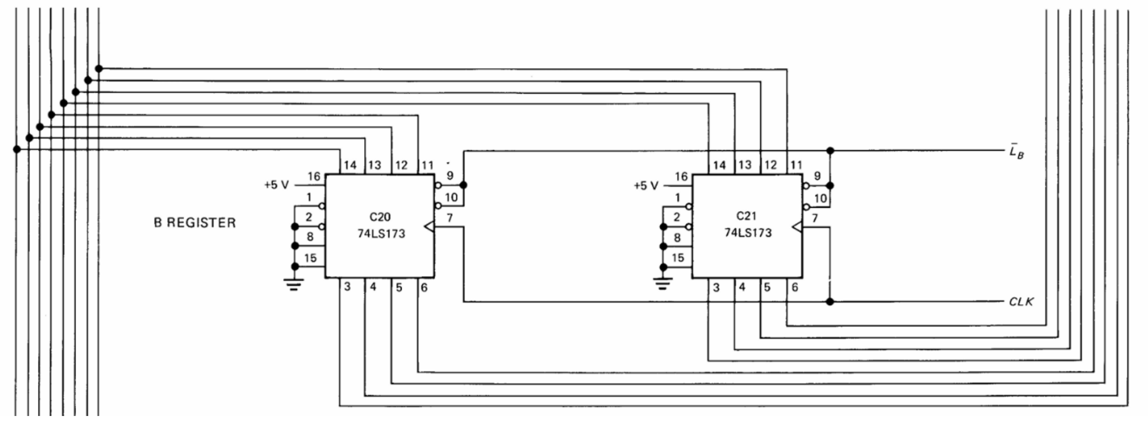


Figure 1.11: B Register

The B Register is a register that stores the second operand of ALU operations. It is required because two values are necessary for ALU operations, one of which is supplied by the B register. The B Register loads data from memory or the W Bus and passes it to the ALU for processing, allowing arithmetic and logical operations to be performed.

Output Register

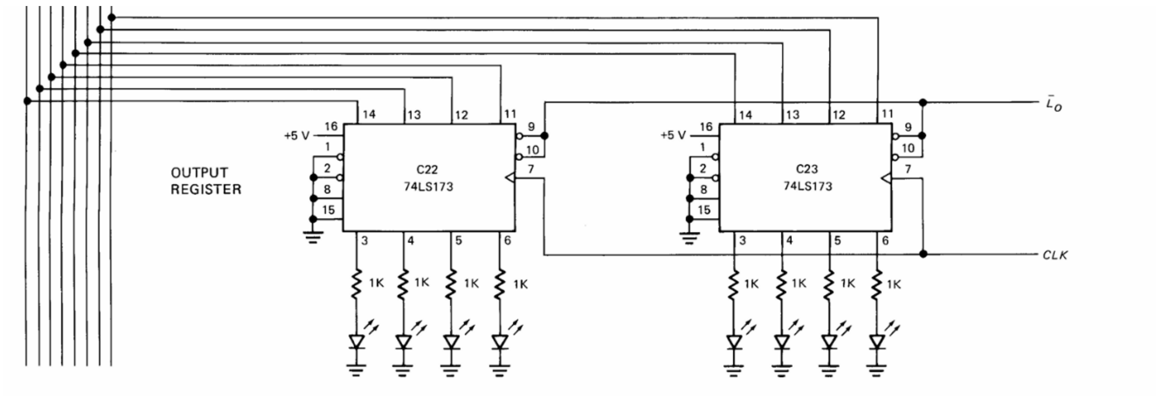


Figure 1.12: Output Register

The Output Register stores the final computed result before delivering it to an external display. It is necessary to verify that the findings are correctly stored before being shown. The process operates by saving the Accumulator's final value in the Output Register, which is then presented as output to the user or other devices.

CHAPTER TWO

DEFINITION OF TERMS

1. **CPU (Central Processing Unit)** - the brain of the computer is in charge of carrying out commands and carrying them out step by step.

2. **Opcode (Operation code)** - the portion of an instruction that instructs the CPU on what to do, like adding two numbers or loading a value.

3. **Operand** - the portion of an instruction that identifies the memory address or data used in the process.

4. **ALU (Arithmetic Logic Unit)** - a portion of the CPU that handles addition and other arithmetic operations. Only simple arithmetic is handled by the ALU in SAP-1.

5. **Arithmetic Operations** - simple arithmetic processes like subtraction and addition.

6. **Logical Operations** - binary logic-based operations include AND, OR, and NOT.

7. **Memory Address** - a specific place in RAM where instructions or data are kept.

8. **Instruction** - a binary-coded operation that the CPU performs using an operand and an opcode that are retrieved from memory.

9. **Fetch** - the procedure by which an instruction is stored in the Instruction Register (IR) after being retrieved from memory using the Program Counter (PC).

10. **Debugging** - the process of finding and correcting program errors include manually verifying control signals and register values.

11. **Data** - the computer processes or stores binary values.

12. **Program** - a set of instructions that are loaded into memory and are executed step-by-step by the CPU.

13. **Circuit** - an arrangement of the electronic parts that make up the SAP-1 system, such as flip-flops and logic gates.

14. **Binary** - a number system with just two digits: 0 and 1.

15. **Transmit** - the process of transmitting control signals or data between parts, like moving data from RAM to the accumulator.

16. **Control Signals** - data flow between SAP-1 components that is coordinated and managed by signals produced by the Control Unit.

17. **Execution** - the stage in which the CPU carries out the instruction, such adding or loading data into a register.

CHAPTER THREE

LIST OF TOOLS AND MATERIALS

2.1 List of Tools and Materials

#	Component	Parts
1	Clock	1x Breadboard 3x 555 1x 74LS04 2x LED Lights 10x 1k V Resistors 3x Capacitors 1x Switch 1x ??? 1x ??? Solid Wires
2	Program Counter	1x Breadboard 2x IC 74LS107 1x IC 74LS126 4x LED Lights 4x 1k V Resistors 2x Capacitors Solid Wires
3	MUX and MAR	1x Breadboard 1x IC 74LS173 1x IC 74LS157 4x LED Lights 4x 1k V Resistors 2x Capacitors 1x Button Solid Wires
4	RAM	1x Breadboard 2x IC 74LS189 8x LED Lights 8x 1k V Resistors Solid Wires
5	Instruction Register	1x Breadboard 2x IC 74LS173 8x LED Lights 8x 1k V Resistors Solid Wires

6	Instruction Decoder	1x Breadboard 2x IC 74LS04 3x IC 74LS20 Solid Wires
7	Ring Counter	1x Breadboard 3x IC 74LS107 6x LED Lights 6x 1k V Resistors Solid Wires
8	Control Matrix	4x Breadboards 6x IC 74CH00 2x IC 74LS04 1x IC 74LS20 1x IC 74LS10 12x LED Lights 12x 220 V Resistors Solid Wires
9	W Bus	13x Breadbords (Terminal) Solid Wires
10	Accumulator	1x Breadboard 2x IC 74LS173 1x IC 74LS126 8x 220 V Resistors 8x LED Lights Solid Wires
11	Adder/Subtractor	1x Breadboard 2x IC 74LS83 3x IC 74LS126 2x IC 74LS86 Solid Wires
12	B Register	1x Breadboard 2x IC 74LS173 8x 220 V Resistors 8x LED Lights Solid Wires
13	Output Register	1x Breadboard 2x IC 74LS173 8x 1k V Resistors 8x LED Lights Solid Wires

CHAPTER FOUR

CIRCUIT DESIGN

Program Counter

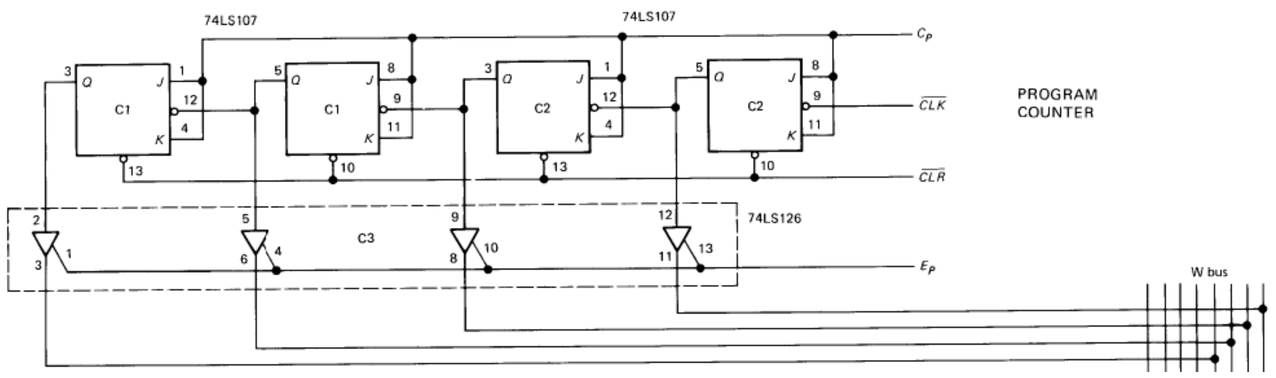


Figure 4.1: Program Counter

Memory Access Register

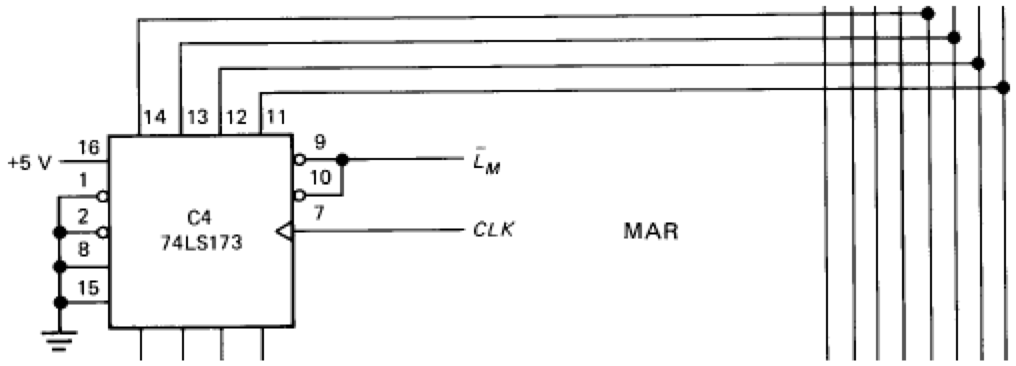


Figure 4.2: Memory Access Register

2 To 1 Multiplexer

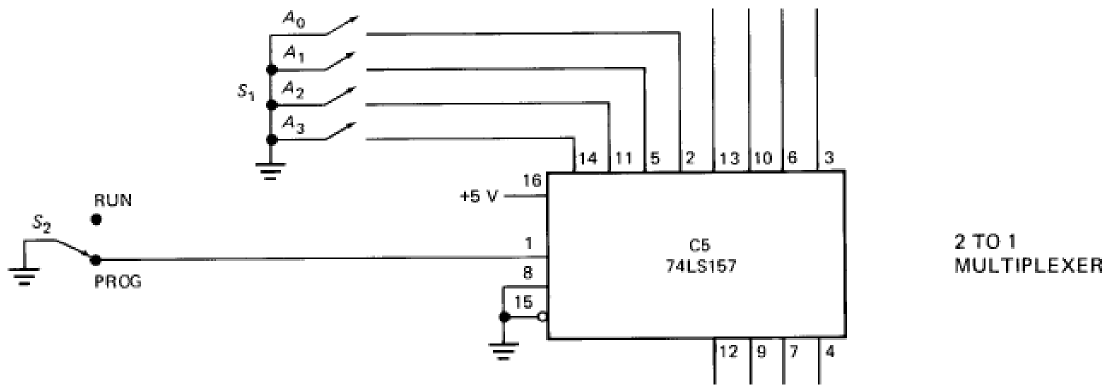


Figure 4.3: 2 To 1 Multiplexer

Random Access Memory

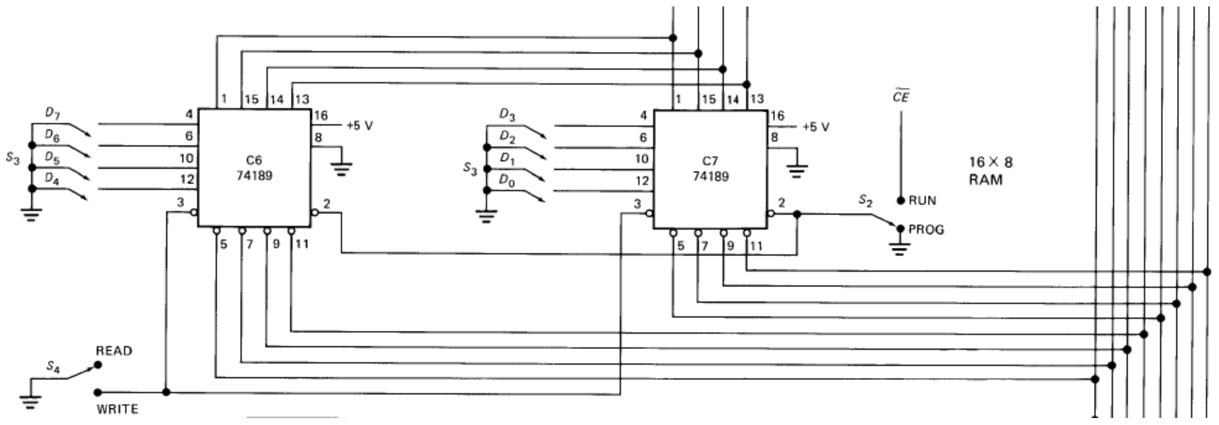


Figure 4.4: Random Access Memory

Instruction Register

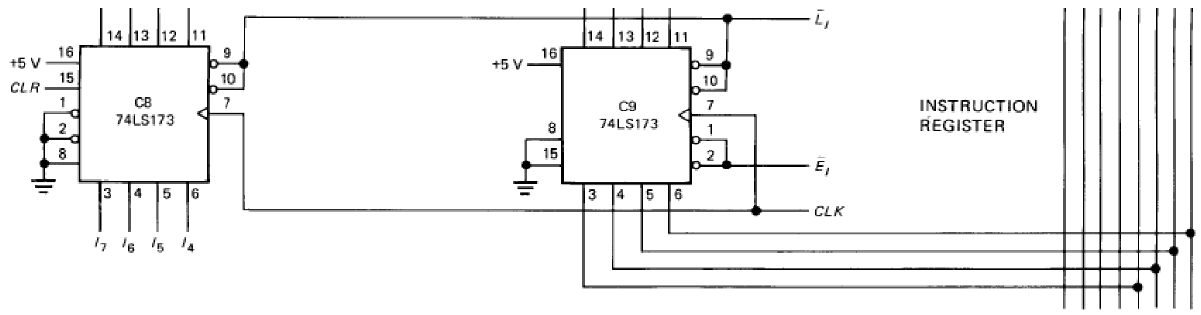


Figure 4.5: Instruction Register

Accumulator

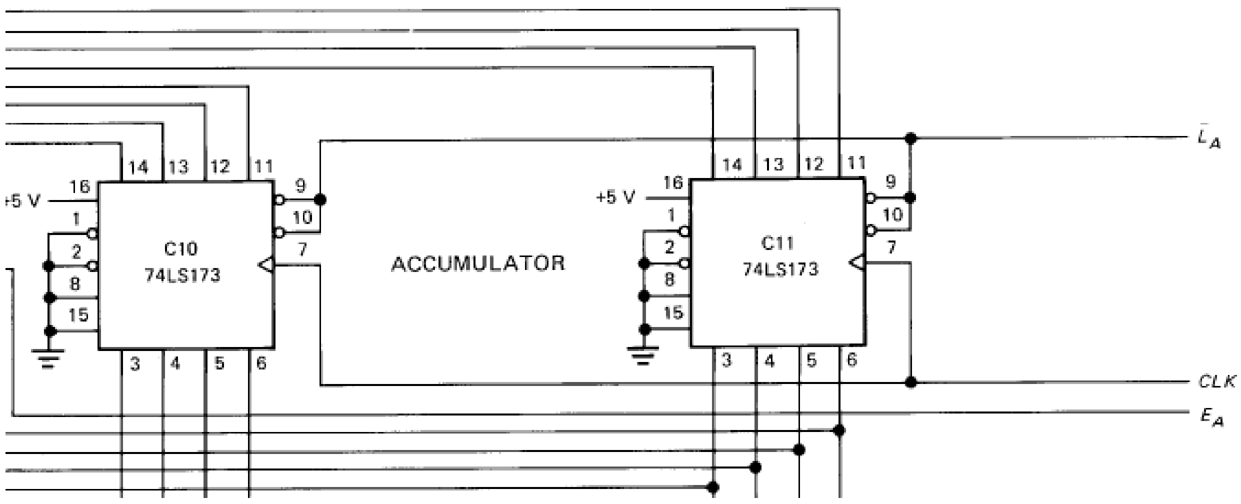


Figure 4.6: Accumulator

Adder-Subtractor

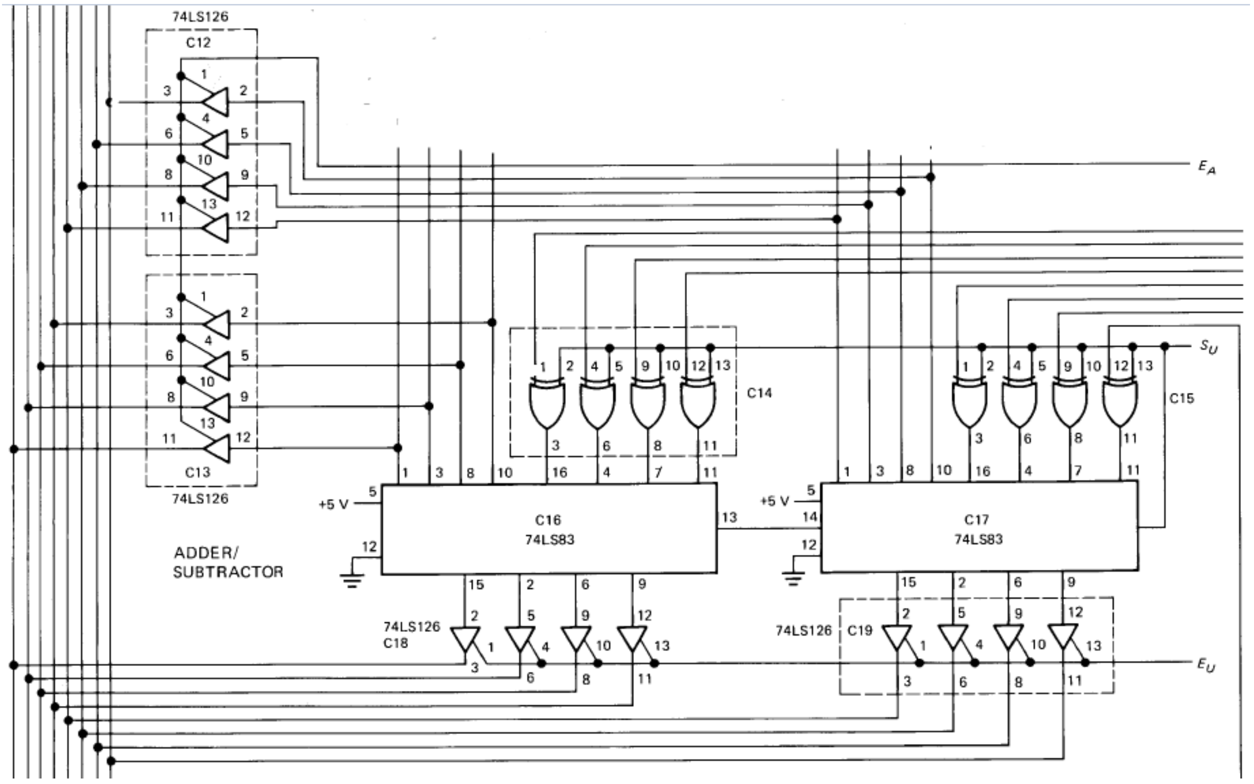


Figure 4.7: Adder-Subtractor

B Register

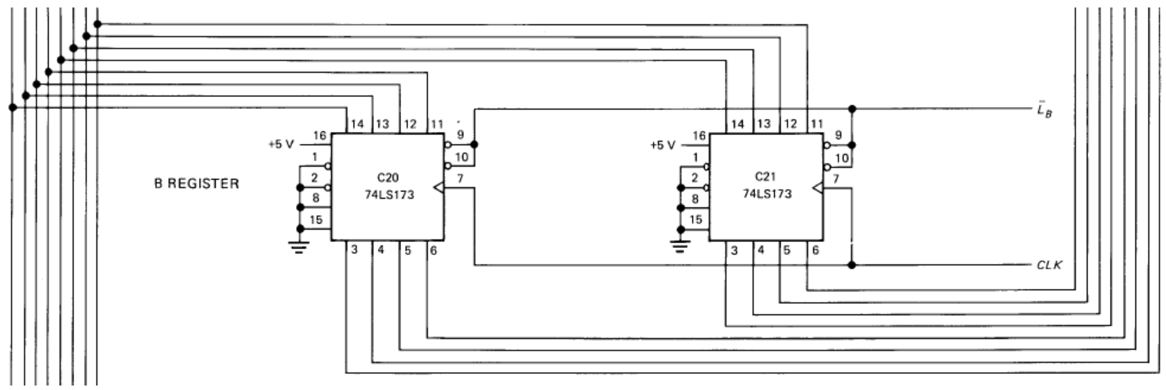


Figure 4.8: B Register

Output Register

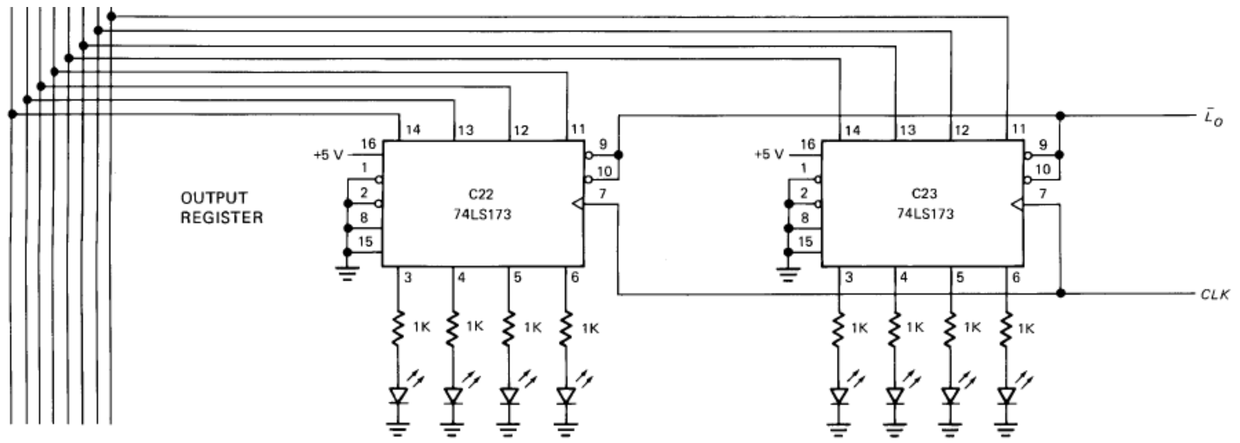


Figure 4.9: Output Register

Clock

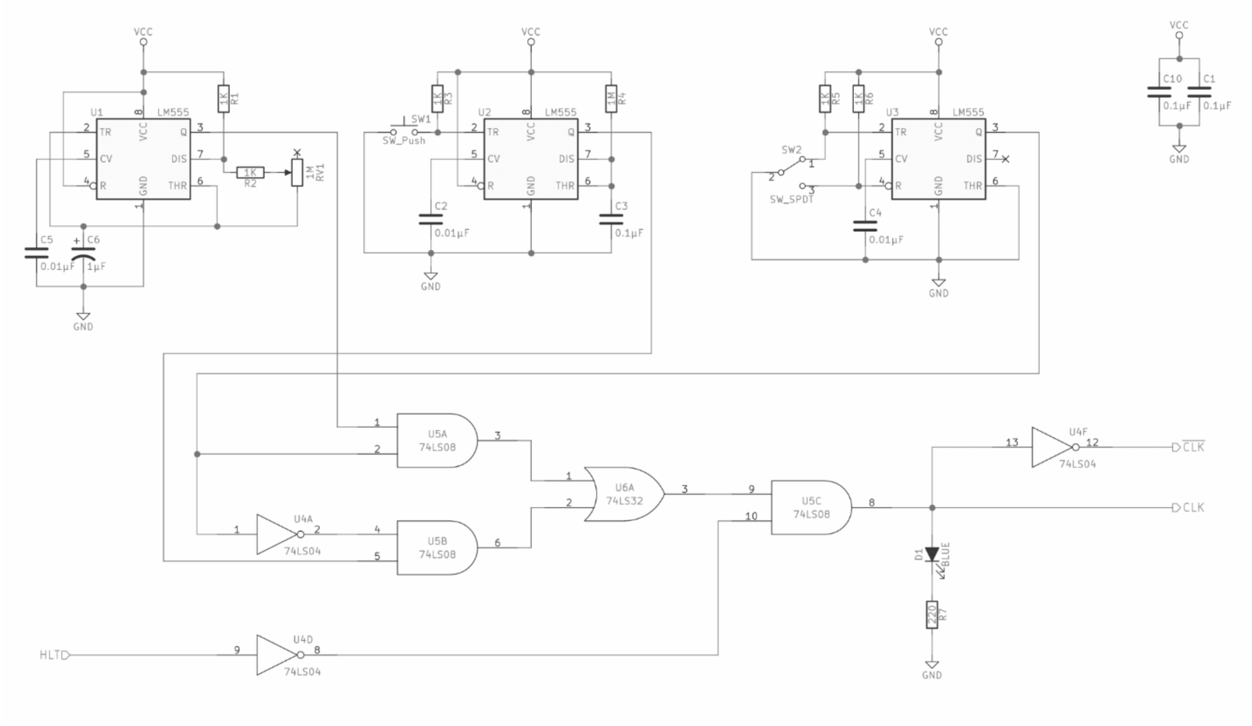


Figure 4.10: Clock

Ring Counter

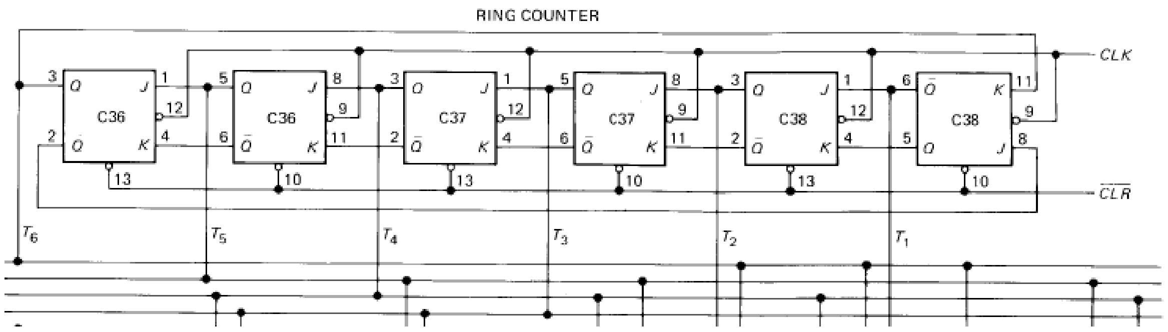


Figure 4.11: Ring Counter

Control Matrix

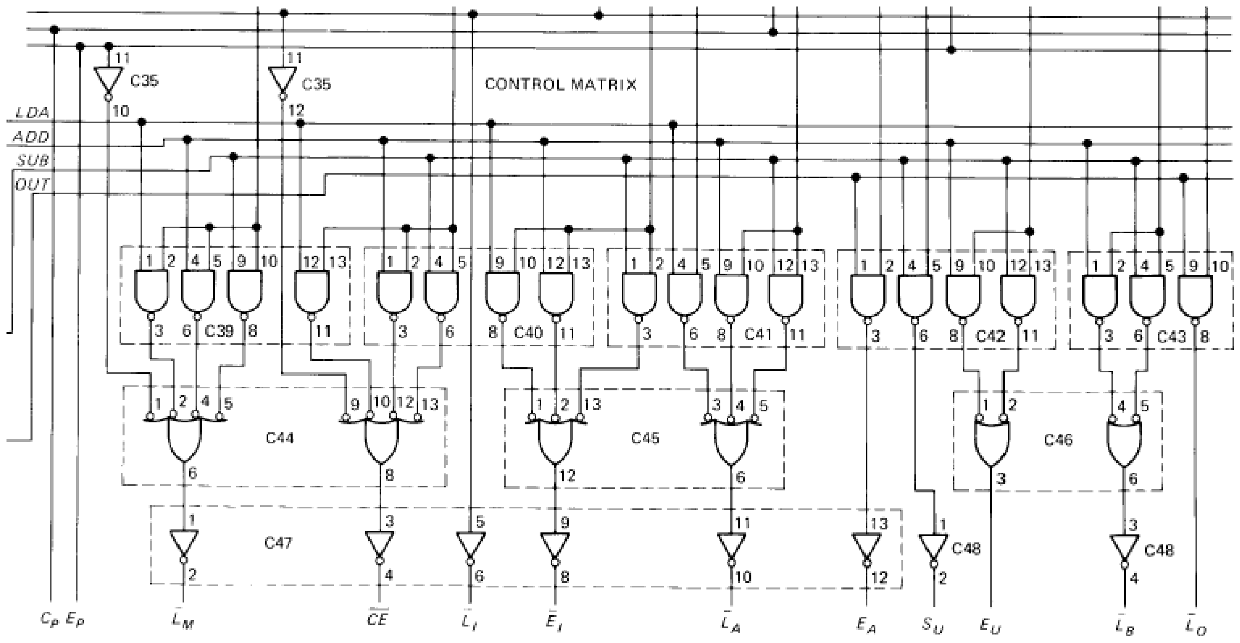


Figure 4.12: Control Matrix

Instruction Decoder

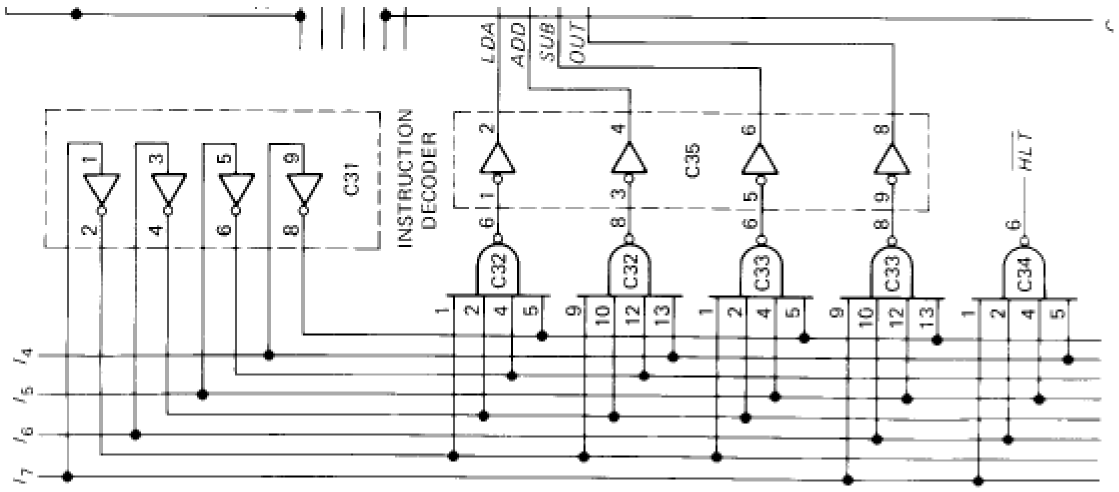


Figure 4.13: Instruction Decoder

CHAPTER FIVE

CIRCUIT DESIGN WITH IC PINS (BREADBOARD LAYOUT)

Program Counter

Pin Diagram:

1. 74LS107

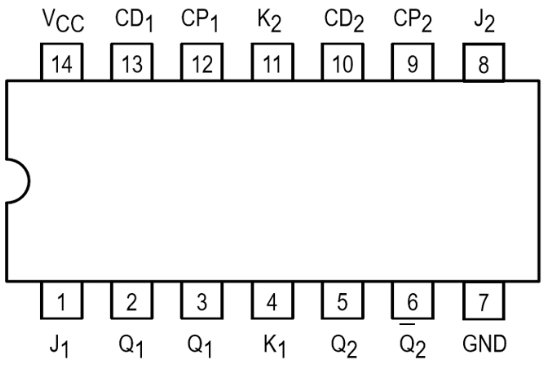


Figure 5.1: 74LS107

2. 74LS126

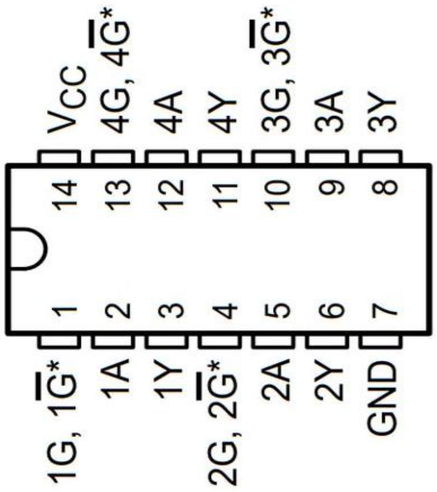


Figure 5.2: 74LS126

Memory Access Register

Pin Diagram:

74LS173

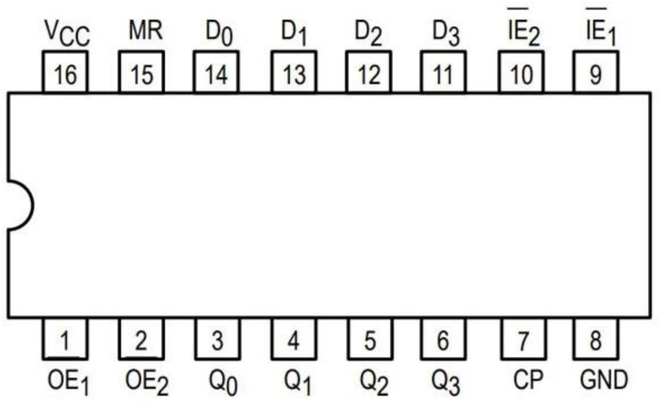


Figure 5.3: 74LS173

2 To 1 Multiplexer

Pin Diagram:

74LS157

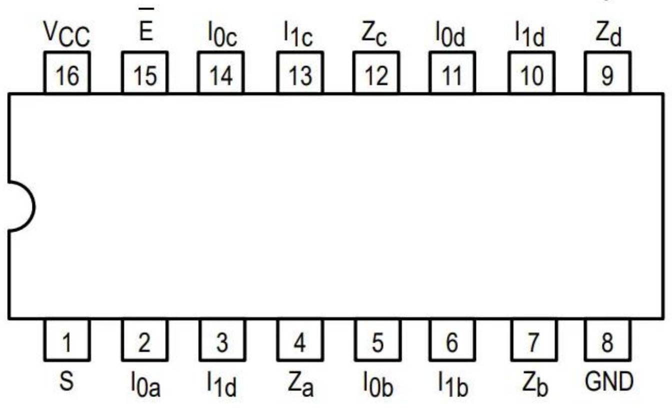


Figure 5.4: 74LS157

Random Access Memory

Pin Diagram:

74LS189

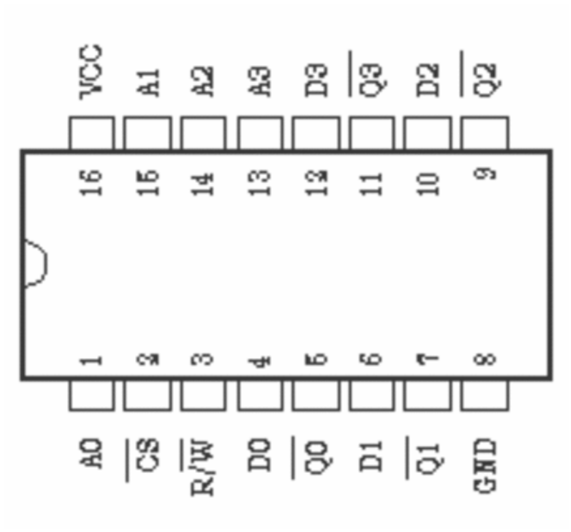


Figure 5.5: 74LS189

Instruction Register

Pin Diagram:

74LS173

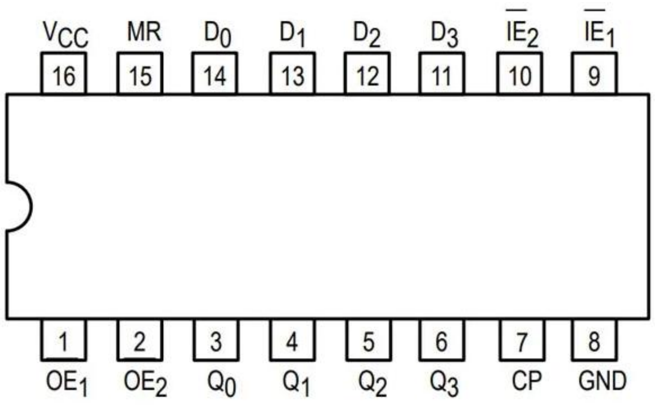


Figure 5.6: 74LS173

Accumulator

Pin Diagram:

74LS173

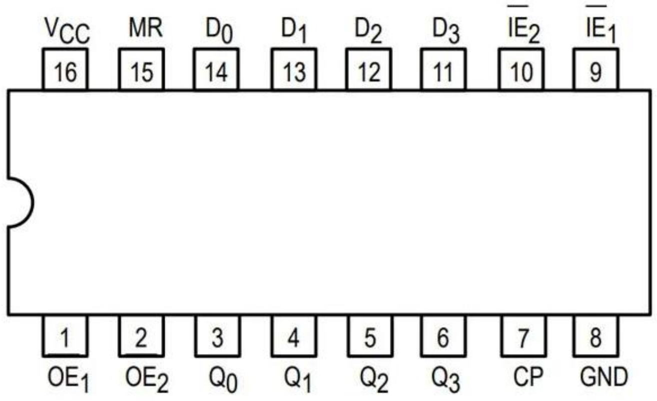


Figure 5.7: 74LS173

Adder-Subtractor

Pin Diagram:

1. 74LS126

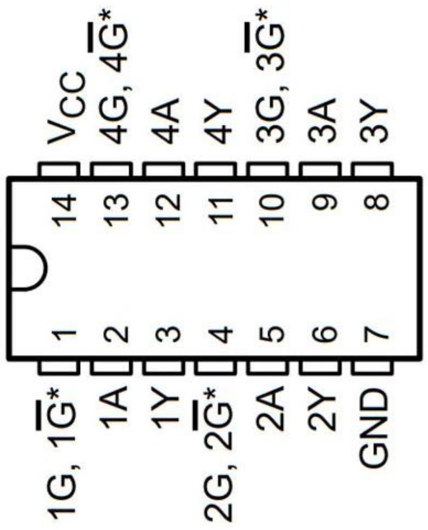


Figure 5.8: 74LS126

2. 74LS83

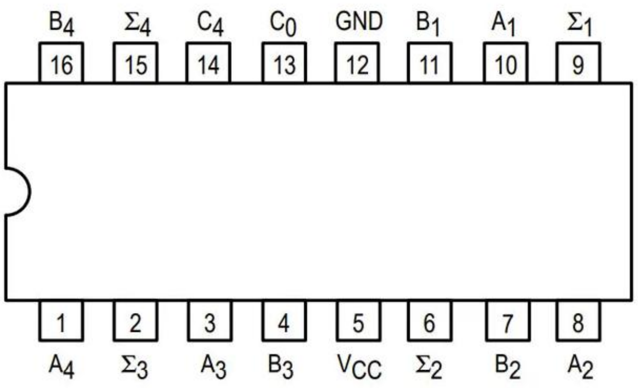


Figure 5.9: 74LS83

B Register

Pin Diagram:

74LS173

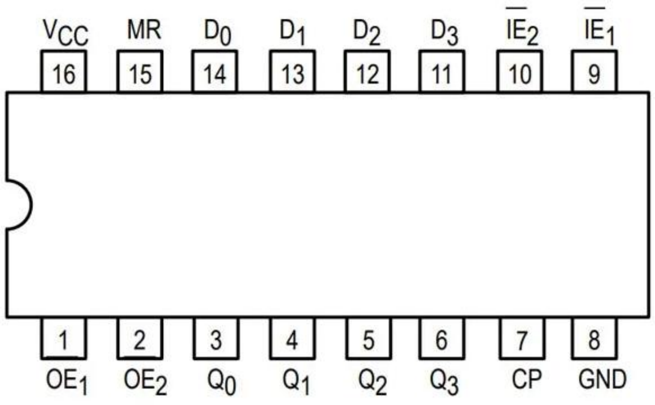


Figure 5.10: 74LS173

Output Register

Pin Diagram:

74LS173

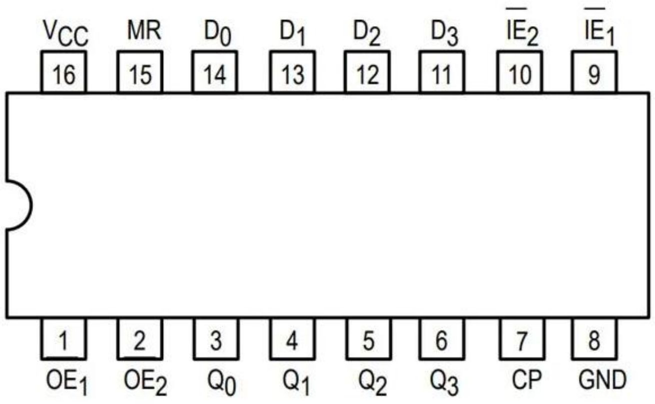


Figure 5.11: 74LS173

Clock

Pin Diagram:

1. 555

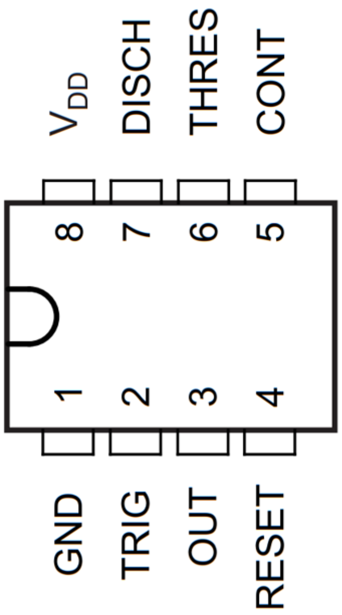


Figure 5.12: 555

2. 74LS04

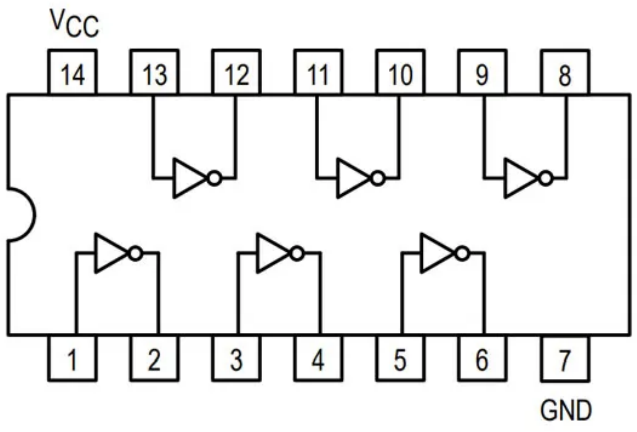


Figure 5.13: 74LS04

3. 74LS08

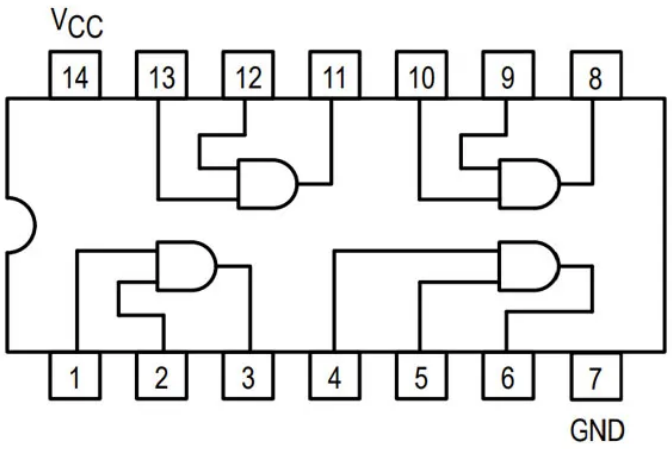


Figure 5.14: 74LS08

4. 74LS32

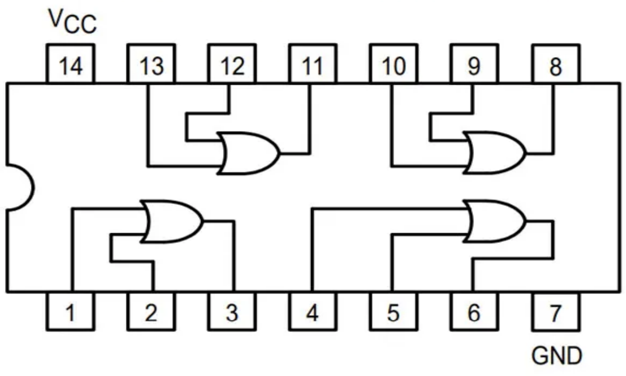


Figure 5.15: 74LS32

Ring Counter

Pin Diagram:

1. 74LS107

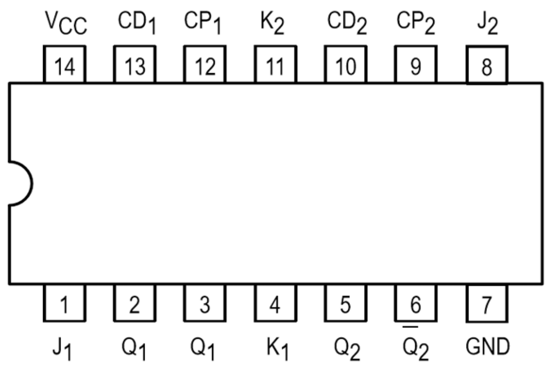


Figure 5.16: 74LS107

Control Matrix

Pin Diagram:

1. 74CH00

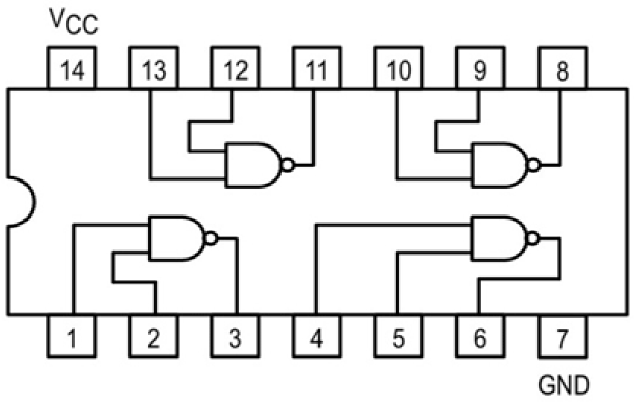


Figure 5.17: 74CH00

2. 74LS04

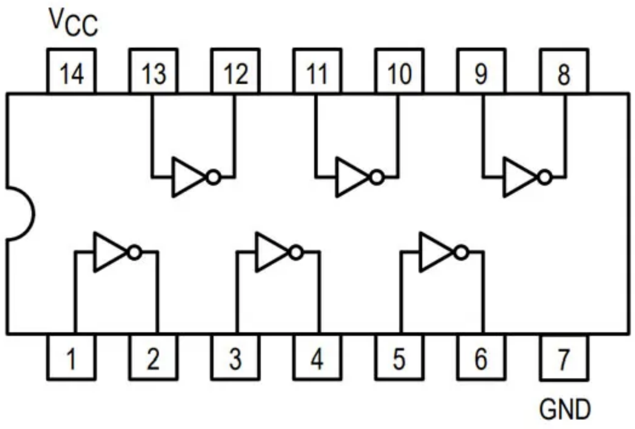


Figure 5.18: 74LS04

3. 74LS20

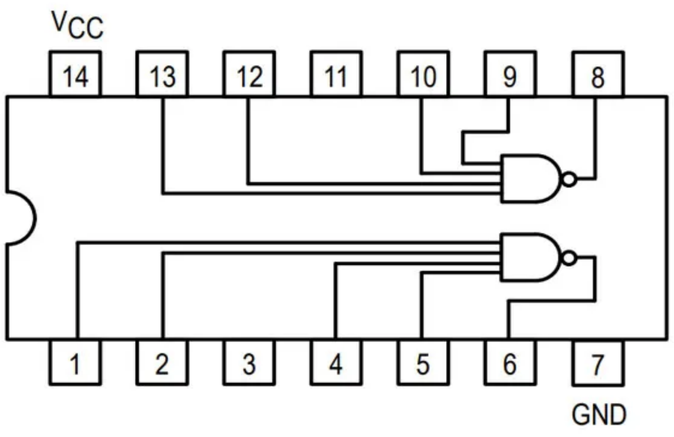


Figure 5.19: 74LS20

4. 74LS10

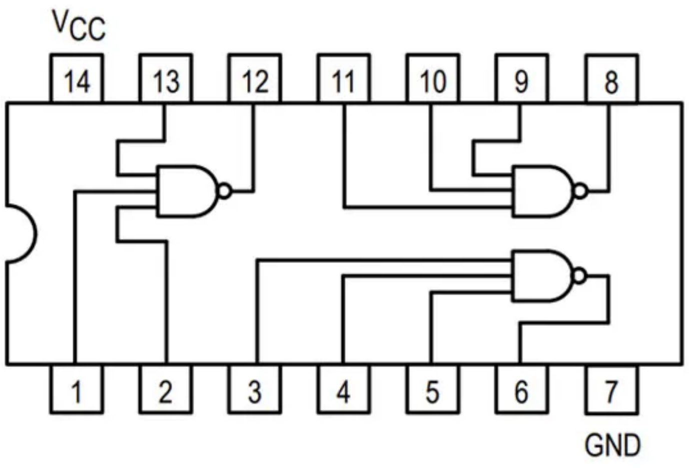


Figure 5.20: 74LS10

Instruction Decoder

Pin Diagram:

1. 74LS04

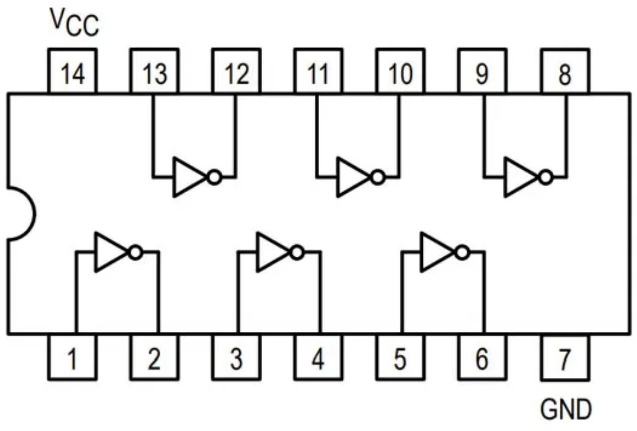


Figure 5.21: 74LS04

2. 74LS20

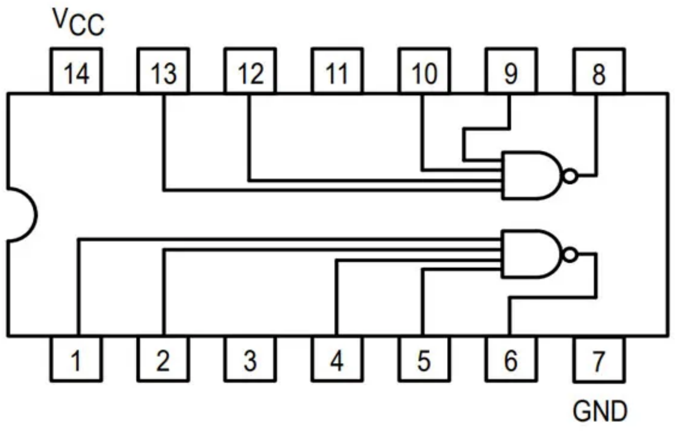


Figure 5.22: 74LS20

CHAPTER SIX

PROJECT DEMONSTRATION

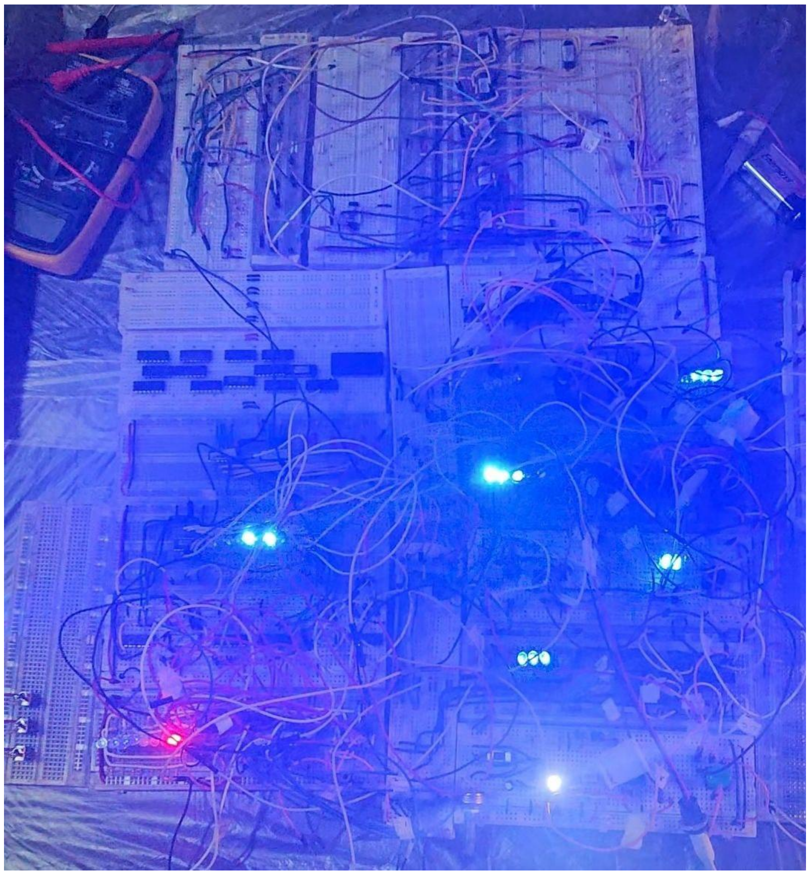


Figure 6.1: SAP-1 Hardware

LIST OF REFERENCES

Ben Eater. (2016). Build an 8-bit Computer From Scratch. https://eater.net/8bit/

.