# PROJECT-3: Design of simple 4-bit microprocessor

1. Introduction

The Simple 4-bit Microprocessor project aims to provide an understanding of fundamental computer architecture concepts. It is inspired by the Simple as Possible (SAP) model, which introduces the basics of how a computer operates, interacts with memory, and executes instructions.

1.1 Simple 4-bit Microprocessor

The SAP-1 microprocessor is a fundamental stage in digital system design. It can execute basic instructions such as loading data from memory, moving data between registers, performing arithmetic operations, and controlling program flow. This project focuses on designing and implementing a functional 4-bit microprocessor using logic gates and integrated circuits (ICs).

1.2 Macro Instruction Set

The microprocessor supports the following macro instructions:

| Sr. | Instruction | Opcode | Function |
|---|---|---|---|
| 1. | IN B | 0000 | Load External input into B register |
| 2. | MVI A, value | 0001 | Move the 4 bit value to A register |
| 3. | AND B | 0010 | Perform AND operation on data of B register with A register and store the result in the B register |
| 4. | JS M | 0011 | Jump to designated memory address M if SF=1 |
| 5. | OUT B | 0100 | Load data of B Register to output register |
| 6. | HLT | 0101 | Stop the program (No operation) |

Table 1: Instructions Of the Microprocessor

1.3 Programming SAP-1

Programming the SAP-1 involves creating a series of instructions for the microprocessor to execute. Each instruction represents an operation like loading data, performing arithmetic, or managing program flow. These instructions are stored in RAM, where the microprocessor retrieves and processes them sequentially.

| Instruction | Operation | Active Signals of T States | | |
|---|---|---|---|---|
| | | Fetch Cycle | | Execution Cycle |
| | | T1 | T2 | T3 |
| IN B | B ← input | Ep, $\overline{\text{LM}}$ | Cp, $\overline{CE}$, $\overline{LI}$ | T3 |
| MVI A, value | A ← value | | | Ex, $\overline{LB}$ |
| AND B | A ← A AND B | | | $\overline{EI}$ , $\overline{LA}$ |
| JS M | PC← M | | | $\overline{LB}$,Eu |
| OUT B | o/p ← B | | | $\overline{EI}$, $\overline{Lp}$ |
| HLT | No operation | | | EB |

Table 2: Programming Of the Microprocessor

2. Architecture

2.1 Block Diagram

**Group-A7-202214011,202214033,202214034,202214049,202214054**

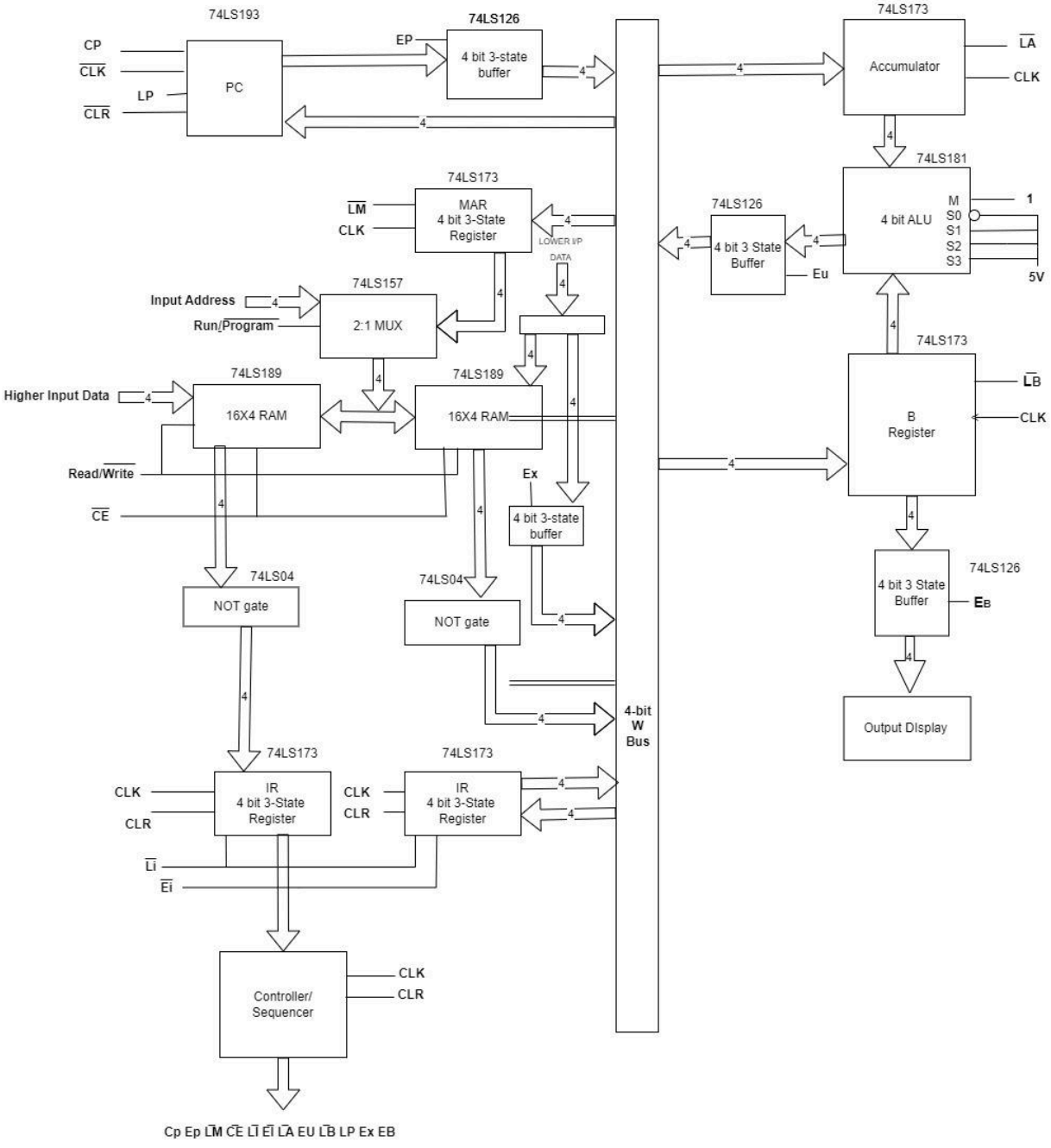


Image 1: Architectural Block Diagram Of the Microprocessor

The block diagram includes:

1. Registers (A, B, OUT, MAR, IR): Temporary storage locations for data and instructions during processing.
    - A & B: General-purpose registers for arithmetic and logic operations.
    - OUT: Stores final results before output.
    - MAR (Memory Address Register): Holds memory addresses for data access.
    - IR (Instruction Register): Stores the current instruction being executed.
    - ALU (Arithmetic Logic Unit): Performs arithmetic (addition, subtraction) and logical (AND, OR, NOT) operations.
2. RAM (Random Access Memory): Stores program instructions and data required for execution.
3. Program Counter (PC): Keeps track of the address of the next instruction to be fetched.
4. MUX (Multiplexer) & Buffers: Direct data flow and manage routing between components.
5. Control Unit: Generates control signals to coordinate operations of the CPU.

3. Fetch Cycle and Active Control Signals

The fetch cycle is responsible for retrieving an instruction from memory for execution. The process follows these steps:

1. PC to MAR: The Program Counter (PC) transfers the address of the next instruction to the Memory Address Register (MAR).
2. RAM to IR: The instruction stored at the address in MAR is fetched from RAM and placed into the Instruction Register (IR).
3. PC Increment: The Program Counter is updated to point to the next instruction.

Active Control Signals:

- PC_OUT: Sends the address from the PC to the MAR.
- MAR_IN: Allows the MAR to store the address from the PC.
- RAM_OUT: Transfers the instruction from RAM to the IR.
- IR_IN: Loads the instruction into the IR.
- PC_INC: Increments the PC to the next instruction address.

4. Design of Execution Unit

4.1 Program Counter, RAM, and ALU

- Program Counter (PC): A 4-bit register that keeps track of the address of the next instruction to be executed, ensuring sequential program flow.
- RAM (Random Access Memory): A 16x4-bit memory unit that stores both instructions and data, allowing the processor to retrieve and execute commands as needed.
- ALU (Arithmetic Logic Unit): Responsible for performing fundamental arithmetic operations such as addition and subtraction, as well as logical operations.

4.2 Registers (A, B, OUT, MAR, IR)

- A Register: Temporarily holds data that is used in arithmetic and logic operations.

- B Register: Stores operand values and intermediate results during computation.
- OUT Register: Holds the final computed values before they are sent as output.
- MAR (Memory Address Register): Stores the memory address of the data or instruction currently being accessed.
- IR (Instruction Register): Holds the current instruction being executed by the processor.

4.3 MUX and Buffers

- MUX (Multiplexer): A control unit that selects between the Program Counter (PC) and the Memory Address Register (MAR) to determine the memory address for instruction fetching.
- Buffers: Manage and regulate the flow of data between different components, ensuring smooth and controlled data transmission within the system.

4.4 Required ICs

| IC Name | IC Number | No of ICs |
|---|---|---|
| PC | 74193 | 1 |
| Register | 74173 | 5 |
| Buffer | 74126 | 4 |
| 2x1 MUX | 74157 | 1 |
| 4 bit ALU | 74181 | 1 |
| 16x4 RAM | 74189 | 2 |
| NOT | 7404 | 2 |
| AND | 7408 | 1 |
| | Total | 17 |

Table 3: IC Required For Execution Unit

5. Design of Control Unit

5.1 Ring Counter

Built using D flip-flops, it generates timing signals (T-states) for fetch and execution cycles.

5.2 Instruction Decoder

Interprets instructions from the IR and activates corresponding control signals.

5.3 Control Matrix

Maps instructions and timing states to the appropriate control signals for execution.

5.4 Required ICs

Includes essential integrated circuits for implementing control logic and sequencing. Below shows the circuitry of the control matrix.

Below Shows the image of the controller Sequence.

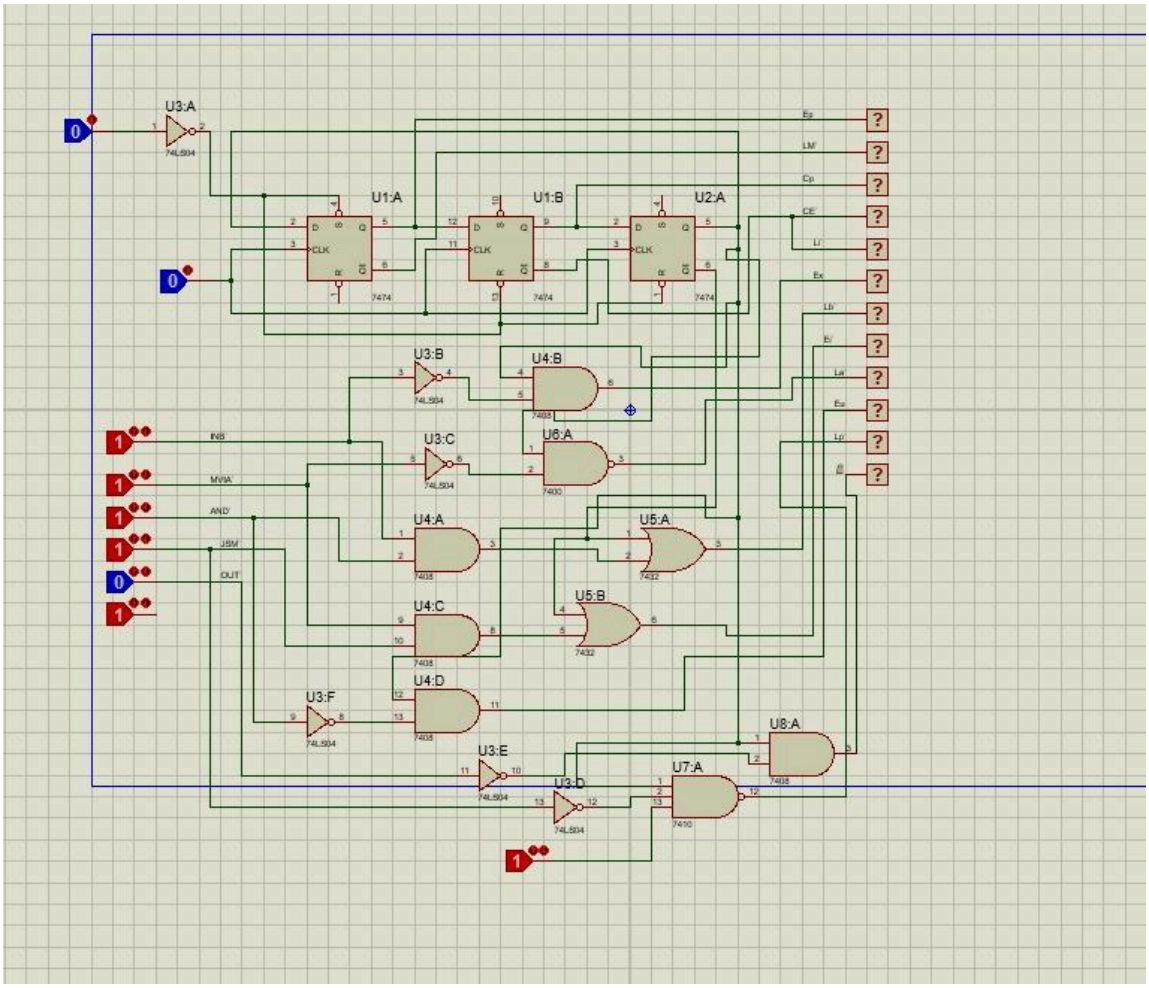


Image 2: Circuit Diagram For COntroller Matrix

| IC Name | IC Number | No of ICs |
|---|---|---|
| AND | 7408 | 2 |
| OR | 7432 | 1 |
| NOT | 7404 | 1 |
| NAND | 7400 | 1 |
| 3 input NAND | 7410 | 1 |
| D Flip Flop | 7474 | 2 |
| 3x8 Decoder | 74138 | 1 |
| | Total | 9 |

Table 4: IC Required For Control Unit

**Equations Of Signals ( Control Matrix )**

Ep = $T1$

$\overline{LM}$ = $\overline{Ep}$

Cp = $T2$

$\overline{CE}$ = $\overline{Cp}$

$\overline{LI}$ = $\overline{Cp}$

Ex = T3.IN B

$\overline{LB}$ = T3 ‘ + (INB’ . AND’)

$\overline{EI}$ = T3’ + (MVI A’ . JSM’)

$\overline{LA}$ = $\overline{T3.(MVI\ A)}$

EU = T3.AND B

$\overline{LP}$ = $\overline{T3.JS\ M.D3}$

EB = T3.(OUT B)

6. Problems Faced

1. Challenges Encountered
    - Timing Synchronization: Ensuring proper alignment of control signals with T-states posed difficulties.
    - Component Failures: Several ICs were found to be faulty, requiring repeated testing and replacements to maintain accuracy.
    - Signal Routing Complexity: Managing the intricate connections between components presented a significant challenge.
    - Aging Equipment Issues: The use of old IC and breadboards introduced internal capacitance and resistance problems, leading to diminished output signals in indicator bulbs.
2. Measures Taken for Improved Accuracy
    - Labeling for Clarity: Masking tape was applied to selection pins, switches, and input bits to minimize wiring errors and enhance clarity during the experiment.
    - Connection Verification: All circuit connections were tested twice, and output values were evaluated with multiple bit inputs before final assessment by the instructor.
    - IC Health Monitoring: Regular inspections were conducted to check for overheating, which could indicate circuit faults or overloads.

3. Simulation and Implementation Approach
    - Pre-Implementation Simulation: Before constructing the circuit, the entire model was first simulated in Proteus and verified under the instructor's guidance.
    - Selection of Wiring: Male-to-male jumper wires were predominantly used instead of twisted pair cables, as the latter were difficult to cut, position, and maintain. Additionally, twisted pair cables were prone to loosening at switch connections, increasing the risk of short circuits and open circuits. The use of jumper wires provided a more stable and efficient alternative.

7. Future Development

- Increased Bit Width: 8-bit or 16-bit enables complicated computation, more precision, and better memory addressing.
- Increased Instruction Set: Adding subtraction, multiplication, logical operations, and branching adds computational power and control.
- Pipeline Architecture: Overlap between fetch-decode-execution makes it more efficient and faster.
- Stack & Subroutines: PUSH, POP, and subroutine call support adds memory management and modular execution support.
- Memory & Addressing: More RAM and addition of direct/indirect addressing adds better data handling.
- Interrupt Handling: Interrupt support improves responsiveness to external events and asynchronous operations.
- Clock Optimization: Dynamic clock speeds improve speed and system stability.

8. Conclusion

The Simple 4-bit Microprocessor project is a culmination of the fundamental concepts of digital system design and microprocessor architecture. Through the designing, building, and debugging of the SAP-1 model, important lessons were learned about instruction execution, data processing, and control signal flow. The project not only reinforced theoretical understanding but also provided first-hand experience with debugging hardware elements and optimizing the efficiency of the circuits.

Upgrades in the future, such as bit width extension, instruction set enhancement, and the inclusion of advanced memory and processing methodologies, can significantly enhance its performance. By adding pipeline execution, interrupts, and better clock management, the system can be made into a more powerful and efficient microprocessor, bridging the gap between basic digital design and real-world computing applications.