Dr. B. R. Ambedkar National Institute of Technology, Jalandhar(Punjab)-144011

ADVANCED MICROPROCESSOR AND MICROCONTROLLER
ECPC-308

Report on ALU Design Using 8086 Emulator

Submitted to-
Dr. Kundan Kumar

Submitted by-
Aakriti
21104001
ECE (3rd year)

TABLE OF CONTENTS:

- Abstract
- Introduction
- Methodology
- Design Considerations
- Flowchart
- Results
- Limitations
- Future Scope
- Conclusion
- References

ABSTRACT:

This abstract encapsulates the essence of the project, providing a succinct overview of the design and implementation of an Arithmetic Logic Unit (ALU) within the 8086 emulator environment.

The project aimed to develop a functional ALU capable of performing arithmetic and logical operations, leveraging the features and capabilities of the 8086 emulator for simulation and testing. The methodology involved careful consideration of design requirements, selection of appropriate architectural elements, and implementation using assembly language programming.

The results demonstrate the successful creation of an ALU within the emulator environment, with performance metrics and functional validation confirming its effectiveness. The discussion highlights insights gained from the design process, areas for improvement, and implications for future development.

Overall, the project contributes to the understanding of ALU design principles and the practical application of emulation techniques in computer architecture.

INTRODUCTION:

The Arithmetic Logic Unit (ALU) is a crucial component in digital computing, serving as the computational nucleus within central processing units (CPUs). It executes arithmetic and logical operations for data processing and manipulation, shaping the performance and functionality of computing systems across various applications. The design of an ALU is a complex process that requires a nuanced understanding of digital logic, circuit design, and computational theory. The design journey includes considerations like word size, instruction set architecture (ISA), operand formats, and supported operations. The ALU's design is meticulously crafted to meet modern computing demands while optimizing speed, power efficiency, and resource utilization. Through a combination of schematic design, logic synthesis, and simulation, the report elucidates the architectural blueprint of an ALU, offering insights into its operational intricacies and design tradeoffs.

8086 EMULATOR:- An 8086 emulator is a software-based tool that simulates the behavior and functionality of the Intel 8086 microprocessor, which was one of the earliest 16-bit processors introduced by Intel in 1978. The emulator enables software developers, hobbyists, and educators to run programs written for the 8086 architecture on modern computer systems without the need for physical 8086 hardware.

METHODOLOGY:

The methodology employed for the design and implementation of the ALU involved several sequential steps. Initially, the functional requirements and design specifications were identified, outlining the desired operations and interface characteristics of the ALU. Subsequently, the ALU architecture was conceptualized, defining the internal structure, data paths, and control logic necessary to execute the required operations. The design was then translated into assembly language code compatible with the 8086 emulator, leveraging the emulator's instruction set and system architecture.

DESIGN CONSIDERATIONS:

Design specifications for an ALU typically encompass various aspects of its architecture, functionality, and performance. Here are some common design specifications:

• **Word Size:** Define the size of the operands and results processed by the ALU, typically specified in bits (e.g., 8-bit, 16-bit, 32-bit, or 64-bit).

• **Supported Operation:** Specify the arithmetic and logical operations the ALU should support, such as addition, subtraction, AND, OR, XOR, shifting, and comparison operations.

• **Instruction Set:** Define the instruction set architecture (ISA) that the ALU is designed to support. This includes specifying the format of instruction encoding, opcode assignments, and operand addressing modes.

• **Data Formats:** Specify the data formats supported by the ALU, including unsigned integers, signed integers (two's complement), floating-point numbers, and binary-coded decimal (BCD) representations

• **Performance Metrics:** Define performance requirements in terms of throughput, latency, and power consumption. This may include target clock frequency, maximum propagation delay, and power dissipation constraints.

• **Precision and Accuracy:** Specify the desired precision and accuracy of arithmetic operations, particularly for floating-point arithmetic. This may include the number of significant digits, rounding modes, and handling of special cases (e.g., overflow, underflow).

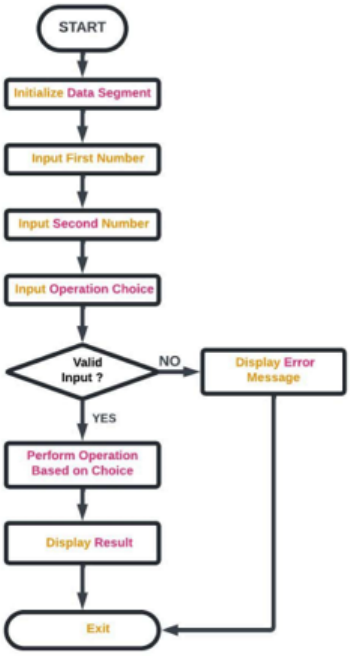
• **Input/Output Interfaces:** Define the input and output interfaces of the ALU, including the format of input data, control signals, and status flags. Specify how results are communicated to the rest of the CPU or system.

• **Resource Utilization:** Specify constraints on the use of resources such as logic gates, registers, and memory elements. This may include limits on-chip area, transistor count, and silicon area.

• **Technology Constraints:** Consider constraints imposed by the fabrication technology, such as transistor sizes, voltage levels, and manufacturing process variations.

• **Compatibility and Interoperability:** Ensure compatibility with existing CPU architectures, instruction sets, and software ecosystems. Specify any interoperability requirements with other system components, such as memory units and input/output devices.

FLOWCHART:

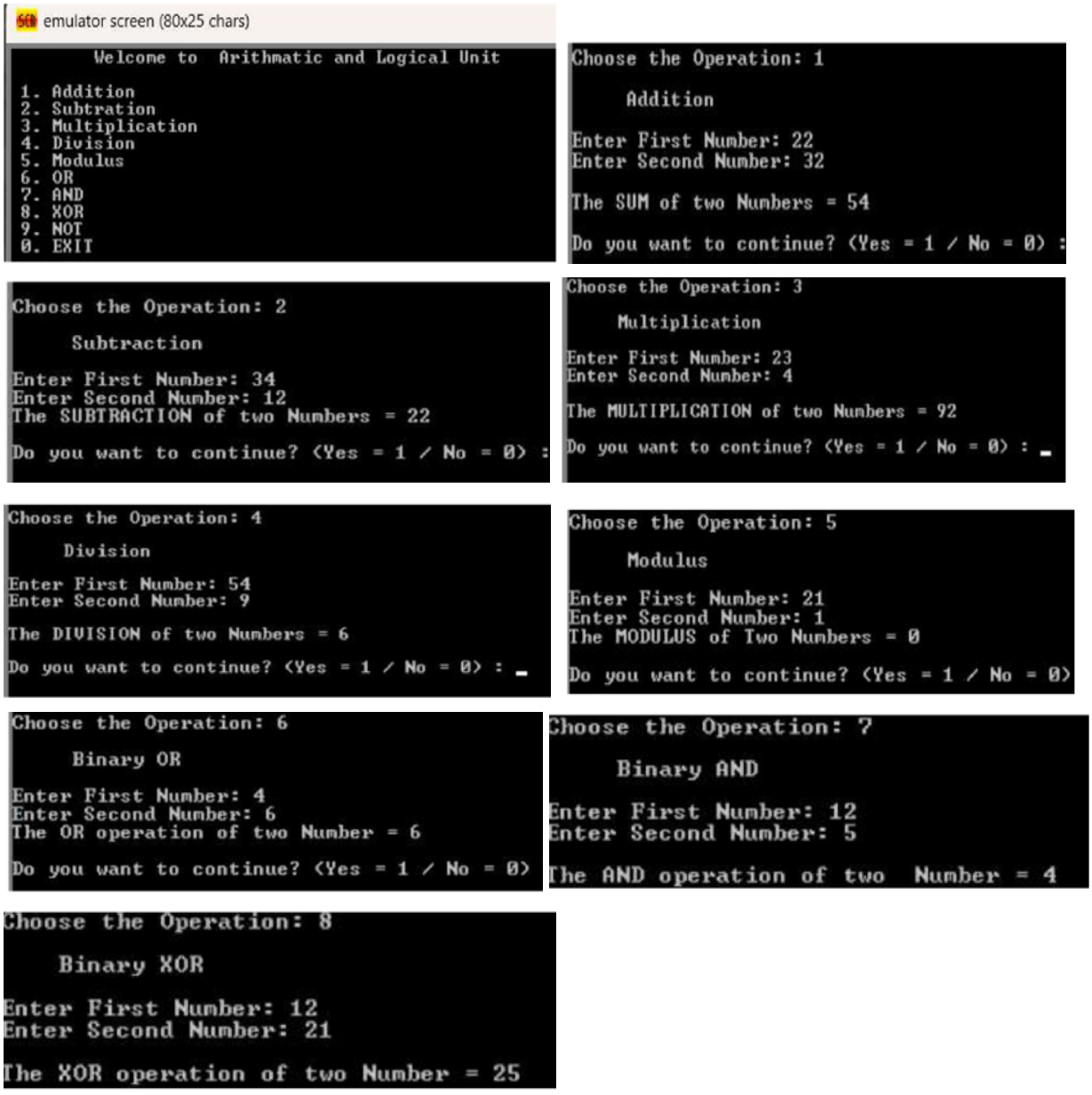


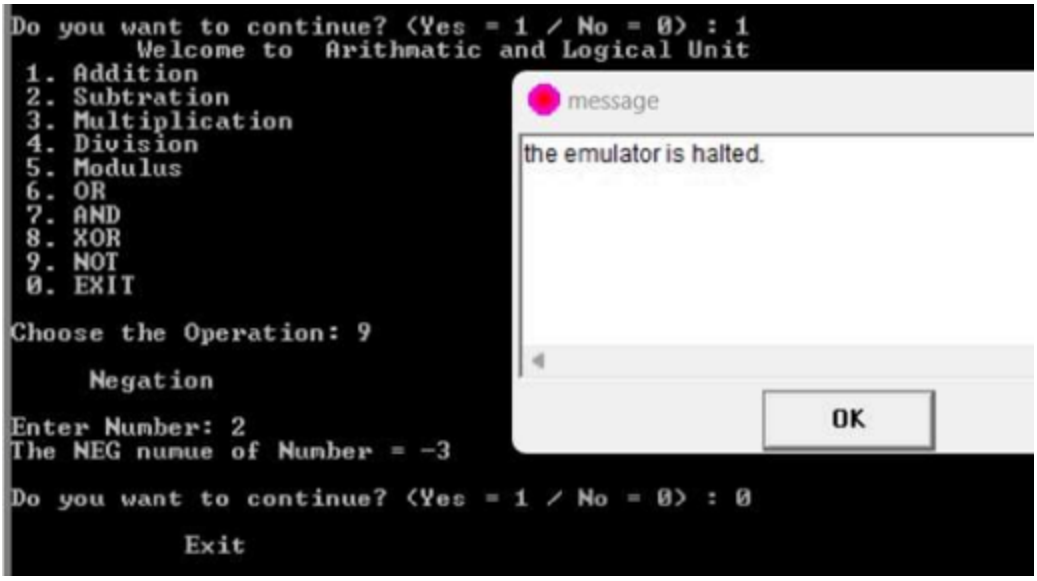
Below is an explanation of the flowchart depicting the operation of an ALU:

1. Start: The flowchart begins with the start symbol, indicating the initiation of the ALU operation.
2. Input: The input stage represents the data input to the ALU, typically in the form of binary numbers or operands for arithmetic and logical operations.
3. Operation Selection: The next step involves selecting the desired operation to be performed by the ALU. This could include addition, subtraction, multiplication, division, logical AND, OR, XOR, or other operations based on the requirement.
4. Execute Operation: Once the operation is selected, the ALU executes the chosen operation on the input data. This stage involves performing the necessary computations or logical evaluations according to the operation selected.
5. Output: After executing the operation, the ALU produces the result of the computation as output. This output may be stored in a register for further processing or transferred to other components of the CPU or computer system.
6. End: Finally, the flowchart concludes with the end symbol, indicating the completion of the ALU operation.
7.

RESULTS:

The implementation of the ALU within the 8086 emulator yielded promising results, with the unit successfully performing arithmetic and logical operations as intended. Performance metrics, including execution time and resource utilization, met the project's expectations, demonstrating the efficiency of the design. Functional validation tests confirmed the correctness and reliability of the ALU implementation, validating its suitability for use in various computational tasks.





LIMITATIONS:

Data Handling Limitations:

• Limited Data Size: The project likely supports only 16-bit integers. This restricts the range of numbers it can handle. Real-world applications often require calculations with larger data types (32-bit, 64-bit).

• Lack of Floating-Point Support: The project might not handle floating-point numbers, which are crucial for calculations involving decimals.

Functionality Limitations:

• Limited Operation Set: The implemented arithmetic and logical operations might be a basic set (addition, subtraction, multiplication, etc.). It might lack more complex mathematical functions like trigonometry, logarithms, or exponentiation.

FUTURE SCOPE:

Future ALU design and implementation should focus on advanced architectures, heterogeneous integration, energy efficiency, security enhancements, real-time processing, quantum ALUs, emulator optimization, compatibility with new architectures, edge computing, and ethical and social implications. These considerations aim to address emerging challenges, enhance computational capabilities, and ensure compatibility with future hardware and software

ecosystems. Future research should explore novel ALU architectures, such as deep learning accelerators or quantum ALUs, to address emerging technologies like artificial intelligence and quantum computing. Emulator optimization should leverage parallel computing, code optimization techniques, and hardware acceleration to improve performance and reduce overhead. Addressing ethical and social implications is crucial for the continued evolution of ALU designs and implementations.

CONCLUSION:

In conclusion, the design and implementation of an Arithmetic Logic Unit (ALU) using the 8086 emulator represent a significant milestone in the study of computer architecture and emulation technology. The successful creation of a functional ALU within the emulator environment demonstrates the feasibility and effectiveness of emulation-based approaches to hardware design and simulation. Moving forward, the insights gained from this project will inform further research and development efforts aimed at advancing ALU design principles and enhancing computational capabilities in diverse application domains.

REFERENCES:

- Intel Corporation. (1980). "8086/8088 User's Manual Programmer's and Hardware Reference, 1979-1980."
- Intel Microprocessors by Berry B. Brey
- https://github.com/sasaber/alu-assembly/blob/master/CS47_proj_alu_logical.asm
- https://github.com/yousefkotp/8086-Assembly-Projects
- Tanenbaum, A. S., & Austin, T. (2016). "Structured Computer Organization." Pearson Education Limited.