HACK ALU

PROJECT REPORT

Submitted by

Adidev MS(AM.AI.U4AID23001)

C.R.Adharsh (AM.AI.U4AID23006)

Gagan Gireesh Krishna (AM.AI.U4AID23008)

Devananda C R (AM.AI.U4AID23036)

Harita Aneesh (AM.AI.U4AID23042)

M V N Prasanna (AM.AI.U4AID23047)

As a part of Project-based evaluation on
23AID113 – Elements of Computing-2

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

AMRITAPURI CAMPUS

JUNE 2024

TABLE OF CONTENTS

ABSTRACT

The 16-bit Hack ALU (Arithmetic Logic Unit) is a fundamental component of the Hack computer architecture, responsible for executing a diverse array of arithmetic and logical operations. Central to its functionality are six control bits—zx, nx, zy, ny, f, and no—which configure the ALU to process two 16-bit inputs, x and y, in various ways. These control bits determine whether inputs are zeroed, negated, added, or subjected to bitwise operations, enabling the ALU to perform essential tasks such as addition, subtraction, and logical conjunctions.

In addition to the versatile control bits, the Hack ALU provides two critical status flags: zr (zero) and ng (negative). The zr flag is set when the ALU's output is zero, which is useful for implementing conditional operations that rely on equality checks. The ng flag is set when the output is negative, indicated by the most significant bit of the 16-bit result being 1. These flags are vital for control flow mechanisms, allowing the Hack computer to perform conditional branching and other decision-based operations effectively.

The combination of the control bits and status flags makes the Hack ALU a powerful computational engine within the Hack architecture. By efficiently handling a wide range of operations and providing detailed status feedback, the ALU supports the execution of complex programs and enhances the overall functionality of the Hack computer. This design not only showcases the elegance of simplicity in computer architecture but also highlights the effectiveness of modular, control-bit-driven computation.

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

The 16-bit Hack ALU is a core component of the Hack computer, designed for educational purposes to demonstrate fundamental computer architecture and operations. It processes arithmetic and logical tasks through configurable control bits and status flags, illustrating key computational principles.

1.2 OBJECTIVES

The objectives of the 16-bit Hack ALU are to demonstrate core computations by performing essential arithmetic and logical operations using configurable control bits. It aims to educate users on computer architecture by providing a hands-on learning tool for understanding fundamental principles of computer design. Additionally, it supports complex control flows by utilizing status flags to enable conditional operations and decision-making processes within the Hack computer.

1.3 RELEVANCE

The 16-bit Hack ALU is highly relevant in the field of computer science education as it provides a practical and accessible way to understand the fundamental principles of computer architecture and digital logic design. By offering a hands-on learning experience, it bridges the gap between theoretical concepts and real-world application, making abstract ideas more concrete and comprehensible. Additionally, the simplicity and modularity of the Hack ALU design allow students to grasp the inner workings of a CPU, fostering a deeper appreciation for how modern computers execute complex operations efficiently. This relevance extends to enhancing problem-solving skills, critical thinking, and innovation in aspiring computer scientists and engineers.

1.4 REPORT ORGANIZATION

The analysis of the 16-bit Hack ALU provides a comprehensive examination of its design, operational principles, and educational significance. By exploring the critical role played by the ALU in computer systems and the complexities of the Hack computer architecture, the report offers valuable insights into the inner workings of this essential component. Additionally, a thorough evaluation of the design, structure, and performance of the Hack ALU sheds light on its efficiency and relevance in an educational context.

CHAPTER 2

REVIEW OF LITERATURE

The concept of implementing a 16-bit Hack Arithmetic Logic Unit (ALU) on a Field-Programmable Gate Array (FPGA) is a well-established approach for education and experimentation in computer architecture. Literature in this area explores various aspects of this process, providing valuable guidance for those undertaking this project.

Bottom-Up Design Methodology

A recurring theme in the literature is the emphasis on a bottom-up design methodology. This approach advocates for starting with the most fundamental building blocks, such as logic gates (AND, OR, NOT), and progressively combining them to construct more intricate components like the ALU. This ensures a thorough understanding of the underlying digital logic principles before tackling a complex system [1].

Construction from Basic Elements

Several resources delve deeper into the practical aspects of constructing the ALU. They detail how to utilize fundamental components like full adders (performing single-bit addition) and multiplexers (acting as programmable data selectors) to build the core functionalities of the ALU. This breakdown allows learners to grasp the relationship between basic logic operations and the overall functionality of the ALU [2][3].

The Advantage of the Hack CPU Design

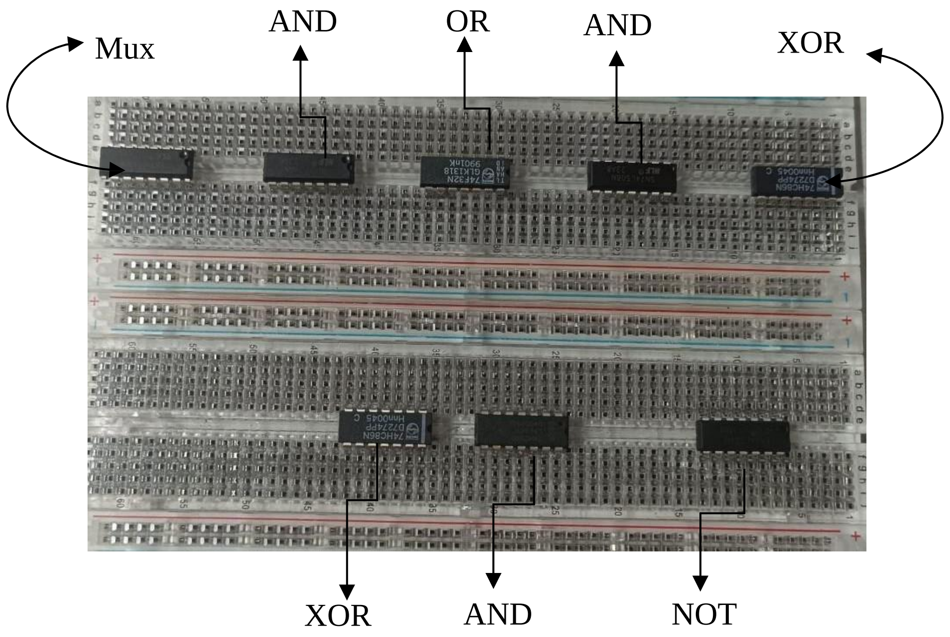
The Hack CPU architecture emerges as a particularly suitable candidate for this project due to its modular nature. This modularity simplifies the process by allowing the isolation and implementation of the ALU as an independent unit. Existing code for these fundamental building blocks within the Hack CPU design can be leveraged, accelerating the development process [2].

Conclusion

Overall, the literature on implementing a 16-bit Hack ALU on FPGA offers a structured approach. By following a bottom-up design methodology, utilizing existing resources on building blocks, and capitalizing on the modularity of the Hack CPU design, this project becomes a feasible and rewarding experience for those seeking to understand computer architecture and digital design principles.

CHAPTER 3

DESIGN CONSIDERATIONS

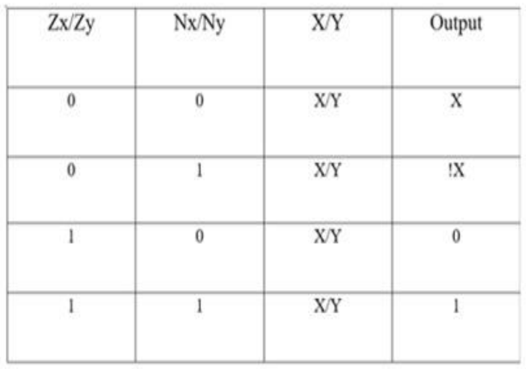


[Figure 1]

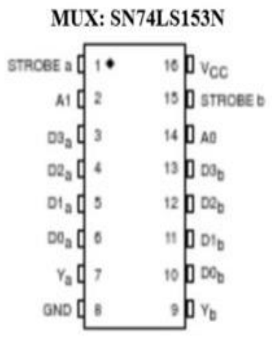
In the construction of a 16-bit Arithmetic Logic Unit (ALU), several key components play integral roles in enabling a wide array of computational tasks and operations. These components include the SN74LS153N multiplexer, 74LS08N AND gate, 74LHC86N XOR gate, 74LS04N NOT gate, and 74F32N OR gate.

The SN74LS153N multiplexer is pivotal in the ALU design as it allows for the selection and routing of specific input signals. This functionality is essential for managing control signals and facilitating the ALU's ability to switch between different operations based on selected inputs. By efficiently directing signals, the multiplexer enhances the ALU's versatility and capability to handle various computational tasks. Figure 2 and Table 1 show the pin diagram and the truth table of MUX.

Truth Table of MUX
Table 1



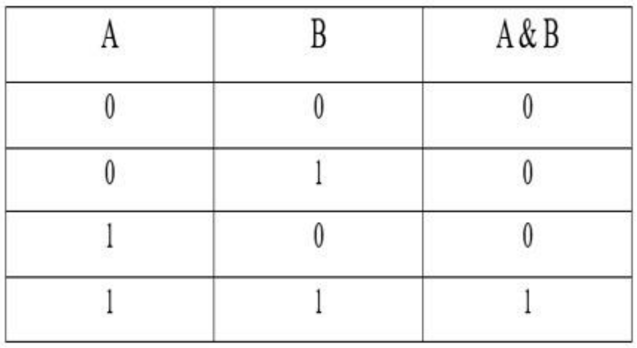
Zx/Zy	Nx/Ny	X/Y	Output
0	0	X/Y	X
0	1	X/Y	!X
1	0	X/Y	0
1	1	X/Y	1



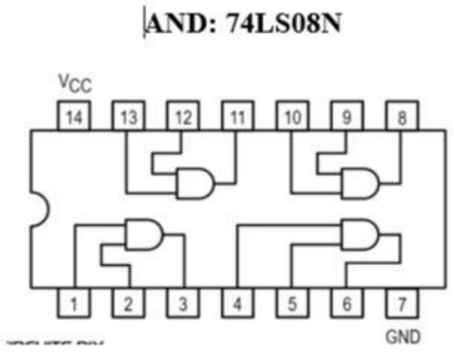
Pin Diagram of Mux
Figure 2

For bitwise logical operations, the 74LS08N AND gate is employed within the ALU. This gate performs logical conjunctions on individual bits, enabling essential operations such as bit masking and logical multiplication. It allows the ALU to manipulate and combine input bits based on the AND logic, which is fundamental for controlling data flow and executing specific computational tasks efficiently. Figure 3 and Table 2 show the pin diagram and the truth table of AND gate.

Truth Table of AND Gate
Table 2



A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

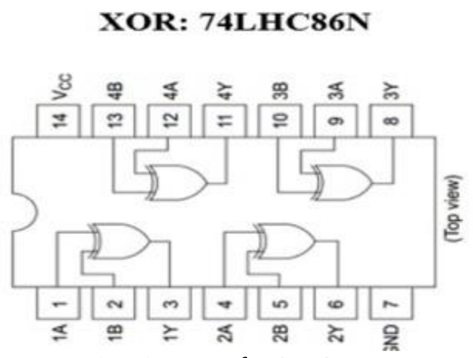


Pin Diagram of AND Gate
Figure 3

The 74LHC86N XOR gate plays a crucial role in the ALU by facilitating bitwise addition and parity checks. XOR gates are fundamental components in creating adders and comparators within the ALU architecture. They enable the ALU to perform arithmetic operations and detect parity errors by comparing corresponding bits across input data streams, thereby supporting accurate and reliable computation. Figure 4 and Table 3 show the pin diagram and the truth table of XOR gate.

Truth Table of XOR Gate
Table 3

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



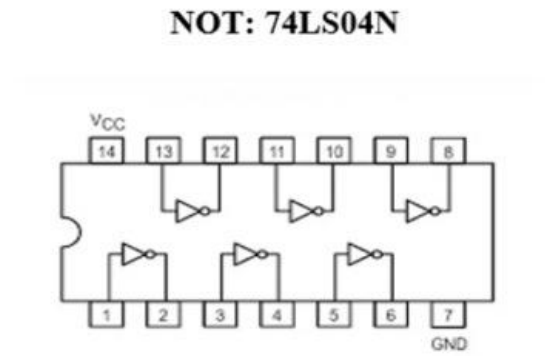
Pin Diagram of XOR Gate
Figure 4

In contrast, the 74LS04N NOT gate serves as an inverter within the ALU. This gate is essential for performing bitwise negation operations, which are necessary for tasks such as logical complementation and subtraction. By inverting the logical state of input bits, the NOT gate enables the ALU to implement various computational algorithms and operations effectively. Figure 4 and Table 4 show the pin diagram and the truth table of XOR gate. Figure 5 and Table 4 show the pin diagram and the truth table of NOT gate.

Truth Table of NOT Gate

Table 4

A	Output
1	0
0	1

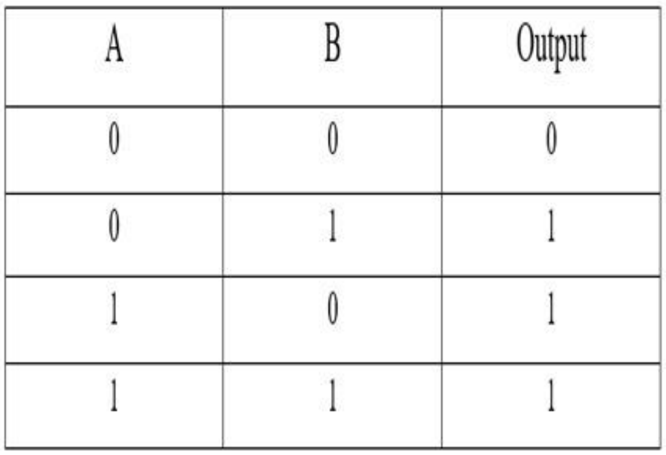


Pin Diagram of NOT Gate
Figure 5

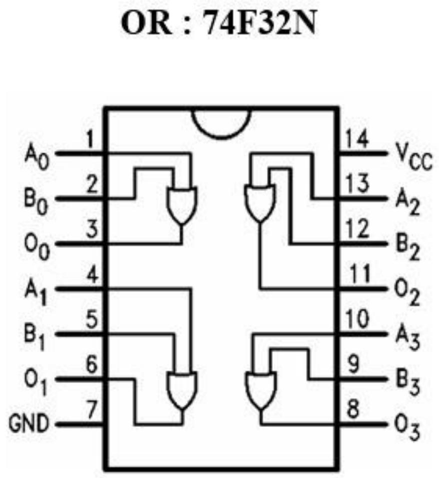
Additionally, the 74F32N OR gate is utilized for inclusive disjunctions within the ALU. By applying the OR logic to input signals, this gate allows the ALU to perform logical addition and merge multiple conditions seamlessly. It enhances the ALU's capability to handle complex decision-making processes and data manipulation tasks, supporting versatile computational operations within digital systems. Figure 6 and Table 5 show the pin diagram and the truth table of OR gate.

Truth Table of OR Gate

Table 5



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

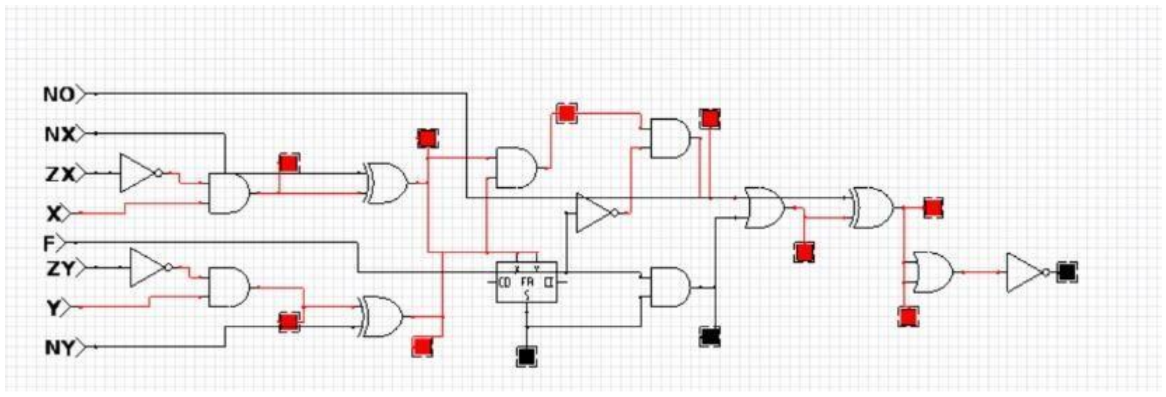


Pin Diagram of OR Gate
Figure 6

Collectively, these components form the foundational framework of a 16-bit ALU, providing the necessary functionalities to execute diverse arithmetic and logical operations with precision and efficiency. Their integration ensures that the ALU can effectively process data, from basic arithmetic calculations to complex algorithmic tasks, thereby enhancing overall computational performance and functionality.

CHAPTER 4

CIRCUIT DIAGRAM



Simulation
Figure 7

This circuit diagram represents a crucial section of a 16-bit Arithmetic Logic Unit (ALU), showcasing the intricate interconnections between various logic components. The design incorporates multiple input signals (NO, NX, ZX, X, F, ZY, Y, NY), which likely serve as both control signals and data inputs for the ALU's operations. These inputs are strategically routed through a network of logic gates to perform complex arithmetic and logical functions. Figure 7 shows the simulation of ALU.

The circuit prominently features several types of logic gates, each serving a specific purpose in the ALU's functionality. AND gates, possibly implemented using the 74LS08N chip, are utilized for bitwise logical operations such as masking and logical multiplication. These gates allow the ALU to manipulate and combine input bits based on AND logic, which is essential for controlling data flow and executing specific computational tasks.

OR gates, likely implemented with the 74F32N chip, are employed for inclusive disjunctions within the ALU. These gates enable logical addition and the merging of multiple conditions, enhancing the ALU's capability to handle complex decision-making processes and data manipulation tasks. The presence of OR gates supports versatile computational operations within the digital system.

NOT gates (inverters), potentially using the 74LS04N chip, play a crucial role in performing bitwise negation operations. These are necessary for tasks such as logical complementation and subtraction, allowing the ALU to invert the logical state of input bits and implement various computational algorithms effectively.

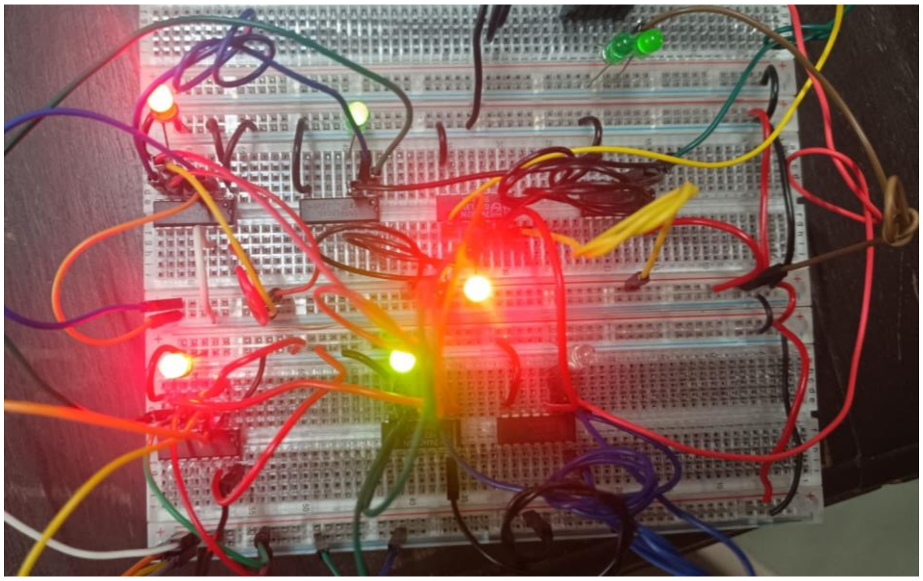
XOR gates, possibly implemented with the 74LHC86N chip, are fundamental components in creating adders and comparators within the ALU architecture. These gates facilitate bitwise addition and parity checks, enabling the ALU to perform arithmetic operations and detect parity errors by comparing corresponding bits across input data streams.

The diagram also includes a component labelled "FF," which likely represents a flip-flop. This element suggests that the circuit incorporates sequential logic, allowing it to maintain state information. The flip-flop could be used for temporary storage of intermediate results or for synchronizing operations within the ALU. Figure 8 shows the implemented circuit.

While not explicitly shown in the diagram, the complex routing of signals implies the presence of multiplexing functionality. This could be implemented using components like the SN74LS153N multiplexer, which is crucial for selecting and routing specific input signals based on control inputs. The multiplexer enhances the ALU's versatility, allowing it to switch between different operations dynamically.

The intricate connections between these components create a sophisticated system capable of performing a wide array of 16-bit arithmetic and logical operations. The design allows for efficient processing of data, from basic arithmetic calculations to complex algorithmic tasks. The strategic placement and interconnection of gates enable the ALU to select between different operations, process multi-bit data in parallel, and produce results with high efficiency and accuracy.

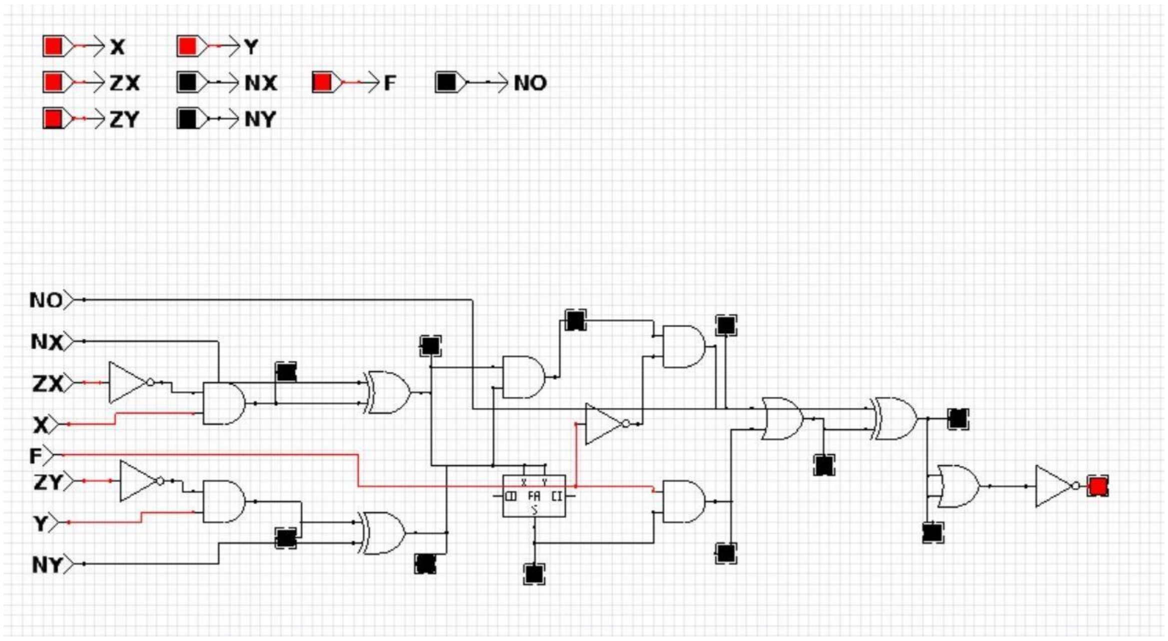
This circuit diagram, while not showing the entire 16-bit ALU, provides a clear insight into the complexity and ingenuity involved in designing such a critical component of a computer's central processing unit. The integration of various logic gates and sequential elements demonstrates how fundamental digital components can be combined to create a powerful computational unit capable of executing diverse and complex operations essential for modern computing systems.


Implemented Circuit
Figure 8

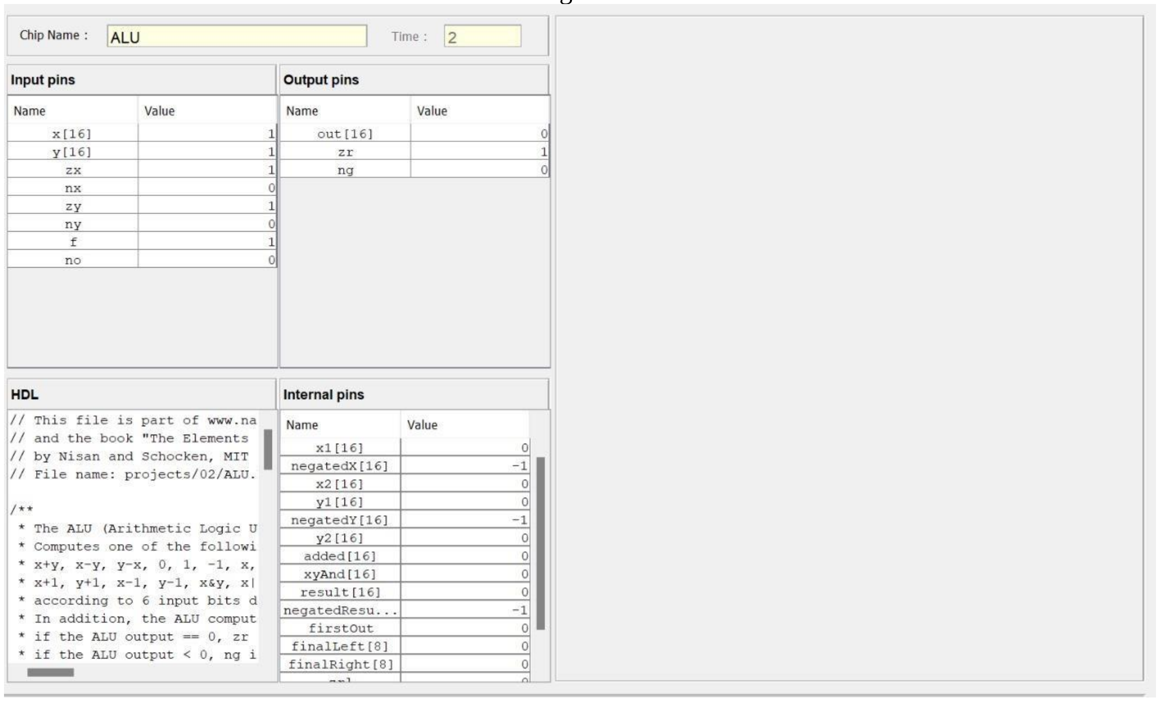
CHAPTER 5

IMPLEMENTATION AND RESULTS

5.1 Software Implementation



[CEDAR Logic Simulator – ALU circuit]
Figure 9



Input pins

Name	Value
x[16]	1
y[16]	1
zx	1
nx	0
zy	1
ny	0
f	1
no	0

Output pins

Name	Value
out[16]	0
zr	1
ng	0

HDL

```
// This file is part of www.na
// and the book "The Elements
// by Nisan and Schocken, MIT
// File name: projects/02/ALU.

/**
 * The ALU (Arithmetic Logic U
 * Computes one of the followi
 * x+y, x-y, y-x, 0, 1, -1, x,
 * x+1, y+1, x-1, y-1, x&y, x|
 * according to 6 input bits d
 * In addition, the ALU comput
 * if the ALU output == 0, zr
 * if the ALU output < 0, ng i
```

Internal pins

Name	Value
x1[16]	0
negatedX[16]	-1
x2[16]	0
y1[16]	0
negatedY[16]	-1
y2[16]	0
added[16]	0
xyAnd[16]	0
result[16]	0
negatedResu...	-1
firstOut	0
finalLeft[8]	0
finalRight[8]	0

[Hardware Simulator- Nand2Tetris]
Figure 10

5.2 Hardware Implementation

The Hack ALU is implemented using a combination of fundamental digital components, including 4-way multiplexers, NOT gates, AND gates, OR gates, and an 8-way OR gate. These components work together to process 16-bit data inputs X and Y, performing various arithmetic and logical operations as dictated by control signals. The key control signals include ZX (Zero X), NX (Negate X), ZY (Zero Y), NY (Negate Y), F (Function), and NO (Negate Output), each modifying the behavior of the ALU to achieve the desired operation.

The multiplexer is pivotal in the ALU design, particularly the 4-way multiplexer, which allows for the selection between different data paths based on the control signals. For instance, the multiplexer helps in choosing between performing an AND operation or an ADD operation on the inputs X and Y, governed by the F control signal. This versatility is essential for the ALU's ability to switch between different types of operations seamlessly. The multiplexer essentially dictates the flow of data through the ALU based on the selected function.

NOT gates are employed to negate inputs or intermediate results, activated by control signals NX and NY. When NX is high, the X input is inverted, and similarly, when NY is high, the Y input is inverted. This negation capability is crucial for operations that require subtraction or logical NOT operations. Additionally, AND gates are used to zero out the inputs X and Y when the ZX and ZY control signals are high, respectively. This ensures that the inputs can be selectively nullified, which is necessary for certain operations like computing constant values.

Finally, the 8-way OR gate is used to determine if the output result is zero. This is done by OR-ing all 16 bits of the result together; if all bits are zero, the output of the 8-way OR gate will be zero, indicating that the result is zero. This zero detection mechanism is vital for setting the zero flag, which is an important status output of the ALU. The combination of these gates and multiplexers forms a robust ALU capable of performing a wide range of operations, demonstrating the power of simple digital logic components when combined effectively.

CHAPTER 6

ADVANTAGES & APPLICATIONS

ADVANTAGES

- The Hack ALU's design, while simple, is highly effective. The use of six control bits to dictate a wide range of operations demonstrates how a minimalistic approach can yield powerful computational capabilities. This simplicity makes it easier to understand, implement, and debug.

- The control bits allow the ALU to perform a variety of arithmetic and logical operations. This versatility ensures that the ALU can handle diverse computational tasks, from basic arithmetic to complex logical operations.

- By integrating functions such as zeroing, negation, addition, and bitwise operations into a single unit, the Hack ALU minimizes the need for multiple, separate computational units, leading to efficient use of resources.

- The status flags (zr and ng) enhance the ALU's functionality by providing crucial feedback on the result of computations. This supports the implementation of conditional operations, which are essential for control flow in programming.

- The modular nature of the ALU, driven by control bits, allows for straightforward integration into larger systems. This modularity also facilitates scalability and reuse in different parts of the architecture or in different projects.

- The Hack ALU serves as an excellent educational tool. Its design is simple enough for students to grasp fundamental concepts of computer architecture and ALU operations, yet powerful enough to demonstrate practical applications.

APPLICATIONS

- The Hack ALU is extensively used in educational environments to teach students about the basics of computer architecture, logic design, and the functioning of an ALU. Its simplicity and clarity make it ideal for instructional purposes.

- The compact and efficient design of the Hack ALU makes it suitable for embedded systems, where resource constraints and efficiency are critical.

- In systems where basic arithmetic and logical operations are required, such as simple calculators, digital signal processing units, and small-scale microcontrollers, the Hack ALU can be effectively implemented.

- The ALU’s ability to provide status flags (zero and negative) is crucial for implementing control flow mechanisms in software. This includes tasks like conditional branching, looping, and decision-making in various programming contexts.

- The Hack ALU is a useful tool for prototyping and research in computer science and digital logic design. Researchers and developers can use it to quickly implement and test new ideas in computational logic and processor design.

- For custom computing solutions where specific operations and optimizations are required, the Hack ALU can be adapted and extended to meet unique requirements, providing a flexible foundation for custom hardware development.

CHAPTER 7

CONCLUSION & FUTURE SCOPE

Conclusion

The 16-bit Hack ALU is a cornerstone of the Hack computer architecture, demonstrating a blend of simplicity and functionality. Through the use of six control bits, the ALU can execute a diverse array of arithmetic and logical operations on two 16-bit inputs, making it a versatile computational tool. The inclusion of status flags such as zr (zero) and ng (negative) further enhances its capability, enabling the implementation of complex control flow mechanisms. This design showcases the power of modular, control-bit-driven computation, providing a solid foundation for executing a wide range of programs efficiently. The Hack ALU exemplifies the elegance and effectiveness of minimalist computer architecture design, ensuring reliable performance and flexibility.

Future Scope

1. **Increased Bit Width:**
    - Expand the ALU to support 32-bit or 64-bit operations to handle larger data sets and more complex calculations.
2. **Additional Operations:**
    - Integrate more arithmetic and logical operations, such as multiplication and division, to enhance the ALU's functionality.
3. **Improved Efficiency:**
    - Optimize the existing design to reduce power consumption and increase processing speed, making the ALU more efficient.
4. **Error Detection:**
    - Implement basic error detection mechanisms, such as parity checks, to ensure reliable operation and data integrity.
5. **Educational Resources:**
    - Develop interactive learning tools and detailed documentation to help students and hobbyists understand and utilize the Hack ALU.
6. **Modular Design:**
    - Create a modular ALU design that can be easily adapted or extended for various applications, promoting flexibility and reusability.

REFERENCES

[1] Patterson, David A., and John L. Hennessy. Computer Organization and Design: The Hardware/Software Interface.

[2] Nisan, Noam, and Shimon Schocken. The Elements of Computing Systems: Building a Modern Computer from First Principles (Nand to Tetris book). Available at: [Nand to Tetris](https://www.nand2tetris.org/)

[3] Harris, David, and Sarah Harris. Digital Design and Computer Architecture.

[4] YouTube: [Vtcm1oSSI0o](https://www.youtube.com/watch?v=Vtcm1oSSI0o)

[5] YouTube: [3bgMUGdGcBU](https://www.youtube.com/watch?v=3bgMUGdGcBU)

APPENDIX

```
CHIP ALU {
  IN
    x[16], y[16], // 16-bit inputs
    zx, // zero the x input?
    nx, // negate the x input?
    zy, // zero the y input?
    ny, // negate the y input?
    f,  // compute out = x + y (if 1) or x & y (if 0)
    no; // negate the out output?

  OUT
    out[16], // 16-bit output
    zr, // 1 if (out == 0), 0 otherwise
    ng; // 1 if (out < 0),  0 otherwise

  PARTS:
    // x zero / negate
    Mux16(a=x, sel=zx, out=x1);
    Not16(in=x1, out=negatedX);
    Mux16(a=x1, b=negatedX, sel=nx, out=x2);

    // y zero / negate
    Mux16(a=y, sel=zy, out=y1);
    Not16(in=y1, out=negatedY);
    Mux16(a=y1, b=negatedY, sel=ny, out=y2);

    // x + y OR x & y
    Add16(a=x2, b=y2, out=added);
    And16(a=x2, b=y2, out=xyAnd);
    Mux16(a=xyAnd, b=added, sel=f, out=result);

    // negate the  output?
    Not16(in=result, out=negatedResult);
    Mux16(a=result,    b=negatedResult,    sel=no,    out=out,    out[15]=firstOut,
out[0..7]=finalLeft, out[8..15]=finalRight);

    // output == 0 (zr)
    Or8Way(in=finalLeft, out=zrl);
    Or8Way(in=finalRight, out=zrr);
    Or(a=zrl, b=zrr, out=nzr);
    Not(in=nzr, out=zr);

    // output < 0 (ng)
    And(a=firstOut, b=true, out=ng);
}
```