**A Project Report *Submitted by*** Group No.:12

Group Members (Name & Roll No.)

C V SHREYAAS ADITYA CB.SC.U4AIE23018

S MATHIVANAN CB.SC.U4AIE23042

V SAI NIVEDH CB.SC.U4AIE23062

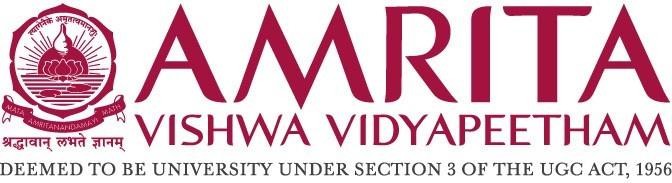
SYED ANEES ASHRAF CB.SC.U4AIE23068

*As a part of the subject*

22AIE102- ELEMENTS OF COMPUTING SYSTEMS 1

**HACK COMPUTER AND**

**4-BIT COMPARATOR**



**Department of Artificial Intelligence**

# AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112 (INDIA)

**December 2023**

**AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE**

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112

**DECLARATION**

We, hereby declare that the project entitled, is the record of the “Hack computer and 4-bit Comparator” the work done by our team under the guidance of Dr. Jyothish Lal G, Assistant Professor, Centre for Computational Engineering and Networking, Amrita, School of Artificial Intelligence, Coimbatore. To the best of our knowledge this work has not formed the basis for the award of B.TECH degree in Amrita Vishwa Vidhyapeetham University.

Project Team: 12

Section – A

C V SHREYAAS ADITYA CB.SC.U4AIE23018

S MATHIVANAN CB.SC.U4AIE23042

V SAI NIVEDH CB.SC.U4AIE23062

SYED ANEES ASHRAF CB.SC.U4AIE23068

**Signature of the Faculty**

**(Dr. Jyothish Lal G)**

Assistant Professor

Department of Artificial Intelligence

Amrita Vishwa Vidyapeetham, Coimbatore.

**Place:** Coimbatore

**Date:** 22. 12. 2023

**INDEX**

**PART 1**

|  |  |
| --- | --- |
| **CONTENTS** | **PAGE NO.** |
| **CHAPTER I: INTRODUCTION** | **6** |
| **CHAPTER II: DESIGN** | **7** |
| **CHAPTER III: METHODOLOGY** | **8** |
| **CHAPTER IV: RESULTS** | **9** |
| **CHAPTER V: APPLICATIONS** | **12** |
| **CHAPTER VI: CONCLUSION** | **14** |

**INDEX**

**PART 2**

|  |  |
| --- | --- |
| **CONTENTS** | **PAGE NO.** |
| **CHAPTER I: INTRODUCTION** | **16** |
| **CHAPTER II: DESIGN** | **17** |
| **CHAPTER III: METHODOLOGY** | **21** |
| **CHAPTER IV: RESULTS** | **23** |
| **CHAPTER V: APPLICATIONS** | **26** |
| **CHAPTER VI: CONCLUSION** | **27** |
| **APPENDIX: CODES** | **28** |
| **REFENCES** | **30** |

**PART 1**

**16-bit HACK CPU**

**CHAPTER I**

**INTRODUCTION**

The project endeavours to design and implement a 16-bit CPU on the Hack platform, a virtual computer architecture introduced in the "From Nand to Tetris" curriculum. The motivation behind this undertaking lies in the pursuit of a deeper understanding of computer architecture and the intricacies of CPU design. By embarking on this project, the goal is to translate theoretical knowledge into practical skills, fostering a comprehensive comprehension of the underlying principles that govern computing systems.

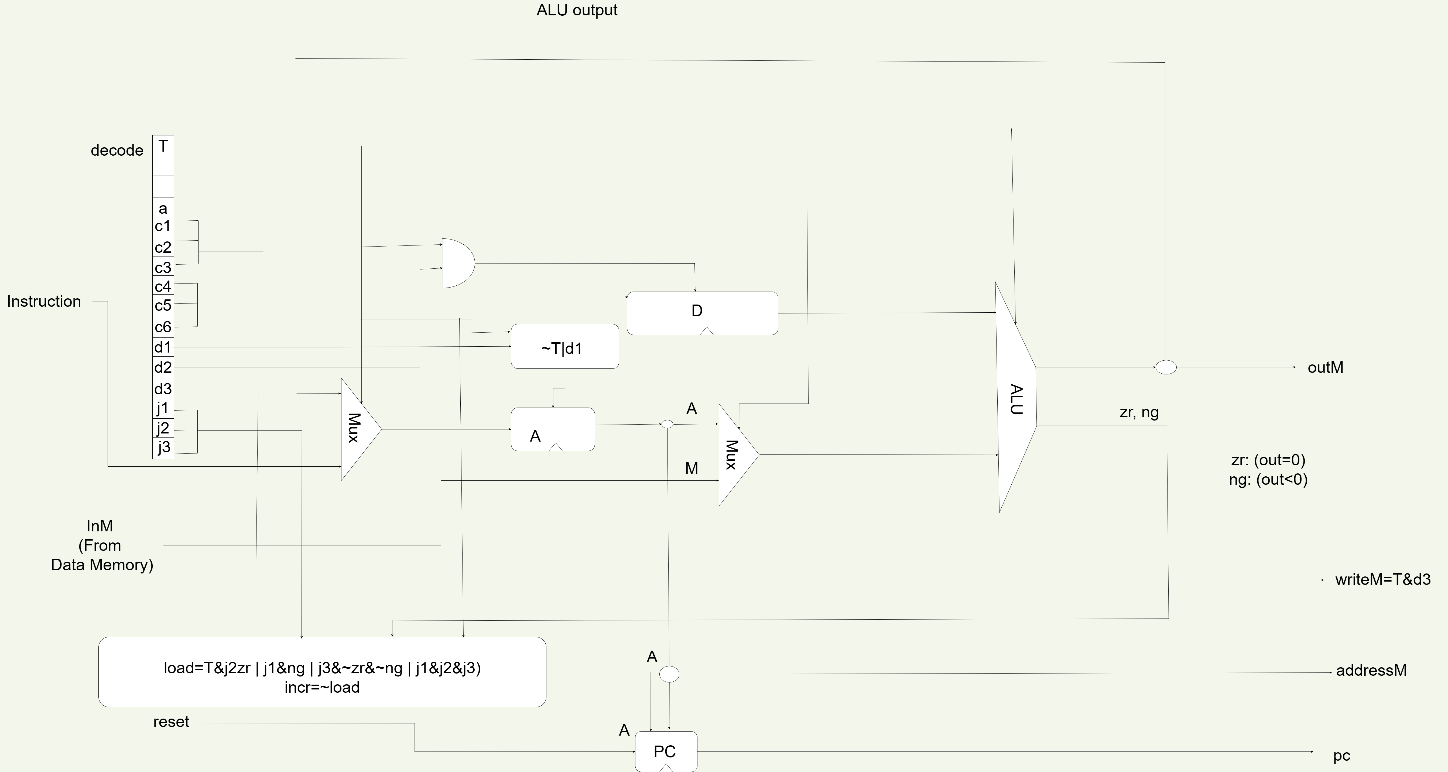
The significance of constructing a 16-bit CPU on the Hack platform extends beyond the immediate scope of the project. It serves as a hands-on exploration of the complexities involved in building a central processing unit, offering insights into the interactions between various hardware components. This endeavour seeks to bridge the gap between theoretical concepts and practical implementation, empowering enthusiasts and learners to delve into the inner workings of computers.

Furthermore, the Hack platform, with its simplicity and elegance, provides a suitable environment for this project. The creation of a 16-bit CPU within this context not only aligns with educational objectives but also opens avenues for experimentation and future extensions. As a foundational project within the broader realm of computer science education, this endeavour promises to equip participants with invaluable skills and a profound appreciation for the intricate dance between hardware and software in computing systems

**CHAPTER II**

**DESIGN**

* FIGURE 2.1

Figure 2.1 is the chip diagram of HACK CPU

**CHAPTER III**

**METHODOLOGY**

* Preliminary Research:

Begin the project with a comprehensive overview of the relevant literature and documentation on CPU architecture, 16-bit computing, and the specifics of the Hack platform. This background research will inform key design decisions and ensure compliance with established principles.

* Design Specifications:

Clearly define the architecture and specifications of a 16-bit CPU. This includes determining the data path, controller structure, and memory organization. Create an instruction set architecture (ISA), including opcodes, addressing modes, and supported operations.

* Logic design:

Translate the design specifications into a detailed logic design. Develop schematics and diagrams representing CPU components, including registers, ALU, control signals, and memory units. Use hardware description language (HDL) to systematically represent logic circuits.

* Implementation of Components:

Build the individual components of a 16-bit CPU based on the logic design. This includes creating modules for the registers, implementing the ALU, and configuring the memory units. Simulate each component to ensure proper functionality and compliance with design specifications.

* Development of the control unit:

Build a control unit responsible for organizing the execution of instructions. Develop the ultimate machine that controls the flow of control signals and ensures synchronization and coordination between the various CPU components.

* Integration:

Integrate individual components into a cohesive 16-bit CPU system. Validate interactions between different units and resolve potential bottlenecks or conflicts. Conduct thorough testing at each stage of integration to identify and resolve potential issues.

* Assembly language implementation:

Implement a symbolic instruction language that corresponds to the defined instruction set. Develop an assembler to translate assembly code into machine code compatible with a 16-bit CPU.

* Testing and Debugging:

Testing is done using the supplied test script and debugging if any fixes need to be made and design optimization if needed.

* Output:

The output is to be displayed as part of the HACK computer.

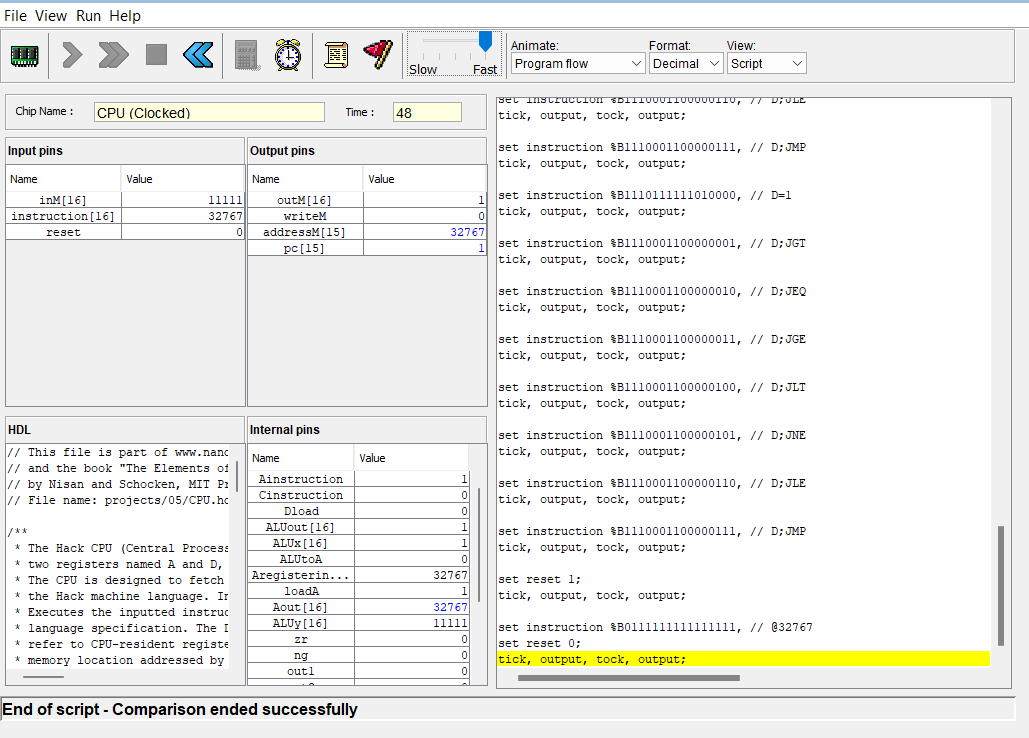
**CHAPTER IV**

**RESULTS**

The result of the project is given as screenshots of the outputs tested using the script file.

Result of the CPU:

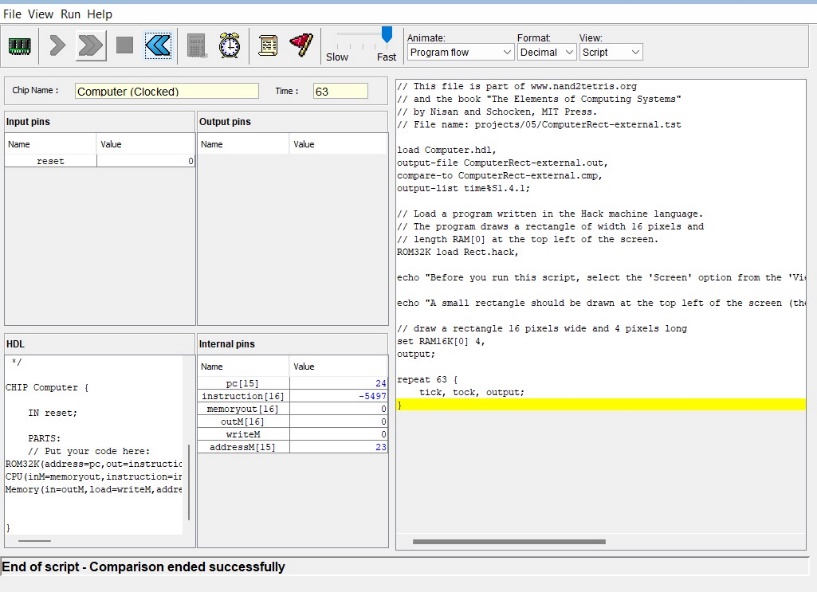
* Figure 4.1



The figure 4.1 is the output of the of the CPU in the Hardware simulator tested with the test script.

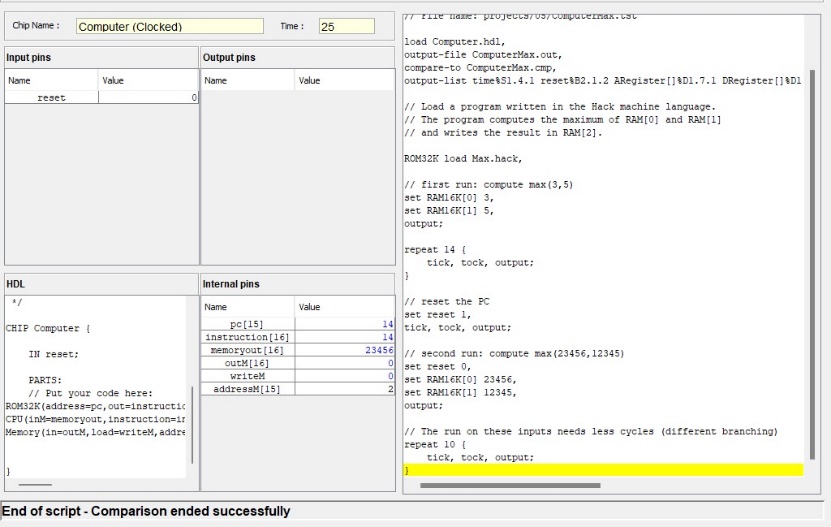
**RESULT: CPU AS A PART OF HACK COMPUTER.**

**Figure 4.2**



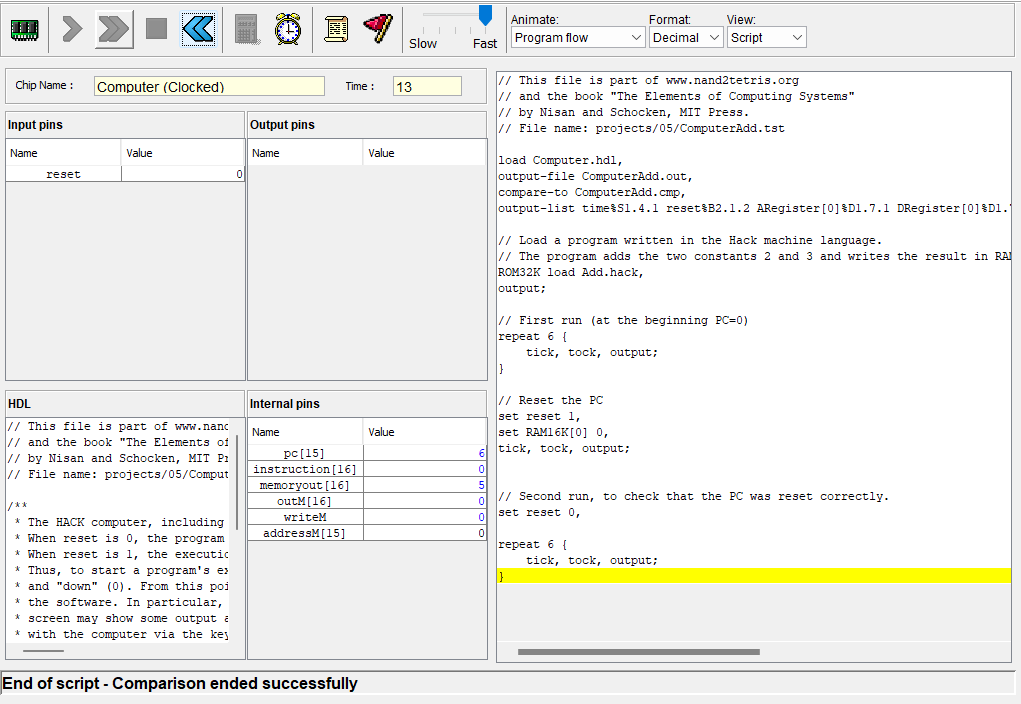
The figure 4.2 is the output of the of the computer tested with the computerAdd.test script.

**Figure 4.3**



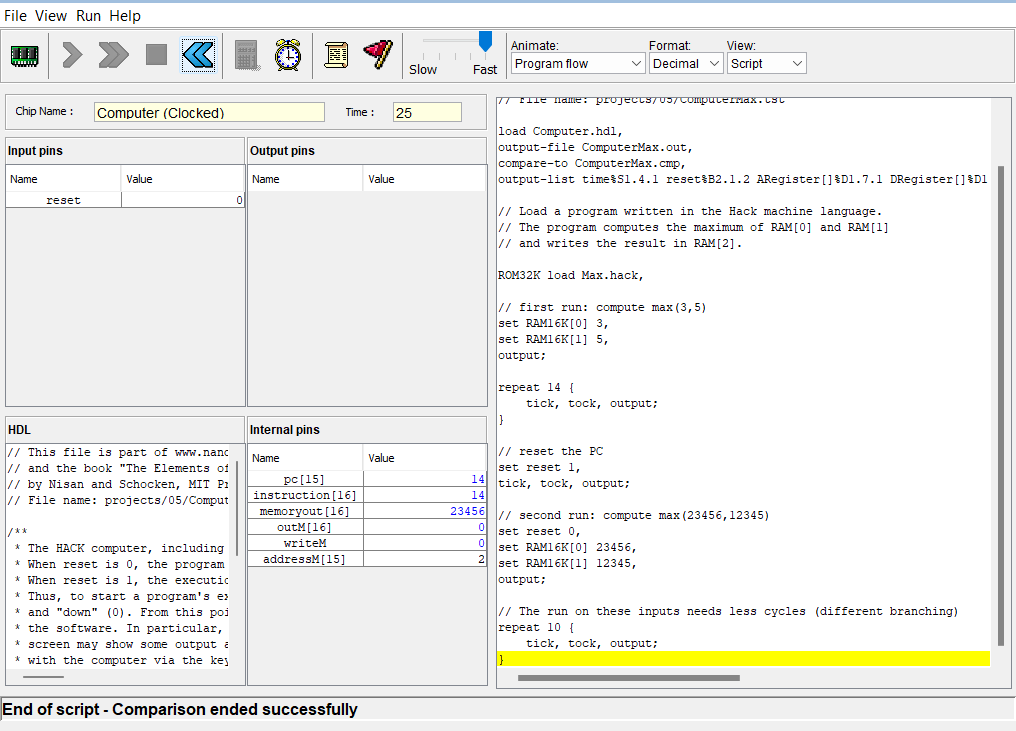
The figure 4.3 is the output of the of the computer tested with the computerAddexternal test script.

**Figure 4.4**



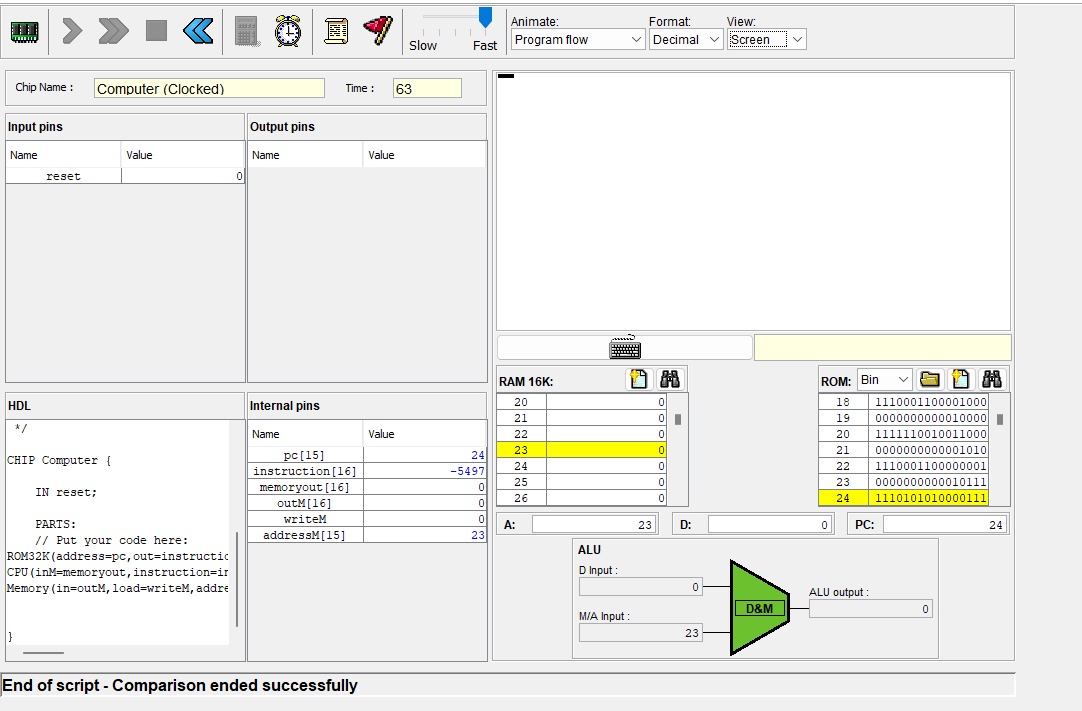
The figure 4.4 is the output of the of the computer tested with the computerMax test script.

**Figure 4.5**

****

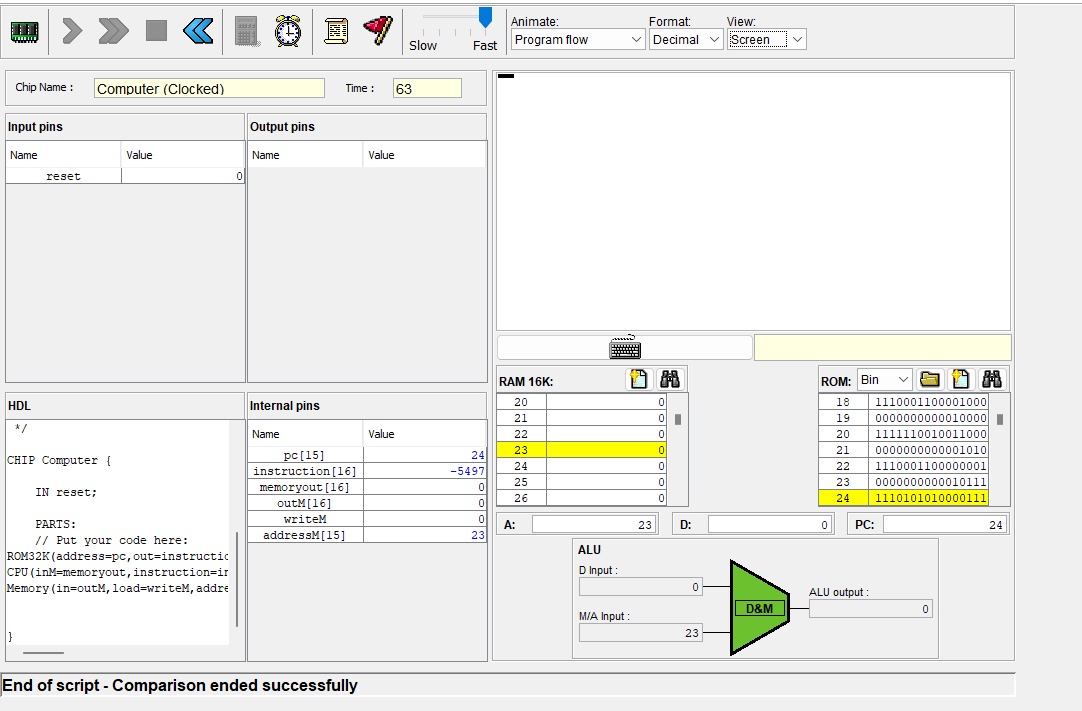
The figure 4.5 is the output of the of the computer tested with the computerMaxexternal test script.

Figure 4.6



The figure 4.6 is the output of the of the computer tested with the computerRect test script.

Figure 4.7



The figure 4.6 is the output of the of the computer tested with the computerRect test script.

**CHAPTER V**

**APPLICTIONS**

The utilization of a 16-bit CPU within the hack platform offers a rich landscape of applications across education, research, platform extension, historical exploration, simulation, and skill development, significantly impacting various realms within computer science and engineering.

* Education and learning:

At its core, the 16-bit CPU serves as a potent educational tool. In computer architecture courses, it provides students hands-on experience, facilitating a deeper comprehension of CPU design and functionality. Additionally, it caters to self-paced learners and enthusiasts seeking practical insights into computer systems, offering a structured guide to delve into the intricacies of CPU architectures.

* Research and development:

For researchers and developers, the 16-bit CPU becomes a valuable prototyping platform. It facilitates experimentation with novel CPU architectures, enabling the testing of new instructions or exploration of optimization techniques. Moreover, it stands as a versatile teaching tool, aiding in workshops and training sessions focused on elucidating complex computer architecture concepts.

* Platform extension:

Beyond its educational and developmental roles, the 16-bit CPU forms a foundational base for extended projects. It can be the cornerstone for constructing comprehensive computer systems integrating i/o devices or even more advanced CPU architectures. Its adaptability allows developers to extend its capabilities by adding peripherals, enhancing instruction sets, or implementing additional features to simulate diverse computing scenarios.

* Understanding legacy systems:

One intriguing facet lies in its contribution to understanding historical computing. The 16-bit CPU on the hack platform serves as a portal to explore the design principles of past 16-bit computing systems, fostering a deeper comprehension of the evolutionary path of computer architectures.

* Simulation and emulation:

Adaptation for virtual environments or simulation and emulation purposes stands as another noteworthy application. This adaptability enables the creation of controlled environments for educational pursuits or even professional software development experimentation within a simulated 16-bit computing environment.

* Demonstration of concepts:

In presentations and demonstrations, the 16-bit CPU serves as a tangible illustration of fundamental computer architecture concepts. It vividly showcases aspects like instruction execution, memory management, and control flow, simplifying the comprehension of complex ideas.

* Skill development:

Lastly, individuals engaged in this project undergo a holistic skill development journey. They gain proficiency in hardware description languages (hdl), logic design, assembly language programming, and system integration, fostering a versatile skill set crucial in the realm of computer engineering.

In summary, the 16-bit CPU on the hack platform transcends its role as a mere educational tool; it becomes a catalyst for exploration, experimentation, and skill enhancement, contributing significantly to the multifaceted landscape of computer science and engineering. Its adaptability and comprehensive functionality make it a versatile asset in various learning, developmental, and exploratory endeavours within these domains.

**CHAPTER VI**

**CONCLUSION:**

In conclusion, the successful design and implementation of the 16-bit CPU on the Hack platform mark a significant achievement in the exploration of computer architecture. This project aimed to bridge the theoretical understanding of CPU design with practical application, providing a comprehensive educational experience for both students and enthusiasts. Through meticulous planning, logic design, and systematic implementation, the 16-bit CPU now stands as a testament to the intricacies of hardware construction.

The project's significance lies not only in its tangible outcome but also in the insights gained during its development. The hands-on experience of building a CPU has deepened our understanding of the coordination between various components, the intricacies of control unit design, and the importance of a well-defined instruction set. Additionally, the assembly language implementation and subsequent testing phases have illuminated the practical challenges inherent in translating high-level concepts into functional hardware.

As a versatile tool, this 16-bit CPU serves not only as a learning resource for computer architecture but also as a foundation for further exploration. Whether used for educational purposes, as a platform for experimentation, or as a starting point for future projects, the CPU on the Hack platform stands as a testament to the intersection of theoretical knowledge and practical application in the realm of computer science.

**PART 2**

**4-bit COMPARATOR**

**CHAPTER I**

**INTRODUCTION**

Build a 4-bit benchmark on NAND for the Tetris framework covers the content of the principles of digital circuits. The comparator is, in essence, the key that allows digital machines to make important decisions by measuring the relative value of two 4-bit binary numbers. By using critical gates, specifically NAND gates in this case, the project aims to reflect the unique possibilities inherent in these simple elements.

The main goal of the project is to carefully configure these doors to create a strong and reliable door. High efficiency circuit. The goal is not to create a functional comparison, but to demonstrate the depth of capabilities achieved with these simple gates. Creating a circuit that can determine whether a 4-bit number is less than, equal to, or greater than another 4-bit binary number not only represents an engineering feat, but also demonstrates the possibilities of digital design.

Importance The industry resonates with its ubiquity in digital systems. In mathematics and arithmetic, comparators play an important role in facilitating comparison of data to support decision making in various computational tasks. The universality of comparisons and branches, ranging from simple mathematical operations to complex systems, is important.

Beyond practical applications, the project also seeks to understand the interactions between digital gateway architectures. It offers applied research that dives into the nuances of energy use. Through this pursuit, professional engineers and advocates better understand and increase their understanding of important concepts of computer engineering.

In fact, creating 4-bit parallel on NAND for Tetris software is more than an academic skill; It is an introduction to the power of digital logic and the role of hardware in creating the functionality and performance of today's computer systems, which play an important role in decision making.

**CHAPTER II**

**DESIGN**

**1-bit Comparator**

**Figure 2.1**

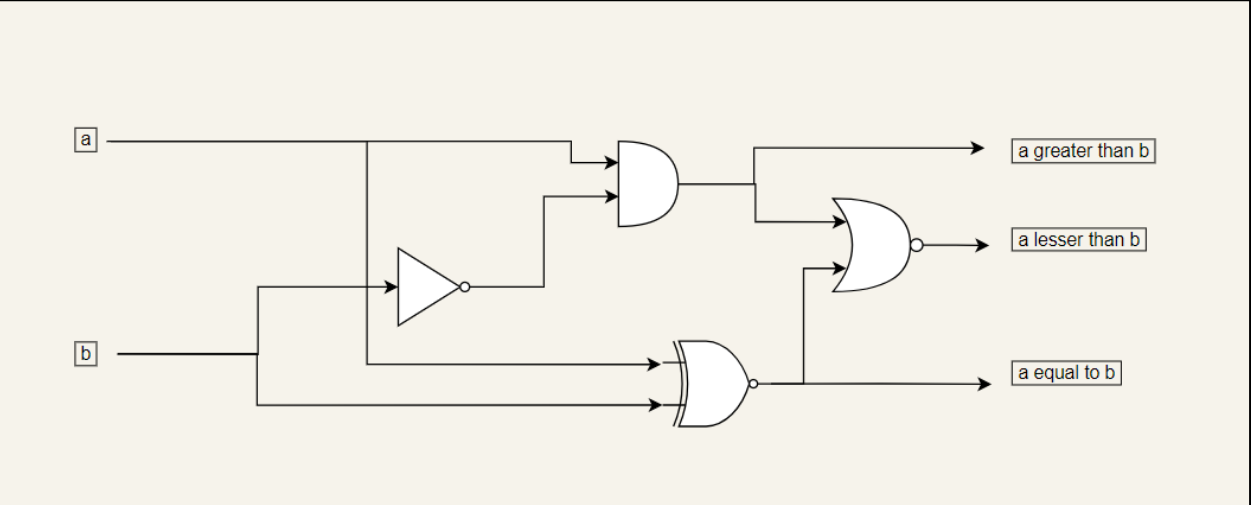


Figure 2.1 is a chip diagram for the 1 bit comparator.

**2-bit Comparator**

**Figure 2.2**

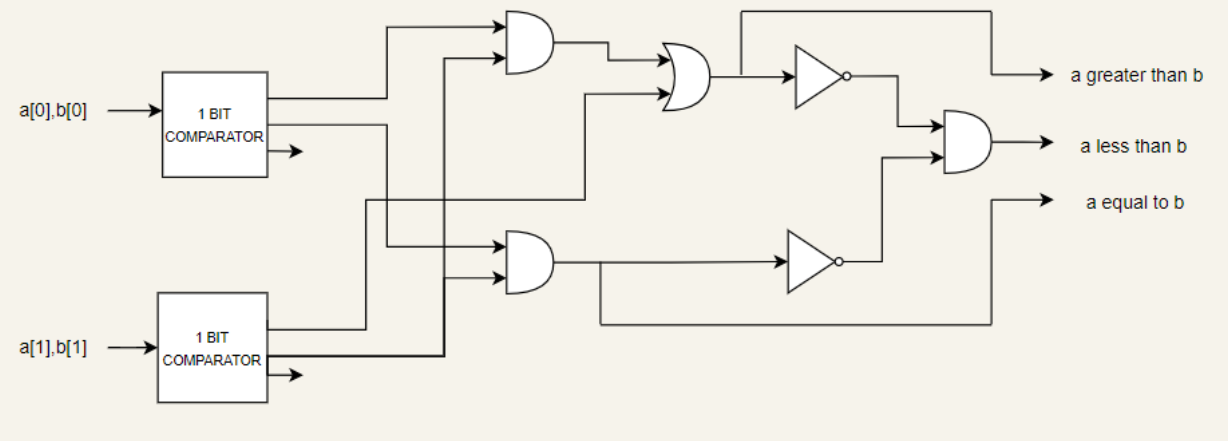


Figure 2.2 is the chip diagram for the 2 bit comparator.

**4-bit Comparator**

**Figure 2.3:**

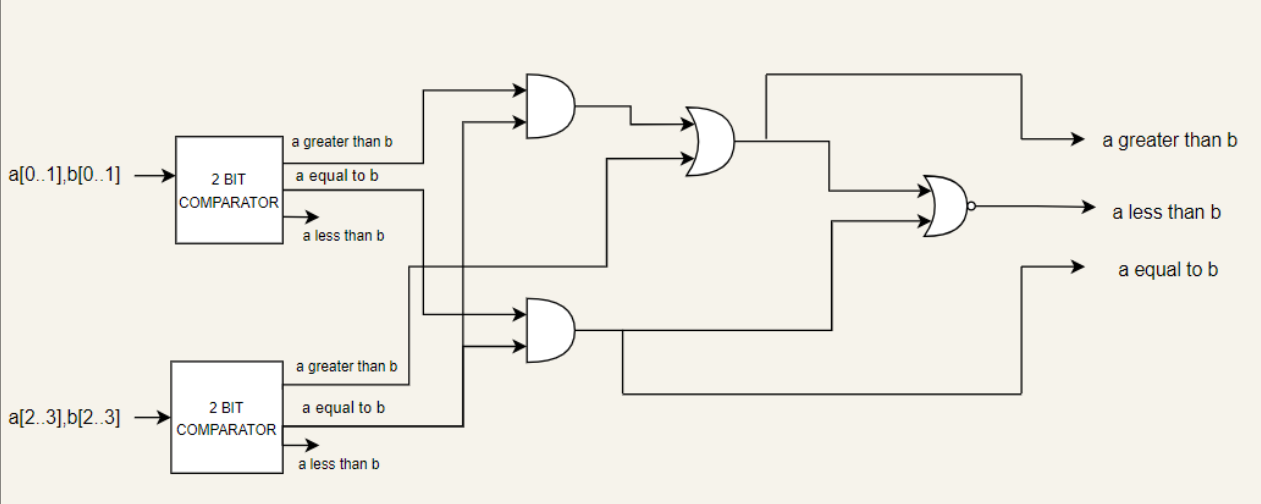


Figure 2.3 is the chip diagram for the I bit comparator.

Figure 2.4

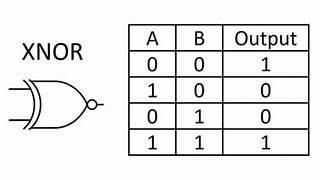


Figure 2.4 is the truth table of XNor gate

Figure 2.5

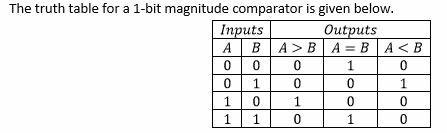


Figure 2.5 is the truth table of 1-bit Comparator

Figure 2.6

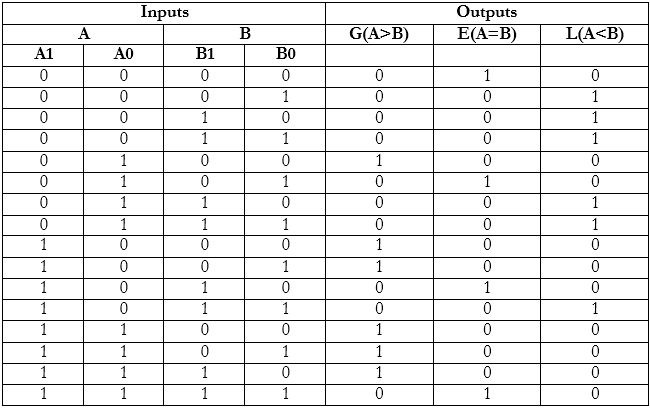


Figure 2.6 is the truth table of 2-bit Comparator

Figure 2.7

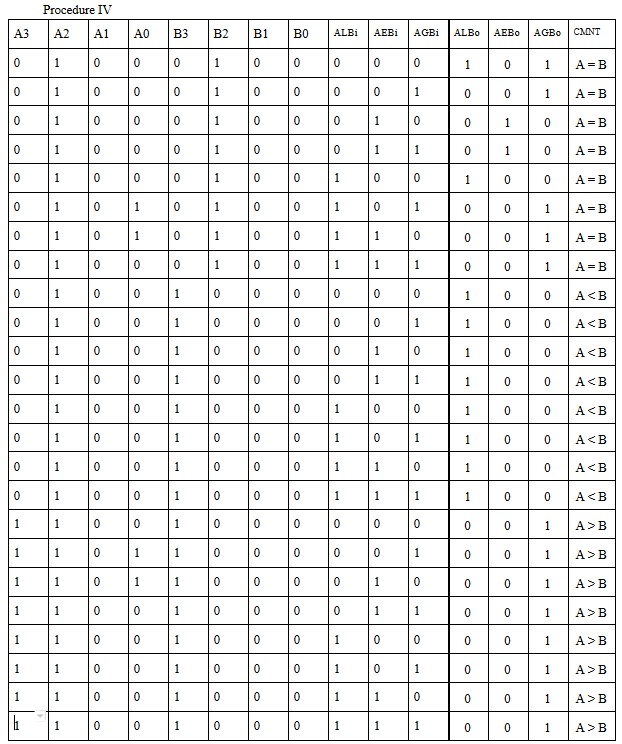


Figure 2.7 is the truth table of 4-bit Comparator

**CHAPTER III**

**METHODOLOGY**

The proposed 4-bit benchmark development process follows a path that allows for further development and rigorous testing at each stage. Detailed instructions are as follows:

1. Define Requirements:

It is important to establish clear requirements. This involves specifying the material, the output, and the desired behaviour. The important thing is to define the "less than," "equals," and "greater than" conditions.

1. Create a 1-bit Comparator:

1 bit comparator uses the Xnor gate for the relative magnitude of “ a equal to b ” and the “a greater than b” can be found by deriving the logic that “ a and not (b) ” which can be found by the truth table of the 1 bit comparator given in figure 2.4. And the a less then b can be found by the logic that if it is a not greater than b and a not equal to b then it is “ a less than b”. Then the chip diagram Figure 2.1 is drawn according to the logic.

1. Follow the 1-bit comparison in hardware:

From hardware to software, from NAND to Tetris, the theoretical design changes to the practical level. The accuracy of the hardware is verified through simulation and testing.

1. Create 2-bit Comparators:

Creation of 2-bit comparators continues, 1-bit truth table continues. The “ a greater than b” can be created by the logic that if the “ Both the 1st bit are equal” then it check “ a greater than b” of the 1 bit comparator which compares the “ 0th it” and gives the relative magnitude of “a greater than b” then “ a equal to b” only when the “ a equal to b” of the two one bit comparator are ‘1’. Then the “ a less than b” only when the “a not greater then b” and “ a not equal to b”. Then the Chip diagram is drawn according to the logic. Truth table is added in figure 2.6

1. Implement a 2-bit comparison in hardware:

Create a 2-bit version in HDL using a 1-bit comparison module. Extensive testing ensures correct operation before proceeding.

1. Create a 4-bit comparison:

Creation of 2-bit comparators continues, 1-bit truth table continues. The “ a greater than b” can be created by the logic that if the “ 3 and 4 bit are equal” then it check “ a greater than b” of the 1 bit comparator which compares the “ 1 and 2 bit” and gives the relative magnitude of “a greater than b” then “ a equal to b” only when the “ a equal to b” of the two one bit comparator are ‘1’. Then the “ a less than b” only when the “a not greater then b” and “ a not equal to b”. Then the Chip diagram is drawn according to the logic. Truth table is added in figure 2.7

1. Implement a 4-bit comparator in hardware:

Use the 2-bit comparator hardware module to create a 4-bit counterpart in HDL. Comprehensive verification methods through simulation and testing ensure reliability and accuracy.

1. Information and Description:

Information about the entire process is important, including charts, facts and opinions. A project report details the process, design decisions, simulation results, and challenges encountered and serves as an important document.

1. Iteration and Optimization:

Today, it is a matter of revisiting the design to improve it by focusing on efficiency and accuracy. Make any necessary adjustments or optimizations to improve the overall performance of the 4-bit benchmark.

1. Final verification:

Perform stringent simulation and testing to ensure accuracy and reliability of 4-bit rate.

From the development of the product from small parts to the end, despite its complex structure, this method ensures that the 4-bit comparison is good, the details are checked correctly and the designs are written to meet the requirements and standards.

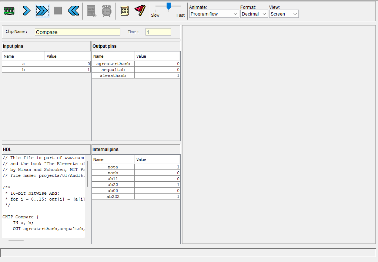
**CHAPTER IV**

**RESULTS**

The result of the project is given as screenshots of the outputs tested using the script file.

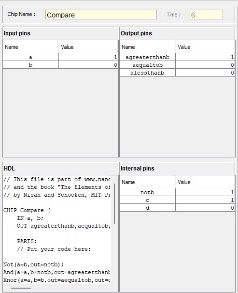
1. 1-bit comparator:

Figure 4.1.a



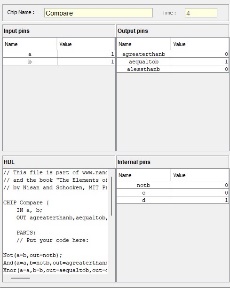
The input a is given 1 and b is 0 and output agreaterthanb is 1 rest 0

Figure 4.1.b



The input a is given 0 and b is 1 and output alesserthanb is 1 rest 0

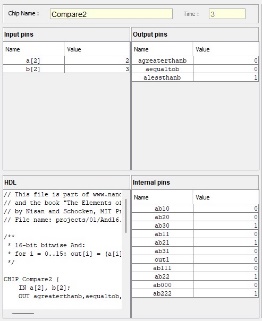
Figure 4.1.c



The input a is given 0 and b is 0 and output aequaltib is 1 rest 0

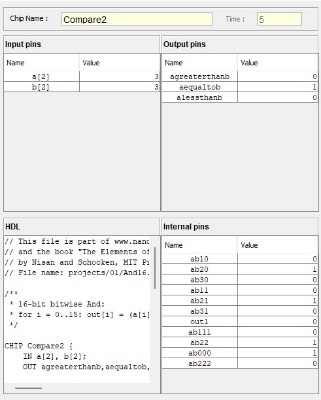
1. 2-bit comparator:

Figure 4.2.a



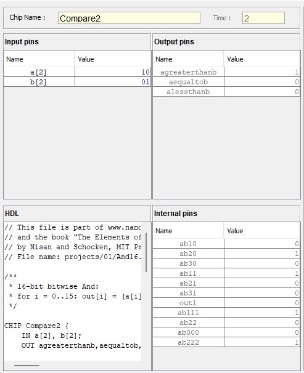
The input a is given 2 and b is 3 and output alesserthanb is 1 rest 0

Figure 4.2.b



The input a is given 3 and b is 3 and output aequaltob is 1 rest 0

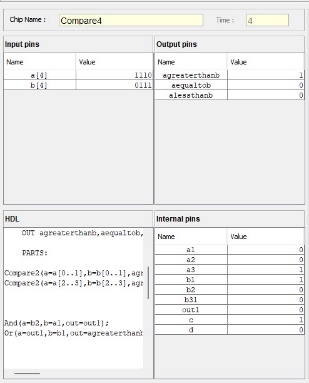
Figure 4.2.c



The input a is given 2 and b is 1 and output agreaterthanb is 1 rest 0

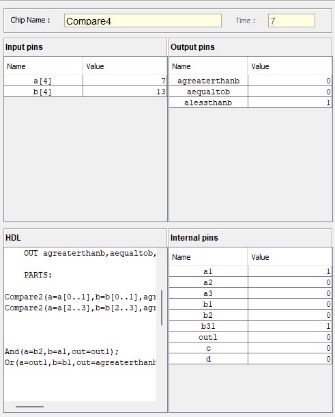
1. 4-bit comparator:

Figure4.3.a



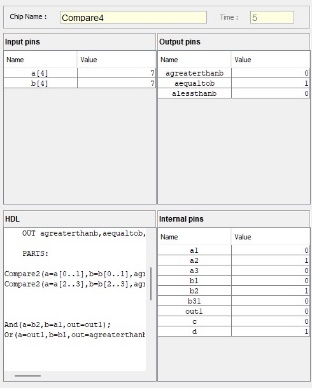
The input a is given 14 and b is 7 and output agreaterthanb is 1 rest 0

Figure4.3.b



The input a is given 7 and b is 13 and output alesserthanb is 1 rest 0

Figure4.3.c



The input a is given 7 and b is 7 and output aequaltob is 1 rest 0

**CHAPTER V**

**APPLICATIONS**

4-bit comparators are important components in digital systems and are important tools for comparing binary numbers and determining their relative values. It is important to expand the scope of digital systems and improve their performance and decision-making capabilities.

* Arithmetic and Logic Unit (ALU):

Comparisons are important in the ALU of the CPU. During an arithmetic operation such as addition or subtraction, it evaluates the size of the binary number or the flag that indicates the result (e.g., equal to, greater than, less than) to help determine the result. The status branch is necessary for the execution of the specified instruction as a result of a comparison and is based on a 4-bit comparison.

* Data analysis and search:

Analysis of algorithms and binary search using the ability of comparators to evaluate the quality of binary results. It helps in quickly comparing elements in an array or data set or storing data according to certain criteria.

* Control circuits:

Control circuits control the flow of information and instructions in digital systems. By utilizing the 4-bit comparator, these circuits make decisions, directing systems down particular paths or executing operations based on binary value comparisons.

* Address Decoding in Memory Systems:

Memory systems utilize address decoding to access specific locations. The 4-bit comparator is instrumental in this process, determining whether accessed memory addresses match the intended ones, thereby enabling seamless data retrieval and storage.

* Digital Signal Processing (DSP):

In DSP applications like audio or image processing, the comparator compares signal attributes (like amplitudes) to facilitate various manipulations or analyses, ensuring accurate processing based on binary value relationships.

**CHAPTER VI**

**CONCLUSION**

The 4-bit benchmark project represents the foundation of digital system design, incorporating both theoretical and practical ideas. Its hierarchical structure, from simple 1-bit logic to 4-bit comparison method, shows the importance of the design. This process reflects real-world performance, providing flexibility, scalability and reusability.

From NAND to Tetris Software using only NAND gates, this project provides hands-on work with digital design principles. Careful preparation of the truth table, logic design, and hardware implementation after ensuring power and accuracy at each stage of development.

At its peak, 4-bit comparison embodied the ability to process and compare complex binary data and provided insight into the arithmetic and arithmetic operations of digital systems. Its versatility reveals its applicability in many technological applications, broadening its relevance to diverse fields from mathematics to storage systems.

The project encourages performance as well as an understanding of the nature of design, abstraction and iteration. Perfectly complex systems. Combining theory with practical applications, this holistic course provides professional engineers with the fundamental knowledge necessary to enter digital systems and computer architecture. Ultimately, the success of the program is the combination of theoretical understanding and practical skills that foster a deep knowledge of digital circuit design.

**APPENDIX: CODES**

1. Hack CPU:

CHIP CPU {

IN inM[16], // M value input (M = contents of RAM[A])

instruction[16], // Instruction for execution

reset; // Signals whether to re-start the current

// program (reset==1) or continue executing

// the current program (reset==0).

OUT outM[16], // M value output

writeM, // Write to M?

addressM[15], // Address in data memory (of M)

pc[15]; // address of next instruction

PARTS:

// Put your code here:

And(a=instruction[15],b=instruction[4],out=Dload);

DRegister(in=outM,load=Dlaod,out=ALUx);

Mux16(b=outM,a=instruction,sel=instruction[15],out=Aregisterin);

Not(in=instruction[15],out=notofinstruction15);

Or16(a=notofinstruction15,b=instruction[5],out=Aload);

ARegister(in=Aregisterin,load=Aload,out=Aout,out[0..14]=addressM);

Mux16(a=Aout,b=inM,sel=instruction[12],out=ALUy);

//ALU

ALU(x=ALUx, y=ALUy, zx=instruction[11], nx=instruction[10],

zy=instruction[9], ny=instruction[8], f=instruction[7],

no=instruction[6], out=outM, zr=zr, ng=ng);

//writeM

And(a=instruction[15],b=instruction[3],out=writeM);

//pc

And(a=instruction[1],b=zr,out=out1);

And(a=instruction[0],b=ng,out=out2);

Not(in=zr,out=notzr);

Not(in=ng,out=notng);

And3input(a=instruction[2],b=notng,c=notzr,out=out3);

And3input(a=instruction[1],b=instruction[0],c=instruction[2],out=out4);

Or4input(a=out1,b=out2,c=out3,d=out4,out=out5);

And(a=instruction[15],b=out5,out=load);

Not(in=load,out=inc);

PC(in=Aout, inc=inc, load=load, reset=reset, out[0..14]=pc);

}

1. 1-bit Comparator:

CHIP Compare {

IN a, b;

OUT agreaterthanb, aequaltob, alessthanb;

PARTS:

// Put your code here:

Not(a=b, out=notb);

// Taking not of 'b'

And(a=a, b=notb ,out=agreaterthanb, out=c);

if a is '1' and b is '0' the "a is greater than b"

Xnor(a=a, b=b, out=aequaltob, out=d);

// if a and b are the same then it is equal so we are using the Xnor for finding "an equal to b" .

Nor(a=c, b=d, out=alessthanb);

// if it is not "a equal to b" and "a greater than b" then it is "a less than b".

}

1. **2-bit Comparator**:

CHIP Compare2 {

IN a[2], b[2];

OUT agreaterthanb, aequaltob, alessthanb;

PARTS:

***Compare(a=a[0], b=b[0], agreaterthanb=a1, aequaltob=a2, alessthanb=a3);***  // giving '0' bit of a and b to the 1-bit comparator.

***Compare(a=a[1],b=b[1], agreaterthanb=b1, aequaltob=b2, alessthanb=b31);***

// giving '1' bit of a and b to the 1 bit comparator

// cases for "a greater than b":

***And(a=b2,b=a1,out=out1);***

// case 1: if both '1' bits are equal it will check the '0' bit and decide the greater number.

***Or(a=out1, b=b1, out=agreaterthanb, out=c);***

// case 2: if '1' bit of a is greater than b then ultimately the a is greater.

// So, using the 'OR' gate for both cases to obtain ' a greater than b '

// case for "a equal to b" :

***And(a=a2, b=b2, out=aequaltob, out=d);***

//case 1: The "a equal to b" only when the '1' and '0' bit of a and be must equal.

// case for " a less than b" :

***Nor(a=c, b=d, out=alessthanb);***

// 'a less than b' only when the 'a not equal to b' and 'a not greater than b'.

}

1. **4-bit Comparator:**

CHIP Compare4 {

IN a[4], b[4];

OUT agreaterthanb, aequaltob, alessthanb;

PARTS:

***Compare(a=a[0..1], b=b[0..1], agreaterthanb=a1, aequaltob=a2, alessthanb=a3);***

***Compare(a=a[2..3],b=b[2..3], agreaterthanb=b1, aequaltob=b2, alessthanb=b31);***

***And(a=b2,b=a1,out=out1);***

// case 1: if both '1' bits are equal it will check the '0' bit and decide the greater number.

***Or(a=out1, b=b1, out=agreaterthanb, out=c);***

// case 2: if '1' bit of a is greater than b then ultimately the a is greater.

// So, using the 'OR' gate for both cases to obtain ' a greater than b '

// case for "a equal to b" :

***And(a=a2, b=b2, out=aequaltob, out=d);***

//case 1: The "a equal to b" only when the '1' and '0' bit of a and be must equal.

// case for " a less than b" :

***Nor(a=c, b=d, out=alessthanb);***

// 'a less than b' only when the 'a not equal to b' and 'a not greater than b'. }

**REFERENCES**

* The Elements of Computing system - Building a Modern Computer

from First Principles

* Digital Design - with an introduction to verilog HDL

M. MORRIS MANO

MICHAEL D. CILETTI