Project A – Hack CPU  
And  
Project B – 16-bit parallel adder/subtractor

A Project Report  
Submitted by

GROUP 15

TEAM MEMBERS

1. SANGGIT SAARAN K C S CB.SC.U4AIE23247

2. K S VENKATRAM CB.SC.U4AIE23236

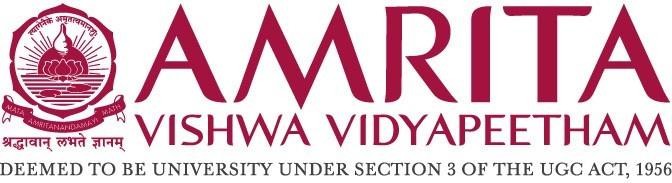
3. VISHAL SESHADRI B CB.SC.U4AIE23260

4. SURYA HA CB.SC.U4AIE23267

5. HARIVAARTHAN TD CB.SC.U4AIE23228

*As a part of subject*

22AIE102 – Elements of Computing 1



**DEPARTMENT OF ARTIFICIAL INTELLIGENCE**

**AMRITA VISHWA VIDYAPEETHAM**

COIMBATORE – 641 112

**DECEMBER - 2023**

**INDEX**

**Topics Pg. No**

1. Introduction……………………………………………………………… 3
2. Objectives ……………………………………………………………….. 4
3. Methodology…………………………………………………..………… 5
4. Results and Discussion…………………………………………..……….15
5. Applications………………………………………………………...…….19
6. Conclusion………………………………………………………….…… 22
7. References ………………………………………………………….…… 23
8. Appendix…………………………………………………………….……24

**INTRODUCTION**

**Project A:**

Designing and implementing a central processing unit (CPU) is the core task in computer architecture, and all aspects of digital systems. The Hack CPU project is a major investigation into the complex structure of modern CPUs. It strives to produce an open-source 16-bit processor that will be convenient and educational at the same time. The Hack CPU project report offers a detailed description of the design choices, implementation methods and results.

One of the primary motivations behind the Hack CPU project is its educational value. The CPU's simplicity and transparency make it an ideal platform for learners to grasp fundamental concepts in computer architecture. Aspiring computer scientists, students, and educators can utilize the Hack CPU as a hands-on tool to bridge the gap between theoretical concepts and practical implementation.

**Project B:**

The 16-bit parallel adder/subtractor is a crucial component in digital computing systems, designed to perform arithmetic operations on binary numbers with a precision of 16 bits. This project report aims to provide a comprehensive overview of the design, implementation, and performance of a 16-bit parallel adder/subtractor. The project involves combining the functionalities of addition and subtraction within a single circuit to enhance computational flexibility.

A high-performance 16-bit parallel adder/subtractor is essential in digital computing applications, including arithmetic operations in microprocessors, signal processing units, and other computational systems. The successful completion of this project contributes to the advancement of digital circuit design and enhances the efficiency of arithmetic processing in digital systems.

**OBJECTIVES**

**Part A:**

**Design and implement a 16bit hack CPU**

The primary objective of the Hack CPU project is to design and implement a simplified yet functional 16-bit CPU architecture. This involves developing a clear understanding of instruction set architecture, control unit design, and memory management. The project aims to provide a hands-on educational experience for learners to grasp the fundamental principles of computer organization and assembly language programming. By creating a CPU with a limited instruction set, the objective is to make the architecture accessible for educational purposes while maintaining its ability to execute instructions and manage data effectively. Rigorous testing and simulation will be employed to ensure the reliability and correctness of the implemented CPU, emphasizing its educational significance and practical application in teaching computer architecture.

**Part B:**

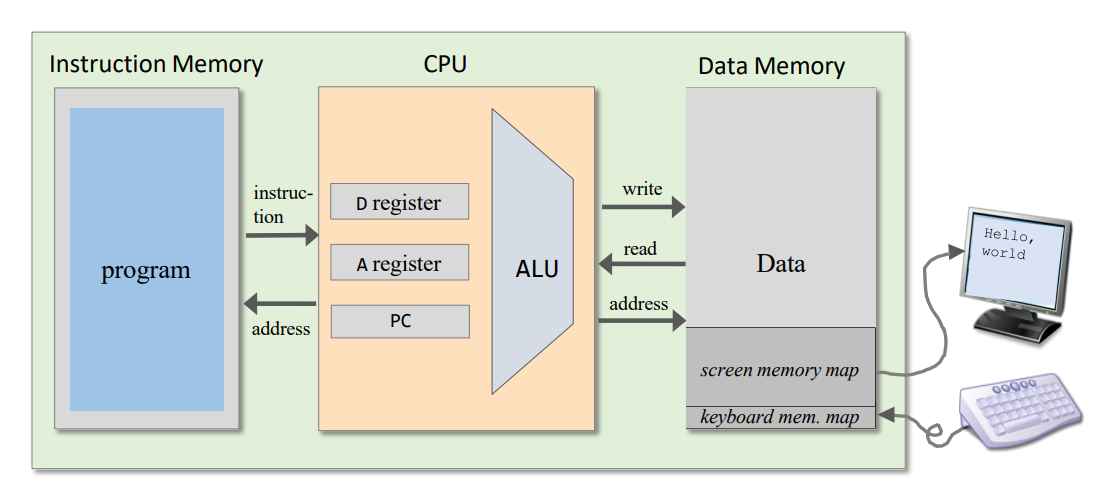
**Design and implement a 16-bit parallel adder/subtractor (a circuit which can perform both operation)**

The primary objectives of the 16-bit parallel adder/subtractor project are centered around designing and implementing an efficient arithmetic circuit capable of performing both addition and subtraction operations on 16-bit binary numbers. This involves exploring various logic gate configurations, addressing carry/borrow propagation, and implementing overflow handling. The project aims to provide a versatile component for digital systems where arithmetic operations are fundamental. Rigorous testing and simulation will be conducted to validate the correctness and reliability of the 16-bit parallel adder/subtractor, ensuring its effectiveness in diverse computational applications. The project seeks to contribute to the advancement of digital circuit design and enhance the precision and efficiency of arithmetic computations in digital systems.

**METHODOLOGY**

Part A:

Basic Implementation of the topmost Hack Computer Chip.



**Basic Working of the Proposed Implementation:**

**The Components:**

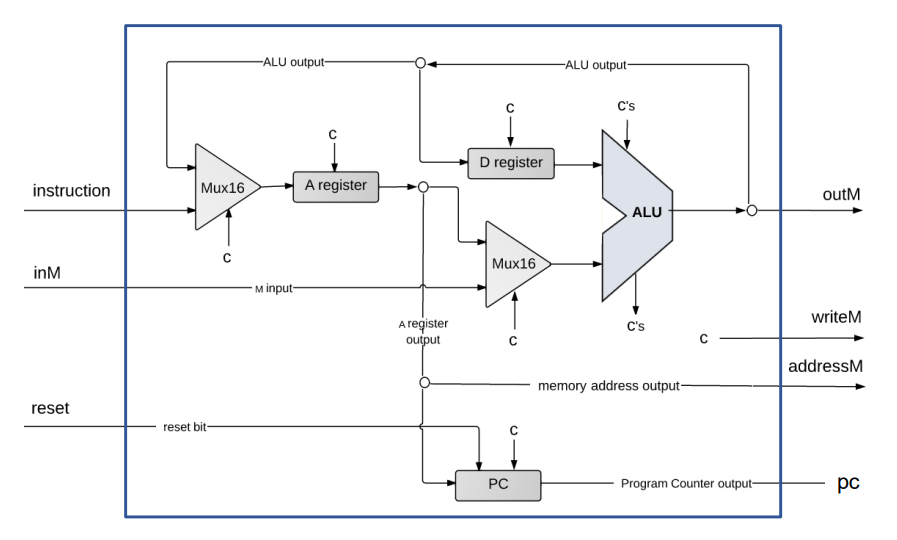
* **PC:** The programming Counter will tell give the address of the instruction it needs to execute. (Addition can be considered as an instruction).
* **Instruction Memory:** The instruction memory stores all the functions that you want to execute. (Examples are addition, subtraction etc.). Each memory address holds a 16-bit instruction. Acts like a read-only memory (ROM) from the CPU's perspective.
* **Data Memory:** It will store all the variables on which you want to execute your function. You can also read the variables stored in it. Stores data that can be changed during program execution. Works like a read-write memory (RAM). Also holds 16-bit values at each address
* **CPU:** CPU reads the 16-bit instruction from the instruction memory at the address specified by the PC. It is the main unit of any computer that helps in processing data. It can do crucial functions such as performing calculations, accessing stored data and so on. The CPU can be considered as the brain of a computer. Hack CPU uses separate instruction and data memories for efficiency and clarity. The fetch-execute cycle ensures instructions are executed in the correct order. Jump instructions allow for flexible control flow within programs. CPU determines the instruction type (A-instruction or C-instruction) and its components.

**Working of the Computer Chip:**

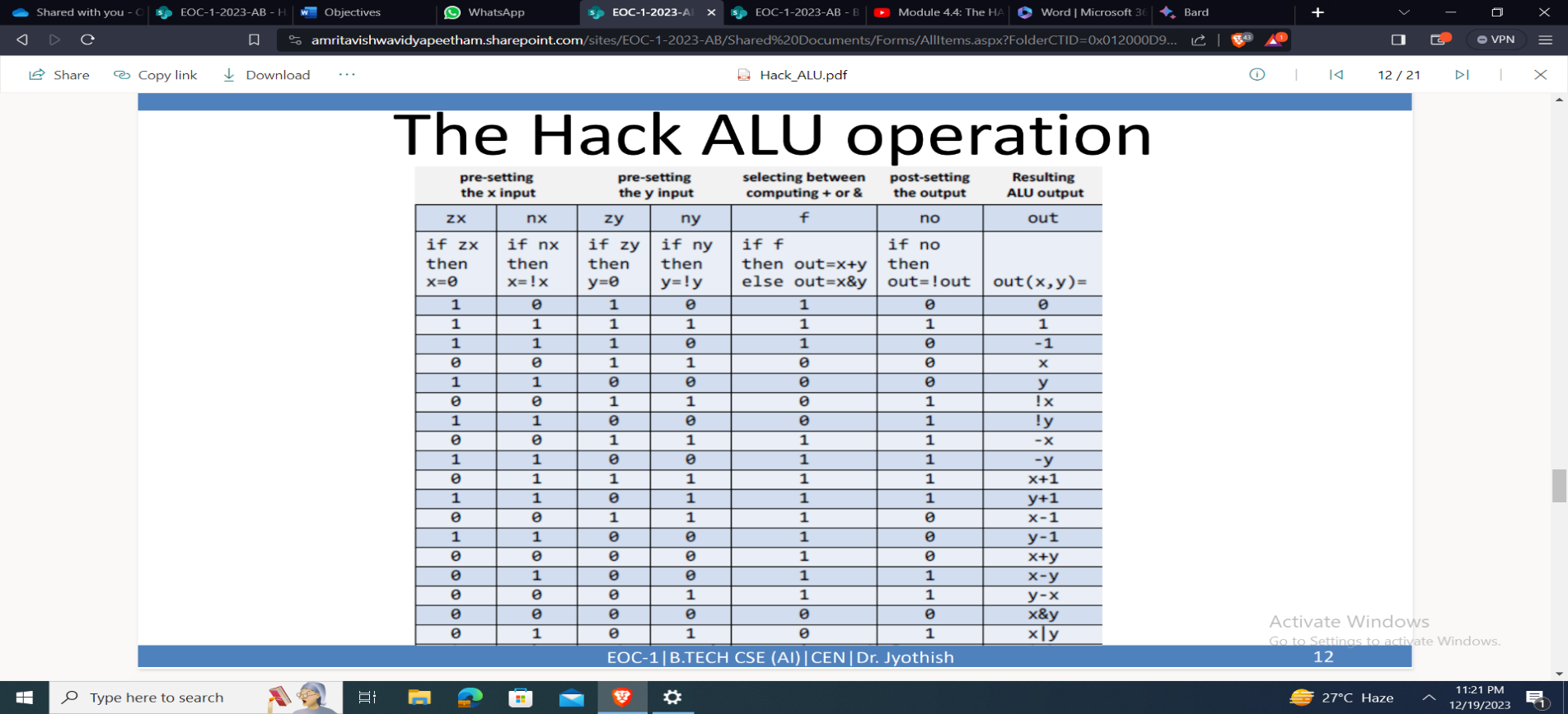
* The PC gives address of the instruction that the CPU should be performing which is stored in the instruction Memory.
* The Instruction Memory provides the CPU with the instructions.
* In the CPU we have write M, Out M, Adress M.
* If you give a particular address and keep, Write M as zero then you will be able to read the data stored in that particular address.
* The data memory provides CPU with IN M, when we provide the address M to the data memory. The value stored in that particular address will be provided to the CPU and the CPU can perform the required operations on the given stored values
* If Write M is 1 then we can write into memory or if an out M if present it can be stored into the data memory.
* The reset function will reset all the previous instructions taken by the CPU.

**Hack CPU**

**DESIGN:**



The CPU is the brain of the Hack computer, responsible for fetching, decoding, and executing instructions. It has these key components:

**Arithmetic Logic Unit (ALU):** Performs mathematical and logical operations like addition, subtraction, and AND/OR on two 16-bit inputs.

Here are a list of functions that can be performed by the hack ALU:

* **Program Counter (PC):** Tracks the address of the next instruction to be fetched. The program counter, like a bookmark in a book, keeps track of which instruction the computer is reading next. As each instruction is done, it flips to the next page (increases the counter) to keep the computer running smoothly through your program. It has 3 main functions they are:
* Reset (Highest Priority)
* Load (Second Highest priority)
* Increment (Least Priority)
* **Instruction Memory:** Holds the current instruction being decoded. It can input 2 types of instruction. They are:
* A instruction
* C Instruction
* A and D Registers: General-purpose 16-bit registers used for storing intermediate data.
* A Register: Primarily functions as an address holder, pointing to locations in memory for data access or jump instructions. Holds temporary values during calculations.
* D Register: Primarily serves as a temporary storage for data, often holding results from ALU operations.
* Both A and D registers can't hold addresses and data simultaneously. When used as an address, the A register's value becomes temporarily unavailable for data processing.

Mux: Generally used for selecting any of the given Data lines.

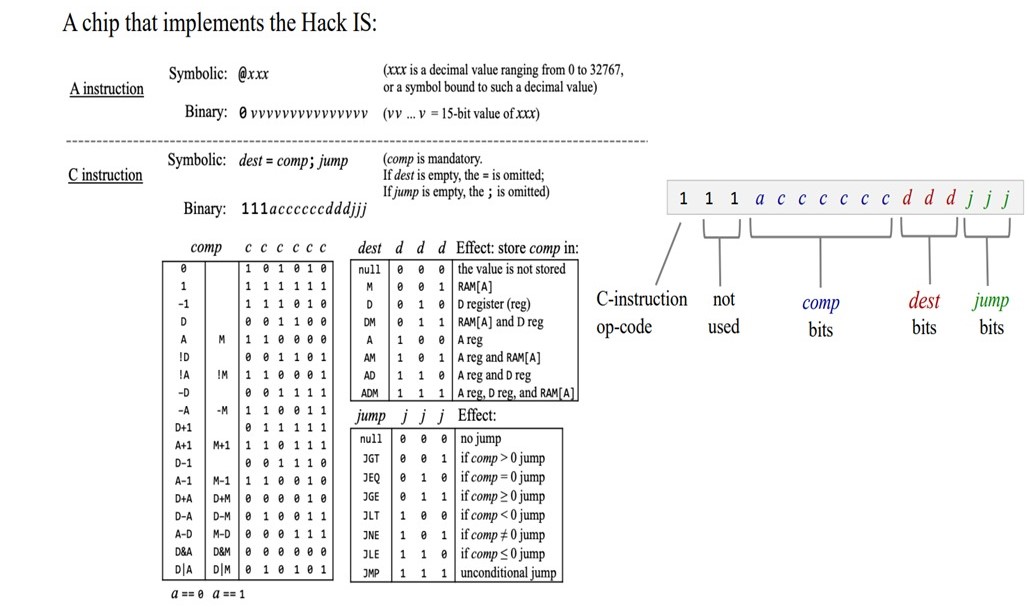
* **Control Unit:** Decodes the instruction in the IR and generates control signals that activate various parts of the CPU to execute the operation.

**Working of the hack CPU:**

* In the CPU there is a reset which is given externally to the CPU that will reset the program counter, the A register &D register (storage elements).
* The CPU will take instruction one by one from the CPU memory pointed out by the program counter. Once the PC is sent to instruction memory it send the instruction to the CPU.
* The CPU interprets this instruction and execute it as a part of that it can read or write the

Memory by specifying the address.

* The 16-bit instruction can be of 2 types A Instruction and C Instruction:
* A – Instruction: Load a 16-bit value into the A-Register (used for memory addresses). Ex: @200 loads the value 200 into the A-Register.
* C – Instruction: Perform computations or data manipulations. Ex: D=D+A adds the values in the A-Register to the D-Register



* In the instructions 6th to 11th bit must be given to ALU
* The first input X we give will always be stored in D register, the other input Y will be stored in M[A] or A register depending upon a bit (12th bit). If a=0 then output of a register will go to ALU else a=1 then InM(from data memory) value will go to ALU.

**Note:** The output of the ALU I.e. outM always to memory, a register and it will also go to d register based on condition set.

**Note:** If the T bit is zero it means it is an instruction, the 15-bit coming from the instruction should go to a register. If T is one it means it is c instruction, then a register will be updated if it’s in destination i.e. d1=1

* So, an A register only load if t=0 or t=1&d1=1
* So, either out m or instruction can be stored in a register and one can be selected using MUX16 gate
* D register will be loaded if T=1 and d2=1 i.e. when it is a destination and it will be loaded with output of ALU (an and gate for t and d2 help realize this condition.)
* The address M will come from A register (since a register store addresses).

**Note:** During Tick of the clock the information from the rom gets into the input of the CPU and then it does computation and at tock the results are stored back in the d and a memory depending upon the destination

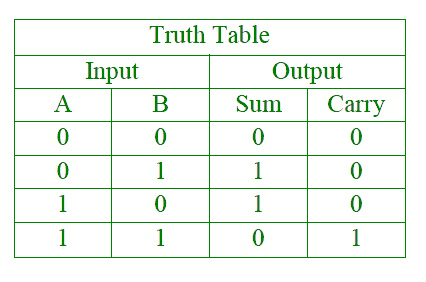
* In case of jump the pc gets updated accordingly and during next tick information flows from ROM

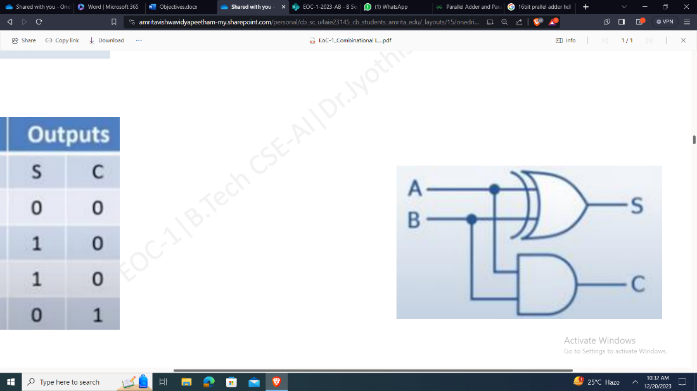
**Pc:**

* In the PC, the load condition is satisfied when it is not in a reset state. It is loaded when a jump instruction is encountered. Therefore, our increment operation will not be a load condition. The conditions for loading are as follows:
* If T is 1, and either J1 and J2 and J3 are true, or it is a conditional jump with ZR (zero) and NG (negative) both equal to zero, and J3 is set, or J1 is 1 and NG is 1, or J2 is 1 and ZR is 1.
* The load condition can be expressed as Load = T & ((J2 & ZR) | (J1 & NG) | (J3 & ~ZR & ~NG) | (J1 & J2 & J3)).
* When the load condition is true, the memory value should be stored in a register for instruction execution; otherwise, it is simply incremented. This logic ensures that the program counter (PC) is updated appropriately based on the specified conditions for loading. For writing in the data memory, it should satisfy the condition T&d3

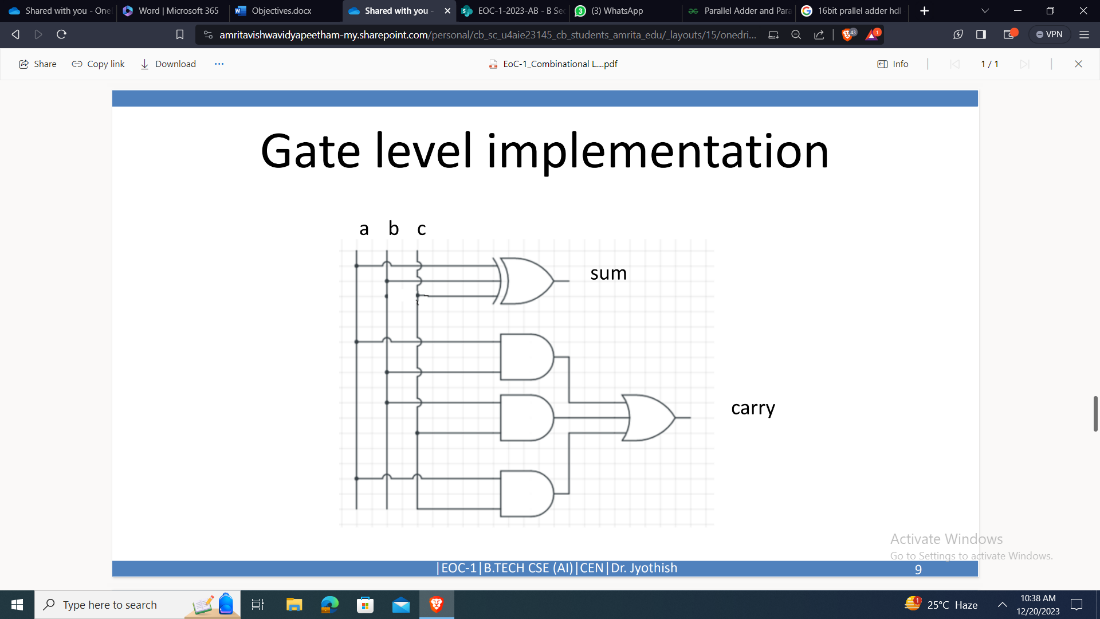
**Part B**

A 16-bit parallel adder and subtractor comprise two primary components: 16 full adders, and additionally, a half-adder. The structure includes 15 full adders in combination with a single half-adder, resulting in a total of 16 full adders collectively contributing to the functionality of the parallel adder and subtractor circuit.

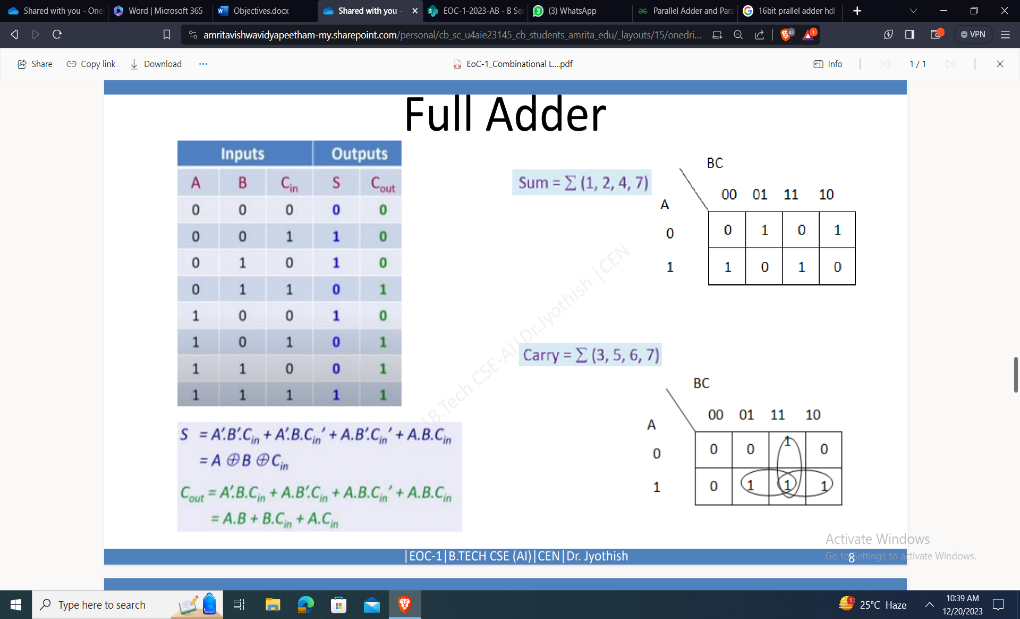
**1)HalfAdder:**

Designed to add two bits

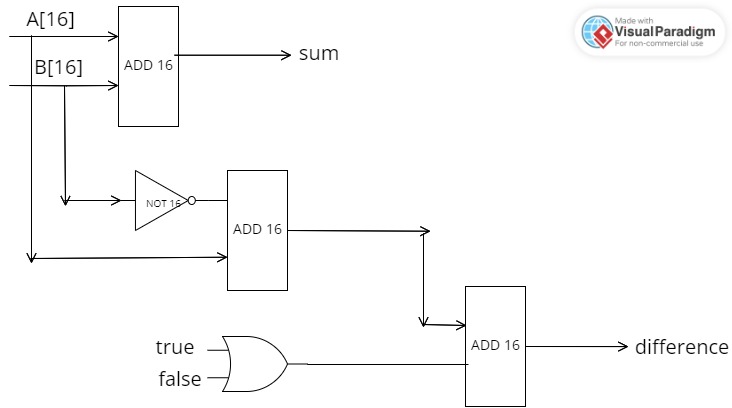
* S=A’B+AB’
* Carry=AB

**2)FullAdder:**

* Has 3 inputs,2 outputs
* Inputs: A, B, carry input C
* Output: Sum, Carry output



**Design of 16-bit parallel adder and subtractor:**



**Components:**

* Add16: used for adding two 16-bit number.
* Not16: used for inverting a 16-bit number
* Or16: used for performing or operation on corresponding bits of 2 numbers.

**Working:**

* Add16(a=a, b=b, out=sum):
  + Performs a 16-bit addition of inputs a and b.
  + Equation: sum = a + b (using 16-bit binary addition)
* Not16(in=b, out=nb):
  + Performs a 16-bit bitwise NOT operation on input b.
  + Equation: nb = ~b (inverts each bit of b)
* Add16(a=a, b=nb, out=ndifference):
  + Adds a to the inverted value of b (nb).
  + Equation: ndifference = a + nb (using 16-bit binary addition)
* Or16(a[0]=false, b[0]=true, out=t):
  + Performs a 16-bit bitwise OR operation with a[0] set to false and b[0] set to true.
  + Equation: t = 0000000000000001 (only the first bit is 1)
* Add16(a=ndifference, b=t, out=difference):
  + Adds ndifference to t.
  + Equation: difference = ndifference + t (using 16-bit binary addition)

**Example:**

This can be explained by taking a simple example with 4 bit numbers

Let us take

6= 0110

2=0010

For sum we can simply add them using general binary addition.

We get 6+2=8 or 1000

Now we also want to find the difference I.e. 6-2

We will be using two complement concept to achieve this(in twos complement final carry ignored and hdl follows twos complement)

We will convert 2 into 1’s complement by using not 4

So, 0010 will become 1101

Now we add 6 and 2

0110

1101

-----------

=1 0011

-----------

Since hdl takes two’s complement we ignore that underlined

Now we add the 1 to the 0011 to get our required value.

0011

+ 1

--------

0100

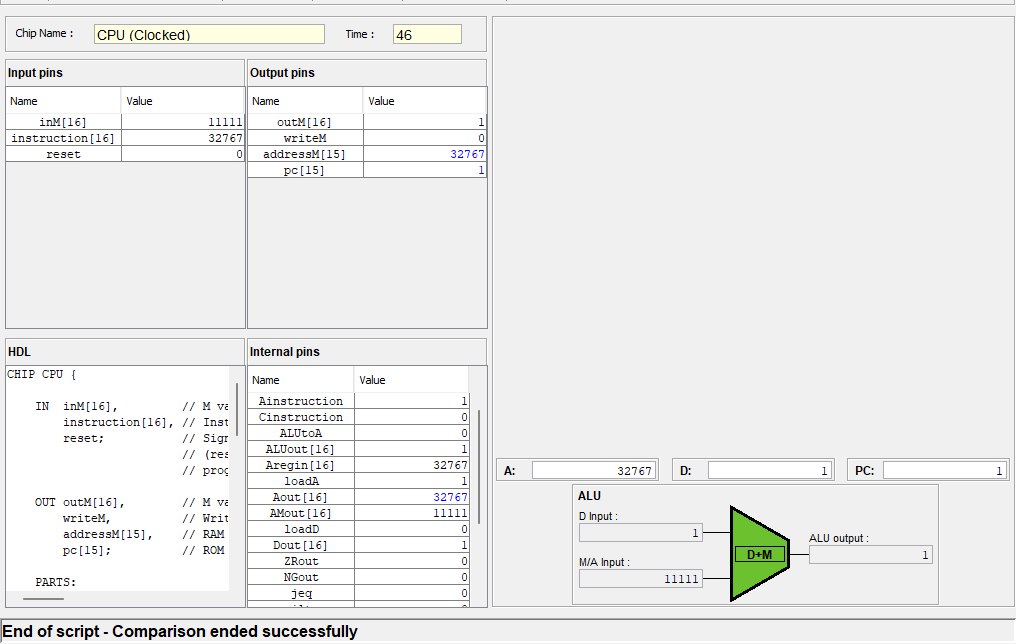
--------

We know that 0100 is 4

This is how the code works

**RESULT AND DISCUSSION**

**Project A**

**Result:**

**Discussion:**

**Basic Understanding of Computer Architecture:**

* + **Instruction Cycle:** we gain firsthand experience with the core concept of the fetch-decode-execute cycle, observing how instructions are fetched from memory, decoded, and executed by the various CPU components.
  + **Control Flow:** Witnessing how branching and jump instructions alter the program counter and control the flow of execution sheds light on conditional logic and program execution paths.
  + **Data Manipulation:** Implementing ALU operations provides a practical understanding of how basic arithmetic and logical operations are performed in hardware.

**Deeper Learning**

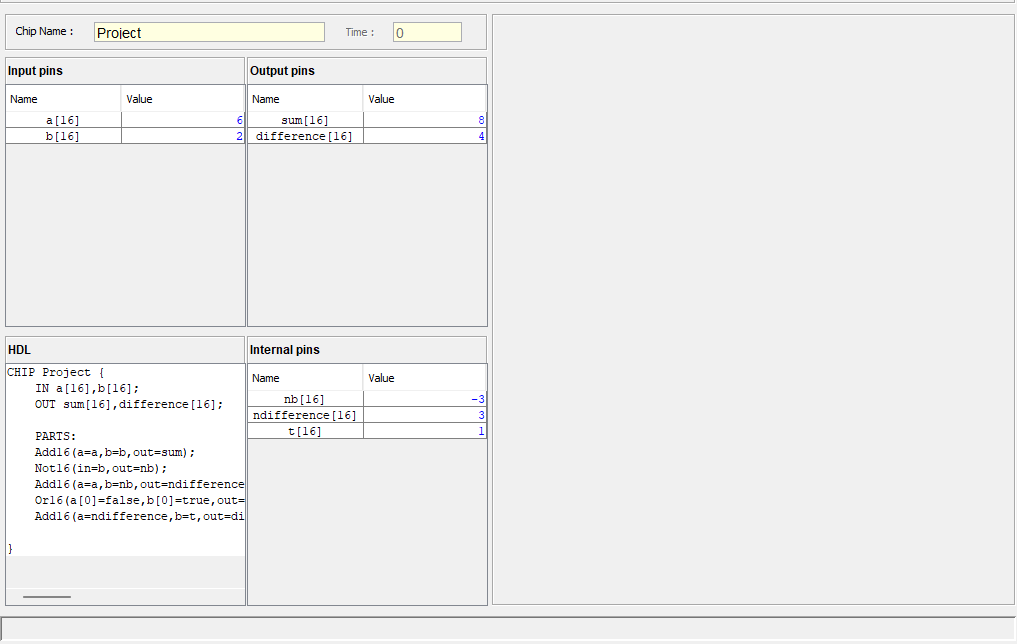
* + **Memory Access:** Exploring memory addressing and data read/write operations enhances your understanding of memory hierarchy and its impact on performance.
  + **Instruction Set Architecture (ISA):** Implementing the specific Hack instruction set provides a concrete example of how an ISA defines the operations a processor can perform and how programmers interact with it.
  + **Hardware-software interaction:** we gain a deeper appreciation for the intricate relationship between hardware components and software instructions, recognizing how specific instructions map to hardware operations.

**Additional benefits:**

* + **Problem-solving and debugging skills:** Building and troubleshooting the Hack CPU hones our problem-solving and debugging skills, as you identify and rectify logic errors and hardware limitations.
  + **Confidence and understanding:** Successfully implementing the Hack CPU can boost our confidence in tackling more complex computer architecture concepts and programming languages.
  + **Foundation for further learning:** This experience serves as a strong foundation for studying more advanced computer architectures and their intricacies.

Remember, the depth of understanding gleaned from implementing the Hack CPU depends on our approach and the level of detail we delve into. It can range from a basic grasp of the core principles to a comprehensive understanding of hardware-software interaction and specific details of the Hack architecture.

**Project B**

**Result:**

**Discussion:**

**Basic Understanding of 16-Bit Parallel Adder/Subtractor Design:**

* **Arithmetic Operation Cycle:** The implementation of a 16-bit parallel adder/subtractor provides hands-on experience with the fundamental arithmetic operations involved in digital computing. The project allows us to observe the sequence of actions within the adder/subtractor, emphasizing the importance of parallel processing for efficient computation.
* **Control Flow in Arithmetic Operations:** Exploring the control flow within the 16-bit parallel adder/subtractor, especially in managing carry and borrow operations, sheds light on how the circuit controls the flow of arithmetic operations. Understanding how specific conditions influence the control signals contributes to a more comprehensive grasp of parallel arithmetic.

**Deeper Learning in 16-Bit Parallel Adder/Subtractor Design:**

* **Memory Access in Arithmetic Circuits:** Delving into the intricacies of memory access within the context of the parallel adder/subtractor deepens our understanding of memory hierarchy. Exploring how data is read from and written to memory in the arithmetic circuit provides valuable insights into the impact of memory operations on overall performance.
* **Custom Instruction Set:** Implementing a specific instruction set for the 16-bit parallel adder/subtractor enhances our comprehension of Instruction Set Architecture (ISA). This experience showcases how the ISA defines the permissible operations and influences the interaction between programmers and hardware components.

**Additional Benefits of Implementing a 16-Bit Parallel Adder/Subtractor:**

* **Problem-Solving and Debugging Skills:** Building and troubleshooting the 16-bit parallel adder/subtractor refines our problem-solving and debugging skills. Identifying and rectifying logic errors and optimizing hardware components contribute to a skill set crucial in digital circuit design.
* **Confidence and In-Depth Understanding:** Successfully implementing the 16-bit parallel adder/subtractor instills confidence in tackling more complex computer architecture concepts. The hands-on experience builds a deeper understanding of parallel processing and arithmetic circuits.

**APPLICATION**

**PROJECT A**

The Hack CPU, while primarily used for educational purposes, offers interesting applications and potential future advancements:

**Current Applications:**

* **Teaching Computer Architecture:** It serves as a simplified yet powerful platform for introducing fundamental concepts like instruction sets, memory access, and control flow. Its open-source nature allows students to tinker and experiment, solidifying their understanding.
* **Hardware Design Teaching:** Building a physical or virtual Hack CPU can provide valuable hands-on experience in digital logic design and computer hardware principles.
* **Software Development Tool:** For small, embedded systems or specific tasks, the Hack architecture can be a starting point for designing a custom and efficient instruction set tailored to a specific use case.
* **Research Platform:** Researchers can use the Hack as a reference architecture to explore concepts in instruction set design, compiler optimization, and other areas of computer architecture.

**Future Scope:**

* **Customizable Architectures:** Building upon the Hack's base, future iterations could explore modularity, allowing users to add custom instructions or components for specific applications like cryptography or signal processing.
* **Educational Expansion:** Integrating advanced topics like pipelining or multiprocessing into the Hack architecture could further enhance its educational value for computer science students.
* **Hardware/Software Co-design:** Exploring deeper integration between the Hack architecture and specialized hardware could push the boundaries of performance and efficiency for specific tasks.
* **Virtualization and Simulation:** Enhanced virtualization and simulation capabilities for the Hack could provide a robust platform for testing and developing software for real-world embedded systems.

**PROJECT B**

The 16-bit parallel adder/subtractor, although primarily developed for educational purposes, exhibits diverse applications and promises potential advancements:

**Current Applications:**

* **Educational Tool for Digital Arithmetic:** The 16-bit adder/subtractor serves as an instructive platform, introducing essential concepts in digital arithmetic, parallel processing, and control flow. Its accessibility and open-source nature empower students to experiment and deepen their understanding of hardware design principles.
* **Hands-On Learning in Hardware Design:** Constructing a physical or virtual 16-bit adder/subtractor provides students with valuable hands-on experience in digital logic design and the fundamentals of computer hardware architecture.
* **Embedded Systems Development Tool:** The 16-bit architecture offers potential as a foundation for designing custom instruction sets tailored to specific applications. This adaptability can be particularly useful for small, embedded systems or specialized tasks.
* **Research Reference in Computer Architecture:** Researchers can utilize the 16-bit adder/subtractor as a reference architecture to investigate various aspects of instruction set design, compiler optimization, and broader computer architecture concepts.

**Future Scope:**

* **Modular and Customizable Architectures:** Future iterations of the 16-bit adder/subtractor could explore modularity, allowing users to incorporate custom instructions or components. This adaptability would cater to specific application domains such as cryptography or signal processing.
* **Expansion of Educational Content:** The architecture's educational value can be further enriched by incorporating advanced topics such as pipelining or multiprocessing. This expansion would provide computer science students with a more comprehensive understanding of modern computing concepts.
* **Hardware/Software Co-Design Exploration:** Delving into deeper integration between the 16-bit architecture and specialized hardware components could push the boundaries of performance and efficiency for specific computational tasks.
* **Enhanced Virtualization and Simulation:** The 16-bit adder/subtractor could be enhanced with advanced virtualization and simulation capabilities, providing a robust platform for testing and developing software targeting real-world embedded systems.

**CONCLUSION**

In conclusion, the successful completion of the Hack CPU and 16-Bit Parallel Adder/Subtractor projects represents a significant milestone in the realm of digital circuit design and computer architecture. Both projects contribute valuable insights into the complexities of creating efficient and functional components for digital systems.

**Hack CPU Project:**

The Hack CPU project involved the design and implementation of a simplified yet powerful 16-bit CPU architecture. The CPU serves as the brain of a computer, executing instructions and managing data flow. Through the Hack CPU project, we gained a deep understanding of instruction set architecture, control unit design, and memory management.

**16-Bit Parallel Adder/Subtractor Project:**

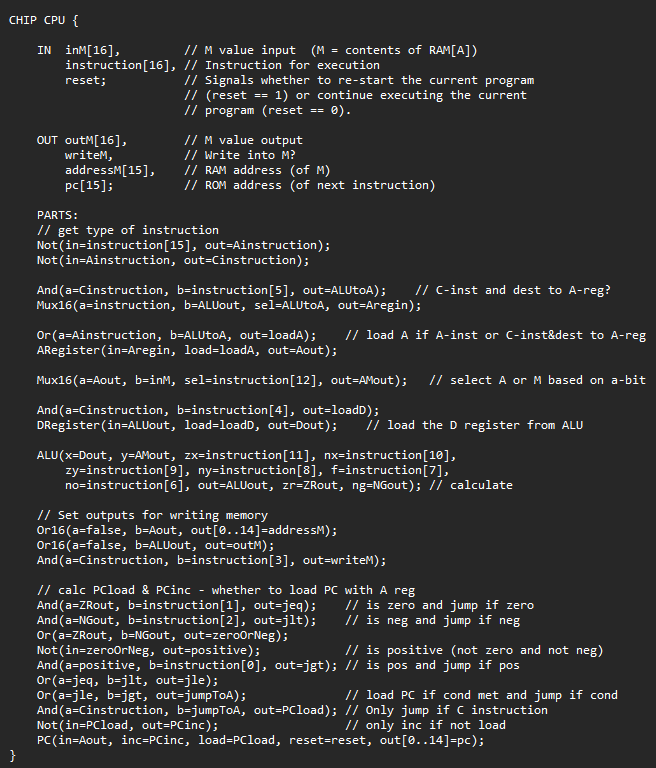
The 16-Bit Parallel Adder/Subtractor project focused on the design and implementation of a versatile arithmetic circuit capable of performing addition and subtraction operations on 16-bit binary numbers. The parallel nature of the circuit allows for efficient processing, essential in digital systems where arithmetic operations are fundamental.

In conclusion, the Hack CPU and 16-Bit Parallel Adder/Subtractor projects have provided a hands-on experience in digital circuit design and computer architecture. The knowledge gained from these endeavors lays the groundwork for future innovations in the field, pushing the boundaries of computational efficiency and advancing the understanding of digital systems.

**APPENDIX**

**Project A:**

Hack CPU



**Project B:**

16-Bit Parallel Adder and Subtractor

