

Design of a CPU FPGA-Based in Verilog

By

Marya Al-Dweik, Aseel Jaber & Abdulaziz AbuSaada

Supervised By

Prof. Fadi Shahroury

Submitted in Partial Fulfillment of the Requirements for the Degree of

BACHELOR OF SCIENCE

in

ELECTRONICS ENGINEERING

at

PRINCESS SUMAYA UNIVERSITY FOR TECHNOLOGY

Amman, Jordan

2023/2024

This is to certify that I have examined this copy of an engineering documentation by

Marya Al-Dweik, Aseel Jaber & Abdulaziz AbuSaada

And have found that it is complete and satisfactory in all respect,

And that any and all revisions required by the final Examining Committee have been made

Supervisor Name

## Acknowledgments

Type your acknowledgments here .

Your acknowledgments section must contain multiple short paragraphs (5-6 para- graphs), start with praises and thanks to Allah almighty, second paragraph for your supervisor , a paragraph for your parents and family, a paragraph for a person that you want to mention in name , a short paragraph for your friends a colleagues and finally a short paragraph for the examiners and committee members...

Enter Your Name

## Abstract

Type your abstract here, you may need to write multiple paragraphs, make sure to follow the instructions of writing an abstract. Describe your project and the results achieved briefly in few paragraphs. The abstract should not exceed one page.

Always make sure to leave a blank space at the start of the paragraph, if you used an abbreviation you need to state the meaning first capitalizing each word then write it between parentheses, for example, I study at Princess Sumaya University for Technology (PSUT).

Abstract lines are single-spaced, only the default 1pt, but paragraphs are, make sure the lines are justified. All of the above instructions are saved in a defined style “Abstract” once you choose it from the Style section it will apply automatically.

Your abstract must provide the reader with a clear and inclusive idea about your study, your ap- proaches, the problem, your methodology, your tools, and a summary of your results and conclusions, all of that must be addressed in brief.

**Keywords:** Keyword1, Keyword2, . . . , Keyword n .

Contents

[Acknowledgments 3](#_Toc166780160)

[Abstract 4](#_Toc166780161)

[Chapter 1 7](#_Toc166780162)

[Introduction 7](#_Toc166780163)

[1.1 Background 7](#_Toc166780164)

[1.2 Objectives 7](#_Toc166780165)

[1.3 Design Requirements 8](#_Toc166780166)

[1.4 Design Constrains 8](#_Toc166780167)

[1.5 Engineering Standards 9](#_Toc166780168)

[1.6 Organization of the Documentation 9](#_Toc166780169)

[1.7 Work Distribution 9](#_Toc166780170)

[Chapter 2 10](#_Toc166780171)

[Literature Review 10](#_Toc166780172)

[2.1 Introduction 10](#_Toc166780173)

[2.1 Literature review 10](#_Toc166780174)

[Chapter 3 13](#_Toc166780175)

[Design 13](#_Toc166780176)

[3.1 Developed Design 13](#_Toc166780177)

[3.1.1 From NAND to Computer 14](#_Toc166780178)

[3.2 Elementary gates to Multi-way Variants 14](#_Toc166780179)

[3.2.1 Elementary Logic Gates 15](#_Toc166780180)

[3.2.2 Multi-Way Variants 17](#_Toc166780181)

[3.2.3 Sequential Logic 20](#_Toc166780182)

[3.3 Arithmetic Logical Unit (ALU) 21](#_Toc166780183)

[3.3.1 The Hack ALU Operations 22](#_Toc166780184)

[3.3.2 Hack ALU Design 24](#_Toc166780185)

[3.4 Memory 24](#_Toc166780186)

[3.4.1 Random Access Memory RAM 26](#_Toc166780187)

[3.4.2 Serial Peripheral Interface SPI 27](#_Toc166780188)

[3.4.3 Universal Asynchronous Receiver / Transmitter UART 28](#_Toc166780189)

[3.5 Control Processing Unit CPU 30](#_Toc166780190)

[3.5.1 Hack CPU Instruction Set 30](#_Toc166780191)

[3.5.2 CPU Design 31](#_Toc166780192)

[3.6 Assembler 32](#_Toc166780193)

[3.7 Hardware Simulation 33](#_Toc166780194)

[Chapter 4 37](#_Toc166780195)

[Results 37](#_Toc166780196)

## List of Tables

[Table 1: Design Constrains 9](#_Toc166780197)

[Table 2: Work Distribution 9](#_Toc166780198)

[Table 3: Comparison table 12](#_Toc166780199)

[Table 4: ALU Operations 23](#_Toc166780200)

[Table 5:ALU Control bits 24](#_Toc166780201)

[Table 6: Memory Addresses 25](#_Toc166780202)

## List of Figures

[Figure 1:The Block Diagram of the system 14](#_Toc166780203)

[Figure 2:Block Diagram of computer system 15](#_Toc166780204)

[Figure 3:NOT Gate 16](#_Toc166780205)

[Figure 4: AND Gate 16](#_Toc166780206)

[Figure 5: OR Gate 16](#_Toc166780207)

[Figure 6: XOR Gate 17](#_Toc166780208)

[Figure 7: MUX Gate 17](#_Toc166780209)

[Figure 8: DMUX Gate 18](#_Toc166780210)

[Figure 9: OR8Way Gate 18](#_Toc166780211)

[Figure 10: MUX 4-Way Gate 19](#_Toc166780212)

[Figure 11: MUX 8-Way Gate 19](#_Toc166780213)

[Figure 12: DMUX 4-Way Gate 20](#_Toc166780214)

[Figure 13: DMUX 8-Way Gate 20](#_Toc166780215)

[Figure 14:Register Block Diagram 21](#_Toc166780216)

[Figure 15:PC Counter Block Diagram 21](#_Toc166780217)

[Figure 16: Hack ALU 22](#_Toc166780218)

[Figure 17: ALU Design 24](#_Toc166780219)

[Figure 18: Memory Block Diagram 26](#_Toc166780220)

[Figure 19: RAM Block Diagram 27](#_Toc166780221)

[Figure 20: SPI Block Diagram 28](#_Toc166780222)

[Figure 21: UART\_TX Block Diagram 29](#_Toc166780223)

[Figure 22: A-instruction examples 30](#_Toc166780224)

[Figure 23: Hack commands reference Card 31](#_Toc166780225)

[Figure 24: CPU Design 32](#_Toc166780226)

[Figure 25: Assembler flow chart 33](#_Toc166780227)

[Figure 26: NOT Gate simulation 34](#_Toc166780228)

[Figure 27: NOT Gate Simulation 34](#_Toc166780229)

[Figure 28: ALU simulation 35](#_Toc166780230)

[Figure 29: Full Adder simulation 35](#_Toc166780231)

[Figure 30: CPU simulation 36](#_Toc166780232)

[Figure 31: Clock Simulation 36](#_Toc166780233)

## Chapter 1

## Introduction

### 1.1 Background

This project aims to design a Central Processing Unit (CPU) from the universal NAND gate and build a computer that runs a game. Building a computer from the NAND gate makes the computer run faster and consume less power due to the NAND gate specifications in terms of power consumption and transistor size. Also, building a computer system from a simple gate enhances the knowledge of both hardware and software parts in any computer since it integrates each aspect of everyday use.

### 1.2 Objectives

The project's goals are outlined as follows:

* Design a computer architecture: The computer architecture needs to meet the main elements of a computer, which are Read Only Memory (ROM), Central Processing Unit (CPU), and Random Access Memory (RAM). This project uses a Reduced Instruction Set Computer (RISC) that is simple and open-source.
* Hardware implementation: The hardware of this computer is implemented using both Verilog and Hardware Description Language (HDL), which have multiple platforms to verify and simulate the design, making it a suitable choice to build a CPU and memory registers from the universal NAND gate.
* Software implementation: The software of this computer is implemented using MATLAB which is responsible for converting the assembly language to machine language.
* Application: The application done to verify the operations of the CPU are working properly is a calculator that computes four operations.

### 1.3 Design Requirements

1.     CPU Core Design:

* The CPU should run at 1MHz speed.
* The CPU should be 16-bit, which supports a subset of the RISC instruction set.

2.     Instruction Set:

* The system should implement a subset of RISC instructions, including 16-bit arithmetic and logic operations.
* The instruction memory should be a 16-bit x 32K size.

3.     Testing and Verification on FPGA:

* The system should create test benches and conduct simulations to test the CPU design, ensuring correctness and functionality thoroughly.

### 1.4 Design Constrains

Table 1: Design Constrains

|  |  |
| --- | --- |
| Economic | The project shall be confined within the budget limit (200 JD). |
| Manufacturability and Sustainability | FPGA Implementation: Synthesize the Verilog design onto an FPGA board. Popular FPGA platforms like Xilinx or Intel FPGA can be used. |

### 1.5 Engineering Standards

The project design should be implemented on the FPGA board to meet the engineering standard of using FPGA technology.

### 1.6 Organization of the Documentation

The rest of the documentation consists of Chapter 2 which includes a background on the project design and a literature review of similar projects, describing the differences between the projects and the aim of this project over the other projects. In Chapter 3, the design requirements are listed, an analysis of its requirements and constraints is conducted, different design approaches are mentioned, and finally, the developed design is presented. At the end of this documentation, the results and discussions are provided in Chapter 4.

### 1.7 Work Distribution

Table 2: Work Distribution

|  |  |  |  |
| --- | --- | --- | --- |
| Tasks | Team Members | | |
|  | Marya Al-Dweik | Aseel Jaber | Abdulaziz AbuSaada |
| Project Research | ✓ | ✓ | ✓ |
| Choosing equipment | ✓ | ✓ | ✓ |
| Documentation | ✓ | ✓ | ✓ |
| Body | ✓ | ✓ | ✓ |
| Hardware | ✓ | ✓ | ✓ |
| Software | ✓ | ✓ | ✓ |
| Presentation | ✓ | ✓ | ✓ |
| Poster | ✓ | ✓ | ✓ |

## Chapter 2

## Literature Review

### 2.1 Introduction

The invention of Field-Programmable Gate Arrays (FPGAs) has brought in a new era of customization and flexibility in hardware development within the fields of engineering and digital design. The design and implementation of Central Processing Units (CPUs) utilizing Hardware Description Languages (HDLs) such as Verilog have attracted a lot of attention among the many applications of FPGAs. Since FPGAs are programmable and flexible, a lot of studies have been conducted on FPGA-based CPU design in Verilog. Verilog is a hardware description language that is useful for building CPUs on FPGAs because it is frequently used to describe digital circuits. The purpose of this literature review is to examine the depths of research and development surrounding the design of FPGA CPU-based systems, focusing on the methods, improvements, and difficulties that come with using Verilog in applications.

### 2.1 Literature review

The first paper covers a straightforward and adaptable FPGA CPU architecture for instructional purposes [1]The authors provide design views and a schematic layout for a graphics framework. This framework is used to create graphics capabilities on 8-bit FPGA processors. Hardware Description Language was used to construct the processor framework (Verilog). This work began with the goal of producing crude real-time projections of simple wireframe models on a single chip [1]

In the second paper, Designing and implementing a dynamically reconfigurable processor for FPGA-based machine learning systems is the main topic of the second paper. The authors propose a new processor architecture for field-programmable gate arrays (FPGAs) that is specifically designed for decision trees (DTs), artificial neural networks (ANNs), and support vector machines (SVMs). The proposed goal is achieved through the use of reconfigurable devices such as field programmable gate arrays (FPGA) [2].

The third paper covers the design and implementation of a 16-bit Reduced Instruction Set Computing (RISC) processor on a Field Programmable Gate Array (FPGA). The study discusses how a 25 MHz Verilog-based microcontroller can benefit from the authors' comprehensive design process methodology. This study is valuable as it addresses the challenges and solutions associated with placing an RISC CPU on an FPGA [3].

The 4’th study describes a new low-power, universal three-input 4T-based gate intended for high-speed applications. The design and modeling of the gate are covered in detail by the authors, who also offered a wealth of information on how well it operates in terms of speed and power usage. The goal of the project is to create a microcontroller based on a universal NAND gate; therefore, the results of this study may provide useful information and suggestions for improving speed and power efficiency in the microcontroller design [4].

In the 5’th paper, Modern technology has been shaped by the discipline of microcontroller design, which is a cornerstone in the fields of electronics and computer engineering. The design and implementation of a 25MHz Verilog-based microcontroller running a 16-bit operating system is the main focus of this literature research. The creation of a universal NAND gate microcontroller, a notion with great promise for the future of microcontroller design, is what sets this project apart [5].

 In the 6’th paper, the comprehensive survey offers a panoramic view of the diverse approaches, architectures, and applications of FPGA-based processors, providing valuable insights into the design space, performance metrics, and emerging trends. By synthesizing insights from a myriad of research works, the authors highlight the versatility and potential of FPGA-based CPUs across various domains, including embedded systems, signal processing, and high-performance computing [6].

In the last paper, microcontrollers are essential components of many electronic gadgets, ranging from simple home appliances to sophisticated industrial gear. The efficiency and usefulness of these devices greatly depend on the design and implementation of these microcontrollers. A hardware description language called Verilog has completely changed how digital circuits, including microcontrollers, are designed and simulated. Verilog offers flexibility and accuracy in the design process by enabling the description of digital systems at multiple levels of abstraction, from the gate level to the architectural level [7].

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Reference | FPGA | cost | speed | open source |
| [1] | Xilinx Spartan-6 LX16 FPGA | 37.86 JD | 25MHz | NO |
| [2] | Xilinx Spartan-6 FPGA | 37.86 JD | 25MHz | NO |
| [3] | Xilinx Spartan-6 SP605 | 86.49 JD | 60MHz | NO |
|  | iCE40HX8K-EVB | 22.98 JD | 25 MHz | YES |

Table 3: Comparison table

## Chapter 3

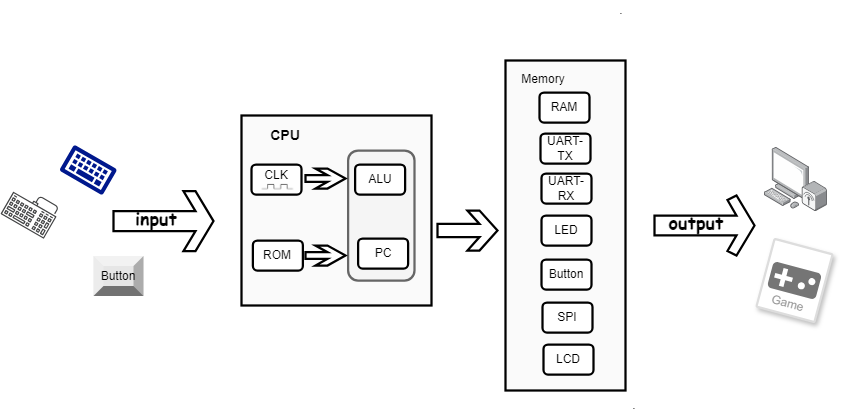
## Design

In this chapter, a complete inspection of the design and its specifications will be provided. Starting from analyzing the proposed design of the project, then discussing the main gate of the system, and finally, explaining the developed design in detail.

### 3.1 Developed Design

The figure below shows the design of the system that could consist of multiple inputs such as a keyboard, push buttons, and touch screen process done inside the computer system first by reading the program stored in the ROM then bypassing the instruction to the CPU and implement it using the ALU and other elements and finally bypassing the information to the memory to display it on the output that could be a computer screen, LCD screen and LEDs.

Figure 1:The Block Diagram of the system



### 3.1.1 From NAND to Computer

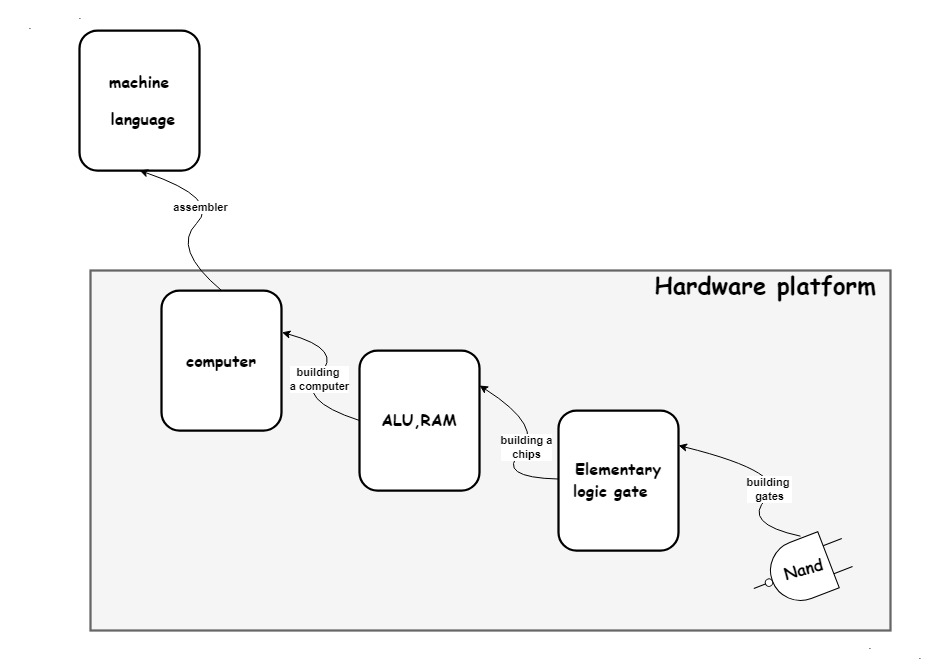
Digital devices operate using a series of specialized chips crafted to both store and process data. Even though these chips are different shapes and forms they are all constructed from the same building blocks ” the elementary logic gates” This project is starting with one primitive logic gate. The NAND gate serves as the foundation for the system computer architecture, upon which all other gates and chips are constructed. NAND gate was chosen because of its low power consumption, transistor size, parallel PMOS, and series NMOS.

Figure 2:Block Diagram of computer system

### 3.2 Elementary gates to Multi-way Variants

The Nand gate serves as the foundation for the computer architecture, upon which all other gates and chips are constructed.

### 3.2.1 Elementary Logic Gates

The simple logic gates were built from the universal NAND gate to build more complicated logic blocks such as ALU, Adder & PC counter.

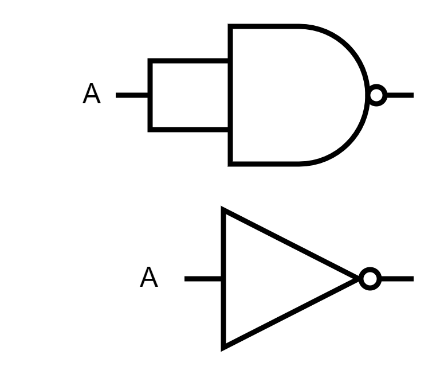
* NOT Gate: Was built by connecting both inputs of the NAND gate to the input signal.

Figure 3:NOT Gate

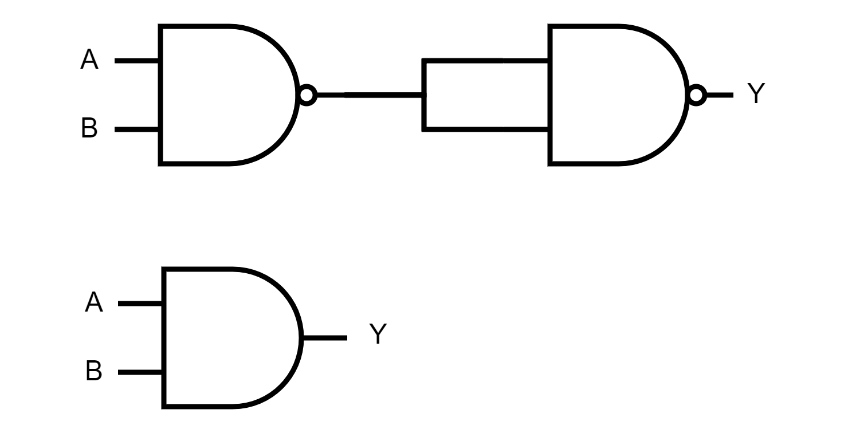
* AND Gate:  Was built by connecting the two inputs A and B to a NAND gate and then connecting the result to both inputs of the NAND gate.

Figure 4: AND Gate

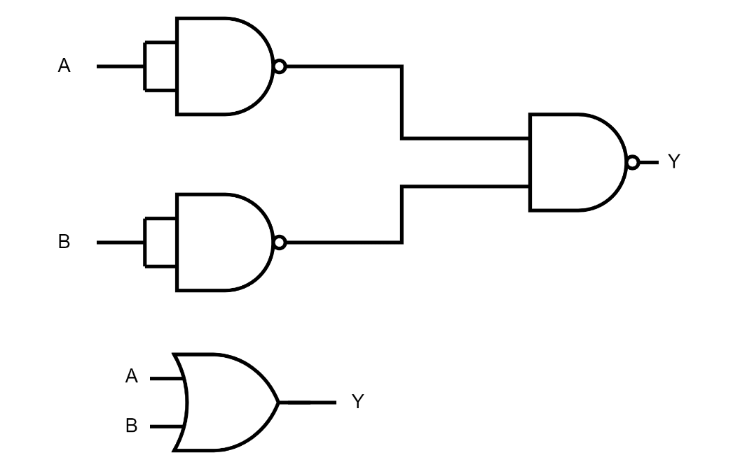
* OR Gate: Was built by taking the NAND of two NAND gates where each input is connected to the same input signal.

Figure 5: OR Gate

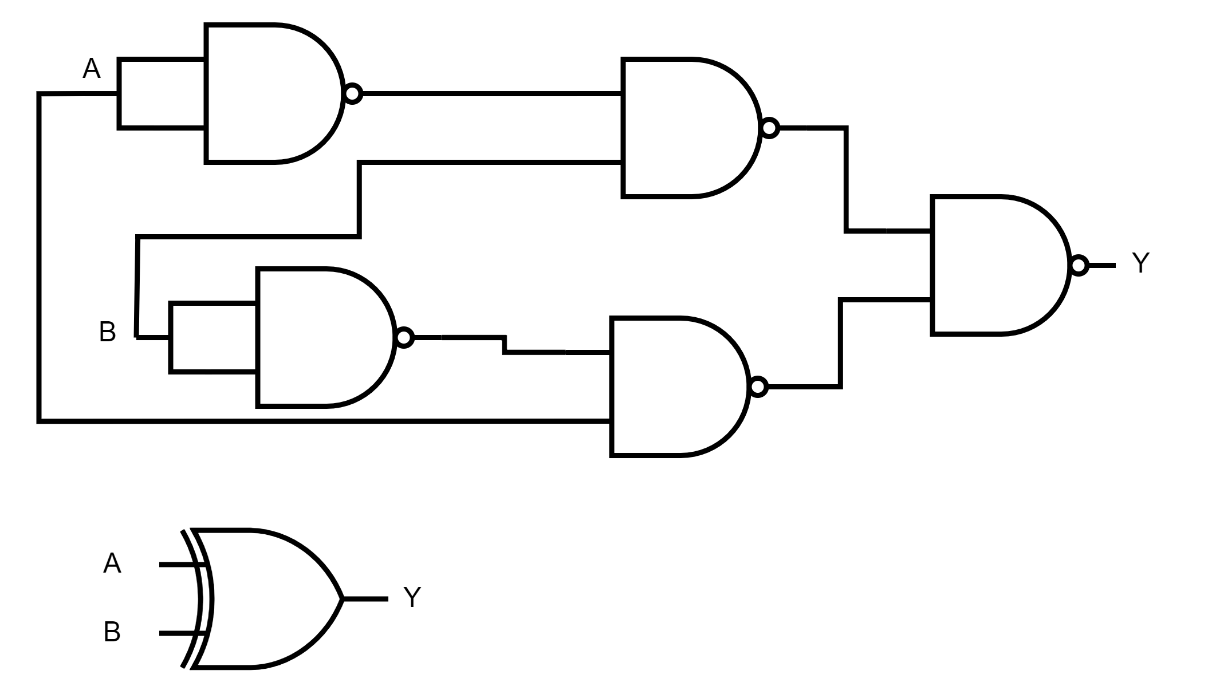
* XOR Gate: Was built by taking the invert of one of the inputs with the other input to a NAND gate and then taking the result of each NAND gate to another NAND gate.

Figure 6: XOR Gate

* MUX Gate: Was built using four NAND gates, input A was connected with an inverted

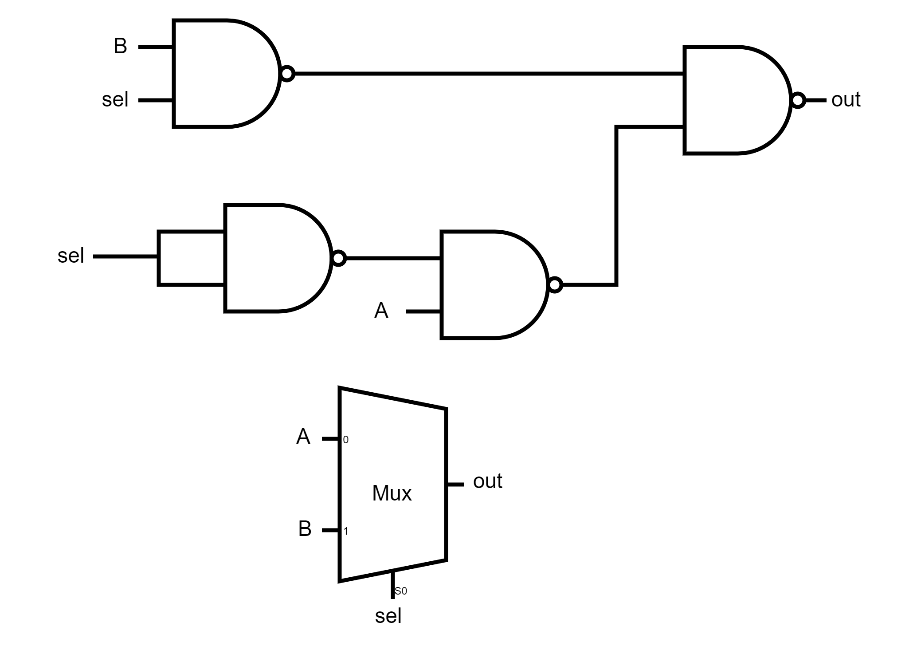
selection line to a NAND gate, B with a non-inverted selection line, and then the results were connected to a NAND gate.

Figure 7: MUX Gate

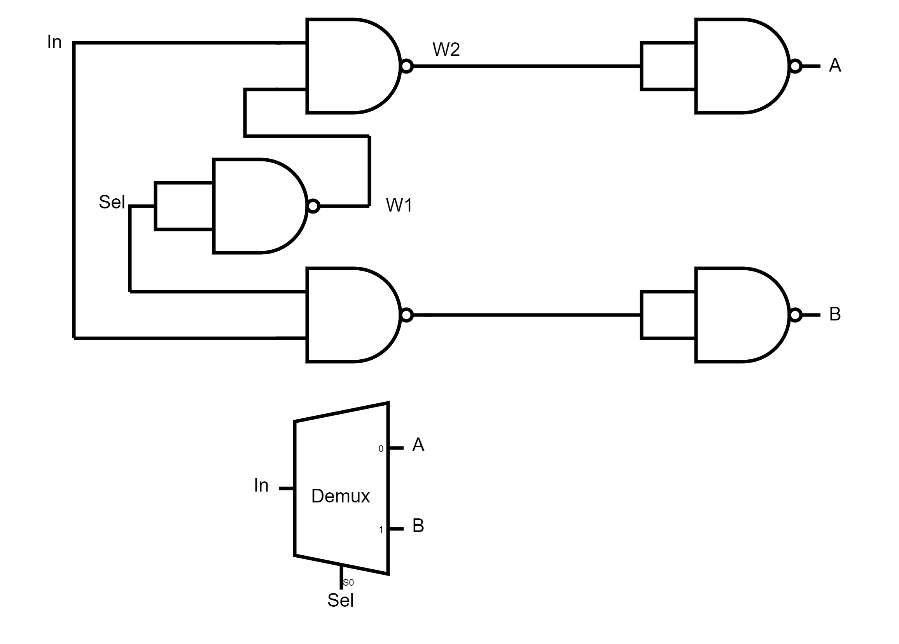
* DMUX Gate: Was built using five NAND gates, the input was connected to a NAND gate with an inverted selecting line, and another NAND gate with a non-inverted selecting line. At the final stage, each result was connected to an inverter NAND gate.

Figure 8: DMUX Gate

### 3.2.2 Multi-Way Variants

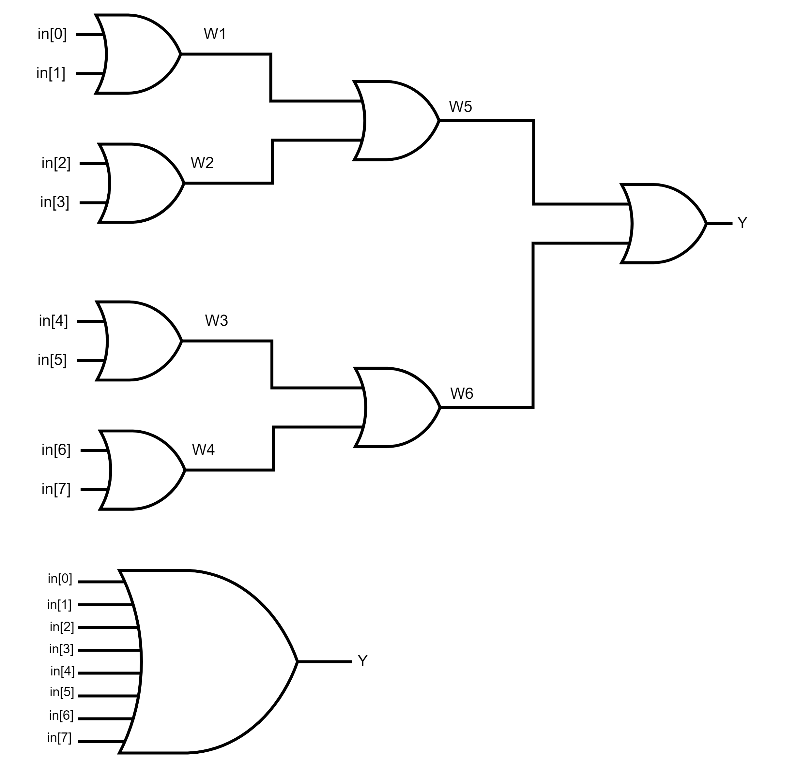
* OR8WAY Gate : Was built using seven elementary OR gates to build OR8WAY. Each OR gate has two inputs, with four of them comparing the inputs and the other three comparing the outputs.

Figure 9: OR8Way Gate

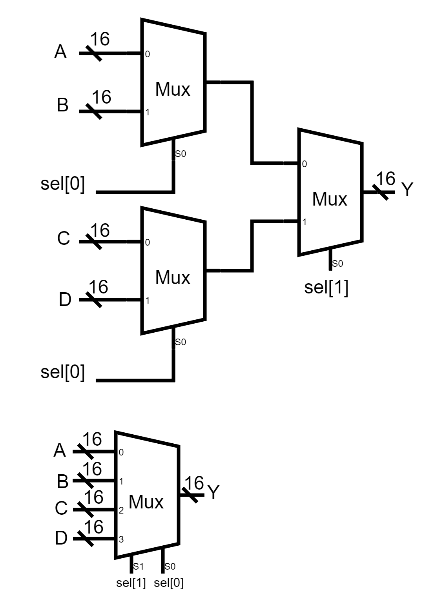
* MUX 4-Way Gate: Was built with three 2:1 MUXs each MUX selects one of two inputs based on the selection line bit 0 and then selects one of the two results based on the selection line bit 1.

Figure 10: MUX 4-Way Gate

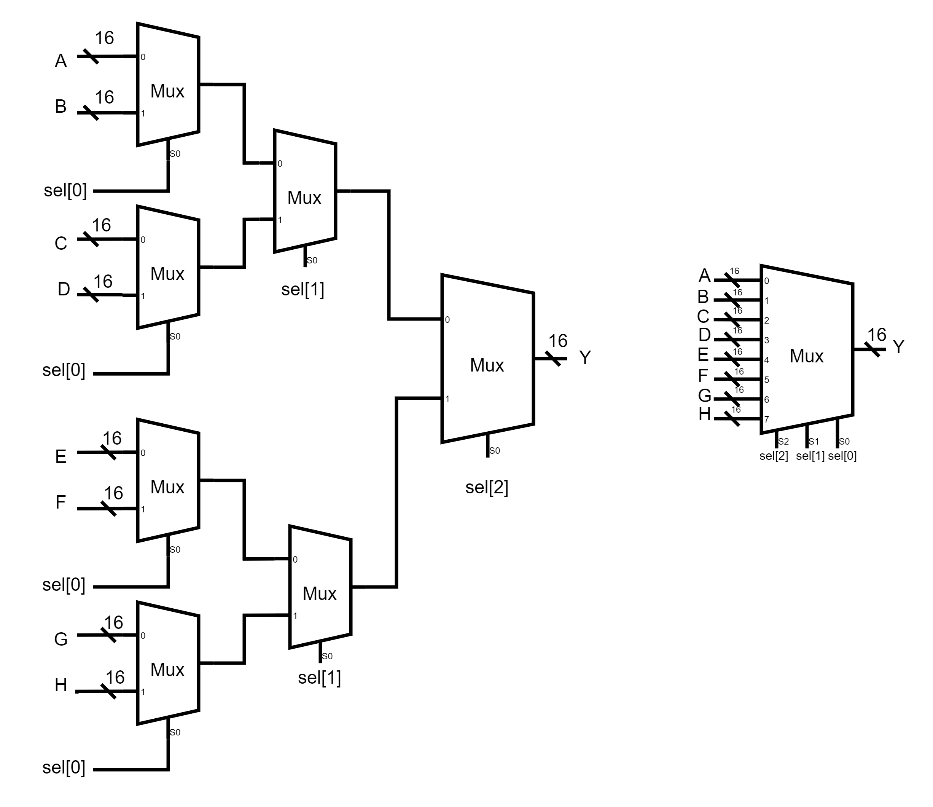
* MUX 8-Way Gate: Was built with seven 2:1 MUXs each MUX selects one of two inputs based on the selection line bit 0 and then selects one of the 2 results based on the selection line bit 1 at the last stage it selects one of the results based on the selection line bit 3.

Figure 11: MUX 8-Way Gate

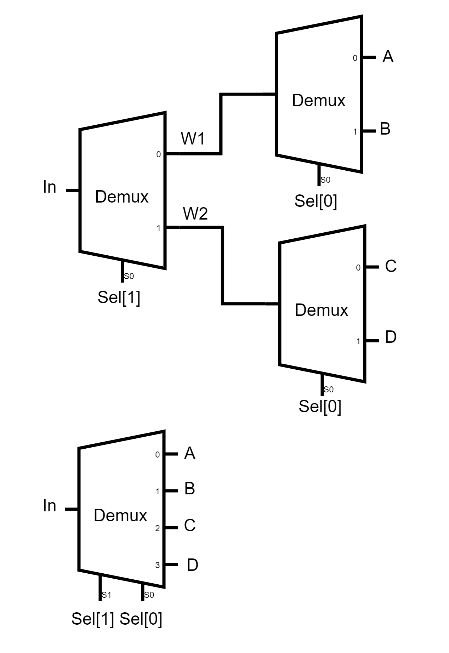
* DMUX 4-Way Gate: Was built with three 2:1 DMUXs, where the input is split into two paths based on the selection line to display the input on one of the outputs.

Figure 12: DMUX 4-Way Gate

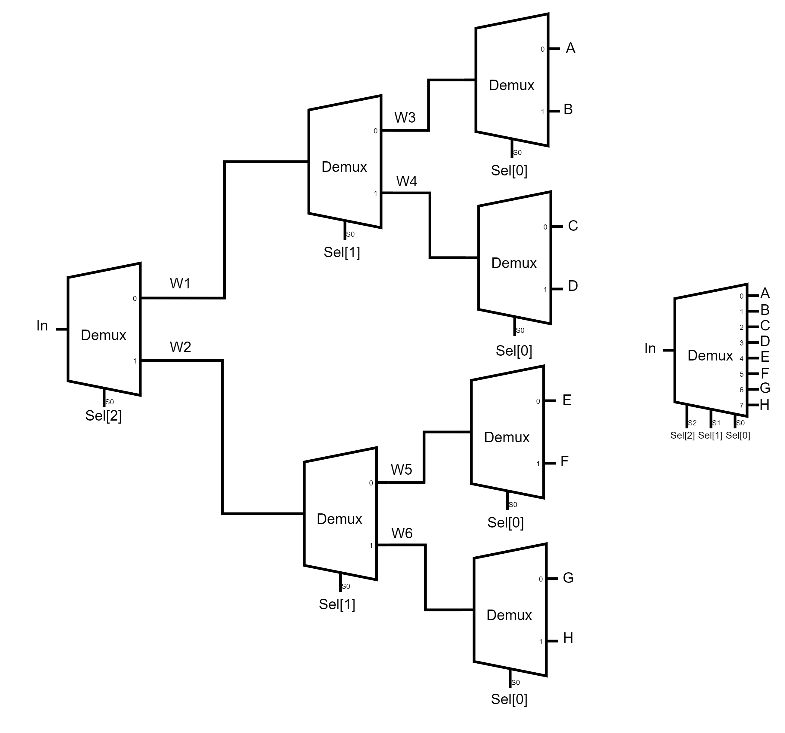
* DMUX 8-Way Gate: Was built with seven 2:1 DMUXs, where the input is split into two paths based on the selection line to pass the input on one of the outputs

Figure 13: DMUX 8-Way Gate

### 3.2.3 Sequential Logic

* Register: Was built using 16 bits each bit consists of a D-flip flop and a MUX the register

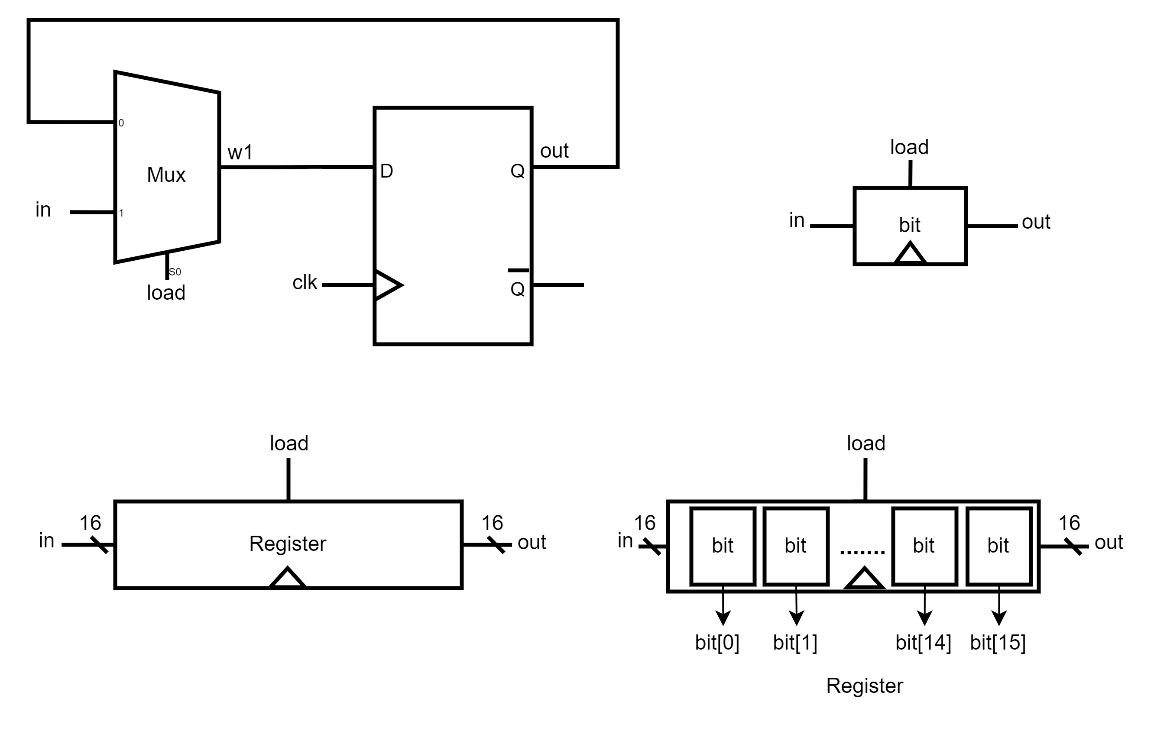
takes input and saves it in the bit and loads it to the output when the load is one.

Figure 14:Register Block Diagram

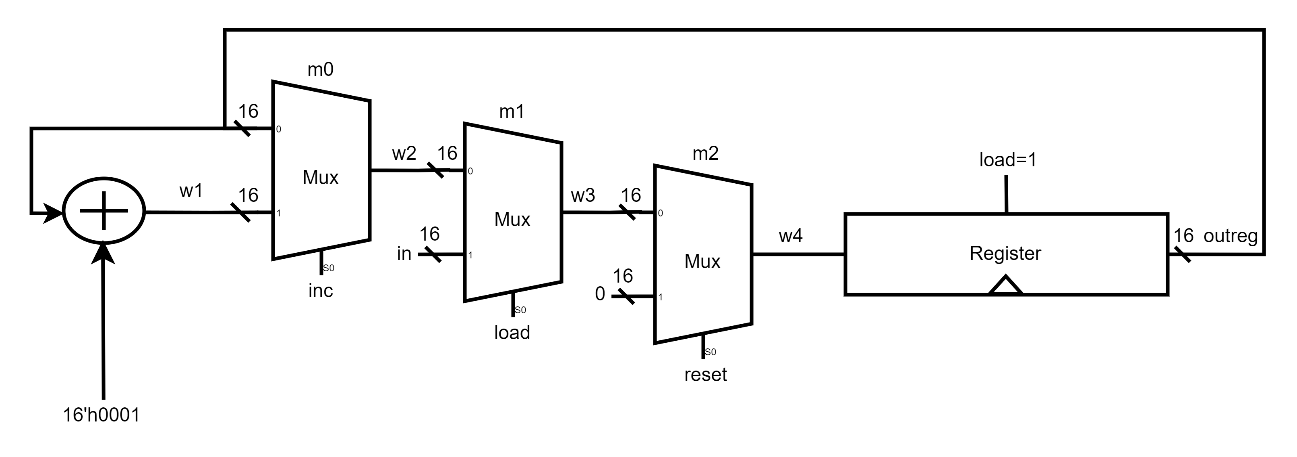
* PC Counter: Was built to increment the PC and point to the next instruction or to load the PC with the address of the next instruction jumping to or to reset the PC and point to the first instruction.

Figure 15:PC Counter Block Diagram

### 3.3 Arithmetic Logical Unit (ALU)

This section explores the construction of gate logic designs for arithmetic operations on numbers, starting with simple logic gates and ending with the creation of an Arithmetic Logical Unit (ALU) that is fully operational. The ALU is the main component of a computer that handles logical and arithmetic processes. The arithmetic logical unit (ALU), performs arithmetic and logical operations based on the control bits. Initially, the basic task the computer was capable of executing was enumerated, then by designing a logic circuit that used the control bits to do the right operation.

### 3.3.1 The Hack ALU Operations

The operations ALU can handle is listed in table 3 by setting the control bits to one of the binary combinations the ALU will compute the function.

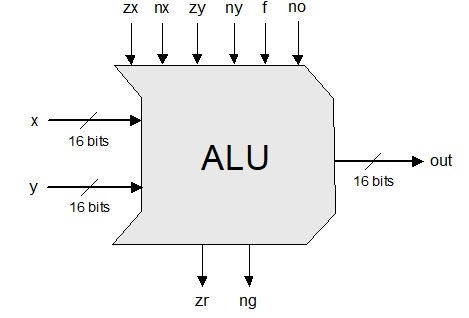


Figure 16: Hack ALU

Control bits bits

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **zx** | **nx** | **zy** | **ny** | **f** | **no** | **out** |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | -1 |
| 0 | 0 | 1 | 1 | 0 | 0 | X |
| 1 | 1 | 0 | 0 | 0 | 0 | Y |
| 0 | 0 | 1 | 1 | 0 | 1 | !x |
| 1 | 1 | 0 | 0 | 0 | 1 | !y |
| 0 | 0 | 1 | 1 | 1 | 1 | -x |
| 1 | 1 | 0 | 0 | 1 | 1 | -y |
| 0 | 1 | 1 | 1 | 1 | 1 | x+1 |
| 1 | 1 | 0 | 1 | 1 | 1 | y+1 |
| 0 | 0 | 1 | 1 | 1 | 0 | x-1 |
| 1 | 1 | 0 | 0 | 1 | 0 | y-1 |
| 0 | 0 | 0 | 0 | 1 | 0 | x+y |
| 0 | 1 | 0 | 0 | 1 | 1 | x-y |
| 0 | 0 | 0 | 1 | 1 | 1 | y-x |
| 0 | 0 | 0 | 0 | 0 | 0 | x&y |
| 0 | 1 | 0 | 1 | 0 | 1 | x|y |

Table 4: ALU Operations

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| pre-setting the x input | | pre-setting the y input | | selecting between computing + or & | post-setting the output | resulting ALU output |
| zx | nx | zy | ny | f | no | out |
| if zx then x=0 | if nx then x=!x | if yx then y=0 | if ny then y=!y | if f  then out=x+y  else out= x&y | if no then out=!out | out(x,y)= |

Table 5:ALU Control bits

### 3.3.2 Hack ALU Design

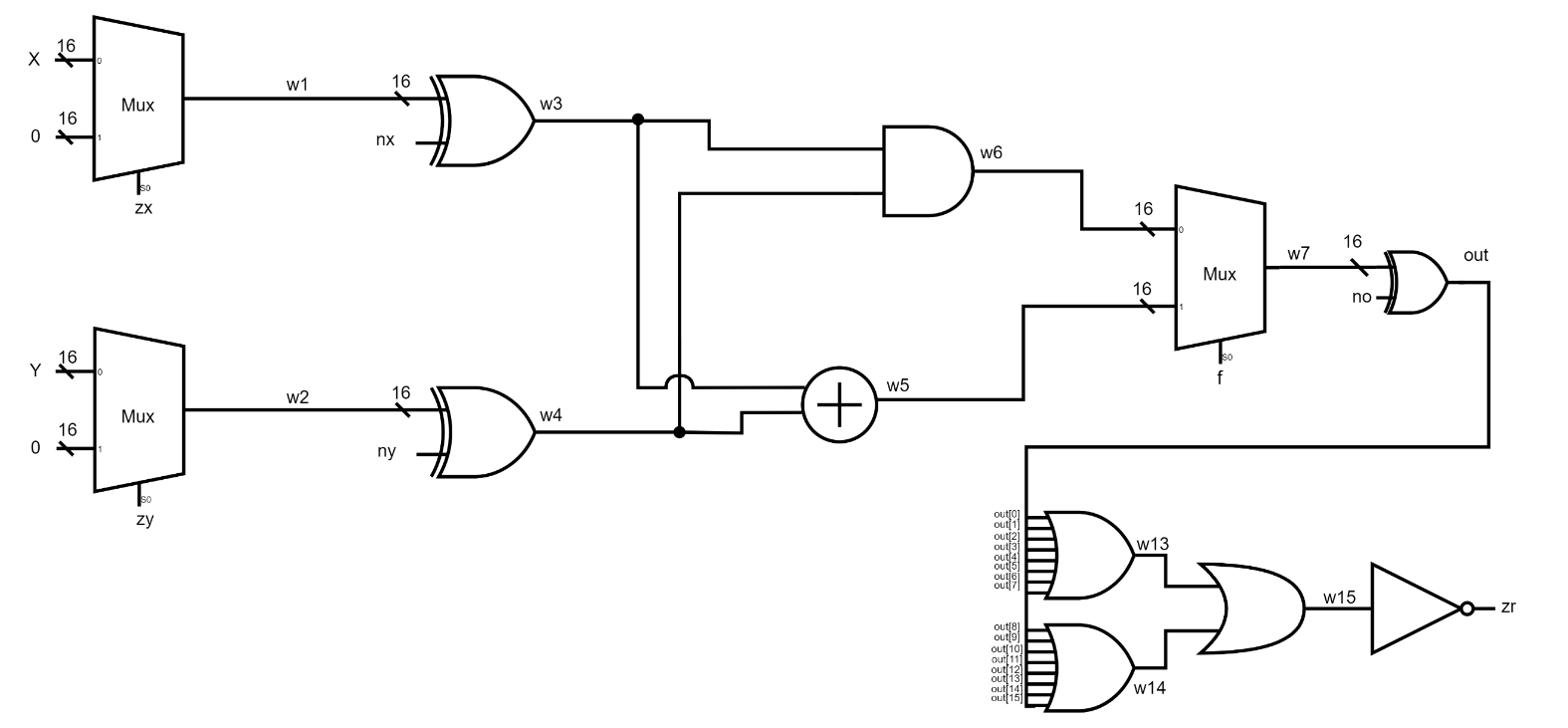
As shown in the block diagram of the ALU design first the two MUXs based on (zx) and (zy) controlling bits choose whether the input or the zero is passing to the next stage and then bypass the result to an XOR gate with the (ny) and (nx) controlling bits to decide if the operation is inverting the inputs then doing the AND and Adding operations with MUX that select which operation will pass based on the (f) control bit at the final stage the result is XORed with (no) control bit to decide if the output has to be inverted also the ALU output (zr) bit that flags if the output is zero, this circuit design perfectly matches operations the ALU expected to do.

Figure 17: ALU Design

### 3.4 Memory

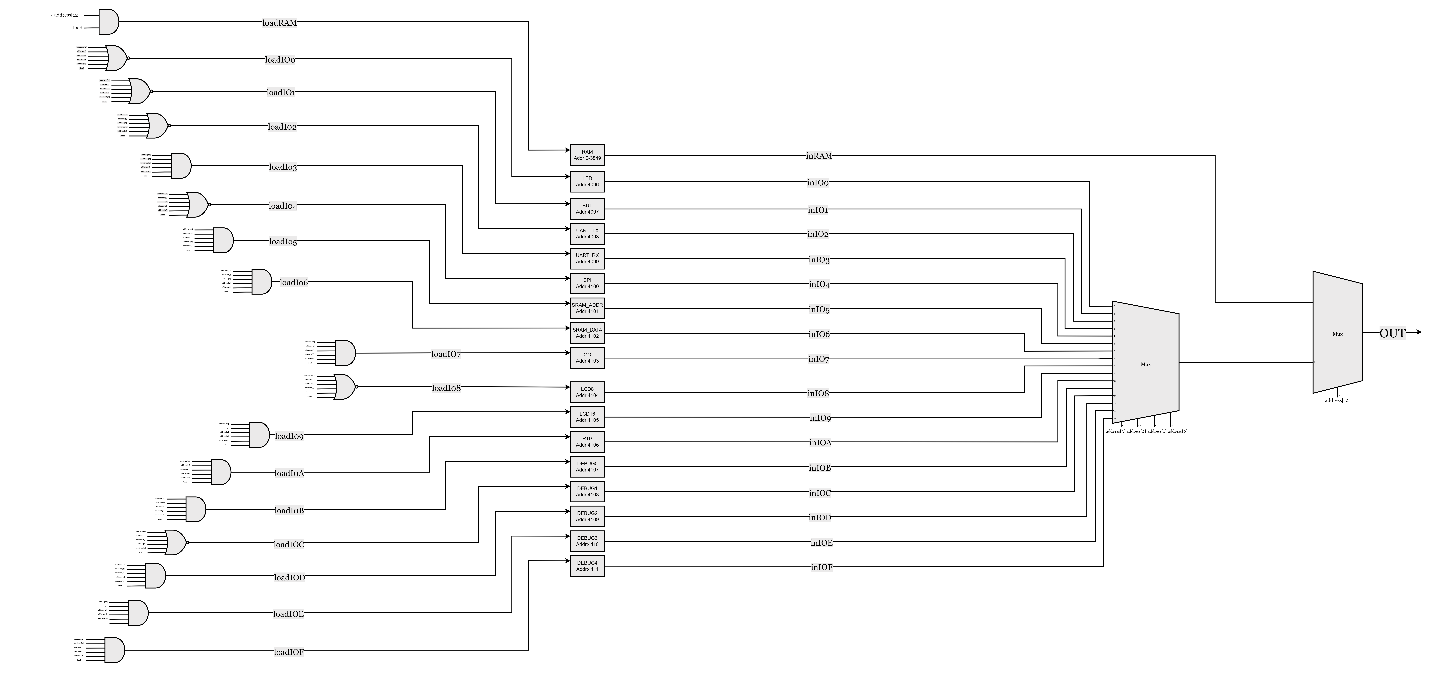
The set of physical components that a computer uses to store data and instructions is referred to as memory informally. From the perspective of the programmer, all memory has the same structure: a word or location is a continuous array of cells with a defined width, each with a distinct address. Therefore, providing the address of a single word (which could be an instruction or a data item) specifies it. For the sake of brevity, we will refer to these single words in the following as Memory[address], RAM[address], or M[address].

Table 6 shows the addresses of I/O devices in the computer system the starred ones are simple registers such as LED & Buttons registers and others are special function registers that are implemented to do special tasks.

|  |  |  |
| --- | --- | --- |
| Address | I/O Dev | Function |
| 0-3839 | RAM | Address=12 bit, X16 bit |
| 4096 | LED\* | 0 = led off, 1 = led on |
| 4097 | BUT\* | 0 = button pressed "down", 1 = button released |
| 4098 | UART\_TX | transmit byte to UART with 115200 baud 8N1 |
| 4099 | UART\_RX | receive a byte from UART with 115200 baud 8N1 |
| 4100 | SPI | read/write spi flash memory chip |
| 4101 | SRAM\_ADDR\* | address of external SRAM chip |
| 4102 | SRAM\_DATA\* | read/write data from/to external SRAM chip |
| 4103 | GO\* | Start execution of instructions from external SRAM |
| 4104 | LCD8 | write 8bit command/data to LCD screen |
| 4105 | LCD16 | write 16bit data to LCD screen |
| 4106 | RTP\* | read/write byte from/to resistive touch panel |
| 4107 | DEBUG0\* | used for debugging |
| 4108 | DEBUG1\* | used for debugging |
| 4109 | DEBUG2\* | used for debugging |
| 4110 | DEBUG3\* | used for debugging |
| 4111 | DEBUG4\* | used for debugging |

Table 6: Memory Addresses

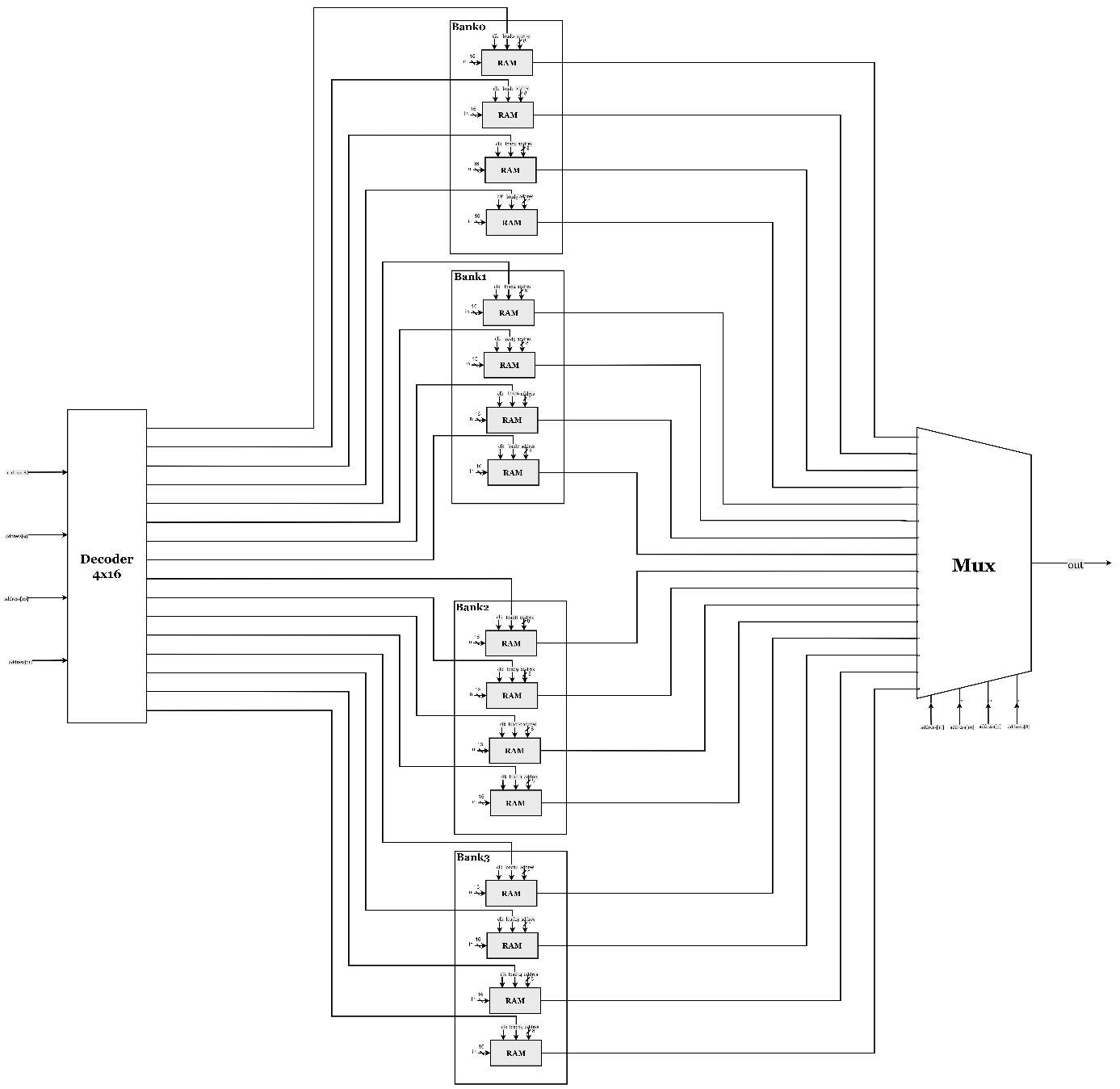
Figure 18: Memory Block Diagram



### 3.4.1 Random Access Memory RAM

This section explores the construction of gate logic designs for arithmetic operations on numbers, starting with simple logic gates and Three inputs accepted by a conventional RAM device: data, address, and load bit. The address specifies which RAM register should be accessed in the current time unit. The value of the specified register is instantly emitted via the RAM's output in the event of a read operation (load=0). When a write operation (load=1) occurs, the RAM's output begins to emit the input value after the chosen memory register commits to it in the subsequent time unit. A RAM device's size is 16-bit x 4K .

Figure 19: RAM Block Diagram



### 3.4.2 Serial Peripheral Interface SPI

HACK can read and write bytes from the SPI flash memory to the special function register SPI memory that is mapped to address 4100.

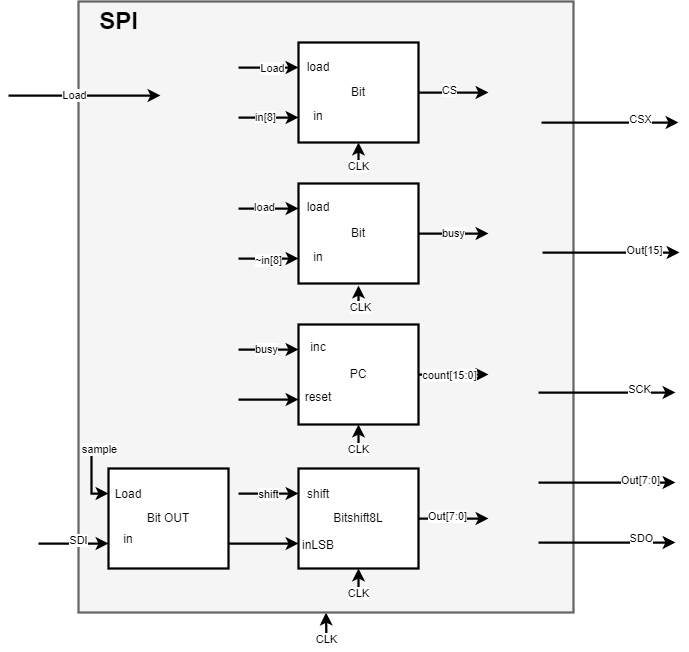
Transmission of bytes in[7:0] begins when load=1 and in[8]=0. CSX drops (and remains low even after the transmission is finished). Together with eight clock signals on SCK, the byte is sent bitwise to SDO. A byte is simultaneously received by the SPI at SDI. Out[15] during transmission is 1. One byte has been transmitted after 16 clock cycles. SPI outputs the received byte to out[7:0] when out[15] becomes low. In the event when load=1 and in[8]=1, CSX goes high and no bits are ****transmitted.

Figure 20: SPI Block Diagram

### 3.4.3 Universal Asynchronous Receiver / Transmitter UART

UART stands for Universal Asynchronous Receiver/Transmitter which is a serial communication protocol to send data between any two interfaces -regardless of their clock frequency- by setting it up by choosing one of the universal baud rates to synchronize between the two devices using only two pins the RX and TX.

For this design 115200 baud rate was chosen and designed according to it, the 115200 baud rate sends 10 bits, a start bit, and the data which consists of 8 bits and a stop bit.

The TX pin sends a bit every=8.68 microsecond, at first the RX is always HIGH when the transmission starts by sending the first bit which is the start bit it will be LOW to tell the other device to start the process to receive the data serially through the RX pin and it will enter a register and start to shift the bit to the right every time a data bit until it finishes, the last bit is the stop bit will it be HIGH to stop the process and save the data.

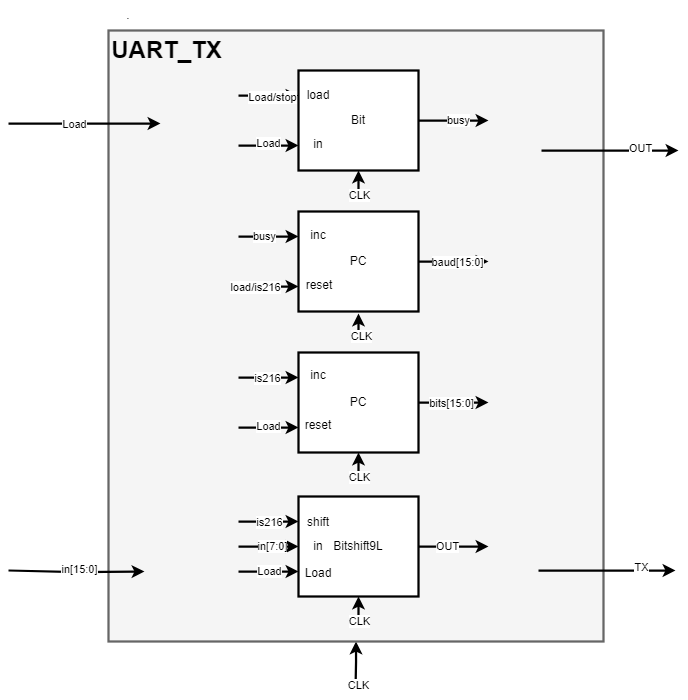
****To implement this in the computer system first how many clock cycles it should take to achieve this synchronization have been calculated, a clock with 25MHz, and instruction time with 40 nanoseconds and a delay have been done to make it close to 8.68 microseconds so by making a counter using the pc module to increment every clock cycle  to reach a specific number of increments to reach 8.68 microseconds calculated by =217 increment so every time the pc for baud rate reach 216 the shift bit module for the TX or RX will shift the bits  and a second counter is used to count how many bits it reached and it also increments every time the baud counter reach 216 to synchronize RX with the clock it will enter a D flip flop and a second D flip flop is used to save the start bit to start the protocol and once the bits counter reach 9 everything will stop and the data will be reflected on the output the default output for the TX is zero when bit 15 is HIGH it indicates that the module is busy and transmitting the data and for the RX bit 15 is HIGH that means the RX module is ready to receive the data and when the bit 15 is LOW then the data  is received.

Figure 21: UART\_TX Block Diagram

### 3.5 Control Processing Unit CPU

A control processing unit (CPU) is a hardware component that executes the instructions that make up a computer program. The CPU acts as the brain of any computing device, Data is transformed into digital signals by servers and other smart devices, which then use those signals to do mathematical operations.

The central component of the computer's architecture, the CPU, is in charge of carrying out the instructions of the software that is presently loaded. These instructions advise the CPU on the calculations it must do, the registers it must read from or write to, and the instructions it must retrieve and carry out next. Three primary pieces of hardware are used by the CPU to carry out these tasks: an ALU (Arithmetic-Logic Unit), a series of registers, and a control unit.

### 3.5.1 Hack CPU Instruction Set

The hack CPU has two types of instructions A-instruction and C-instruction each is used to do specific operations and can access special registers.

* A-instruction: This instruction used for entering a constant value (A = value), selecting a RAM location (register = RAM[A] ) or selecting a ROM location (PC = ROM[A] ).

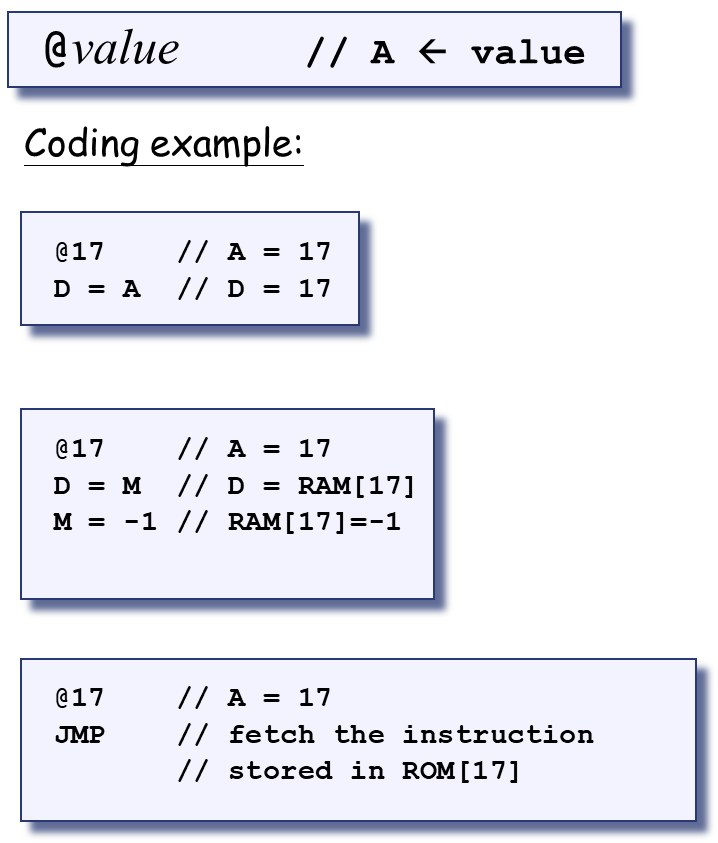
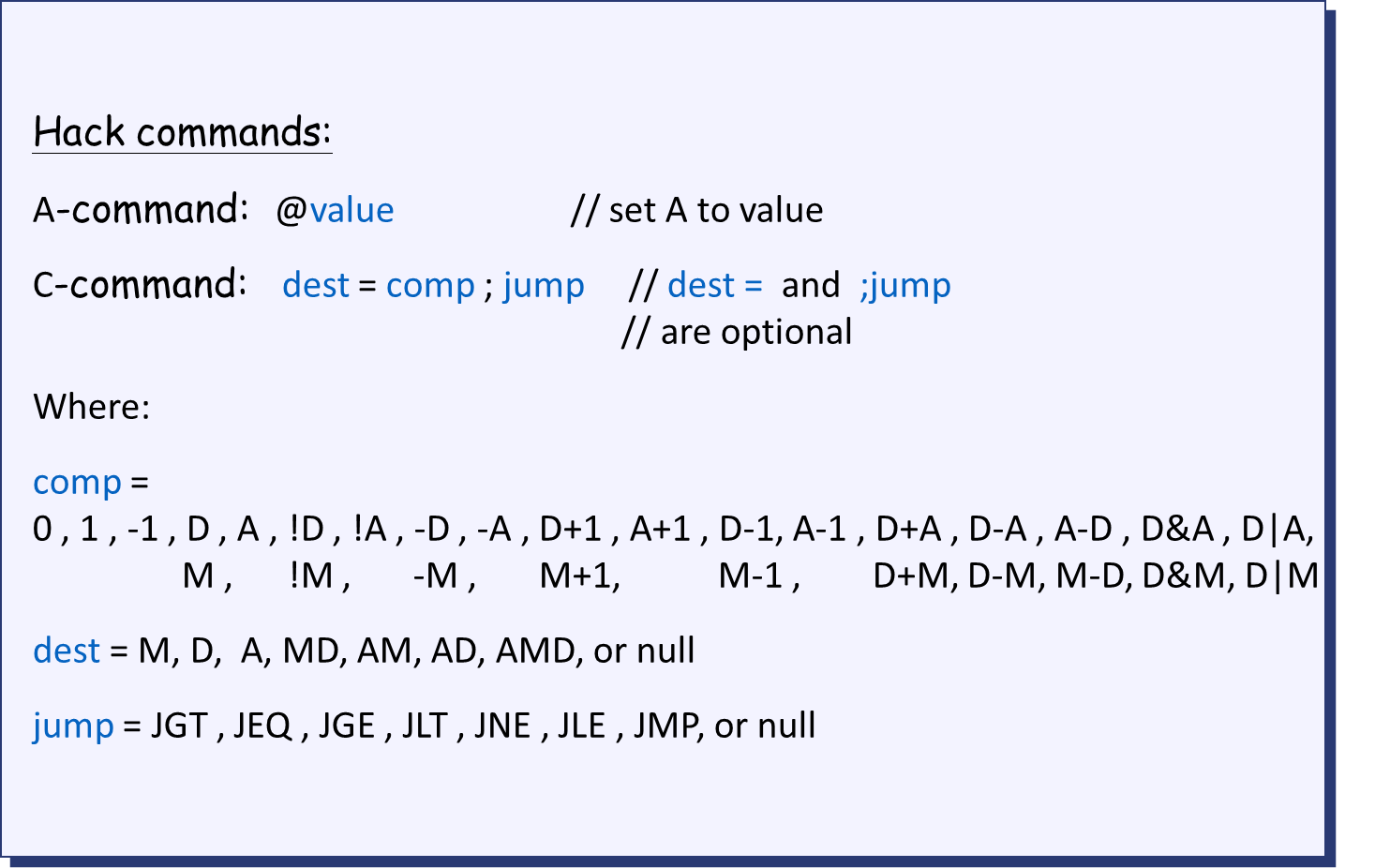


Figure 22: A-instruction examples

* C-instruction: This instruction is used to compute the value of an operation and store the result in the specified destination then go to the instruction stored in ROM[A] if the jump condition is true.

Figure 23: Hack commands reference Card



### 3.5.2 CPU Design

 The CPU loads the 16-bit constant that the instruction represents into the A register if the instruction is an A-instruction. If an instruction is a C-instruction, the CPU will cause the ALU to carry out the computation, and then cause this value to be stored in the subset of {A,D,M} registers that the instruction specifies. The CPU asserts the writeM control bit output if one of these registers is M (when writeM is 0, any value may appear in outM).

When the reset input is zero, the CPU computes the address of the subsequent instruction using the ALU output and the jump directive given by the instruction. It then outputs this address to the PC output. The CPU sets the PC to 0 if the reset input is 1.

 The execution of the instruction has an instantaneous effect on the combinational outputs, writeM and outM. Although they are impacted by the execution of the instruction, the addressM and PC outputs are timed; they commit to their changed values only in the subsequent time step.

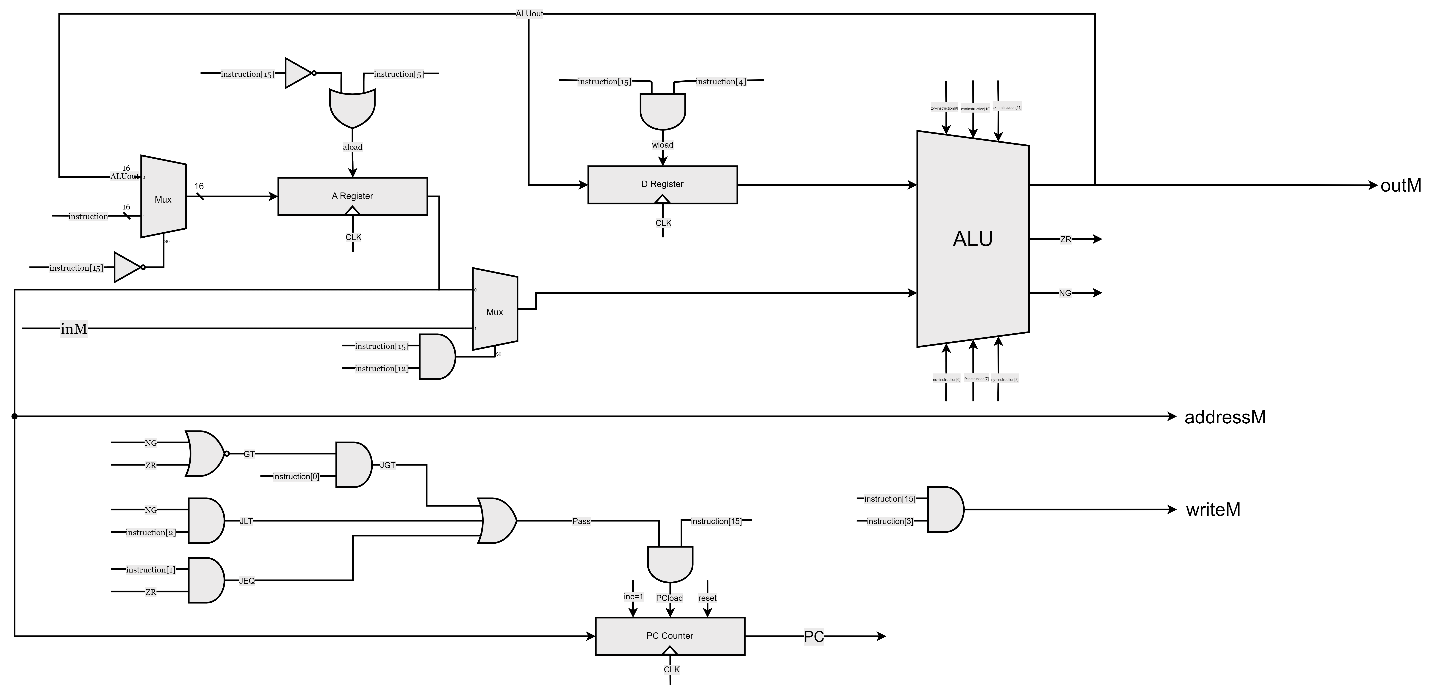


Figure 24: CPU Design

### 3.6 Assembler

The assembler is the more important part of the software interface because it converts the instruction written in assembly language to machine code that the computer can read and execute.

The flow chart in figure 25 shows how the assembler is implemented using MATLAB.

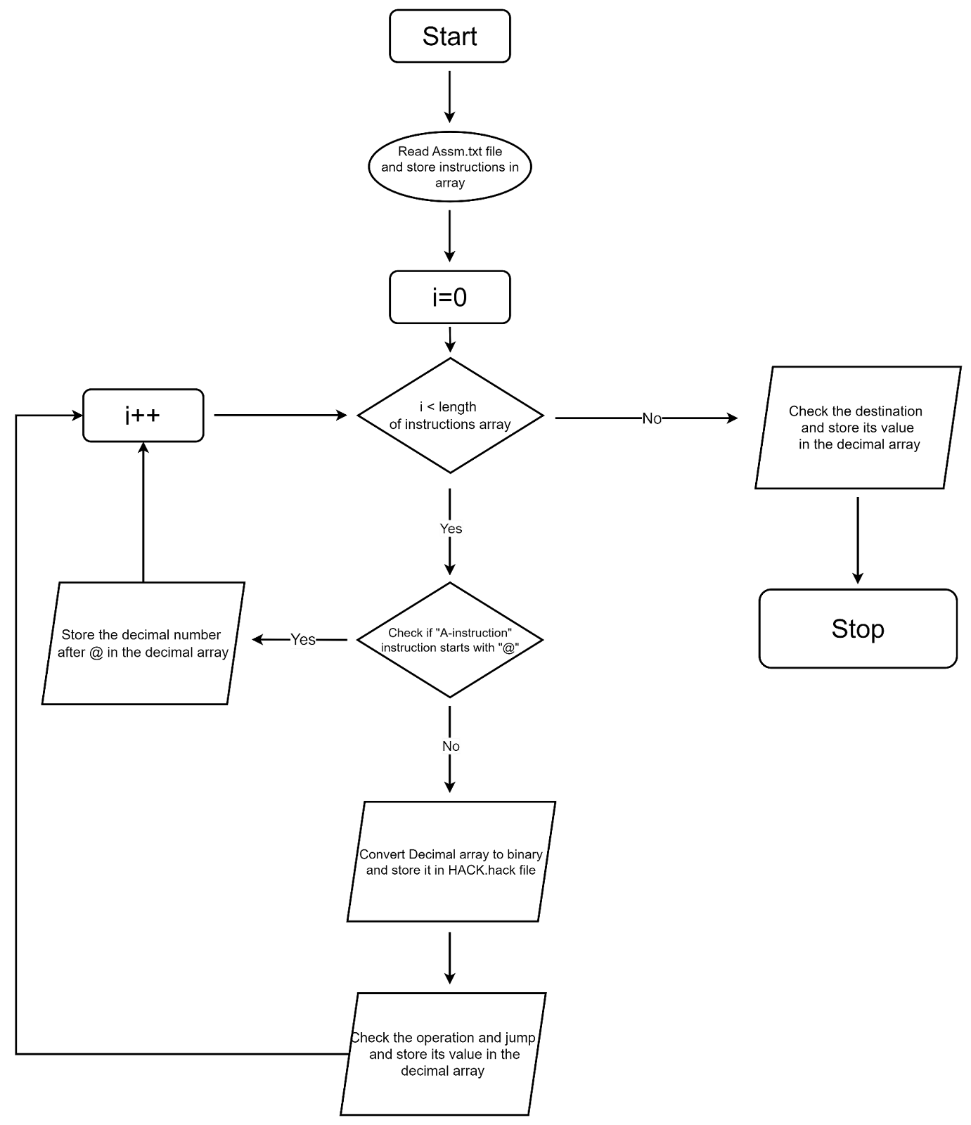


Figure 25: Assembler flow chart

### 3.7 Hardware Simulation

At first, gates and chip architectures were built using the Hardware Description Language( HDL). It specifies the chip structure by writing an HDL program and testing it using a software tool called a hardware simulator. It starts by using the NAND gate as the main gate all other gates and chips were built using it as shown in Figure 26.

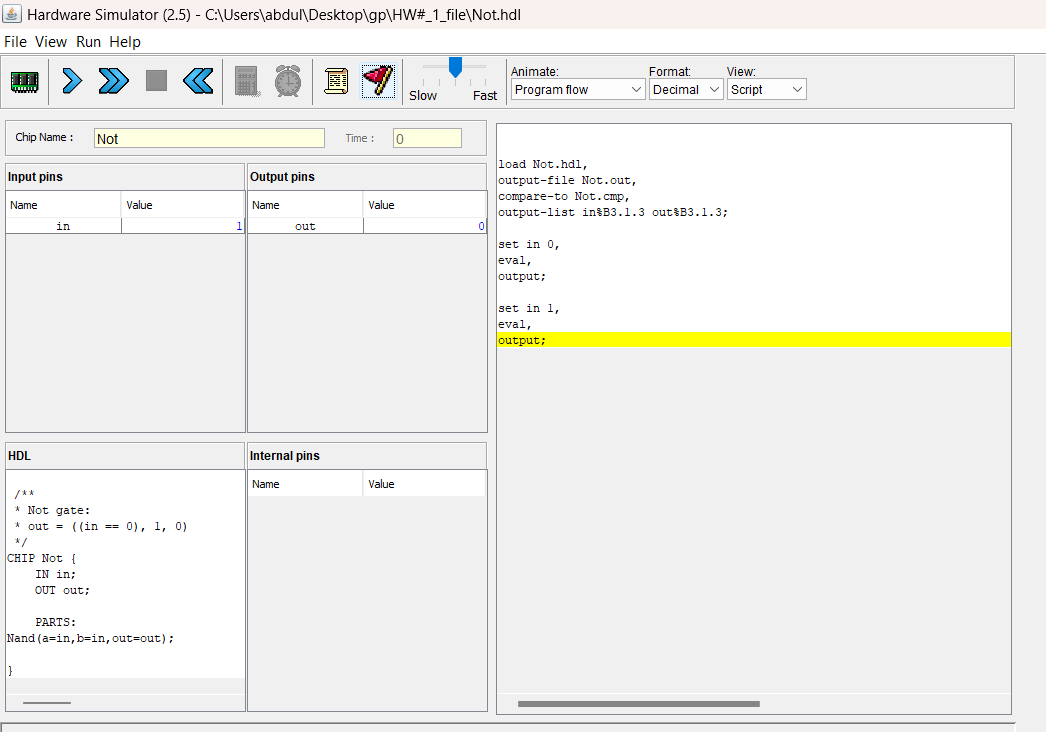


Figure 26: NOT Gate simulation

A screenshot of a computer

Description automatically generatedFirst, the hardware simulator is a program to simulate HDL codes by writing a test file.tst to help see if the code is right or wrong and it is easy to debug when there is something wrong by uploading a truth table file.cmp It will compare the output of the code and the truth table which is the result theoretically in the next figures it's the program window and simulation to a NOT gate that was written in HDL code and the output truth table.

Figure 27: NOT Gate Simulation

In the other step,  the code was written in Verilog, and the Apio sim was the test tool for the codes starting with testing the Elementary logic gates by viewing the waveform.

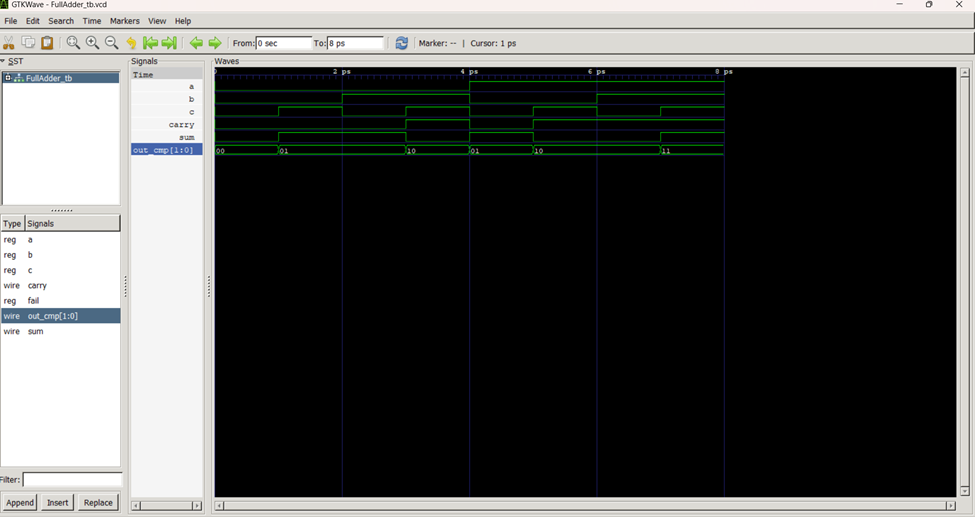
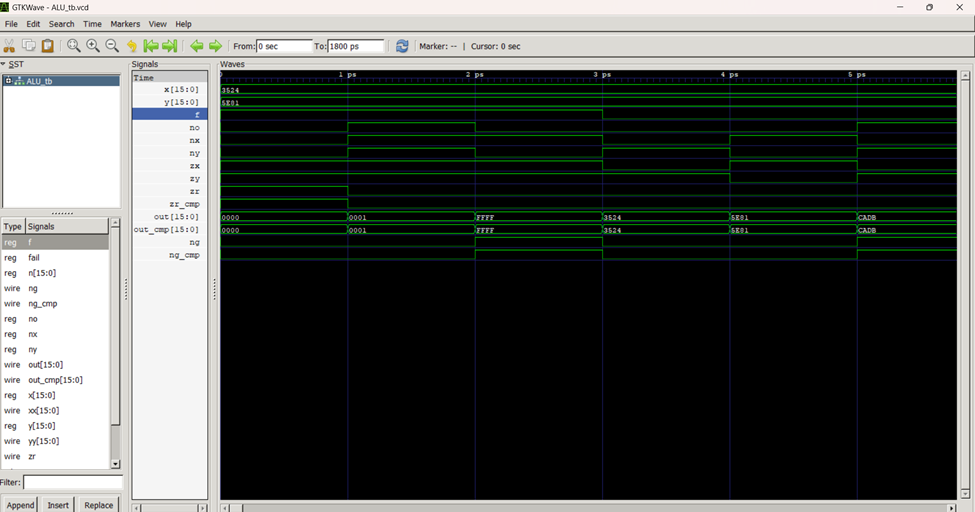


Figure 28: ALU simulation

Figure 29: Full Adder simulation

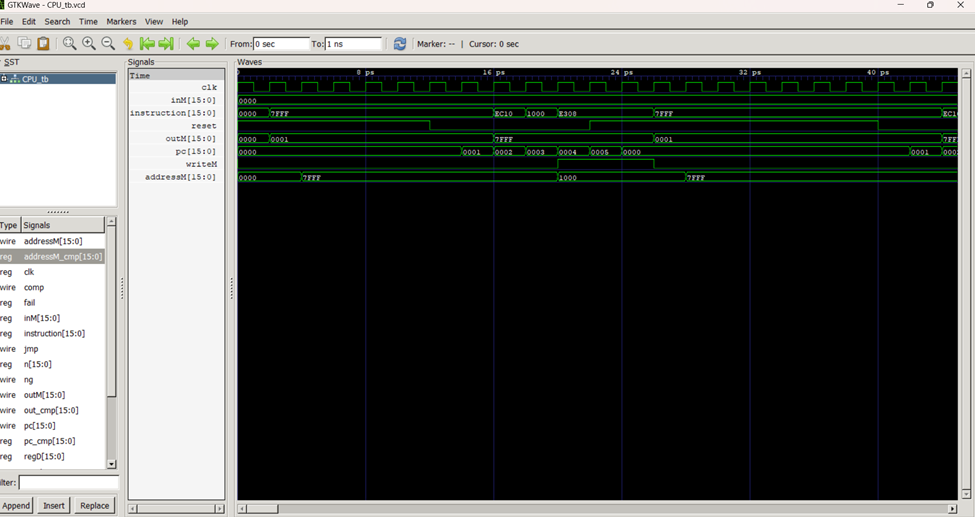


Figure 30: CPU simulation

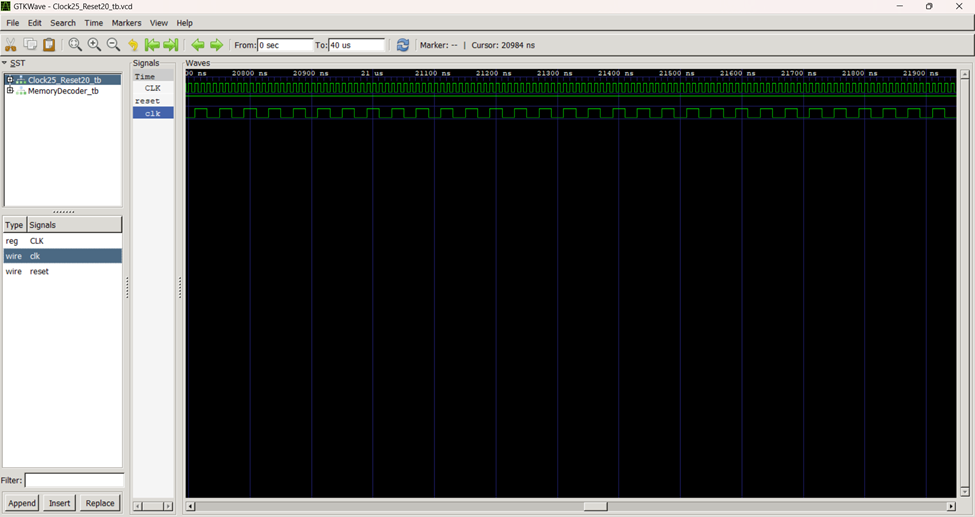


Figure 31: Clock Simulation

## Chapter 4

## Results