### **DESIGN AND IMPLEMENTATION OF HACK CPU AND SYNCHRONOUS COUNTER THAT COUNTS DOWN FROM DECIMAL DIGIT 9 TO 0**

### **A PROJECT REPORT**

*Submitted by*

**CHAITANYA VARMA - CB.SC.U4AIE24017**

**HARSHITH REDDY - CB.SC.U4AIE24018**

**JISHNU TEJA DANDAMUDI - CB.SC.U4AIE24019**

**JIVITES DAMODAR - CB.SC.U4AIE24020**



### 

**CENTER FOR COMPUTATIONAL ENGINEERING AND NETWORKING**

**AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE**

**COIMBATORE-641112**

**2024**

**AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE**

**AMRITA VISHWA VIDYAPEETHAM**

**COIMBATORE-641112**

**DECLARATION**

We, **Chaitanya Varma – CB.SC.U4AIE24017**, **Harshith Reddy - CB.SC.U4AIE24018**, **Jishnu Teja Dandamudi - CB.SC.U4AIE24019** and **Jivites Damodar - CB.SC.U4AIE24020**, hereby declare that the project work entitled “**DESIGNING AND IMPLEMENTATION OF HACK CPU AND SYNCHRONOUS COUNTER THAT COUNTS DOWN FROM DECIMAL DIGIT 9 TO 0**” is the record of the original work done by us under the guidance of **Dr. Lekshmi C. R.**, Assistant Professor, Department of Artificial Intelligence, Amrita Vishwa Vidyapeetham, Coimbatore.

**Chaitanya Varma - CB.SC.U4AIE24017 Harshith Reddy - CB.SC.U4AIE24018**

**Jishnu Teja Dandamudi - CB.SC.U4AIE24019 Jivites Damodar - CB.SC.U4AIE24020**

**Place:** Coimbatore – 641112

**Date:**

**COUNTERSIGNED**

**Dr. Lekshmi C R,**

Assistant Professor,

Department of Artificial Intelligence

**TABLE OF CONTENTS**

**S. No TOPIC Pg. No**

1. **Introduction**
   1. Hardware Description Language 4
   2. Hardware Simulation 4
   3. HACK CPU 21
   4. MOD 10 30
2. **List of Gates Used** 5
3. **Chips with their hdl codes with their truth tables**

* And 7
* Or 8
* Not 9
* And16 10
* Or16 11
* Not16 12
* Or8Way 13
* Mux 14
* Mux16 15
* Xor 16
* Or16Way 17
* Half Adder 18
* Full Adder 19
* Add16 20
* Inc16 21
* ALU 24
* Bit 26
* Register 26
* PC 27
* Ram8 27
* Ram64 27
* HACK CPU 28
* And3 34
* JK Flip Flop using D Flip Flop 34
* MOD10 Synchronous Down Counter 35

1. **List of Figures**

* And 7
* Or 8
* Not 9
* And16 10
* Or16 11
* Not16 12
* Or8Way 13
* Mux 14
* Mux16 15
* Xor 16
* Or16Way 17
* Half Adder 18
* Full Adder 19
* Add16 20
* Inc16 21
* Top Most Computer HACK CHIP 23
* Design of HACK CPU 23
* ALU 24
* HACK CPU 27
* Block diagram for mod-10 synchronous down counter 32
* State Diagram for MOD10 33
* K-MAPS for JK Flip Flops Used 33

1. **Outputs**

* ALU 25
* HACK CPU 29
* MOD10 36

1. **Conclusion**  37

**INTRODUCTION**

1. **HARDWARE DESCRIPTION LANGUAGE(HDL):**

Today, hardware designers no longer build anything with their bare hands. Instead, they plan and optimize the chip architecture on a computer workstation, using structured modelling formalisms like Hardware Description Language, or HDL (also known as VHDL, where V stands for Virtual). The designer specifies the chip structure by writing an HDL program, which is then subjected to a rigorous battery of tests. These tests are carried out virtually, using computer simulation: A special software tool, called a hardware simulator, takes the HDL program as input and builds an image of the modelled chip in memory. Next, the designer can instruct the simulator to test the virtual chip on various sets of inputs, generating simulated chip outputs. The outputs can then be compared to the desired results, as mandated by the client who ordered the chip built. In addition to testing the chip’s correctness, the hardware designer will typically be interested in a variety of parameters such as speed of computation, energy consumption, and the overall cost implied by the chip design. All these parameters can be simulated and quantified by the hardware simulator, helping the designer optimize the design until the simulated chip delivers desired cost/performance levels. Thus, using HDL, one can completely plan, debug, and optimize the entire chip before a single penny is spent on actual production. When the HDL program is deemed complete, that is, when the performance of the simulated chip satisfies the client who ordered it, the HDL program can become the blueprint from which many copies of the physical chip can be stamped in silicon. This final step in the chip life cycle - from an optimized HDL program to mass production - is typically out-sourced to companies that specialize in chip fabrication, using one switching technology or another.

1. **HARDWARE SIMULATION:**

Since HDL is a hardware construction language, the process of writing and debugging HDL programs is quite similar to software development. The main difference is that instead of writing code in a language like Java, we write it in HDL, and instead of using a compiler to translate and test the code, we use a hardware simulator. The hardware simulator is a computer program that knows how to parse and interpret HDL code, turn it into an executable representation, and test it according to the specifications of a given test script. There exist many commercial hardware simulators on the market, and these vary greatly in terms of cost, complexity, and ease of use. Together with this book we provide a simple (and free!) hardware simulator that is sufficiently powerful to support sophisticated hardware design projects. In particular, the simulator provides all the necessary tools for building, testing, and integrating all the chips presented in the book, leading to the construction of a general-purpose computer.

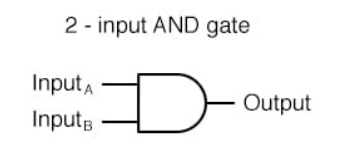
**LIST OF GATES USED IN THE OVERALL PROJECT**

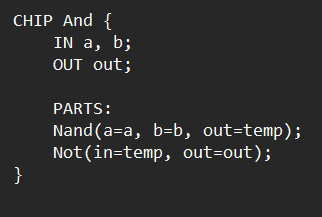
1. And
2. Or
3. Not
4. And16
5. Or16
6. Not16
7. Or8Way
8. Mux
9. Mux16
10. Xor
11. Or16Way
12. Half Adder
13. Full Adder
14. Add16
15. Inc16

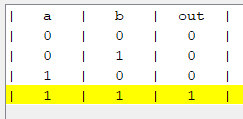
**CHIPS USED**

1. **AND**

The And function returns 1 (true) only when both of the inputs are 1 (true).

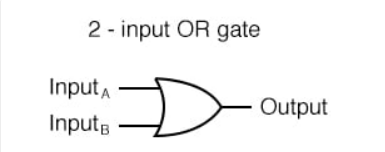


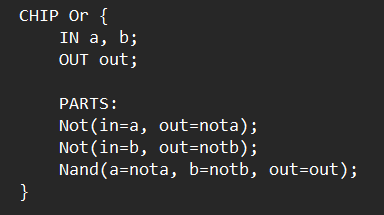
****

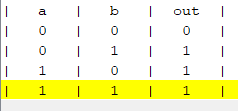
****

1. **OR**

The Or function returns 1 (true) when either of the inputs are 1 (true).

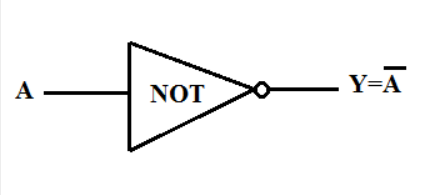


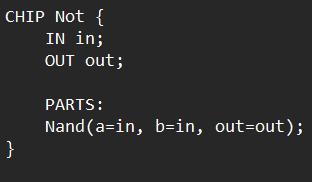
****

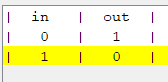
****

1. **NOT**

The single-input Not Gate, also referred as Inverter, inverts the value of input.

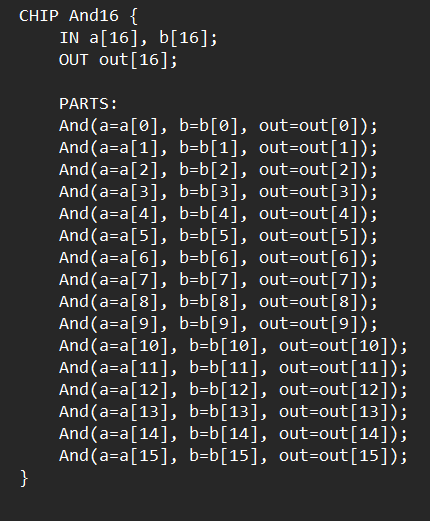


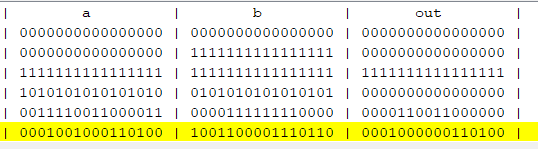
****

****

1. **AND16**

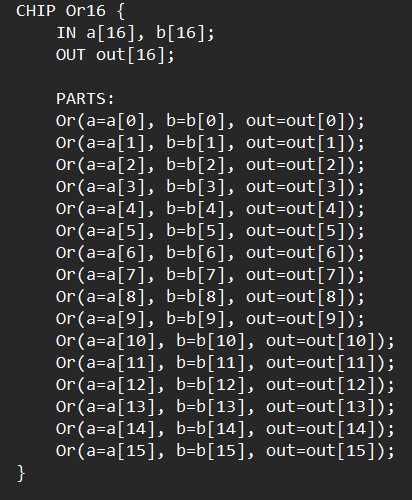
An 16-bit And Chip applies the Boolean operation And to every one of the bits in its 16-bit input bus.

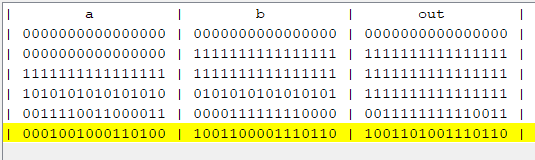
****

****

1. **OR16**

An 16-bit Or Chip applies the Boolean operation Or to every one of the bits in its 16-bit input bus.

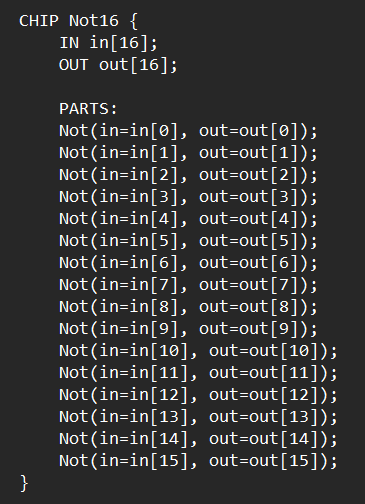
****

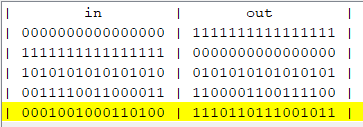
****

1. **NOT16**

Unlike Not gate, Not16 takes 16-bit input, inverts each bit and produces output.

In other words, An 16-bit Not Chip applies the Boolean operation Not to every one of the bits in its 16-bit input bus.

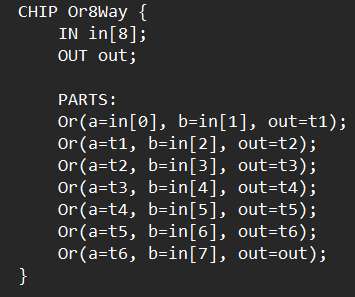
****

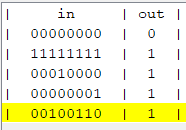
****

1. **OR8WAY**

An 8-way or gate outputs 1 when at least one bit of its 16-bit input is 1.

Unlike n-bit input logic gates, n-way logic gates use the same output iteratively over the Boolean operation, which means, it uses the previous output as input for the next similar Boolean operation.

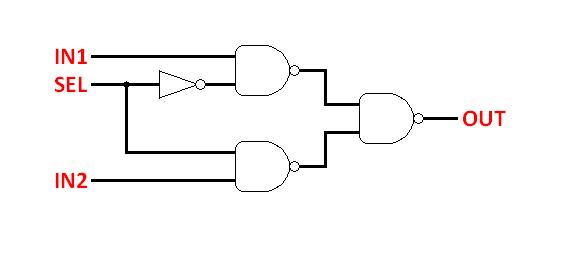
****

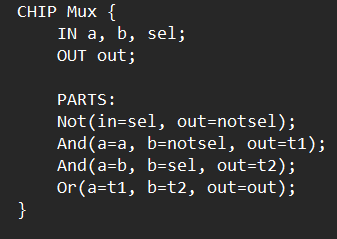
****

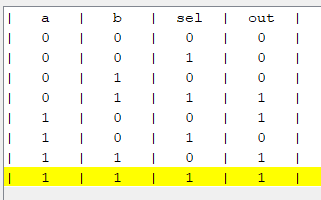
1. **MUX**

A Multiplexor, also known as Selector, is a three-input bit gate that uses one of the inputs called "selection bit" to select one of the given two inputs for outputting them.

The name multiplexor was adopted from communications systems, where similar devices are used to serialize (multiplex) several input signals over a single output wire.

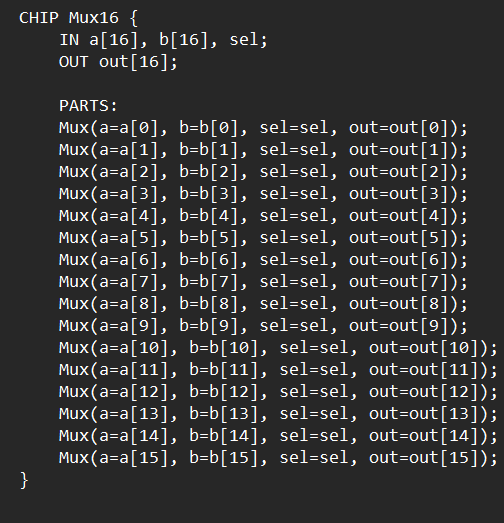


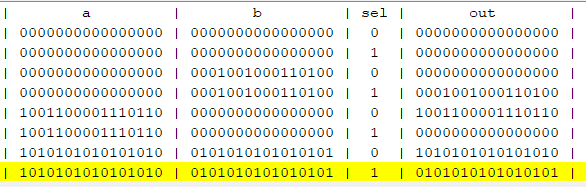
****

****

1. **MUX16**

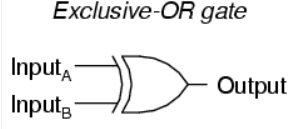
An 16-bit multiplexor is almost similar to the Multiplexor Chip described previously, except that both inputs are each 16-bit wide, while the selector is a single bit.

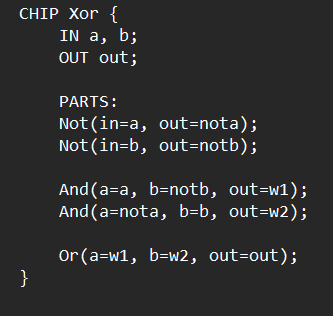
****

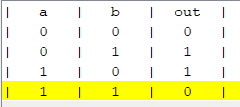
****

1. **XOR**

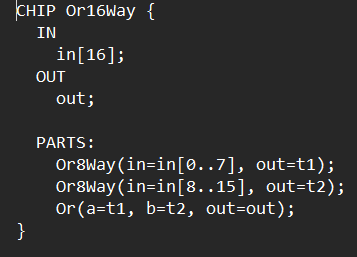
The Xor function, also known as Exclusive-OR, returns 1 (true) when either of the inputs are same (both are 0 or both are 1 simultaneously).

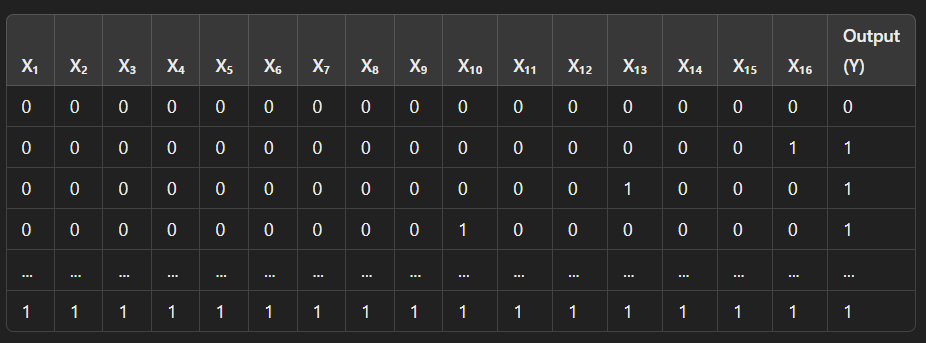


****

****

1. **OR16WAY**

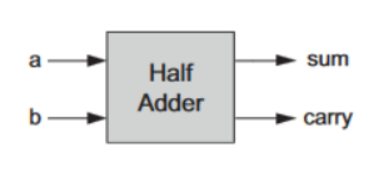
****

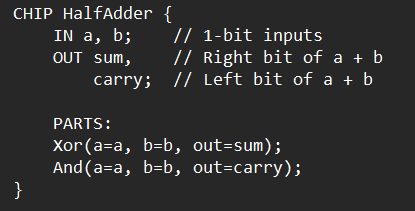
****

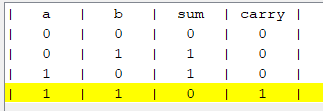
1. **HALF ADDER**

The chip used to add two n-bit numbers is known as Adder, also known as n-bit Adder.

Half Adder chip is used to add 2-bits.

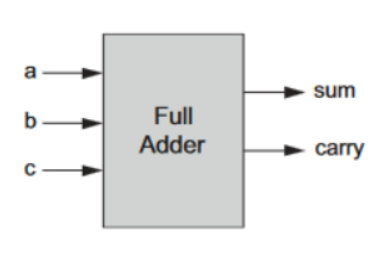


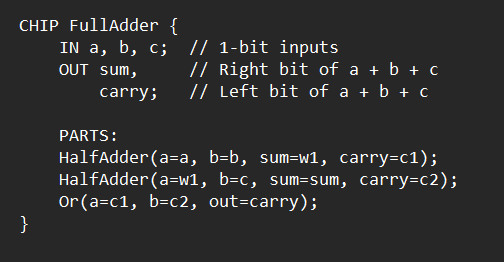
****

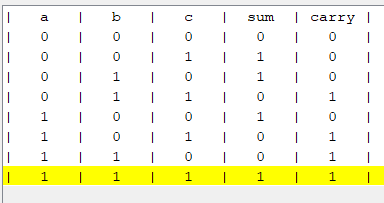
****

1. **FULL ADDER**

Full Adder chip is used to add 3-bits.

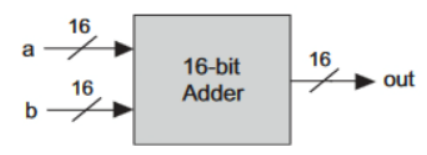


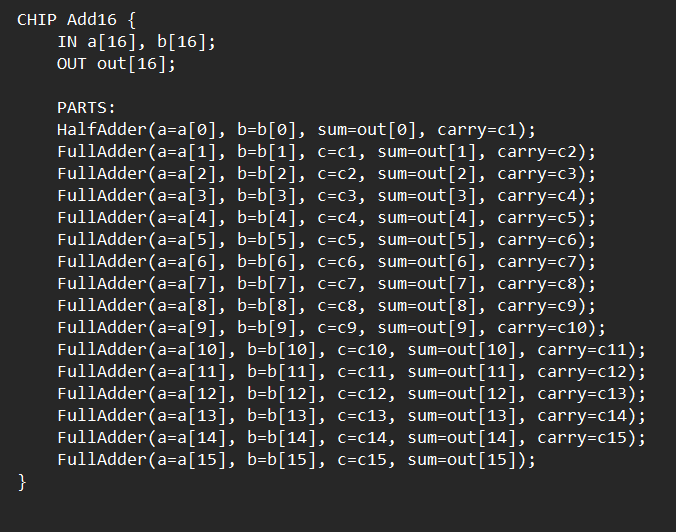
****

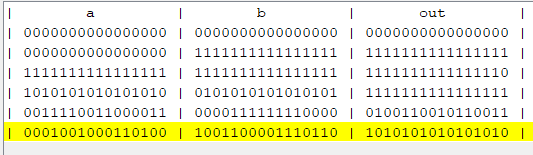
****

1. **ADD16**

16-bit Adder chip is used to add two 16-bit numbers.

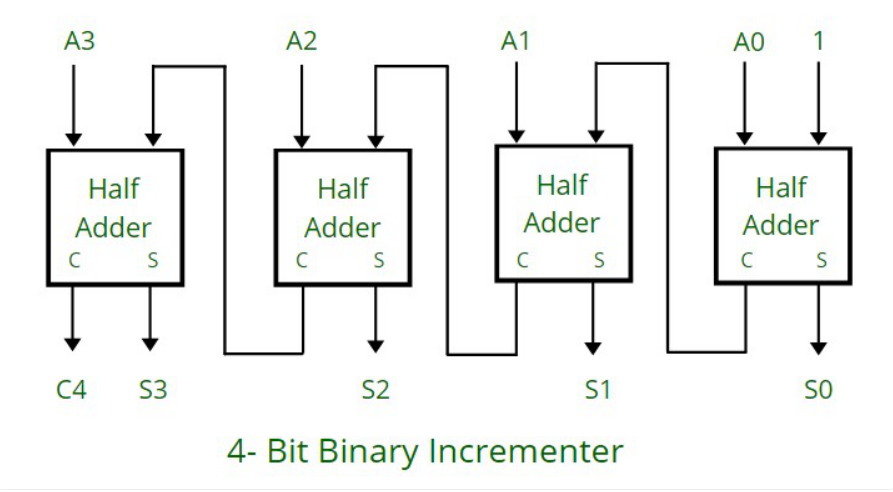


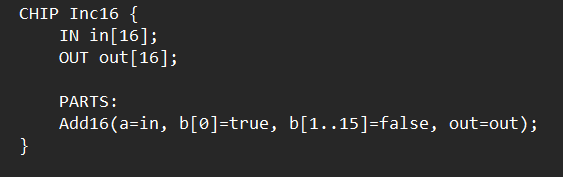
****

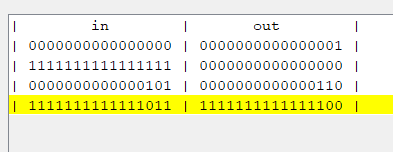
****

1. **INC16**

16-bit Incrementor chip is a special kind of Adder, used to increment a 16-bit input by 1.



****

**.** ****

**DESIGNING AND IMPLEMENTING A 16 BIT HACK CPU**

1. **HACK CPU:**

The Hack CPU consists of the ALU specified in chapter 2 and three registers called data register (D), address register (A), and program counter (PC). D and A are general-purpose 16-bit registers that can be manipulated by arithmetic and logical instructions like A=D-1, D=D|A, and so on, following the Hack machine language specified in chapter 4. While the D-register is used solely to store data values, the contents of the A-register can be interpreted in three different ways, depending on the instruction’s context: as a data value, as a RAM address, or as a ROM address. The Hack machine language is based on two 16-bit command types. The address instruction has the format 0vvvvvvvvvvvvvvv, each v being 0 or 1. This instruction causes the computer to load the 15-bit constant vvv...v into the A-register. The compute instruction has the format 111accccccdddjjj. The a- and c-bits instruct the ALU which function to compute, the d-bits instruct where to store the ALU output, and the j-bits specify an optional jump condition, all according to the Hack machine language specification. The computer architecture is wired in such a way that the output of the program counter (PC) chip is connected to the address input of the ROM chip. This way, the ROM chip always emits the word ROM[PC], namely, the contents of the instruction memory location whose address is “pointed at” by the PC. This value is called the current instruction.

1. **Contents of HACK CPU:**

The HACK CPU has the following key features:

* 16-bit word size: The CPU processes data in 16-bit chunks.
* Registers: It includes a program counter (PC), instruction register (IR), and a set of general-purpose registers (R0 to R7).
* ALU: An Arithmetic Logic Unit to perform arithmetic and logical operations.
* Memory: Support for RAM and ROM.
* Input/Output: Basic I/O operations through the RAM.

a. Registers

* Program Counter (PC): Points to the address of the next instruction.
* Instruction Register (IR): Holds the current instruction being executed.
* General-purpose registers (R0 to R7): For temporary data storage.

b. ALU Operations

The ALU supports several operations, such as:

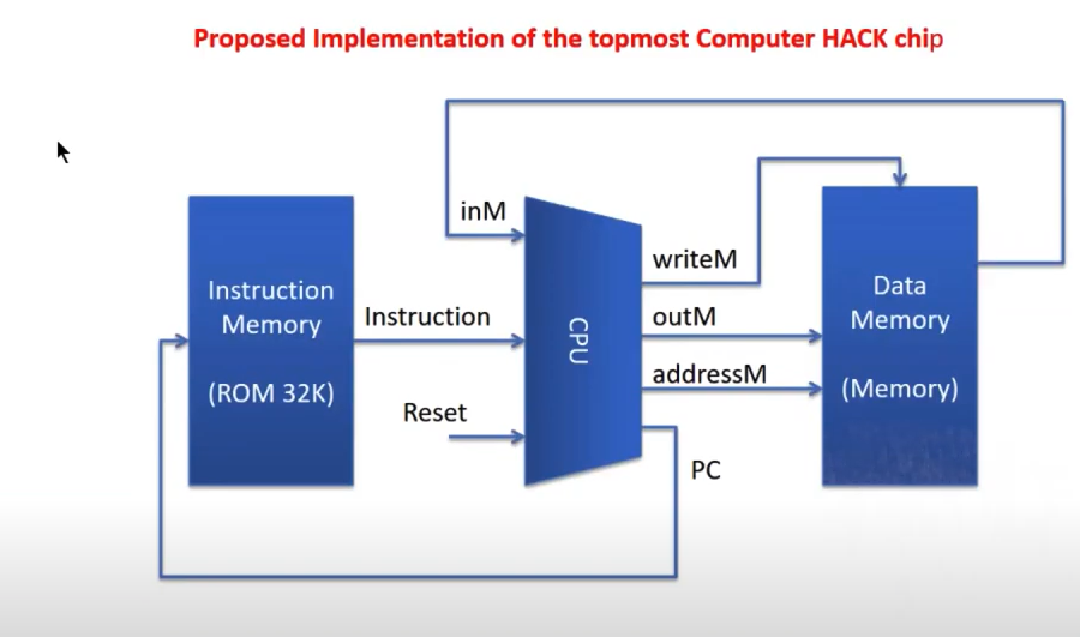
* Addition
* Subtraction
* Bitwise AND, OR, NOT

c. Control Unit

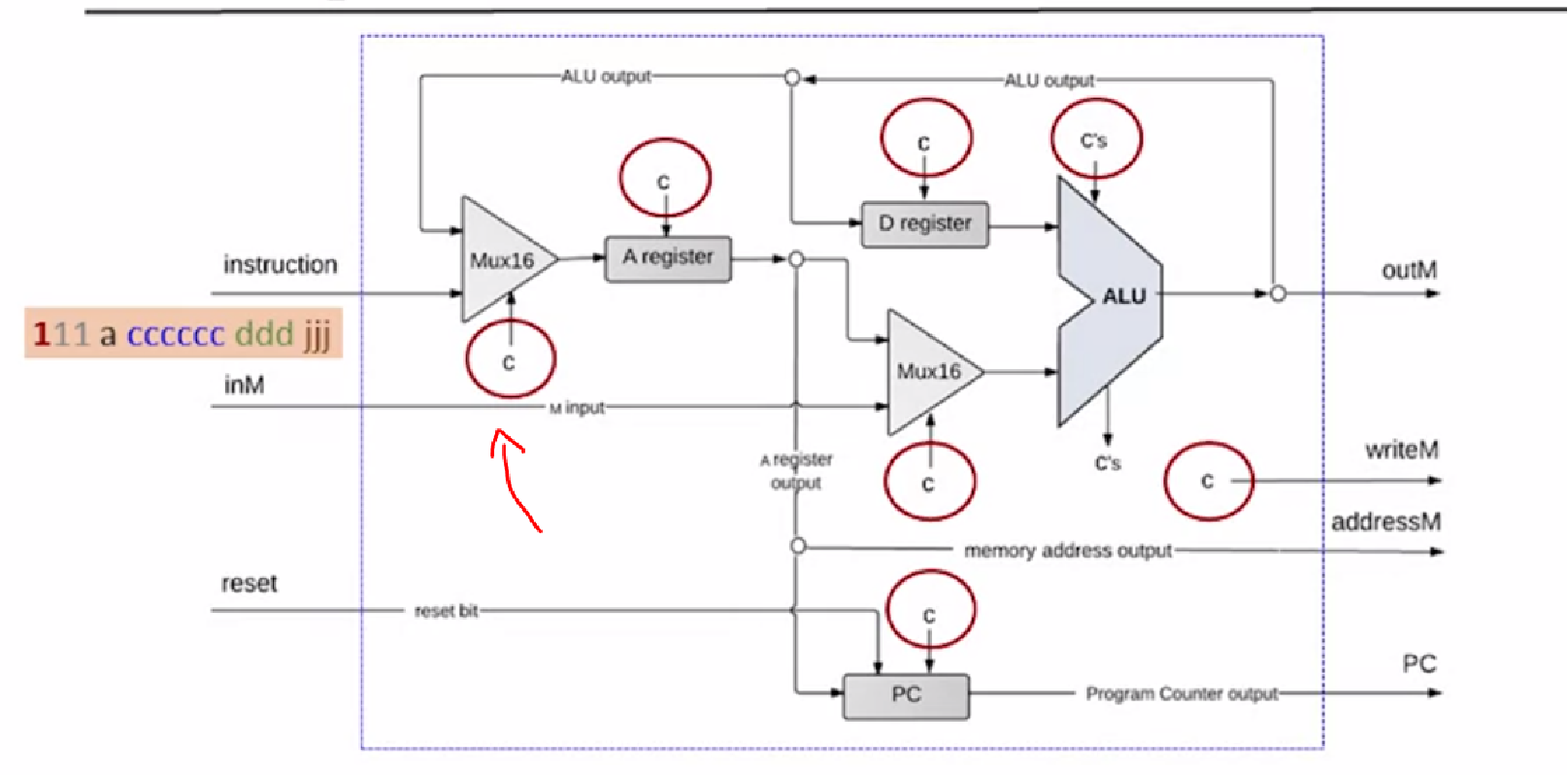
* Decodes instructions and controls the execution of operations.

d. Memory

* RAM: Stores data and instructions.
* ROM: Stores the program to be executed.

****

**CENTRAL PROCESSING UNIT**

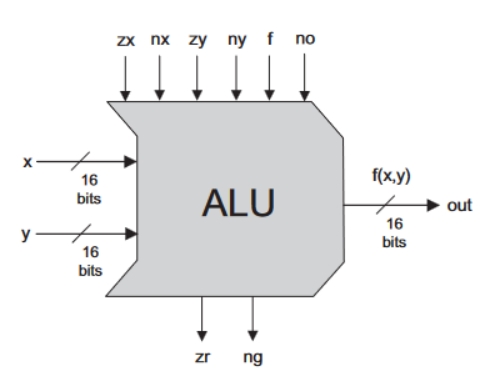
****

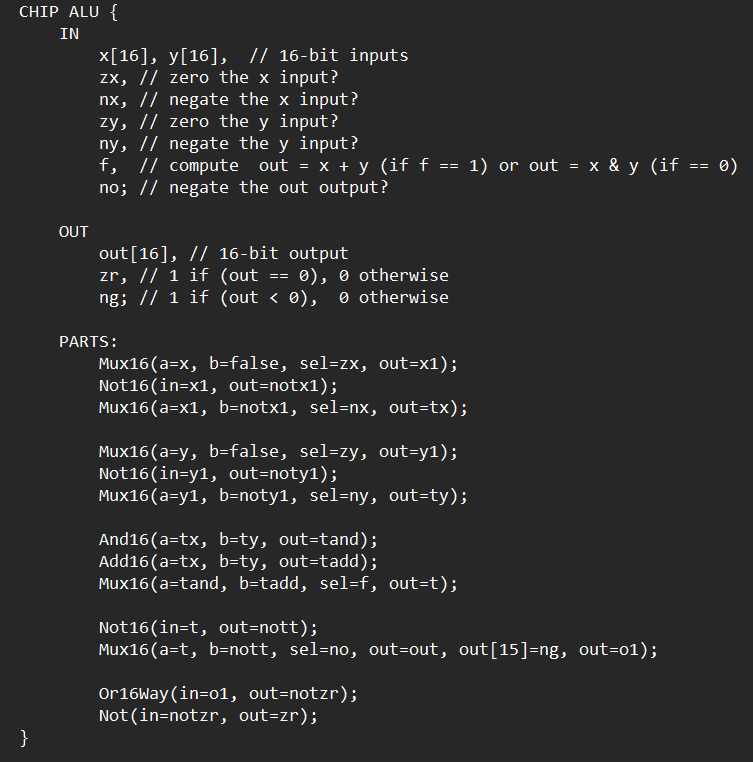


**ALU**

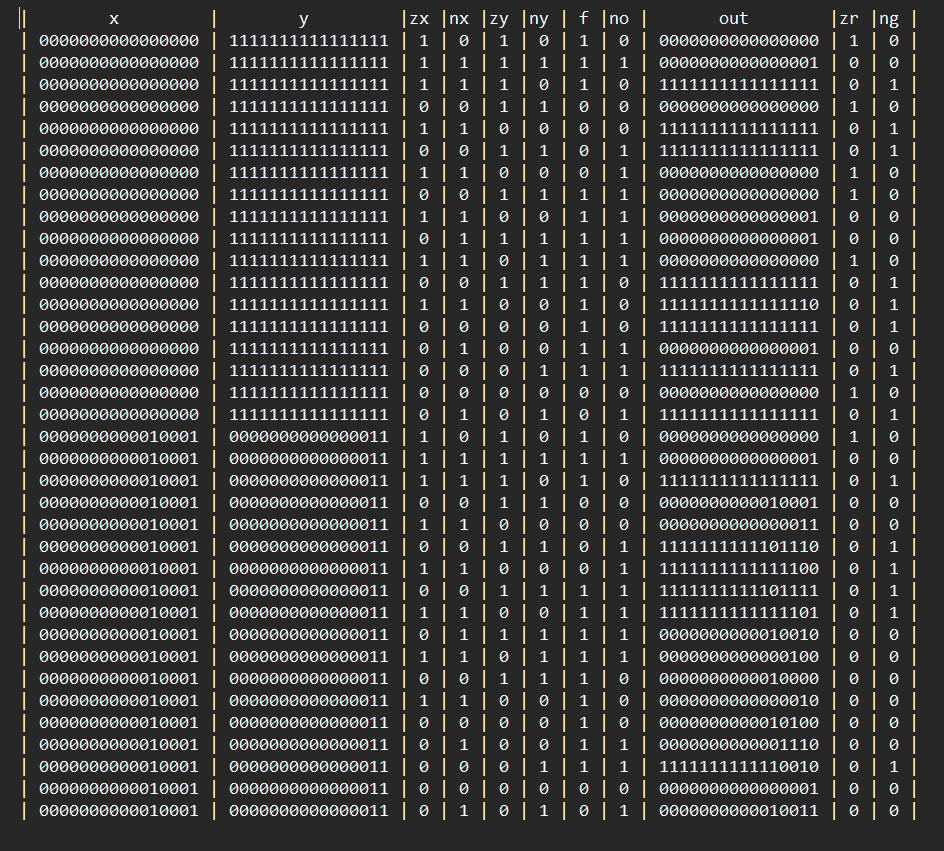
The Hack ALU computes a fixed set of functions on given two 16-bit inputs, out of which the function can be one of the possible eighteen functions.

We instruct the ALU which function to compute using six input bits, called control bits to the selected binary values.

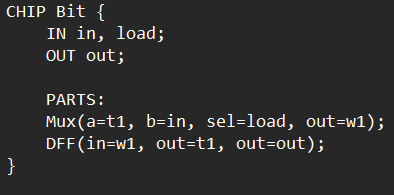


****

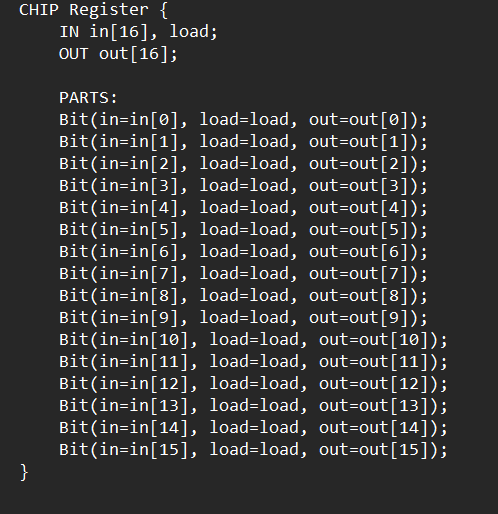
**OUTPUT**

****

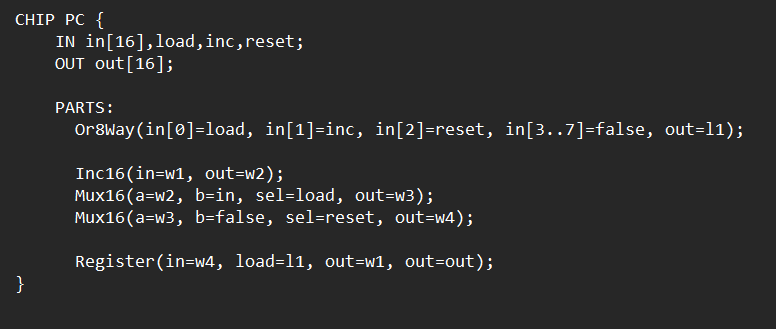
**BIT**

****

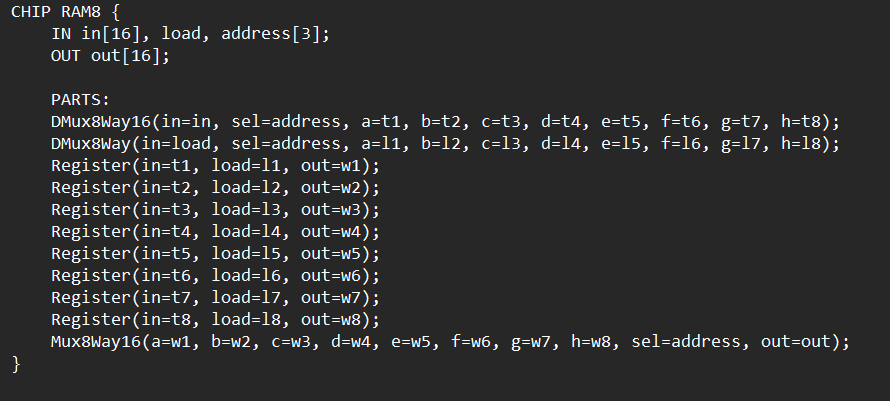
**REGISTER**

****

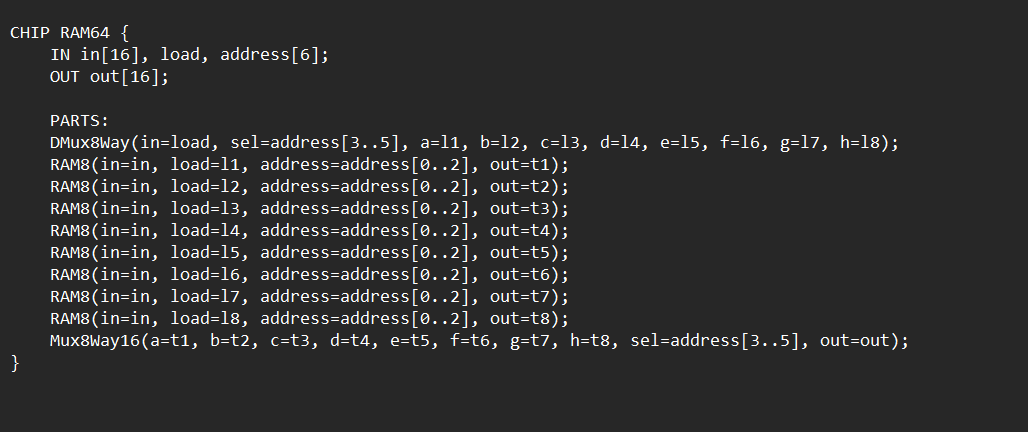
**PC**

****

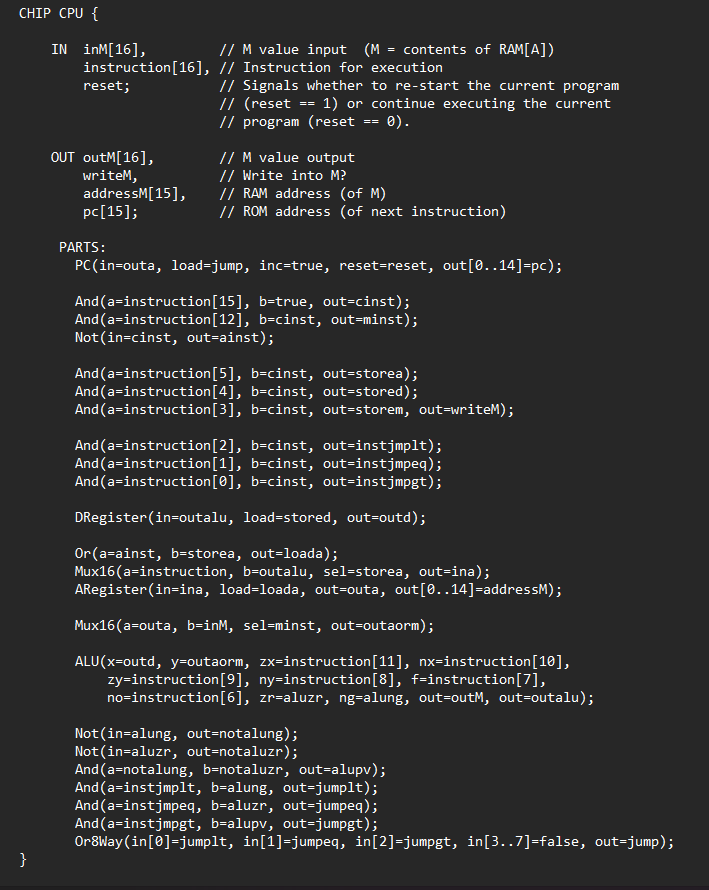
**RAM8**

****

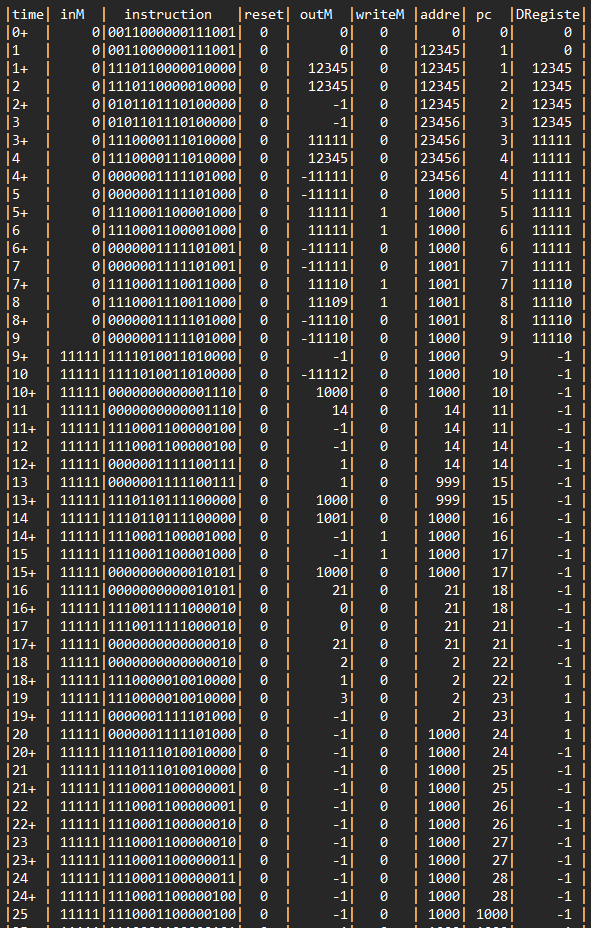
**RAM64**

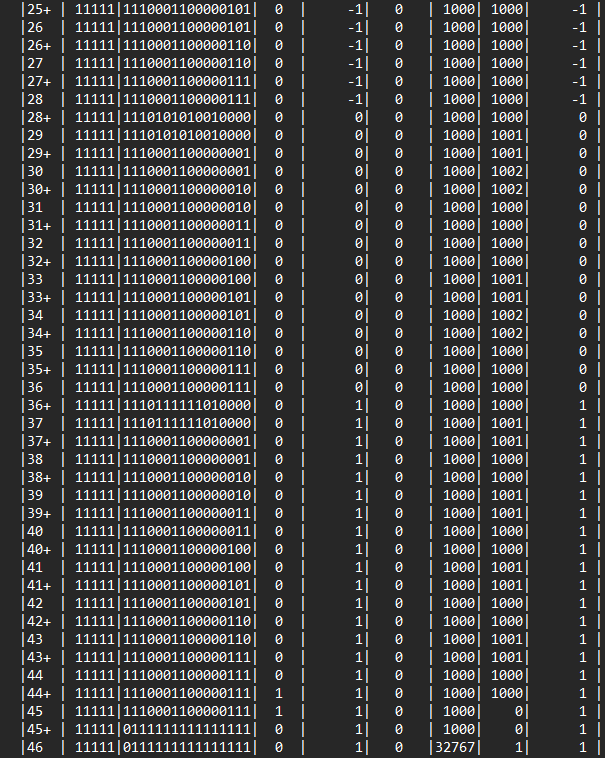
****

**HACK CPU**

****

**OUTPUT**

****

****

**DESIGN AND IMPLEMENT A SYNCHRONOUS COUNTER THAT COUNTS DOWN FROM DECIMAL DIGIT 9 TO 0**

**Introduction:**

A 4-bit 9-0 Synchronous Counter is a digital circuit that counts in a binary sequence from 9 to 0 and then resets to 9, essentially counting decimal digits from 9 to 0. It’s built using flip-flops, which change states in synchronization with the clock signal, ensuring all bits update at the same time. Synchronous counters have several advantages over asynchronous counters, including reduced propagation delays and increased reliability at higher clock frequencies. These counters find applications in digital clocks, timers, and other devices requiring precise counting.

This project aims to design and implement a 4-bit synchronous counter that limits the counting to a range from 9 to 0 (binary sequence 1001 to 0000). Such a counter is also known as a decade counter because it counts ten states. Beyond the zeroth count (0000), the counter resets to zero on the next clock pulse.

**Objectives:**

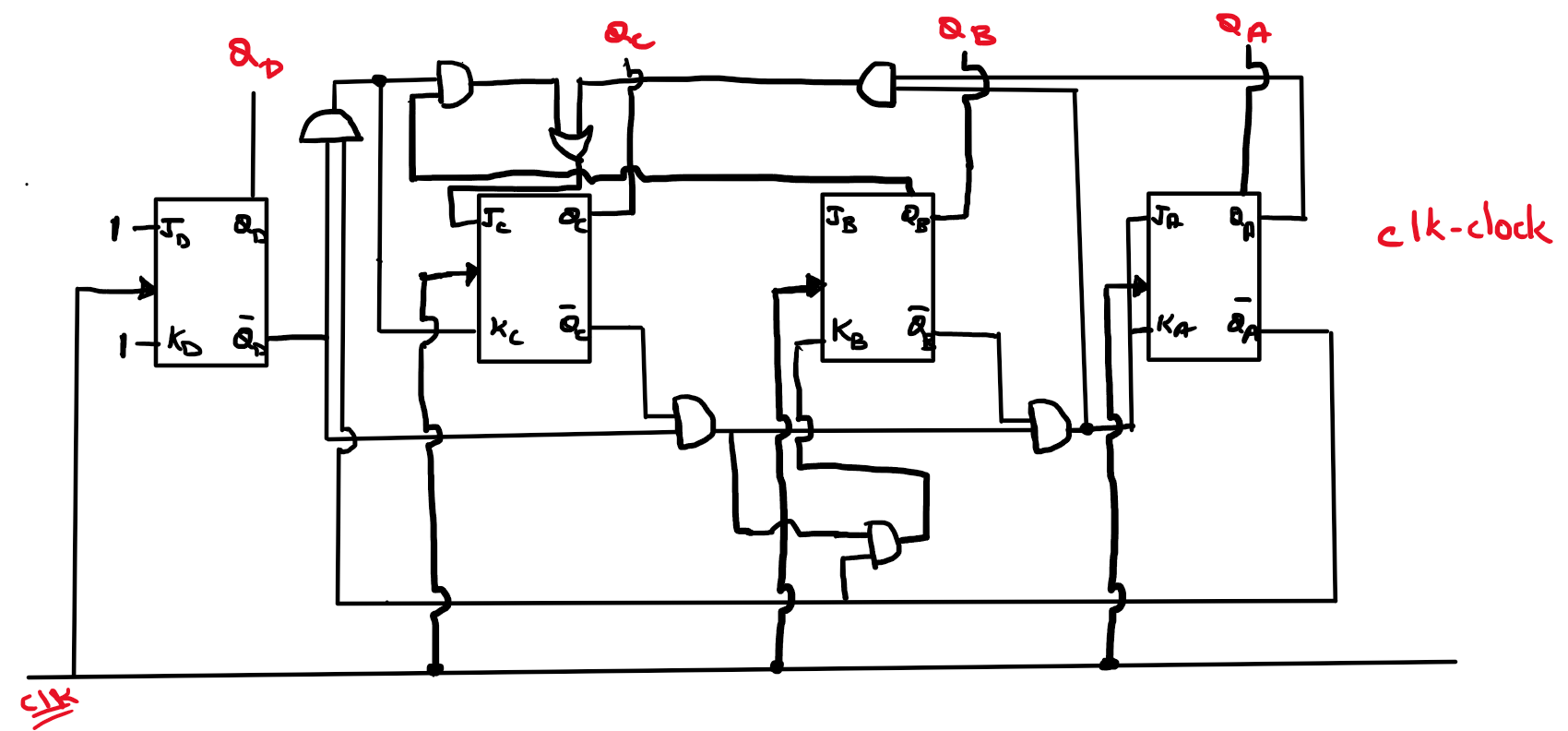
Understand the fundamentals of synchronous counters - Gain insight into how synchronous counters differ from asynchronous counters, focusing on the use of a clock pulse to synchronize state changes.

Design a 4-bit binary counter circuit - Develop a binary counter using JK flip-flops to count sequentially from 9 to 0.

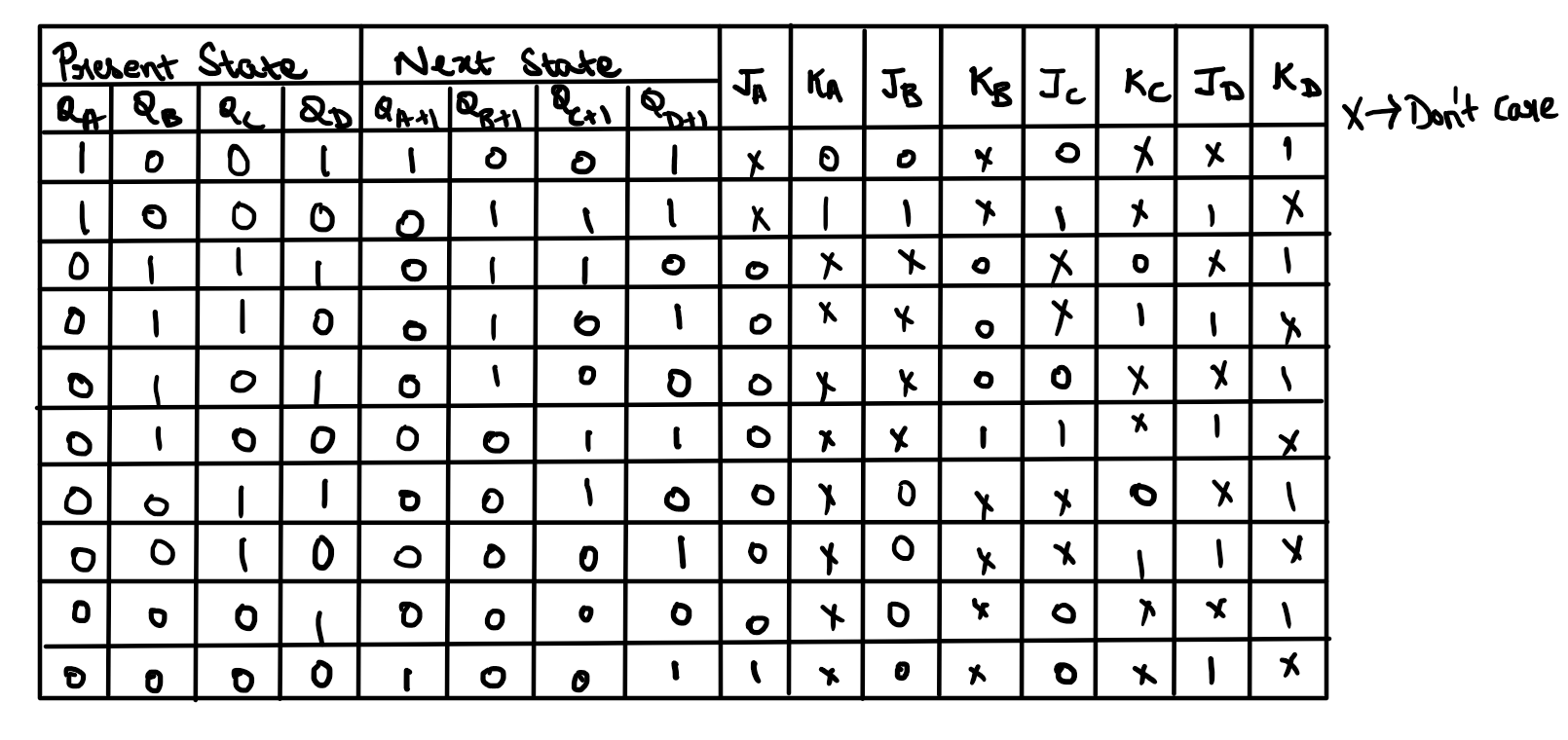
Implement a reset mechanism - Ensure the counter resets back to 0 after reaching the binary equivalent of 0 (0000).

**Analyse applications** - Explore the practical applications of synchronous counters in devices requiring controlled counting mechanisms, such as digital clocks, frequency dividers, and electronic timers.

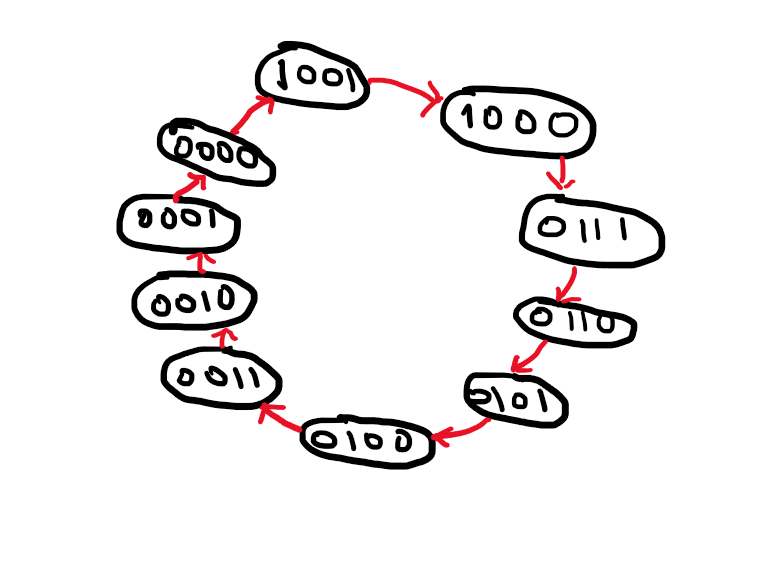
**BLOCK DIAGRAM FOR MOD-10 SYNCHRONOUS DOWN COUNTER**



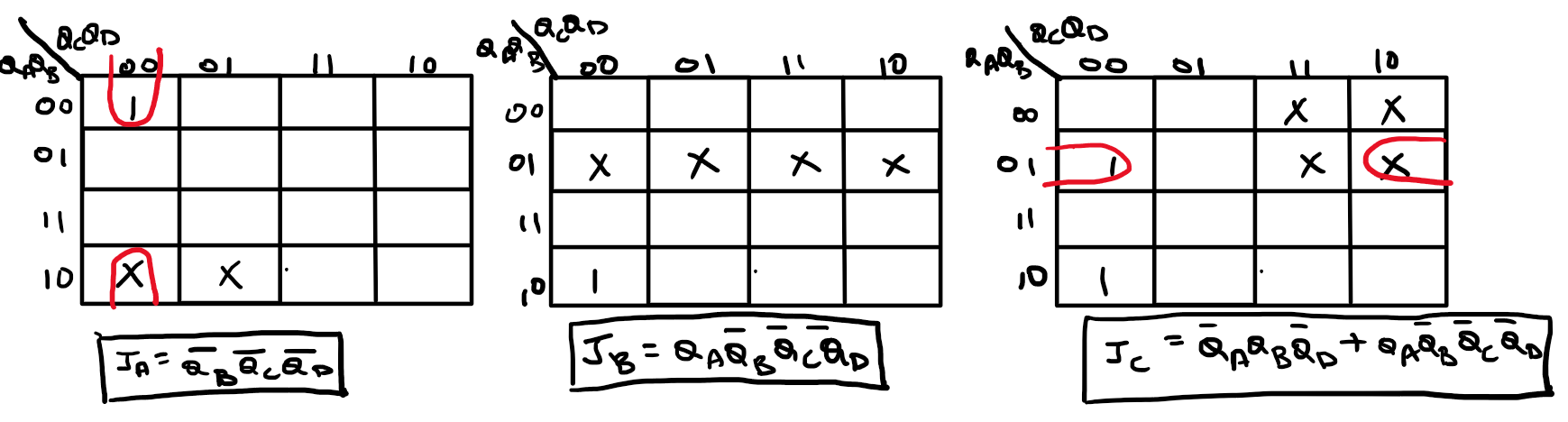
**TRUTH TABLE**

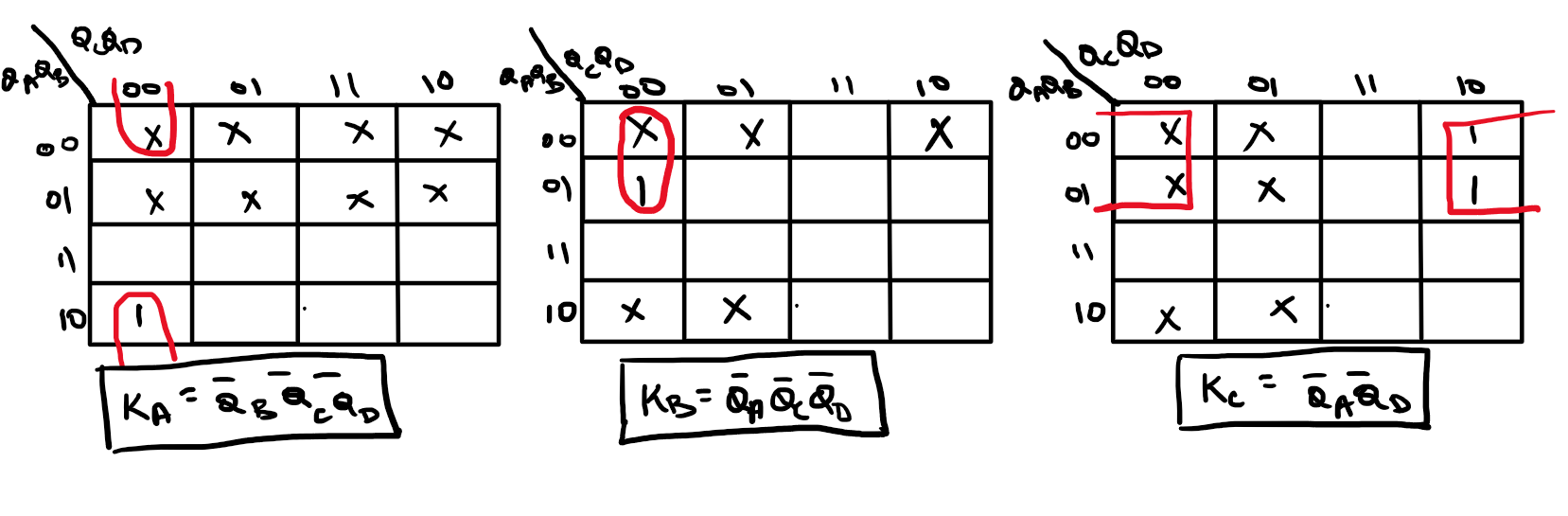


**STATE DIAGRAM**



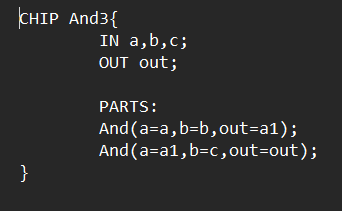
**K-MAPS**



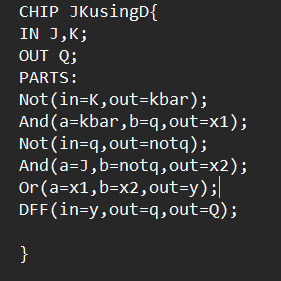


**GATES USED**

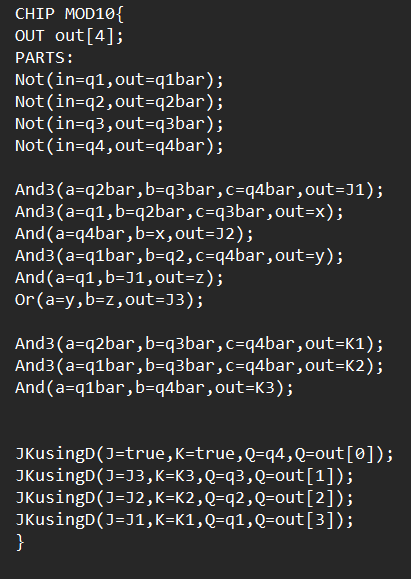
**AND3**

****

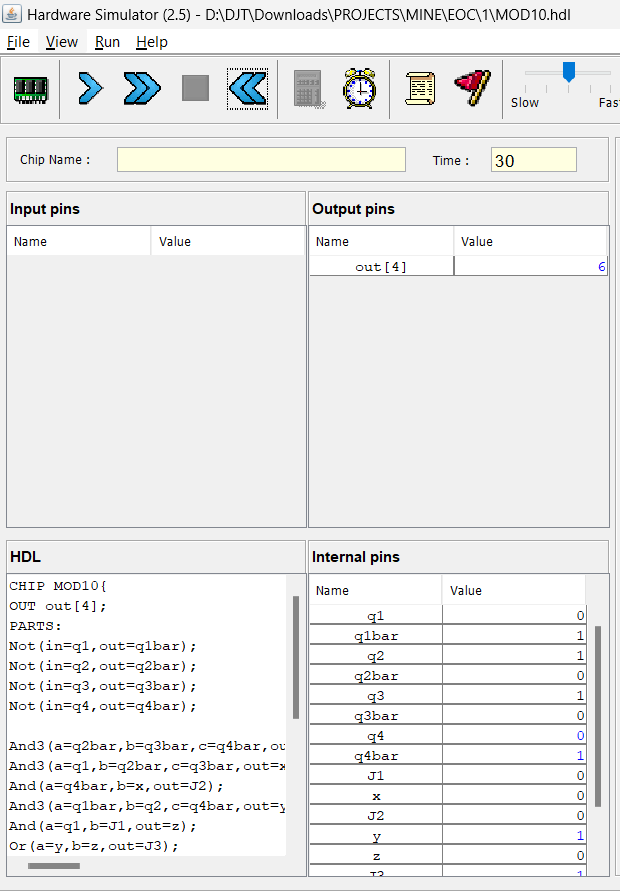
**JK FLIP FLOP USING D FLIP FLOP**

****

**MOD10**

****

**OUTPUT**

****

**CONCLUSION**

We have designed and implemented a HACK CPU. The Hack CPU is the target for the Jack language compiler later in the course. By compiling Jack code to Hack machine language, students see how high-level code translates to machine-level operations, solidifying their understanding of compilation and CPU operation.

We have also designed and implemented a SYNCHRONOUS COUNTER that counts down from DECIMAL DIGIT 9 TO 0. This type of counter is useful for visual countdowns, such as those seen on display boards at events, sports arenas, or conferences, where it counts down each second or minute. It is often coupled with a display to show the countdown, providing a visual cue that counts down from a set time until an event, like the start of a race or a break in a program. Many household and consumer devices use down counters in settings like microwave timers, washing machine cycles, or cooking timers, decrementing through a sequence until the task completes. Since all flip-flops in a synchronous counter are triggered simultaneously, they offer a predictable and stable output, ideal for precise applications. Synchronous counters are faster than asynchronous counters because they do not suffer from propagation delay between stages, making them suitable for high-speed digital applications.