Project Report

Building of HACK CPU

&

Design and implement a digital system that takes the Gray code output from a rotary encoder and displays the rotary shaft's position on a 7-segment display

---

‘

PART A
Design and implement 16-bit HACK CPU

Design and implement 16-bit HACK CPU

Abstract:-

Through this project we aim to design and implement an 16-Bit HACK CPU using Nand2Tetris.This 16-Bit HACK CPU must be capable of perform arithmetic and logical operations, handle memory access, and execute programs

Through this project we aim to build a computer system from very basic steps ,like building logic gates to more complex construction of fully functional computer architecture.The initial step is to build basic logic gates using Nand gates and progressively use these basic gates to built more convulted ones like multiplexers,decoders and adders.Following this we combine these available logics gates to create memory units,registers and ALU(arithmetic-logic unit)

Next step is to design the CPU architecture (using Von Neumann model as reference).The architecture includes a program counter (PC), an instruction memory, a data memory, and various registers. Further control logic is implemented to interpret the instructions, control the flow of data, and coordinate the execution of various operations. Additionally, the CPU is designed to support basic arithmetic and logical operations, branching, and memory access.

To ease the process of testing and integration we use a hardware description language (HDL),where we describe the components of the CPU and define the interconnections between them.Additionally,Simulation tools are used to verify the correctness and functionality of each of the designed module, and a hardware simulator is utilized to test the entire CPU system.

Finally, this project intends to provide hands-on experience in building a computer system.It further helps one nurture a deep understanding of digital logic, computer architecture, and the principles of computation. The resulting 16-bit HACK CPU serves as a foundation for further exploration and experimentation, enabling learners to study and learn the intricacies of computer organization and gain practical insights into the inner workings of a simple yet functional computing system.

Keyword

Central Processing Unit(CPU),Computer Architecture,Arithmetic Logical Unit(ALU), Program Counter(PC),Registers,Assembler,Compiler,simulation,Nand2Tetris

I. INTRODUCTION

In the constantly growing domain of computer architecture,there have been multiple researches and studies, that have paved way to exploration of less investigated areas.One of the major finding over the year is undoubtedly the design and implementation of a 16-bit hack CPU

The term "hack" in this context signifies a diversion from conventional designing principles, encouraging the formulation of new ideas in designing a functional and efficient central processing unit.The central processing unit in a computer system, executes instructions, manages data, and oversees the flow of information between various components. Its 16-bit architecture provides a balance between computational power and simplicity,it poses a unique set of challenges and opportunities.

As there is a rising demand for specialized and customized computing solutions, the need for specialized processors also rises. This project aims to construct a 16-bit CPU, with the help of foundational logic gates and a complex instruction set.

The significance of this study extends beyond mere academic exploration. Building a 16-bit hack CPU not only helps hone the skills of hardware designers but also provides a platform for understanding the inner workings of computers at a fundamental level.In this project we will delve into the concepts of digital logic, assembly language programming, and optimization strategies.

As we delve into constructing a 16-bit Hack CPU, this report seeks to unravel the motivations guiding this undertaking, the methodological considerations shaping design choices, and the valuable lessons garnered through the project's progression. By sharing our experiences and findings, we aspire to contribute meaningfully to the scholarly conversation on CPU design and inspire further exploration in the ever-evolving landscape of computer architecture.

II.Methodology

For the development of the 16-Bit HACK CPU we utilized the NAND2TETRIS software suite.The Nand2Tetris contains various project files and tools that are helpful in the designing and implementation of the HACK CPU.

The methodological approach behind the project is pretty simple and easy to follow even for beginners,we simply just have to follow the structured order and path used in the project files of Nand2Tetris.The method involves breaking the projects into steps ,which progress from being simple to more complex one.We initially start by building basic gates like AND,OR,NOR,XOR ,etc using NAND gates.Further we utilize these gates to build combinational gates and finally the ALU(arithmetic-logic unit).We also implement memory components and registers. After completion of above steps we progress to build the CPU architecture which includes a program counter (PC), an instruction memory, a data memory, and various registersThis sequential approach ensures a clear understanding of each component before integrating them into a cohesive and functional whole.

For gathering information about project requirements, specifications, and underlying architecture we rely primarily on the official NAND2Tetris documentation.It gives information regarding the Hack computer architecture, instruction set, and overall system design. Additionally, engagement with the NAND2Tetris community forums and resources contributes to an iterative process of refining our design based on collective knowledge and experiences shared by other participants.Further we used some github codes and prior research papers as our reference.

Next for analyzing and validating each designed component and the CPU as a whole. We employ rigorous testing and simulation using the NAND2Tetris hardware simulator. Individual components such as ALUs and memory units are subjected to specific test cases to ensure their correctness and functionality. The analysis extends to integrating these components into the overall CPU architecture, involving comprehensive testing of instruction execution and system behavior. The hardware simulator allows us to observe signals, registers, and memory states, aiding in identifying and resolving any anomalies or errors in our design.

The methodology aligns with the structured and educational nature of the NAND2Tetris project, which emphasizes a bottom-up construction approach for a computer system.This method and practice helped us gain a deep understanding of computer architecture and digital logic design. The step-by-step progression ensures that each participant comprehensively understands the purpose and functionality of individual components before integrating them into a larger system. Furthermore, the use of the hardware simulator facilitates efficient testing and debugging, supporting an iterative development process. The methodology balances hands-on learning with theoretical understanding, providing a holistic and effective approach to designing and implementing the 16-bit Hack CPU using the NAND2Tetris framework.

III.Research and implementation

A.Building of HACK CPU

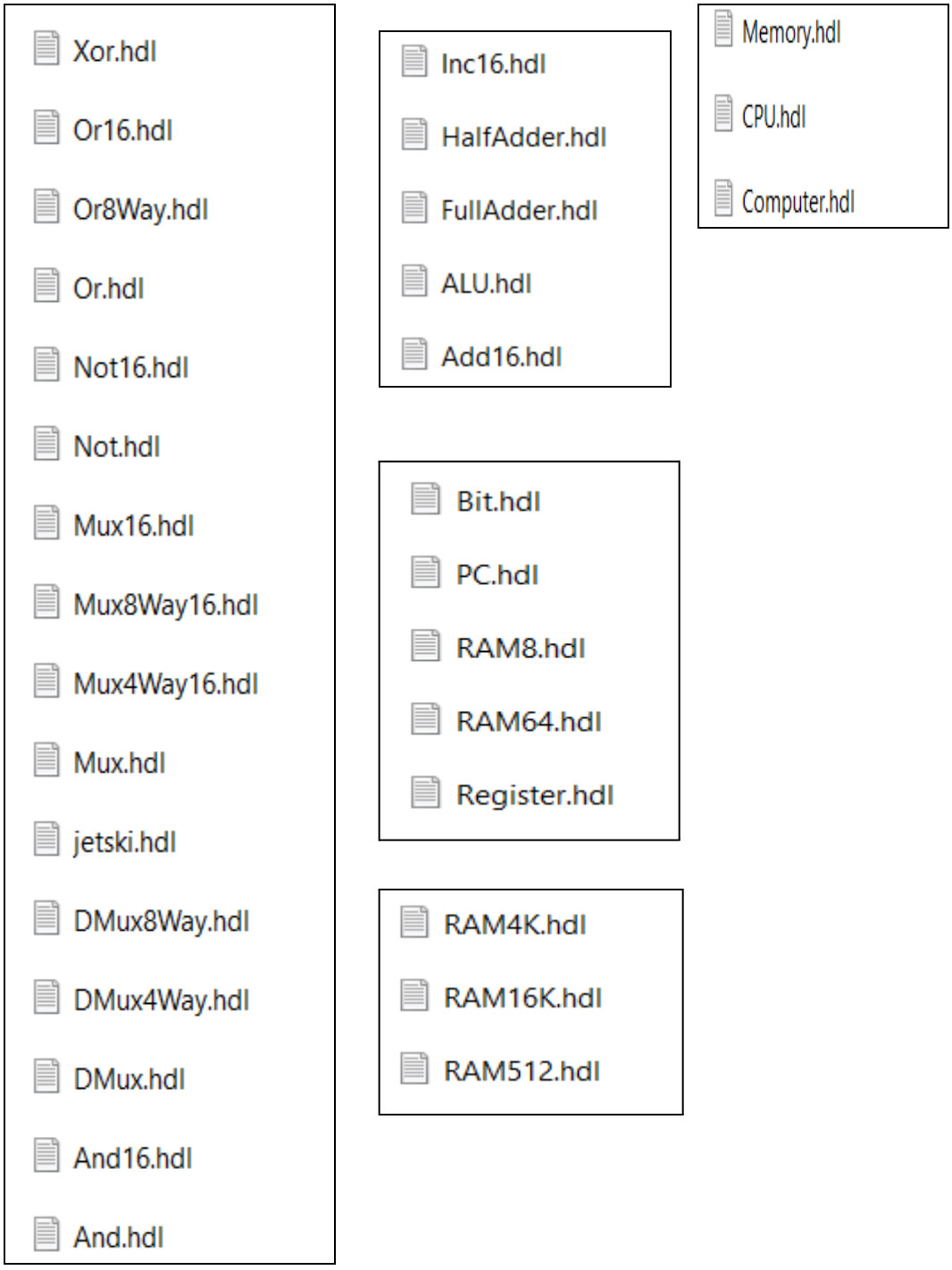
1.Instruction types to inform our CPU design

- *A-instructions :-*
  - Need to be able to load values specified with the @value syntax
- *C-instructions:-*
  - Need to be able to perform different computations
  - Need to be able to store the result of those computations in different destination
- *Flow Control:-*
  - Need to keep track of our current address and know what address to execute next

2.Chips to be used

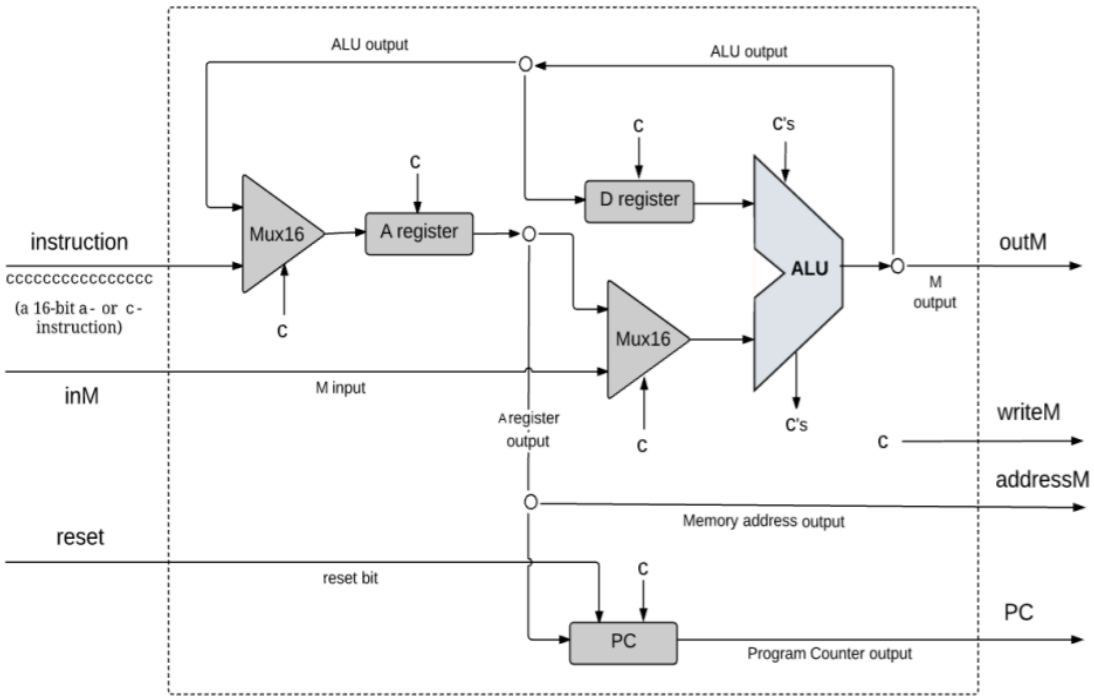
- *A Register:* For storing values in our A register
- *D Register:* For storing values in our D register
- *ALU:* For performing computations
- *Program Counter (PC):* For keeping track of flow control

3.*Chips implemented to built it in Nand2Tetris*

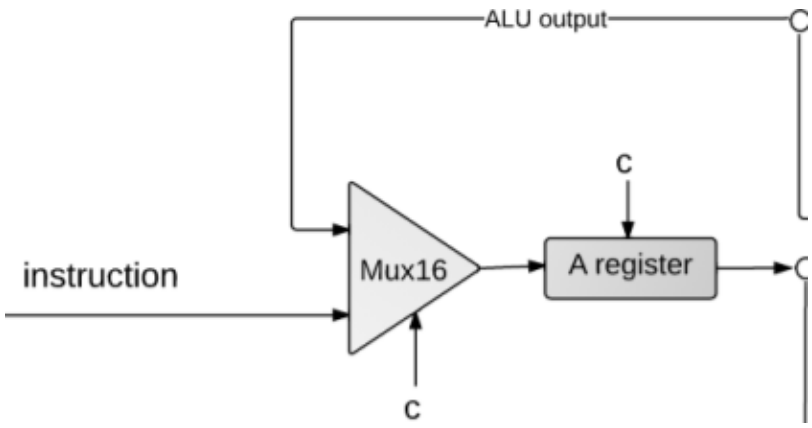


B..Block Diagram Used And explanation:-

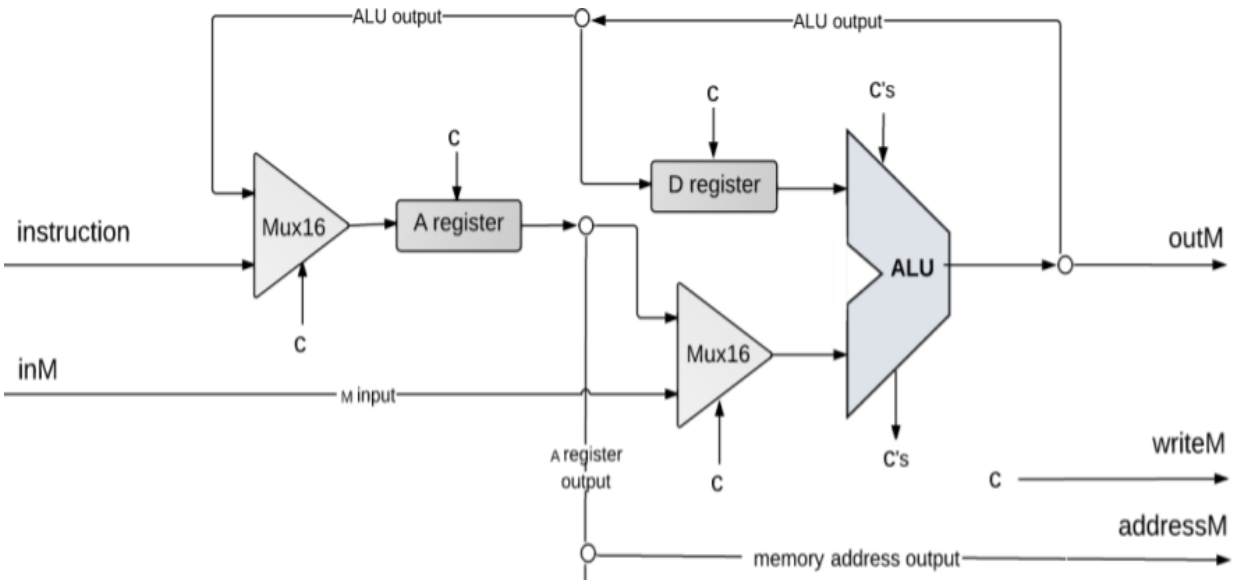
➢ *HIGH LEVEL DIAGRAM*



➢ *A-instruction Design*



➢ *C instruction design:-*



C. NAND2TETRIS code for CPU

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/05/CPU.hdl
/**
 * The Hack Central Processing unit (CPU).
 * Parses the binary code in the instruction input and executes it according to the
 * Hack machine language specification. In the case of a C-instruction, computes the
 * function specified by the instruction. If the instruction specifies to read a memory
 * value, the inM input is expected to contain this value. If the instruction specifies
 * to write a value to the memory, sets the outM output to this value, sets the addressM
 * output to the target address, and asserts the writeM output (when writeM == 0, any
 * value may appear in outM).
 * If the reset input is 0, computes the address of the next instruction and sets the
 * pc output to that value. If the reset input is 1, sets pc to 0.
 * Note: The outM and writeM outputs are combinational: they are affected by the
 * instruction's execution during the current cycle. The addressM and pc outputs are
 * clocked: although they are affected by the instruction's execution, they commit to
 * their new values only in the next cycle.
 */
```

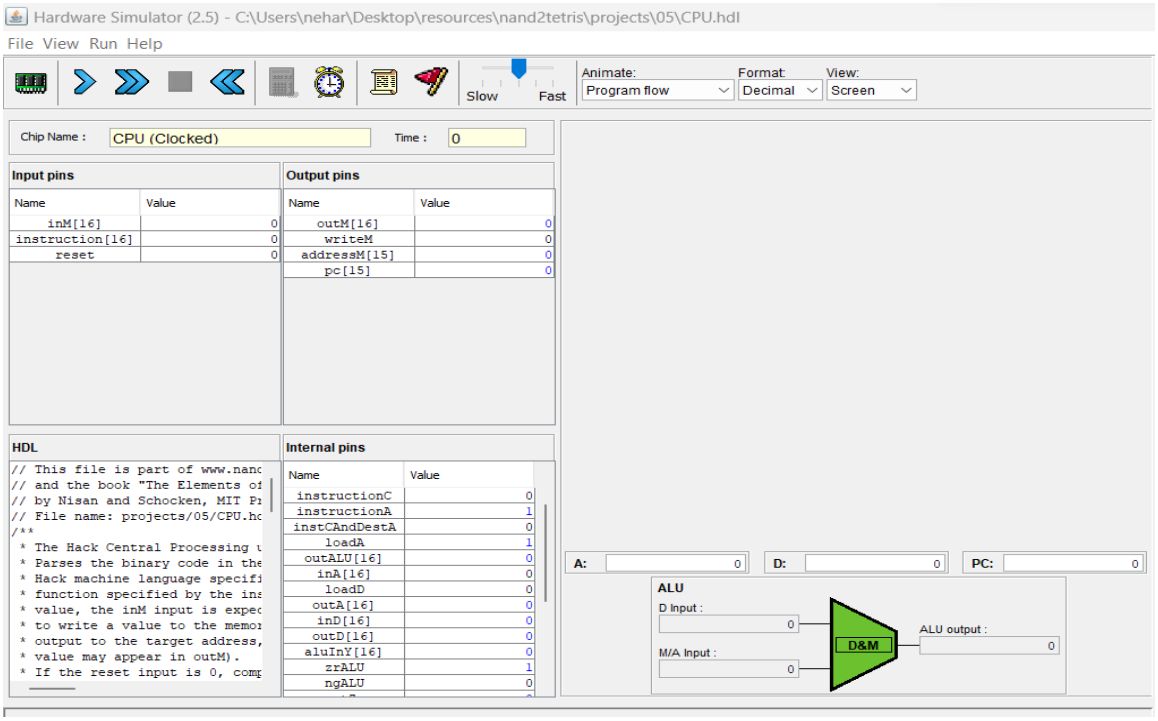
```
CHIP CPU {

    IN  inM[16],         // M value input  (M = contents of RAM[A])
        instruction[16], // Instruction for execution
        reset;           // Signals whether to restart 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:
    And(a=instruction[15], b=true, out=instructionC);
    Not(in=instructionC, out=instructionA);

    // Calculate inA and loadA
    And(a=instructionC, b=instruction[5], out=instCAndDestA);
    Or(a=instructionA, b=instCAndDestA, out=loadA);
    Mux16(a=outALU, b=instruction, sel=instructionA, out=inA);

    // Calculate loadD
    And(a=instructionC, b=instruction[4], out=loadD);

    // A and D registers
    ARegister(in=inA, load=loadA, out=outA, out[0..14]=addressM);
    DRegister(in=inD, load=loadD, out=outD);

    // Calculate y input of ALU
    Mux16(a=outA, b=inM, sel=instruction[12], out=aluInY);
    ALU(x=outD, y=aluInY, zx=instruction[11], nx=instruction[10], zy=instruction[9],
ny=instruction[8], f=instruction[7], no=instruction[6],
            out=outALU, out=inD, out=outM, zr=zrALU, ng=ngALU);

    // Calculate if output of ALU is positive
    Not(in=zrALU, out=notZr);
    Not(in=ngALU, out=notNg);
    And(a=notZr, b=notNg, out=psALU);

    // Calculate loadPC
    And(a=instruction[1], b=zrALU, out=zrMatched);
    And(a=instruction[2], b=ngALU, out=ngMatched);
```

```
    And(a=instruction[0], b=psALU, out=psMatched);

    And(a=instructionC, b=zrMatched, out=jumpZR);
    And(a=instructionC, b=ngMatched, out=jumpNG);
    And(a=instructionC, b=psMatched, out=jumpPS);
    Or(a=jumpZR, b=jumpNG, out=jumpZROrNG);
    Or(a=jumpZROrNG, b=jumpPS, out=loadPC);

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

    // Calculate writeM
    And(a=instructionC, b=instruction[3], out=writeM);
}
```

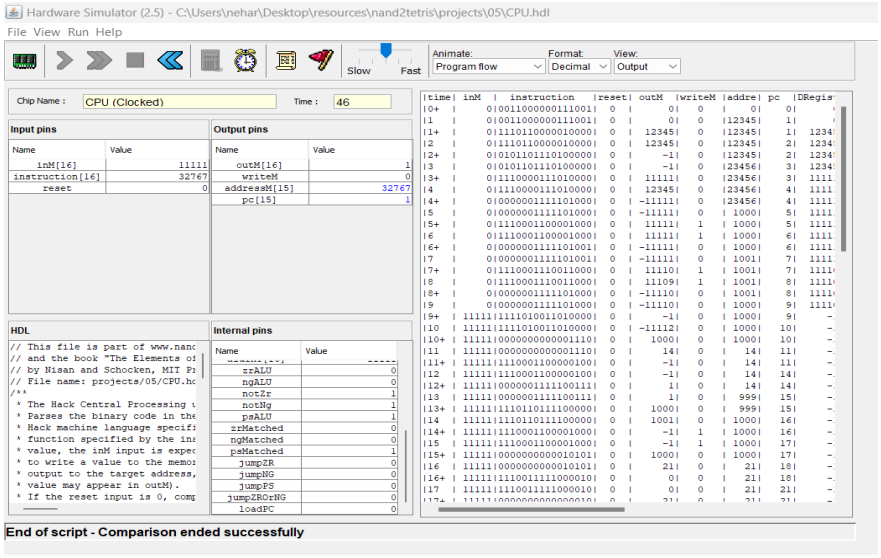
IV.RESULT

A.NAND2TETRIS OUTPUT

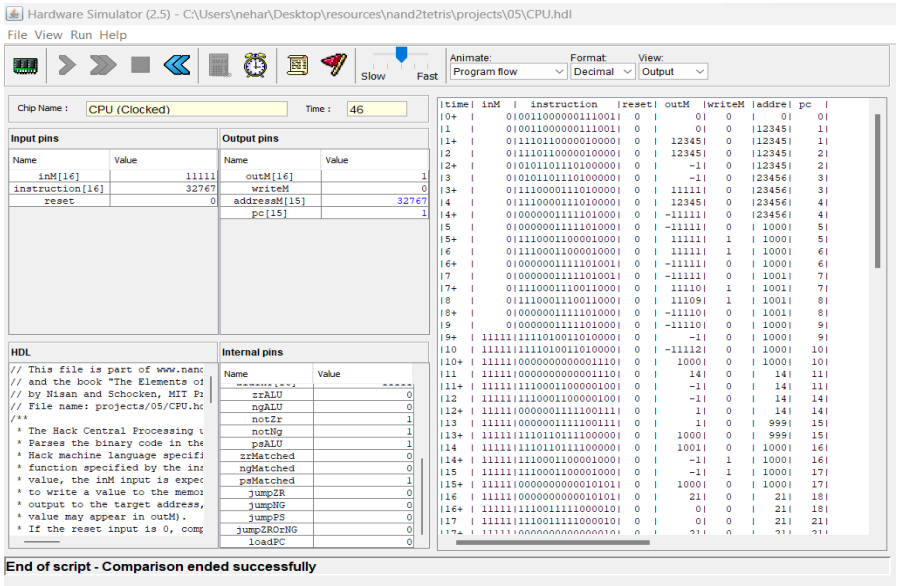
- ➢ *CPU OUTPUT*
  - *Step 1:-Loading chip*



- *Step 2:Comparison with test file*



- *Step 3:Comparison with test file(cpu ext)*



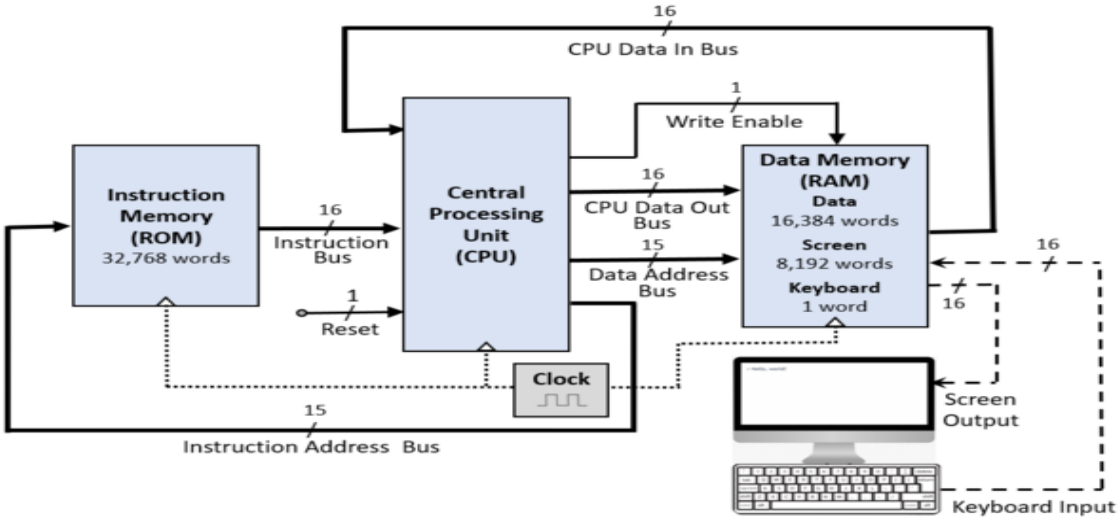
V.DEVELOPMENTS:-

As an attempt to take our project one step ahead ,we used our newly built and implemented 16-Bit HACK CPU to Design an Hack Computer using the NAND2TETRIS

A.Theory

- **The Hack Computer**: The Hack Computer is a theoretical computer design created by Noam Nisan and Shimon Schocken and described in their book "The Elements of Computing Systems: Building a Modern Computer from First Principles". It consists of a central processing unit (CPU), a memory unit, and an input/output unit.
- **The Purpose of the Hack Computer**: The Hack Computer is intended for hands-on virtual construction in a hardware simulator application as a part of a basic, but comprehensive, course in computer organization and architecture.
- **The Structure of the Hack Computer**: The Hack Computer has a 16-bit word size and a 32K-word memory. It has two registers: A and D, and an instruction memory that holds the program. The CPU executes two types of instructions: A-instructions and C-instructions. A-instructions load a constant value into the A register. C-instructions perform computations and jumps based on the values of the A and D registers and the memory. The Hack Computer also has a screen memory that displays black-and-white graphics on a 256 x 512 pixel screen, and a keyboard memory that receives input from a standard keyboard.

B.Block diagram

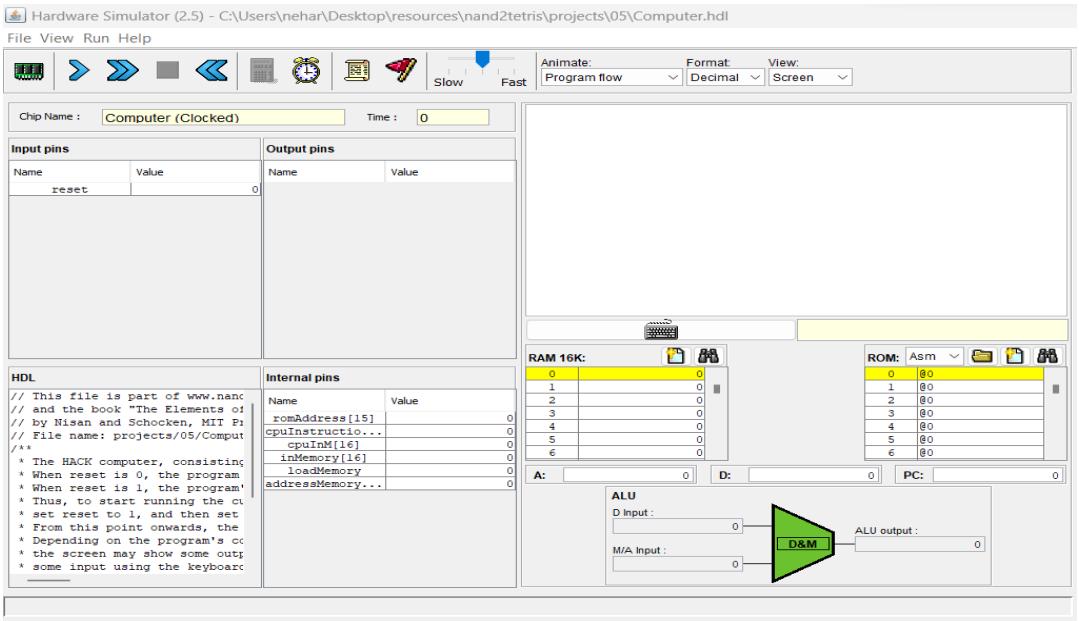


C.Code Used

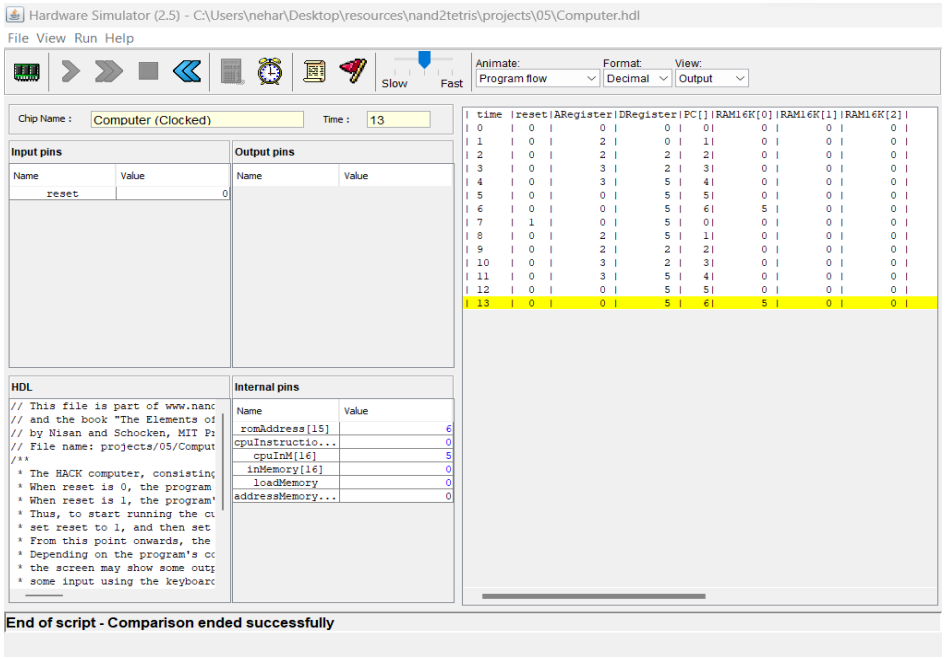
```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/05/Computer.hdl
/**
 * The HACK computer, consisting of CPU, ROM and RAM.
 * When reset is 0, the program stored in the ROM executes.
 * When reset is 1, the program's execution restarts.
 * Thus, to start running the currently loaded program,
 * set reset to 1, and then set it to 0.
 * From this point onwards, the user is at the mercy of the software:
 * Depending on the program's code, and whether the code is correct,
 * the screen may show some output, the user may be expected to enter
 * some input using the keyboard, or the program may do some procerssing.
 */
CHIP Computer {
    IN reset;

    PARTS:
    ROM32K(address=romAddress, out=cpuInstruction);
    CPU(inM=cpuInM, instruction=cpuInstruction, reset=reset, outM=inMemory, writeM=loadMemory, addressM=addressMemory, pc=romAddress);
    Memory(in=inMemory, load=loadMemory, address=addressMemory, out=cpuInM);
}
```

D.Testing and Output

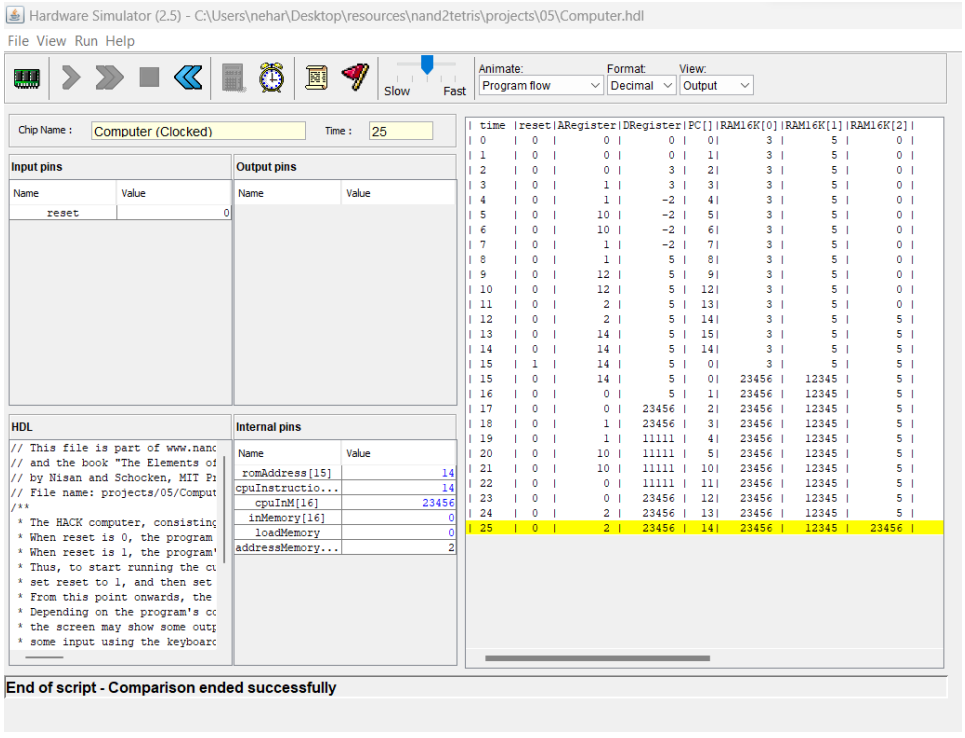
- *Step 1:-Load chip*



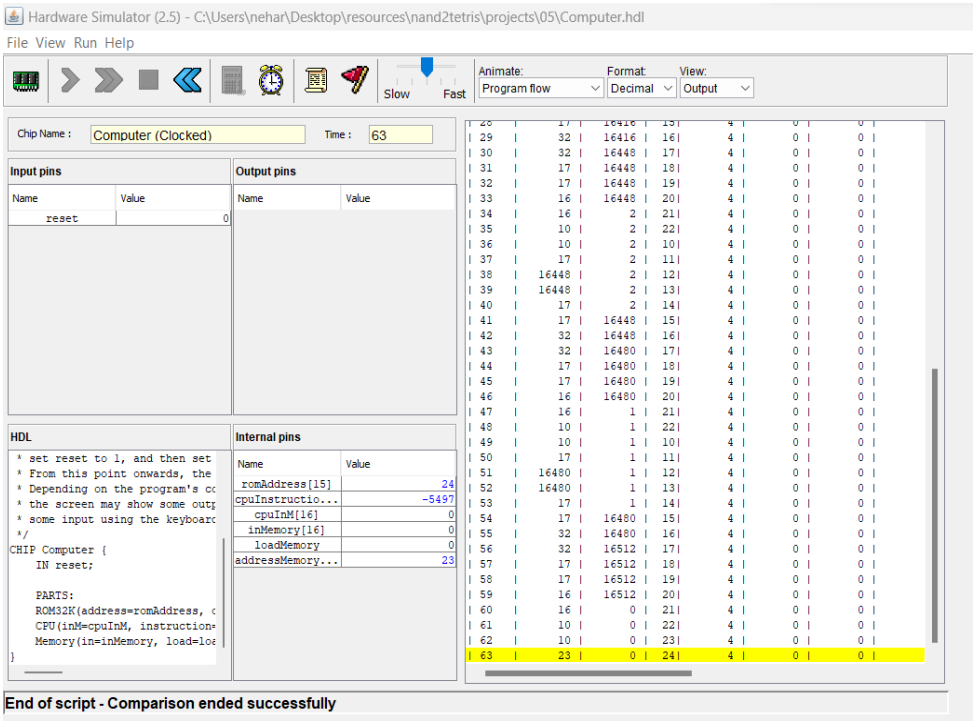
- *Step 2:-Compare with computerAdd.tst*



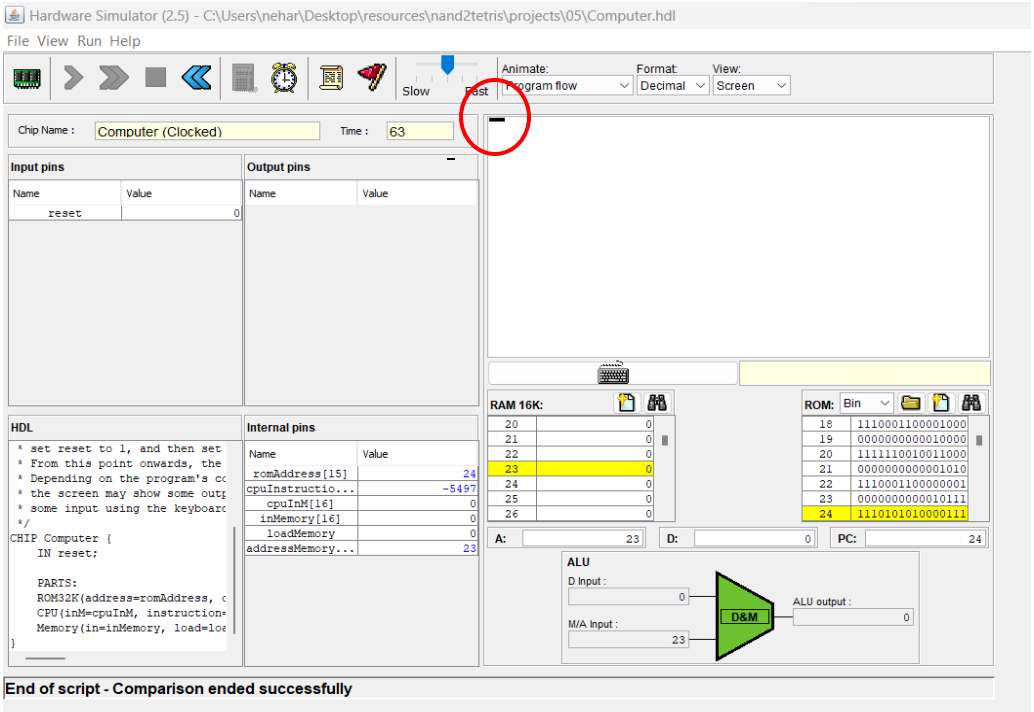
- *Step 3:Compare with ComputaerMax.tst*



- *Step 4:Compare with ComputerRect.tst*

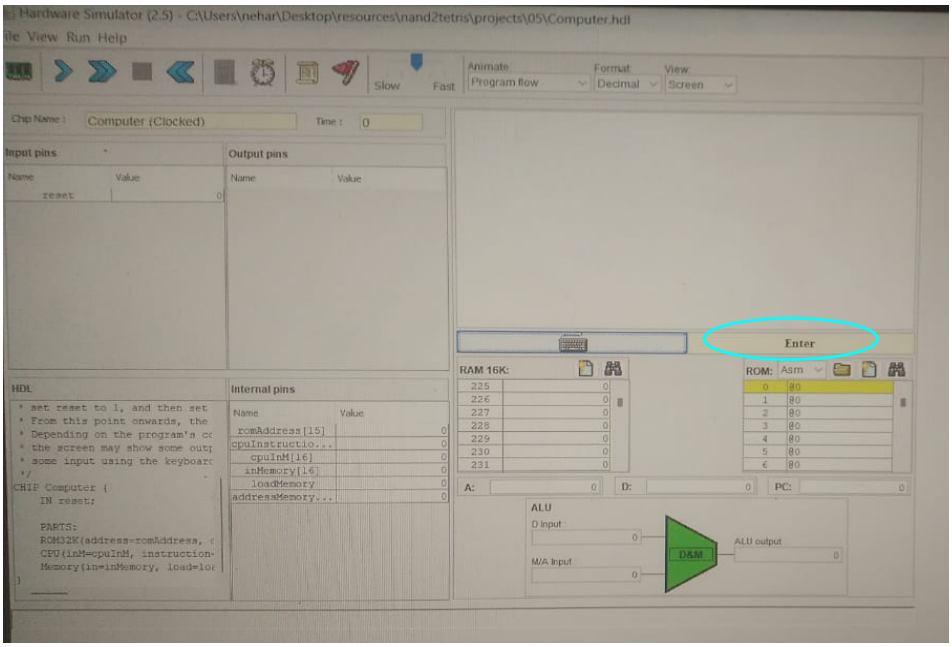


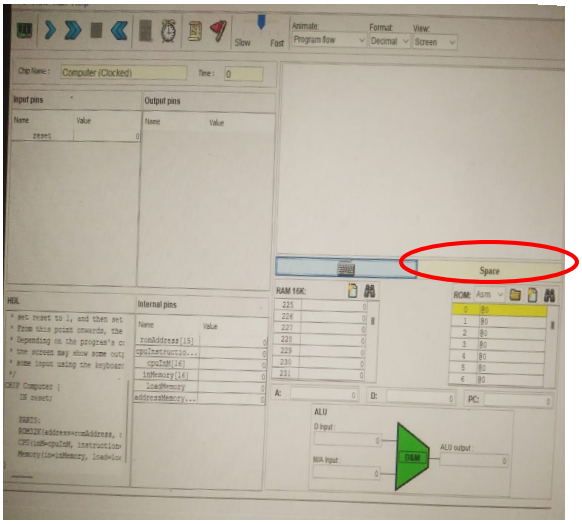
Observation: A small rectangle on top left hand side

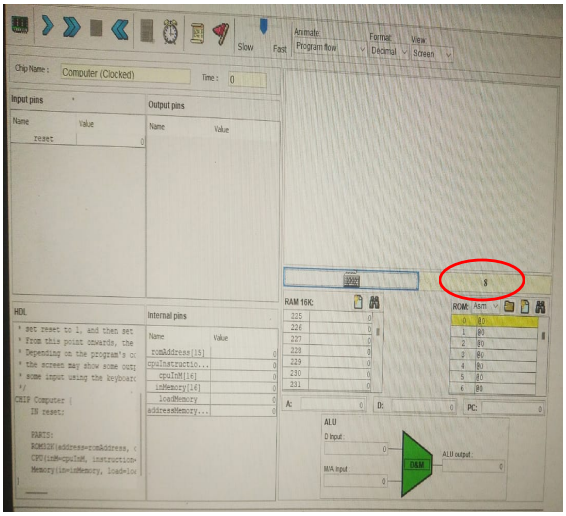


- *Step 5:Testing keyboard*

  Corresponding keys identified successfully







IV. CONCLUSION:-

The design and implementation of the Hack CPU combined with the construction of the Hack computer stands as a milestone in the computer and electrical field .The meticulous planning and recursive development process involved in creating each of these components showcase the minute yet complex juggle and balance between hardware and software. Nand2Tetris, with its emphasis on building a computer from the very basics using a basic NAND gate, served as an efficient and helpful educational tool. The hands-on experience of constructing a complete Hack computer, from individual logic gates to a functional computing system, offers a unique perspective on the inner workings of computers.

The clear and elegant design principles of the Hack architecture not only provide a unique and helpful platform for understanding fundamental concepts but also empower learners to appreciate the complete and universal nature of computer systems. The Hack CPU and computer represent a bridge between theory and practical, offering young learners a comprehensive understanding of computer architecture. Through this study , we gained insights into the fascinating details of digital design and programming, fostering a deep appreciation for the balance between hardware and software. In essence, the Hack CPU and computer, within the context of Nand2Tetris, exemplify the transformative potential of experiential learning in the field of computer science.

PART B

Design and implement a digital system that takes Gray code output from a rotary encoder and display the rotary shaft's position on a 7 segment display

Design and implement a digital system that takes Gray code output from a rotary encoder and display the rotary shaft's position on a 7 segment display

Abstract:

The basic idea of this project is to take in gray code which is the output of the Rotary encoder to display the shaft's position in a 7 segment display. The steps involve converting the gray code to binary code and then converting it to decimal and finally into a 7 segment display. The key components are gray to binary converter,binary to decimal converter and seven segment interface.

A microcontroller can be used to simplify the process as a microcontroller can handle both gray to binary and binary to decimal conversions. The digital system's primary objective is to provide a reliable and precise representation of the angular position of the shaft.

To validate the working of the system rigorous testing and debugging are done. This will ensure that the seven segment display accurately reflects the position of the rotary encoder. Another most important process is to carefully consider the connections between the components. This is very much needed so that our system works. The gray code output from the rotary encoder is linked to the gray to binary converter and it is connected to the binary to decimal converter. The binary to decimal converter is connected to a seven segment display.

Logic gates play a very important role in these conversions. Additionally through the development of this system a comprehensive understanding of rotary encoders, gray code to binary conversions, Binary to decimal conversions and also decimal to seven segment display.

.Introduction

A rotary encoder is an electro-mechanical device that converts the angular position or rotation of a shaft into an electrical signal. It provides feedback as if a mechanical part is moving in accordance to the command given to it. The shaft would be on a motor and would read the rotation or angular position of a dial, knob or any other electric instruments.

A seven segment display is an output display device that displays the information in the form of images or texts or decimal numbers. It consists of seven LEDs which are assembled like the number ‘8’. It is used in digital clocks, calculators, electronic meters etc.

The output of a rotary encoder is often in gray code where two successive values differ only in one bit. To convert the gray code into a seven segment display we need to first convert it to binary and decimal then to seven segment display.

The motivation behind this project is the increase in systems which are capable of monitoring and controlling the rotational components. Rotary encoders are known for their precision and reliability. It captures the angular displacement using Gray code.

If we integrate a microcontroller into this, it will provide better adaptability and make it handle several encoder specifications. A seven segment translates the rotation into a code which can easily be understood by humans and machines.

As the technology has changed drastically the need for precise monitoring is very much essential. The digital system addresses the need and provides better adaptability which can help us in integrating it in the fields of robotics, industrial automation etc.

The project dives into digital logical circuits, principles of gray to binary and binary to decimal and seven segment interfacing. Through repeated monitoring and validation, the accuracy and reliability can be measured.

Methodology

The steps are actually simple, the process involves converting gray code to binary and then binary to decimal and finally seven segment display. Let us understand this with the help of a truth table. The following gray code is the gray code representation of the binary numbers 0 to 9.

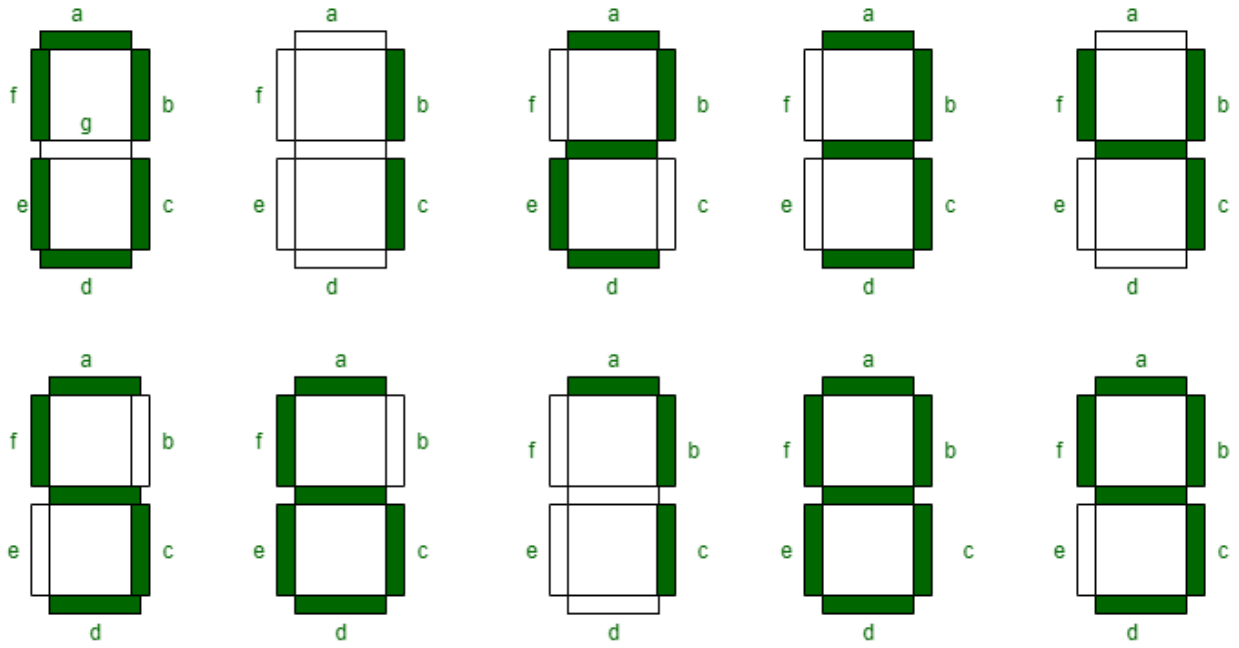
Gray code	Binary
0000	0000
0001	0001
0011	0010
0010	0011
0110	0101
0111	0100
0101	0111
0100	0110
1100	1010
1101	1011

Thus our first step of converting from gray to binary is over. Next we are supposed to convert the binary numbers we have obtained into decimal.

Let us make a truth table for binary to decimal conversion

Binary	Decimal
0000	0
0001	1
0010	2
0011	3
0101	5
0100	4
0111	7
0110	6
1010	10
1011	11

Seven segment display is an electric component that consists of seven individual LEDs arranged in the form of the number 8 labeled from a to g. The primary function of the seven segment display is to visually represent the decimal numbers from 0 to 9 by illuminating the seven segments



The above given picture has 7 segments, let us understand it.

Segment a= Topmost horizontal segment

Segment b= Top right vertical segment

Segment c= Bottom right vertical segment

Segment d= Lowermost horizontal segment

Segment e= Bottom left vertical segment

Segment f= Top left vertical segment

Segment g= Central horizontal segment

With this knowledge, we are converting the decimal code to seven segment display and Let us compare the original gray code and the seven segment display and then make k map

Decimal	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
5	0	1	1	0	0	1	1
4	1	0	1	1	0	1	1
7	1	0	1	1	1	1	1
6	1	1	1	0	0	0	0
10	1	1	1	1	1	1	1
11	1	1	1	1	0	1	1

Consider each digit of Gray code as A,B,C,D respectively

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	1	1	1	0	1	1	0	1
0	0	1	0	1	1	1	1	0	0	1
0	1	1	0	0	1	1	0	0	1	1
0	1	1	1	1	0	1	1	0	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	0	0	1	1	1	0	0	0	0
1	1	0	0	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	0	1	1

Now considering Kmap

ABCD	00	01	11	10
00	1	0	1	1
01	1	1	1	0
11	1	1	X	X
10	X	X	X	X

a= BC'+B'D'+BC

ABCD	00	01	11	10
00	1	1	1	1
01	1	0	0	1
11	1	1	X	X
10	X	X	X	X

b=D'+B'+A

ABCD	00	01	11	10
00	1	1	0	1
01	1	1	1	1
11	1	1	X	X
10	X	X	X	X

c=C’+D’+B

ABCD	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	1	1	X	X
10	X	X	X	X

d= AB+BD+CD+B’D’

ABCD	00	01	11	10
00	1	0	1	0
01	0	1	0	0
11	1	0	X	X
10	X	X	X	X

e= A’BC’D+AD’+BCD+B’C’D’

ABCD	00	01	11	10
00	1	0	0	0
01	0	1	1	1
11	1	1	X	X
10	X	X	X	X

f= BD+BC+A+B’C’D’

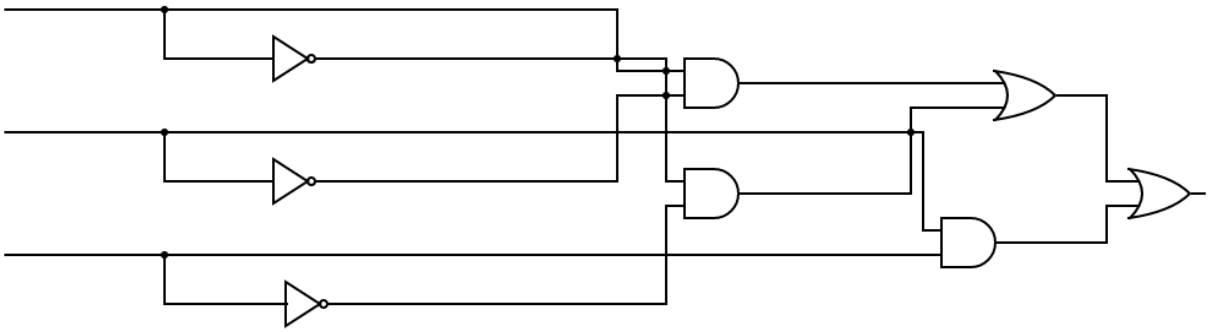
ABCD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	1	1	X	X
10	X	X	X	X

Circuit diagram

For segment ‘a’

The three major lines represent B,C,D

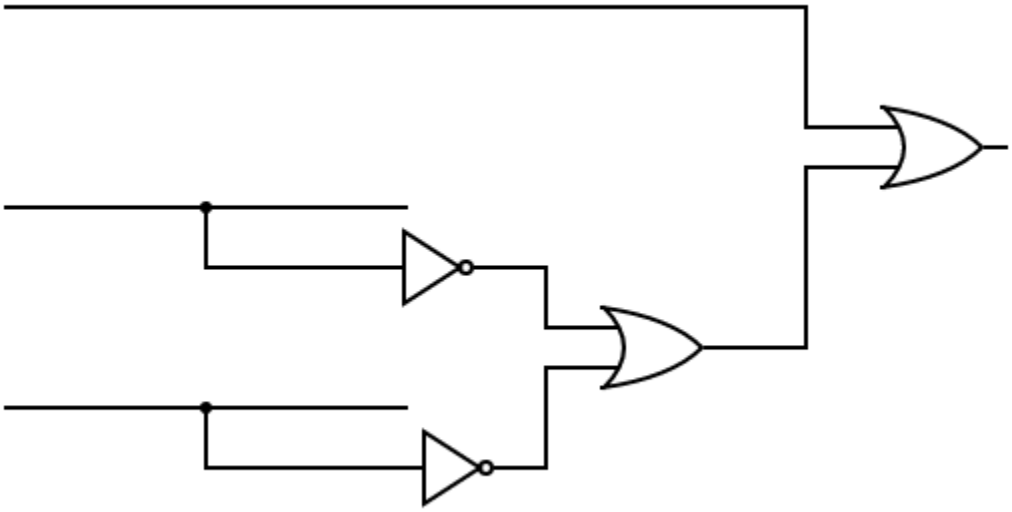
a= BC’+B’D’+CD



For segment ‘b’

The three major lines represents A,B,D

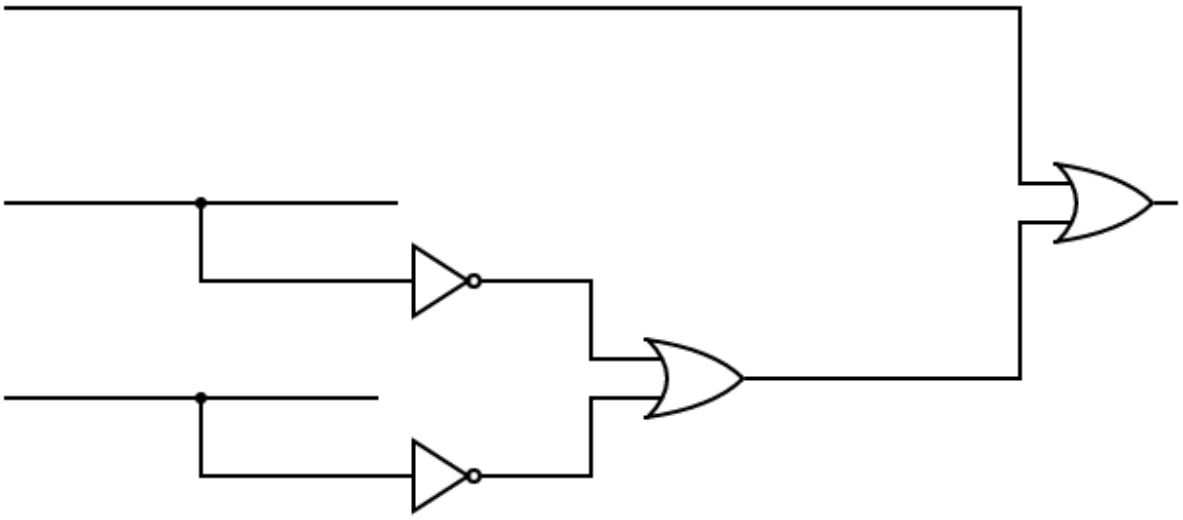
b= C’+D’+B



For segment ‘c’.

The three major lines represents B,C,D.

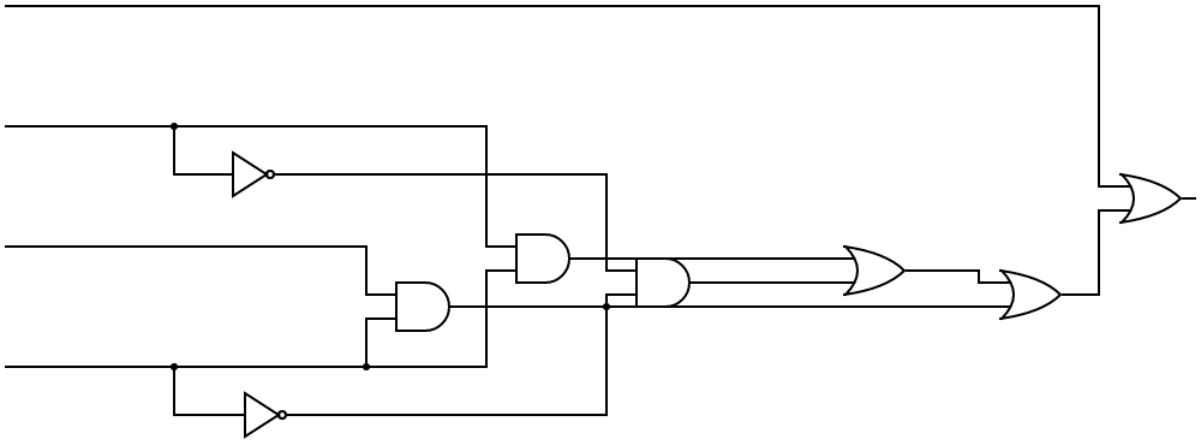
c= C’+D’+B



For segment d

The three major lines represent A,B,C,D.

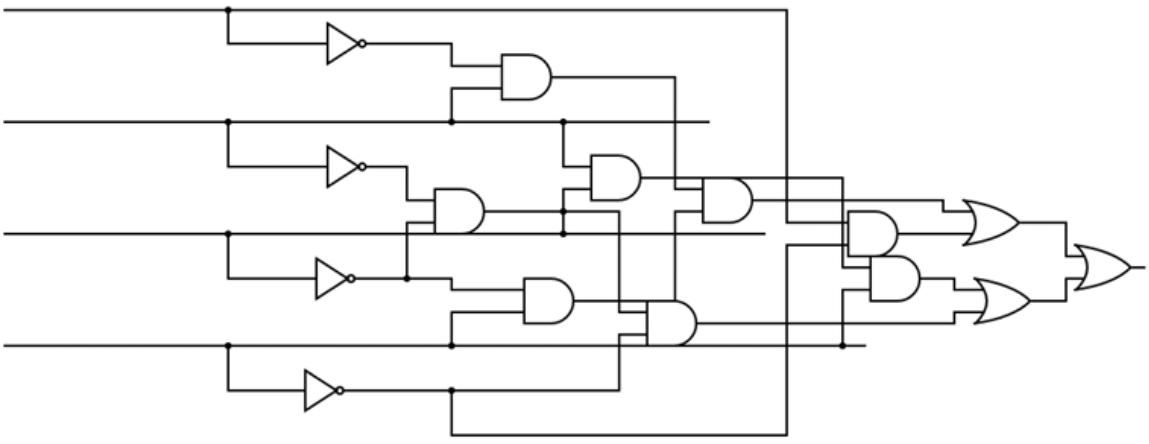
d= A+BD+CD+B’D’



For segment ‘e’:

The four major lines are A,B,C,D

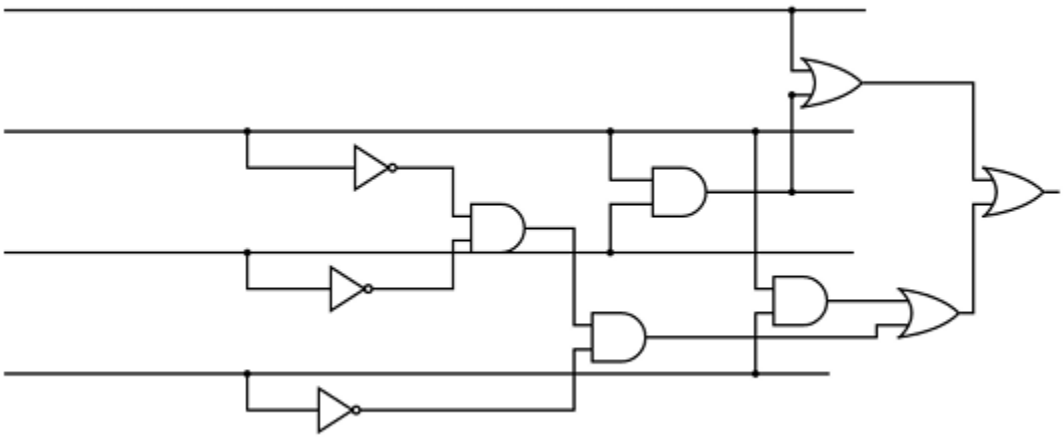
e= A’BC’D+AD’+BCD+B’C’D’



For segment ‘f’

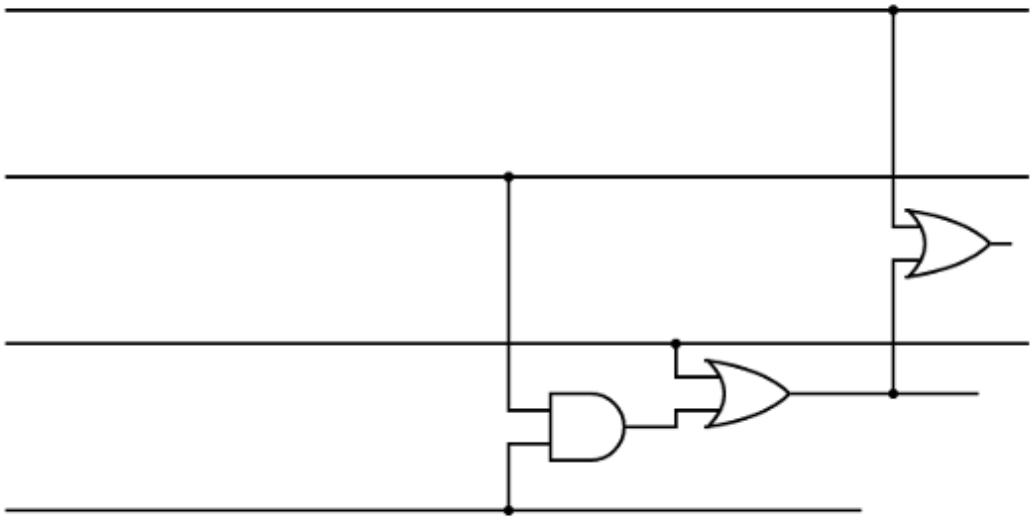
The major lines are A,B,C,D.

f= BD+BC+A+B’C’D



For segment ‘g’

The major lines are A,B,C,D
g= BD+A+C

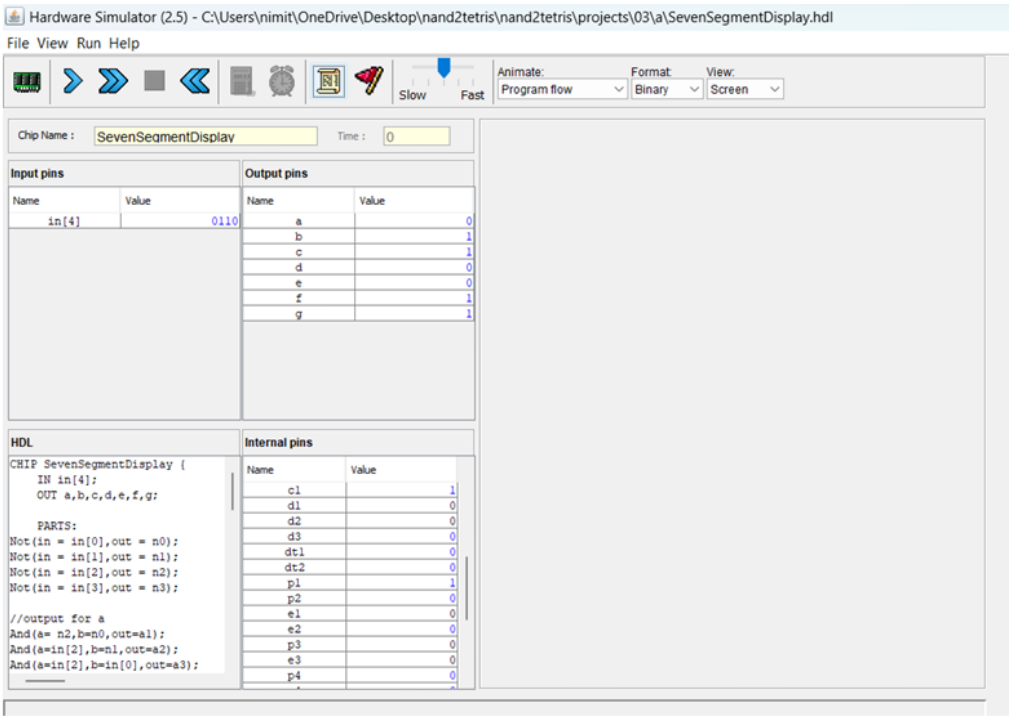


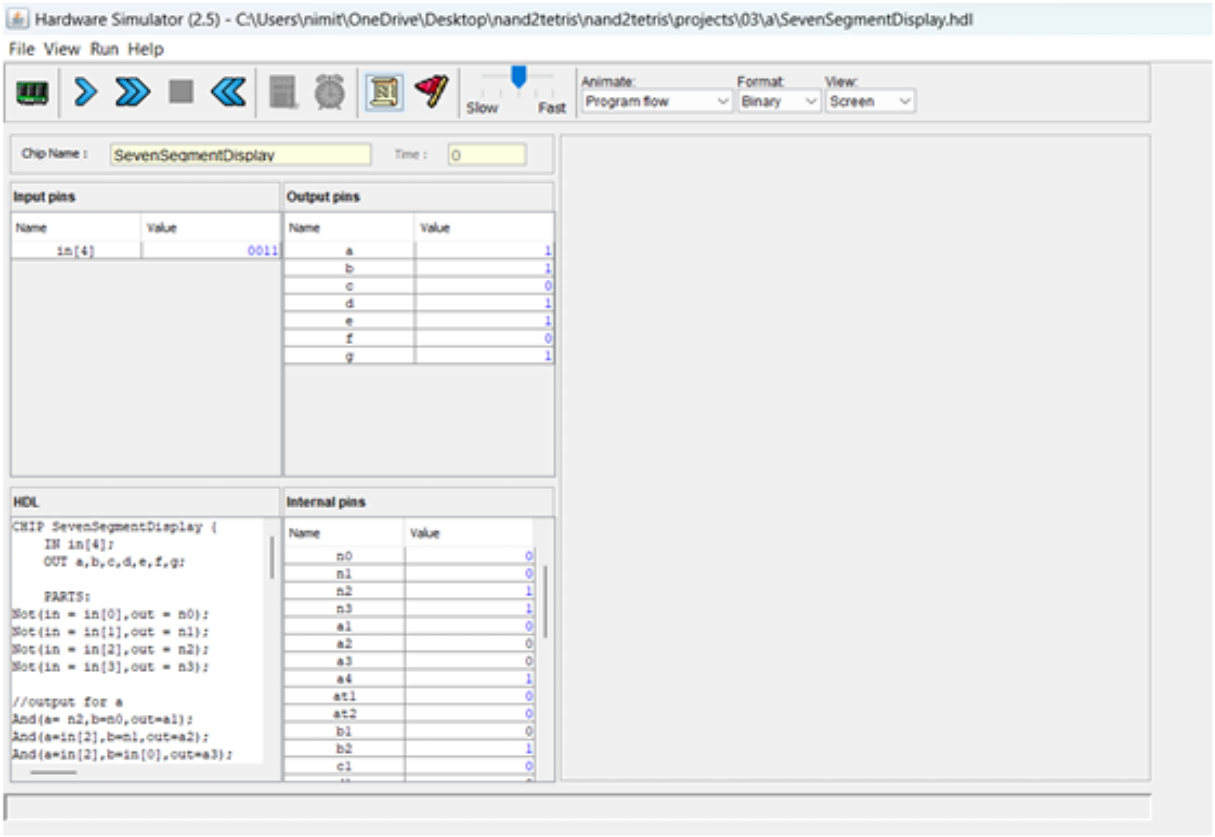
NAND2TETRIS CODE :-

```
CHIP SevenSegmentDisplay
{
IN in[4];
OUTa,b,c,d,e,f,g;

PARTS:

Not(in = in[0],out = n0);
Not(in = in[1],out = n1);
Not(in = in[2],out = n2);
Not(in = in[3],out = n3);

 //output for a
And(a= n2,b=n0,out=a1);
And(a=in[2],b=n1,out=a2);
And(a=in[2],b=in[0],out=a3);
And(a=n2,b=in[1],out=a4);
Or(a=a1,b=a2,out=at1);
 Or(a=at1,b=a3,out=at2);
Or(a=at2,b=a4,out=a);

 //Output for b
And(a=in[2],b=n0,out=b1);
 Or(a=n2,b=in[3],out=b2);
Or(a=b2,b=b1,out=b);

//Output for c
 Or(a=in[2],b=n1,out=c1);
Or(a=c1,b=n0,out=c);

//Output for d
And(a=in[2],b=in[0],out=d1);
And(a=in[1],b=in[0],out=d2);
And(a=n2,b=n0,out=d3);
Or(a=in[3],b=d1,out=dt1);
Or(a=dt1,b=d2,out=dt2);
Or(a=dt2,b=d3,out=d);

//Output for e
And(a=n3,b=in[2],out=p1);
And(a=p1,b=n1,out=p2);
And(a=p2,b=in[0],out=e1);
And(a=in[3],b=n0,out=e2);
```

```
And(a=n2,b=in[1],out=p3);
And(a=p3,b=in[0],out=e3);
And(a=n2,b=n1,out=p4);
And(a=p4,b=n0,out=e4);
Or(a=e1,b=e2,out=et1);
Or(a=et1,b=e3,out=et2);
 Or(a=et2,b=e4,out=e);

 //Output for f
And(a=n2,b=n1,out=u1);
And(a=u1,b=n0,out=f1);
And(a=in[2],b=in[0],out=f2);
And(a=in[2],b=in[1],out=f3);
Or(a=in[3],b=f1,out=ft1);
Or(a=ft1,b=f2,out=ft2);
Or(a=ft2,b=f3,out=f);

//Output for g
And(a=in[2],b=in[0],out=g1);
 Or(a=in[3],b=in[1],out= g2);
Or(a=g2,b=g1,out=g);
}
```

OUTPUT




Hardware Simulator (2.5) - C:\Users\nimit\OneDrive\Desktop\nand2tetris\nand2tetris\projects\03\a\SevenSegmentDisplay.hdl
File View Run Help
Slow
Fast
Animate:
Program flow
Format:
Binary
View:
Screen
Chip Name :
SevenSegmentDisplay
Time :
0
Input pins
Name
Value
in[4]
0011
Output pins
Name
Value
a 1
b 1
c 0
d 1
e 1
f 0
g 1
HDL
CHIP SevenSegmentDisplay {
IN in[4];
OUT a,b,c,d,e,f,g;
PARTS:
Not(in = in[0],out = n0);
Not(in = in[1],out = n1);
Not(in = in[2],out = n2);
Not(in = in[3],out = n3);
//output for a
And(a= n2,b=n0,out=a1);
And(a=in[2],b=n1,out=a2);
And(a=in[2],b=in[0],out=a3);
Internal pins
Name
Value
n0 0
n1 0
n2 1
n3 1
a1 0
a2 0
a3 0
a4 1
at1 0
at2 0
b1 0
b2 1
c1 0