“FROM NAND TO TETRIS”

PROJECT REPORT

Submitted for CAL in B.Tech Operating Systems (CSE2005)

By

G. MUKKESH – 17BCE1128

C. MAHESHVAR - 17BCE1172

Slot: A1

Name of faculty: PROF. RENUKA DEVI

(SCHOOL OF COMPUTER SCIENCE ENGINEERING: SCSE)

1) ABSTRACT

From Nand to Tetris is a project oriented online course taught by professors Shimon Schocken and Noam Nisan from the University of Jerusalem. It makes students build a modern computer system from ground up. This course is divided into two and both the parts have six projects in it which takes students from building logic gates on a hardware simulated environment to creating a fully functional general purpose machine

2) INTRODUCTION

Building a computer begins at the hardware level and building a platform for software to work on. This works with multiple levels of abstraction. The course offers students to build a computer using rudimentary steps. All the projects use the working of the previous ones. The bottom up approach makes every project easily implementable. Test scripts are given to test our work in software before submitting the code online.

3) MODULE DESCRIPTION

In the first part of Nand to Tetris, the course begins with the instructors given us a Nand gate and making us build other logic gates and latches using a hardware simulated software. The code is implemented by specifying what gets inputted into the gate and what exits the gate depending on the operation being done inside the gate. An ALU was built using these gates with predefined specifications given by the course. The RAM of the computer was built using registers combined together with MUX and DEMUX gates. The CPU was finally made by using the ALU, program counter and memory together to operate on binary numbers input in it. The course ends with introduction to a machine level coding language called hack and running basic programs in it. Finally, an assembler was built to convert the language into binary codes which can be input into the system. This course also talks about how the keyboard and screen are interfaced to the lower

In the second part of Nand to Tetris, starts with learning a Virtual Machine language and building a compiler to convert it into hack. This brings us to the usage of stacks in the computer. Process switching was implemented by saving the contents of variables in a fixed location and moving the stack pointer further. Then, the high level language which is used to build the operating system and to run programs in the machine, called jack was taught. This language is similar to Java. This succeeded with building a compiler for converting jack programs to Virtual Machine Code .Finally the course ends with building an operating system for the machine. The Operating System offers 8 classes whose making offers structure to the computer and makes it run. It is written in the jack programming language. Operations such as memory allocation, mathematical functions with efficient algorithms, building shapes in screen, getting input from the screen etc were made in this module.

4) HARDWARE AND SOFTWARE REQUIREMENT

The software were provided by the course and are listen below:

- Assembler
- CPU Emulator
- Hardware Simulator
- Jack Compiler
- Text Comparator
- VMEmulator

5) SAMPLE CODING

Below is a sample program written in the jack language

```
// This file is part of the materials accompanying the book

// "The Elements of Computing Systems" by Nisan and Schocken,

// MIT Press. Book site: www.idc.ac.il/tecs

// File name: projects/12/Screen.jack


/**
```

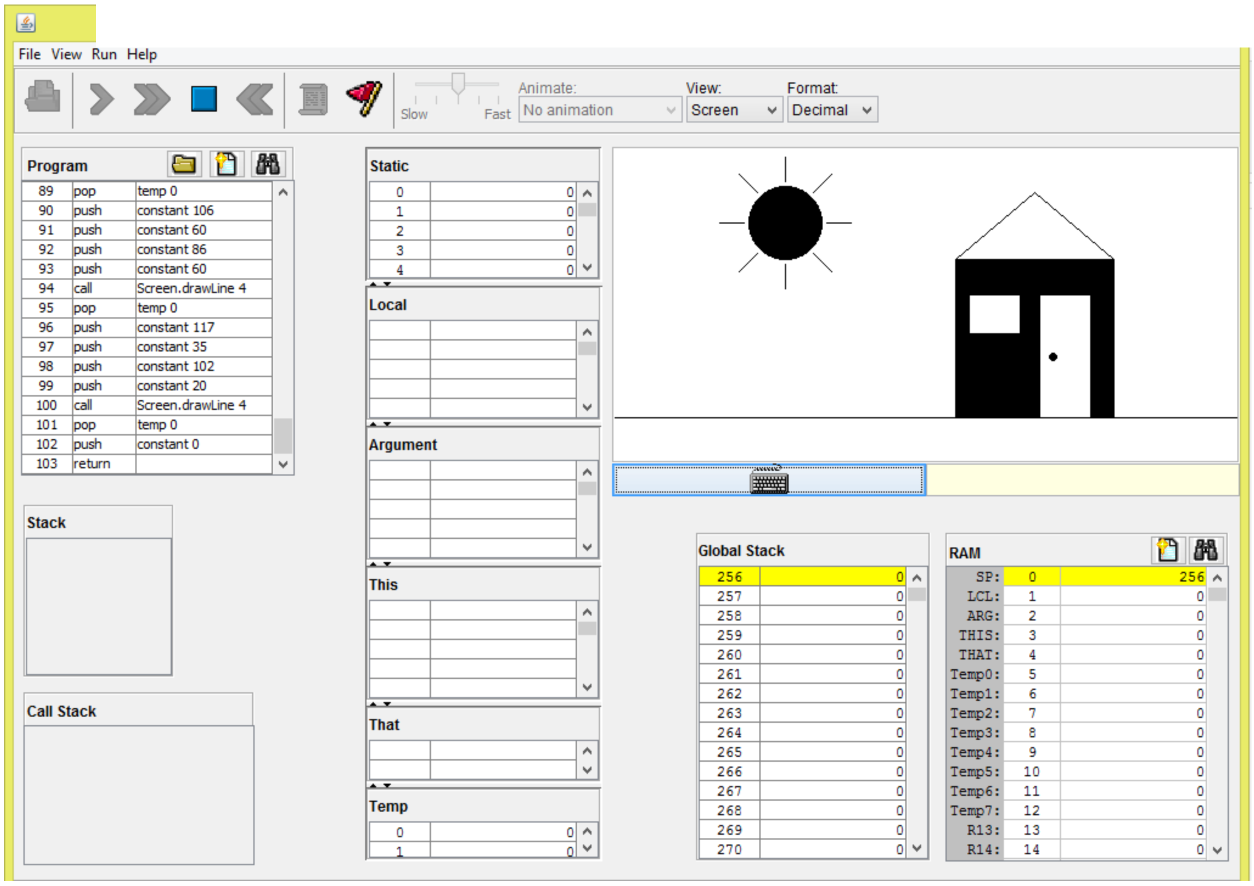
```
 * Graphic screen library.
 */
class Screen
{
    static boolean isBlack;
    static int SCREEN_ADDR;
    static int KEYBOARD_ADDR;
    static Array twoToThe;

    /** Initializes the Screen. */
    function void init()
    {
        var int i;
        var int currentVal;

        let twoToThe = Array.new(16);
        let isBlack = true;
        let SCREEN_ADDR = 16384;
        let KEYBOARD_ADDR = 24576;

        let currentVal = 1;
        let i = 0;
        while(i < 16)
        {
            let twoToThe[i] = currentVal;
            let currentVal = currentVal + currentVal;
            let i = i + 1;
        }
        return;
    }

    /** Erases the whole screen. */
    function void clearScreen()
    {
        var int i;

        let i = SCREEN_ADDR;
        while(i < KEYBOARD_ADDR)
        {
            do Memory.poke(i, 0);
            let i = i + 1;
        }
        return;
    }

    /** Sets the color to be used in further draw commands
     *  where white = false, black = true. */
    function void setColor(boolean b)
    {
        let isBlack = b;
        return;
    }
```

```
/** Draws the (x, y) pixel. */
function void drawPixel(int x, int y)
{
    var int addr;
    var int pixelValue;
    var int bitIndex;
    var int xDiv16;

    if((x < 0) | (x > 511) | (y < 0) | (y > 255))
    {
        do Sys.error(7);
    }

    let xDiv16 = x / 16;
    let bitIndex = x - (xDiv16 * 16);
    let addr = SCREEN_ADDR + ((y * 32) + xDiv16);
    let pixelValue = Memory.peek(addr);
    if(isBlack)
    {
        let pixelValue = twoToThe[bitIndex] | pixelValue;
    }
    else
    {
        let pixelValue = (~twoToThe[bitIndex]) & pixelValue;
    }
    do Memory.poke(addr, pixelValue);

    return;

}


/** Draws a line from pixel (x1, y1) to (x2, y2). */

function void drawLine(int x1, int y1, int x2, int y2)

{

    var int adyMinusbdx; //autoinitialized to 0

    var int a, b, dx, dy;

    var int xBase, xEnd;

    var int yBase, yEnd;

    let dx = x2 - x1;

    let dy = y2 - y1;


    if(dx = 0)

    {

        let yBase = y2;

        let yEnd = y1;

        if(y1 < y2)

        {

            let yBase = y1;

            let yEnd = y2;

        }

        while(~(yBase > yEnd)) //yBase <= yEnd

        {

            do Screen.drawPixel(x1, yBase);
            let yBase = yBase + 1;
        }
    }
    if(dy = 0)
    {
        let xBase = x2;
        let xEnd = x1;
        if(x1 < x2)
        {
            let xBase = x1;
            let xEnd = x2;
        }
        while(~(xBase > xEnd)) //xBase <= xEnd
        {
            do Screen.drawPixel(xBase, y1);
            let xBase = xBase + 1;
        }
    }
    while((~(a > dx)) & (~(b > dy)))
    {
        do Screen.drawPixel(x1 + a, y1 + b);
        if(adyMinusbdx < 0)
        {
            let a = a + 1;
            let adyMinusbdx = adyMinusbdx + dy;
        }
        else
        {
            let b = b + 1;
            let adyMinusbdx = adyMinusbdx - dx;
        }
    }
    return;
}

/** Draws a filled rectangle where the top left corner
 *  is (x1, y1) and the bottom right corner is (x2, y2). */
function void drawRectangle(int x1, int y1, int x2, int y2)
{
    var int yCounter;
    let yCounter = y1;
    while(yCounter < y2)
    {
        do Screen.drawLine(x1, yCounter, x2, yCounter);
        let yCounter = yCounter + 1;
    }
    return;
}

/** Draws a filled circle of radius r around (cx, cy). */
function void drawCircle(int cx, int cy, int r)
{
```

```
            return;
        }
    }
```

6) SCREENSHOTS

The output of the program given in the top is given below



Animate: No animation
View: Screen
Format: Decimal

Program

89	pop	temp 0
90	push	constant 106
91	push	constant 60
92	push	constant 86
93	push	constant 60
94	call	Screen.drawLine 4
95	pop	temp 0
96	push	constant 117
97	push	constant 35
98	push	constant 102
99	push	constant 20
100	call	Screen.drawLine 4
101	pop	temp 0
102	push	constant 0
103	return	

Static

0	0
1	0
2	0
3	0
4	0

Local

Argument

This

That

Temp

0	0
1	0

Stack

Call Stack

Global Stack

256	0
257	0
258	0
259	0
260	0
261	0
262	0
263	0
264	0
265	0
266	0
267	0
268	0
269	0
270	0

RAM

SP:	0	256
LCL:	1	0
ARG:	2	0
THIS:	3	0
THAT:	4	0
Temp0:	5	0
Temp1:	6	0
Temp2:	7	0
Temp3:	8	0
Temp4:	9	0
Temp5:	10	0
Temp6:	11	0
Temp7:	12	0
R13:	13	0
R14:	14	0

7) RESULTS AND CONCLUSIONS

Thus a fully functioning modern computer was realized using basic gates to building an operating system on top of which programs can be run in. The whole process was simplified drastically has the computer was viewed as modules in an abstracted manner.

8) REFERENCE

- **https://www.coursera.org/learn/build-a-computer**
- **https://www.coursera.org/learn/nand2tetris2**
- **https://www.nand2tetris.org/**