Name: MIDN 2/C McKenzie Eshleman

Course / Section: SY303 / 3321

Enclosures: (1) Gate Diagram for CPU chip

(2) HDL Code for CPU chip

(3) Screenshot for CPU Chip

(4) Gate Diagram for computer chip

(5) HDL Code for Computer Chip

(6) Screenshot for Computer Chip

**RUBRIC**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Section** | **0%** | **50%** | **80%** | **100%** | **Max Points** |
| Used Template | No | Partially |  | Yes | 10 |
| Introduction | Purpose and objectives unclear | Discussion lacking | Progressing | Well discussed | 5 |
| Procedure | Discussed procedure could not be reproduced | Some steps and tools missing | Progressing | Comprehensive and clear | 10 |
| Results | Not present or not discussed | Incorrect results without explanation | Progressing | Correct, well-explained results | 10 |
| Discussion | Questions not answered or answers are off-topic . | Discussions lack complete consideration. Terse responses. | Progressing | Well-developed discussions with interesting insights | 30 |
| Gate Diagrams | Sloppy, incomplete drawings | Not properly labeled, wrong gates | Progressing | Included chip name, all labels and connections correct | 10 |
| HDL Code | Missing or not commented | Incomplete, has errors, missing name or comments | Progressing | Includes identification and clear descriptions of all code | 10 |
| Simulator Screenshots | Missing | Does not capture full simulator screen or wrong chip | Progressing | All present, shows status of test script, full legible screen capture | 10 |
| Grammar/Professionalism | Poor grammar or use of slang |  | Progressing | Professional writing | 5 |

**INTRODUCTION:**

The premise of Project 5 is to build a hack computer platform. This is the seen in the culminating chip computer. The main objectives is to first build the memory chip, which was completed in lab 5. The next chip needed is the CPU chip which can execute instructions and also fetch instructions. The next objective is to build the computer chip which is the topmost chip in the hack platform.

**PROCEDURE:**

On this project, I followed along with Professor Brown’s video for the CPU chip. For the CPU chip we broke it down into eight steps. For the first step we use a Mux16 chip to determine if we want to use an A or C instruction for input into the CPU. The next two steps is using the A register and D register to input the A and D instructions. For step four we are determining if we are using an A or M instruction. For step five is inputting the c instructions into the ALU chip, using the different instructions. Step six happens when a C instruction and the D3 bit is M, then we write logic to Ram memory, this part checks to make sure the inputs are c instruction and destination memory M if both are true then it writes into memory. Step seven takes the output of the A address and converts it into a 15 bit address M. The final step is first defining the loads there are three loads that we need to define using jump functions. For the jump functions we make jumps for is the input is negative, zero, or positive. Once the different loads are defined then we have to verify if the input is a c instruction, then we put all the loads into the PC chip.

The Computer Chip is pretty simple, I based my chip off of the diagram in the lecture notes on slide 31 also in class we discussed that the chip was simple and uses the chips that we just created. We first use a ROM32K chip which was a pre-built chip, where we use the address of the PC, and output the instructions. Then we use our newly made CPU chip and set all the inputs equal to data that is shown on the gate diagram. The last portion of the computer chip is the data memory chip which was created in Lab 5, the final output it the outputted data memory.

**RESULTS:**

|  |  |
| --- | --- |
| Succeeded | Failed |
| * CPU Chip * Computer Chip |  |

**DISCUSSION (For these questions you may seek quality information from the internet or other resources.  Be sure to cite all sources used for research, give the exact URL(s) visited):**

1. **In a computer system, there are relatively few registers but many RAM locations to hold data. We have discussed that access to data already in a register is much faster. Quantify the relative difference in access time to CPU registers versus RAM in a modern device. Explain how cache memory provides a useful middle ground and why some systems even employ multiple levels of cache.**

**Cache memory is a useful middle ground for memory because it helps reduce the average cost of time and or energy to access data from the main memory. It is used to speed up and synchronizing with high speed CPU, it typically acts as a buffer between RAM and the CPU. A cache is smaller and faster memory that can store copies of data from frequently used main memory locations. Most CPU’s have a hierarchy of multiple cache levels, that have separate instruction-specific and data-specific caches.**

**https://www.geeksforgeeks.org/cache-memory-in-computer-organization/**

**https://en.wikipedia.org/wiki/CPU\_cache**

1. **Compare and contrast Reduced Instruction Set Computers (RISC) with Complex Instruction Set Computers (CISC). Which one best describes our Hack machine? What reasons would lead you to choose one paradigm over the other?**

**Reduced Instruction Set Computer (RISC) based machine executes one instruction per clock cycle. Complex Instruction Set Computer (CISC) can have special instructions as well as instructions that take more than one cycle to execute on an RISC machine. The RISC architecture needs more working RAM memory that a CISC to hold values as its loads each instruction and acts upon it, then it loads the next one. The hack computer is similar to the RISC machine since our Hack machine uses a single clock cycle for one instruction.**

**https://www.microcontrollertips.com/risc-vs-cisc-architectures-one-better/#:~:text=One%20of%20the%20major%20differences,efficiency%20in%20instructions%20per%20program.&text=RISC%20needs%20more%20RAM%2C%20whereas,less%20RAM%20overall%20than%20RISC.**

1. **Describe IN DETAIL how the CPU you designed implements the fetch-decode-execute cycle.**

**Fetch: Our CPU fetches the information from instructions that are inputted and will check with the program counter to see which instruction to run next. The Program counter will give an address in memory of where our next instruction is located at, it also uses the different loads that we created for the specific jumps.**

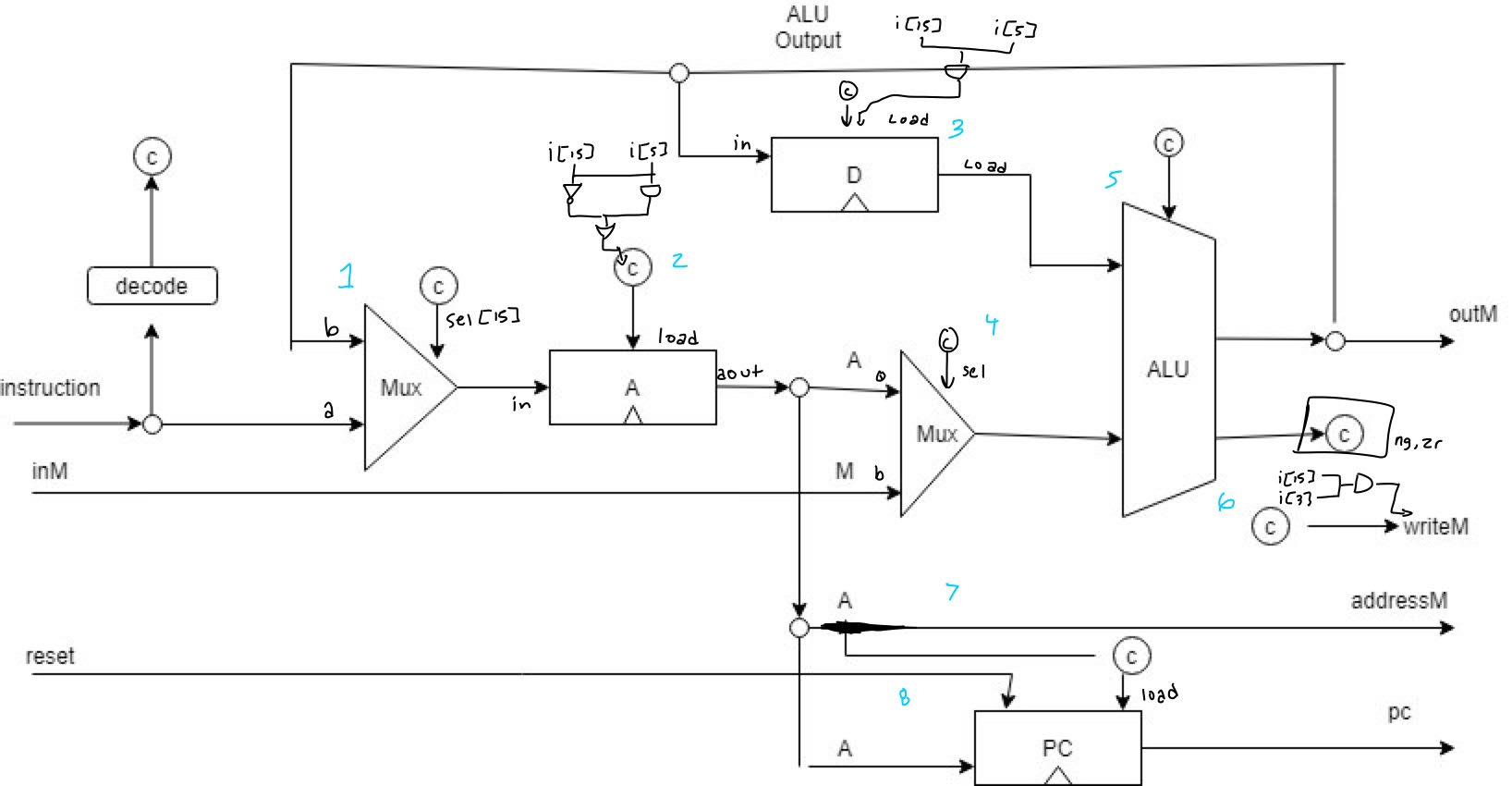
**Decode: The decode process is used when the instruction goes through the A and D registers, because this is determining what type the instruction is. We also decode when we take a 16 bit input and then output 15 bits.**

**Execute: The execute process is seen when we take a value and then input it into the ALU chip, then taking a different value from the register and adding them together. This cycle is repeated until the cycle ends.**

**COMMENTS:**

I felt like this project was very important because we have taken all of our work from previous labs and projects and created an entire running computer, which I believe is very eventful. I thought following along with Professor Browns video was very helpful because it helped fill some holes that I had in the logic of this chapter. The CPU chip was fairly difficult to program and there was not too much information on the computer chip. There was only a review on what the computer chip does on black board so I believe more useful resources on the Computer chip would have been helpful.

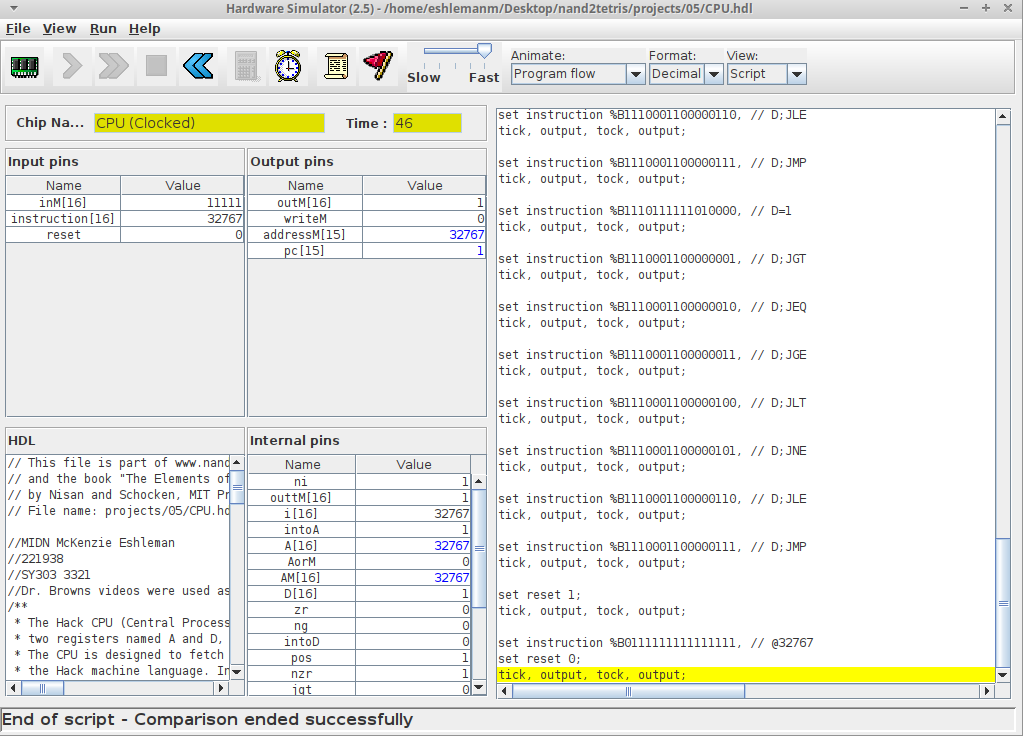
**ENCLOSURE (1): CPU Gate Diagram**

****

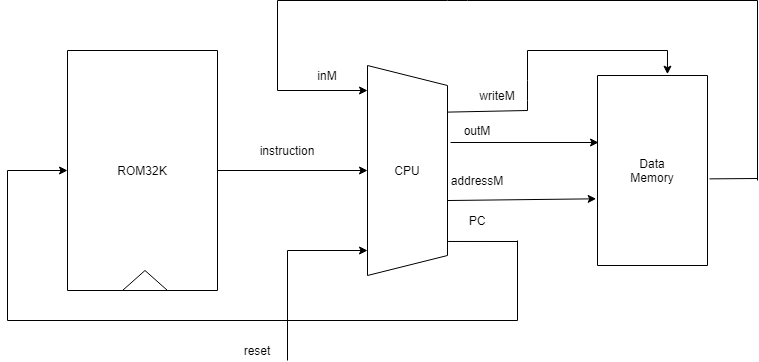
**ENCLOSURE (2): CPU hdl code**

|  |
| --- |
| **// 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**  **//MIDN McKenzie Eshleman**  **//221938**  **//SY303 3321**  **//Dr. Browns videos were used as reference**  **/\*\***  **\* The Hack CPU (Central Processing unit), consisting of an ALU,**  **\* two registers named A and D, and a program counter named PC.**  **\* The CPU is designed to fetch and execute instructions written in**  **\* the Hack machine language. In particular, functions as follows:**  **\* Executes the inputted instruction according to the Hack machine**  **\* language specification. The D and A in the language specification**  **\* refer to CPU-resident registers, while M refers to the external**  **\* memory location addressed by A, i.e. to Memory[A]. The inM input**  **\* holds the value of this location. If the current instruction needs**  **\* to write a value to M, the value is placed in outM, the address**  **\* of the target location is placed in the addressM output, and the**  **\* writeM control bit is asserted. (When writeM==0, any value may**  **\* appear in outM). The outM and writeM outputs are combinational:**  **\* they are affected instantaneously by the execution of the current**  **\* instruction. The addressM and pc outputs are clocked: although they**  **\* are affected by the execution of the current instruction, they commit**  **\* to their new values only in the next time step. If reset==1 then the**  **\* CPU jumps to address 0 (i.e. pc is set to 0 in next time step) rather**  **\* than to the address resulting from executing the current instruction.**  **\*/**  **CHIP CPU {**  **IN inM[16], // M value input (M = contents of RAM[A])**  **instruction[16], // Instruction for execution**  **reset; // Signals whether to re-start 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:**  **// Put your code here:**  **//1**  **//if A-instruction is pass address to A, else pass ALU output as dest**  **Mux16(a=instruction, b=aluout, self=instruction[15], out=mux1);**  **//determines if we are using the A instruction or c instruction for input for our CPU**  **//2**  **//A instruction with the A register**  **And(a=instruction[15], b=instruction[5], out=adest); //If both true c instruction (15) and the destination 1 (5)instruction**  **Not(in=instruction[15], out=ainst); //checks to see if it is an A instruction**  **Or(a=dest, b=ainst, out=aload); //loads which ever instruction is being used**  **ARegister(in=mux1, load=aload, out=aout); //using the ARegister chip with our determined inputs**  **//3**  **//load logic for the D register using C-insruction and destination 2**  **//will load D if the C instruction has D destination**  **And(a=instruction[4], b=instruction[15], out=dload); //combines together the C instruction and destination**  **DRegister(in=aluout, load=dload, out=dout); //implementing the D register**  **//4**  **//determines if we are using the A or M input of the instruction**    **Mux16(a=aout, b=inM, sel=instruction[12], out=aorm); //bit 12 is the A bit**  **//5**  **//inputting the C bits that will go into the ALU**  **//instructions are the address for the index of c instructions**    **ALU(x=dout, y=aorm, zx=instruction[11], nx=instruction[10], zy=instruction[9], ny=instruction[8], f=instruction[7], no=instruction[6], out=outM, out=aluout, zr=zero, ng=neg);**  **//6**  **//write logic to write to RAM memory, happens when it is a C instruction and the D3 bit being M**  **//checks if both are true and writes into memory**    **And(a=instruction[3], b=instruction[15], out=writeM);**  **//7**  **//takes the output of the A address and converts it into address M**  **//converts the 16 bits into 15 bits of A**  **And16(a=aout, b=true, out[0..14]=addressM);**  **//8**  **//defining our loads for the program counter C instruction**  **//input is from the A register**  **//load 1 jump if j1 is negative & ng confirms it being negative**    **And(a=instruction[2], b=neg, out=jump1);**  **//load 2 jump if j2 is zero & zr confirms it is negative**  **And(a=instruction[1], b=zero, out=jump2);**  **//load 3 jump is j3 is positive as long as the number is not negative or zero it jumps**  **Or(a=zero, b=neg, out=negz); //checks if number is zero or negative**  **Not(in=negz, out=pos); //if not negative or zero then it must be positive**  **And(a=instruction[0], b=pos, out=jump3);**  **Or(a=jump1, b=jump2, out=jumping);**  **Or(a=jump3, b=jumping, out=pcloadifc); //if loading is true then we jump**  **//only jump if it is a C instruction**  **And(a=instruction[15], b=pcloadifc, out=pcload);**  **PC(in=aout, inc=true, load=pcload, reset=reset, out[0..14]=pc); //implementing the PC chi**  **}** |

**ENCLOSURE (3): CPU chip test**

****

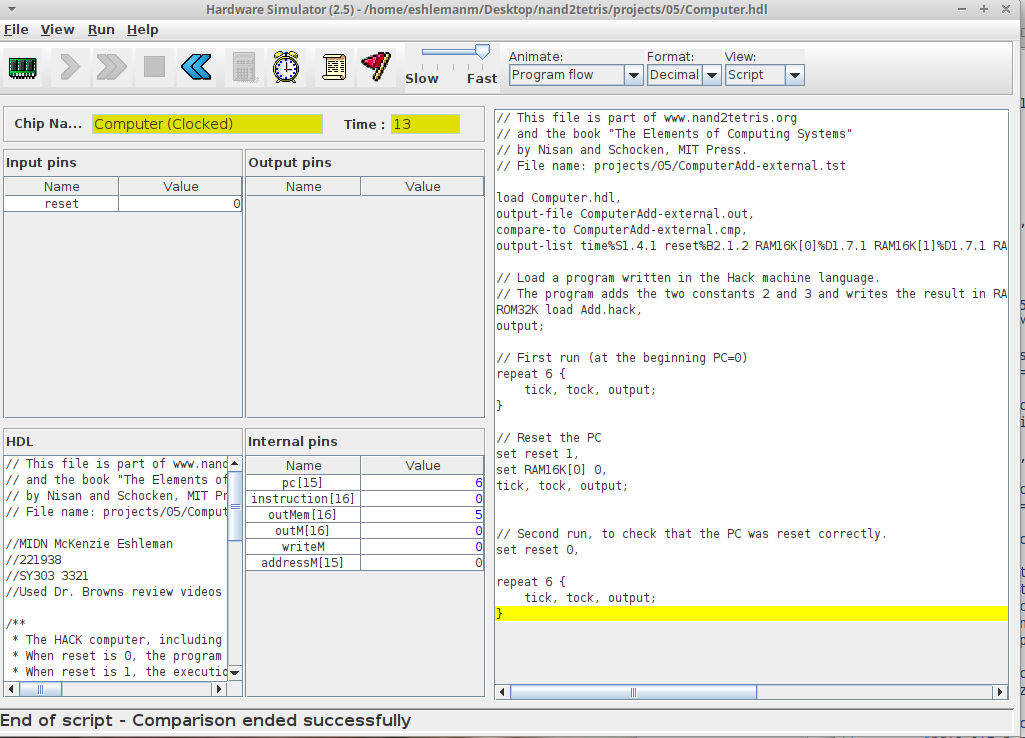
**ENCLOSURE (4): Computer Gate Diagram**

****

**ENCLOSURE (5): Computer hdl code**

|  |
| --- |
| **// 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**  **//MIDN McKenzie Eshleman**  **//221938**  **//SY303 3321**  **//Used Dr. Browns review videos**  **/\*\***  **\* The HACK computer, including CPU, ROM and RAM.**  **\* When reset is 0, the program stored in the computer's ROM executes.**  **\* When reset is 1, the execution of the program restarts.**  **\* Thus, to start a program's execution, reset must be pushed "up" (1)**  **\* and "down" (0). From this point onward the user is at the mercy of**  **\* the software. In particular, depending on the program's code, the**  **\* screen may show some output and the user may be able to interact**  **\* with the computer via the keyboard.**  **\*/**  **CHIP Computer {**  **IN reset;**  **PARTS:**  **// Put your code here:**  **//uses ROM32K to set the address to PC and fetches the outputs instructions**  **ROM32K(address=pc, out=instruction);**  **//sets all the inputs for the CPU chip**  **CPU(inM=outMem, instruction=instruction, reset=reset, outM=outM, writeM=writeM, addressM=addressM, pc=pc);**    **//after the inputs run through CPU they go into the memory**  **Memory(in=outM, load=writeM, address=addressM, out=outMem);**  **}** |

**ENCLOSURE (6): Computer chip test**

****