­­­­­Name: McKenzie Eshleman

Partner(s): Jon Robinson (in company)

Section: 3321

**Objective:** In this lab, you will build on experience gained with the previous projects and build the full von Neumann computing platform. These parts will be built using HDL, documentation using Gate Diagrams and commented code will be required.

1. **Pre-Lab: Big Picture**
   1. Page 97 shows the full intended implementation of the final Computer chip you will design.

|  |
| --- |
|  |

* 1. For each signal in this diagram, **describe** its source and destination as well as its purpose in the overall design of the Computer. Avoid superficial responses (eg. the source is the CPU). (20 pts)

**reset**: The reset comes from the Computer chip, this input is related to the PC in our CPU chip. If the reset is equal to zero then there should be a jump and the PC value is set to A, if the reset is 1 the PC is set to zero and the computer is restarting.

**pc**: The PC comes out of the CPU chip and goes through the reset, as stated above the PC will jump and set the value of PC equal to A , else the PC is set to PC + 1 and goes back into instruction memory and restarts the computer. This is part of the fetch logic and resetting the computer.

**instruction**: The instruction comes from the instruction memory which is ROM that is preloaded with a program that is written in the Hack Machine Language. The instruction is interpreted as the current instruction and is then inputted into the CPU chip.

**writeM**: WriteM is a single bit output from the CPU chip, it then goes into the data Memory. WriteM functions as a control bit for the data memory, if a current instruction needs to write a value to M, when writeM is zero any value will appear in outM.

**addressM**: M refers to the memory location addressed by A (Memory[A]), the address comes from the CPU chip and is inserted into data memory.

**inM**: The M value is inputted as the contents of RAM[A], it holds the value of what is stored in Memory[A], this again comes from the CPU chip and is inserted into the Data Memory.

**outM**: If the current instruction requires to write a value to M, the value is placed as outM, it comes from the CPU chip and is inserted into Data Memory. WriteM and OutM are effected similarly by the execution of the current instruction.

1. **Part: Memory**

**Purpose:** Describe the function and purpose of this chip in your own words. Describe, at a high level, its internal implementation as well as how it will fit in to your final Computer product. (5 pts) The memory chip is composed of three separate chips, including the RAM16K, Screen, and Keyboard. The chip puts these three chip parts into a single address space. Conceptually, the memory records or recalls values using the first 16K words of the memory. It also writes or reads the screen using the next 8K words of memory. Lastly it reads which key is currently pressed and uses the next word of the memory.

You should go straight into conceptualizing your gate diagram now. Memory is made up of three main chips: **RAM16K**, **Screen**, and **Keyboard**. Each of these chips is mapped to a different region of the memory address space.

For the **RAM16K**, record the lowest memory address that maps to this chip in both decimal and binary (15-bit) form. (3 pts)

|  |  |  |
| --- | --- | --- |
| **Decimal** |  | **Binary** |
| 0 |  | 000 0000 0000 0000 |

Now record the highest memory address that maps to this device in both decimal and binary form. (2 pts)

|  |  |  |
| --- | --- | --- |
| **Decimal** |  | **Binary** |
| 16383 |  | 011 1111 1111 1111 |

Compare these two binary numbers carefully. Recall from some earlier truth tables that you can discriminate between options by recognizing which bits change and which bits stay the same. List the bit index(es) of the bit(s) which remain constant during all accesses of the **RAM16K**. **HINT**: Make sure to properly count from **LSB** to **MSB**! (5 pts)

Constant bit(s):

For each of the binary addresses for ROM16K the 14th bit is both zero.

For the **Screen**, record the lowest memory address that maps to this chip in both decimal and binary (15-bit) form. (3 pts)

|  |  |  |
| --- | --- | --- |
| **Decimal** |  | **Binary** |
| 16384 |  | 100 0000 0000 0000 |

Now record the highest memory address that maps to this device in both decimal and binary form. (2 pts)

|  |  |  |
| --- | --- | --- |
| **Decimal** |  | **Binary** |
| 24575 |  | 101 1111 1111 1111 |

Compare these two binary numbers carefully. Recall from some earlier truth tables that you can discriminate between options by recognizing which bits change and which bits stay the same. List the bit index(es) of the bit(s) which remain constant during all accesses of the **Screen**. **HINT**: Make sure to properly count from **LSB** to **MSB**! (5 pts)

Constant bit(s):

For the screen both the 14th bit is a one.

For the **Keyboard**, record the memory address that maps to this chip in both decimal and binary (15-bit) form. (5 pts)

|  |  |  |
| --- | --- | --- |
| **Decimal** |  | **Binary** |
| 24576 |  | 110 0000 0000 0000 |

Now compare this binary address with the ones from the two other chips. Recall from some earlier truth tables that you can discriminate between options by recognizing which bits change and which bits stay the same. List the bit index(es) of the bit(s) which are unique to **Keyboard** when compared against the constant bits from the previous two chips. **HINT**: Make sure to properly count from **LSB** to **MSB**! (5 pts)

For the keyboard the 13th and 14th bit are ones.

Unique bit(s):

As you attempt to conceptualize the Gate Diagram for Memory, consider these questions.

1. What chip has been designed to allow you to write a single input to one of three possible chips? (3 pts) A DMux chip is ideal to use for Memory because we can have one input use it for three separate variables. This allows us to send the variables to the appropriate registers.
2. What chip has been designed to allow you to read output from three different chips and select just one of them? (2 pts) A Mux4Way16 can be used because we can select which output we want to use, this can be from the RAM, Screen, or keyboard, and we have a one output.

**Gate Diagram (Insert Drawing or Use Word Art):**

**Diagram:** (10 pts)

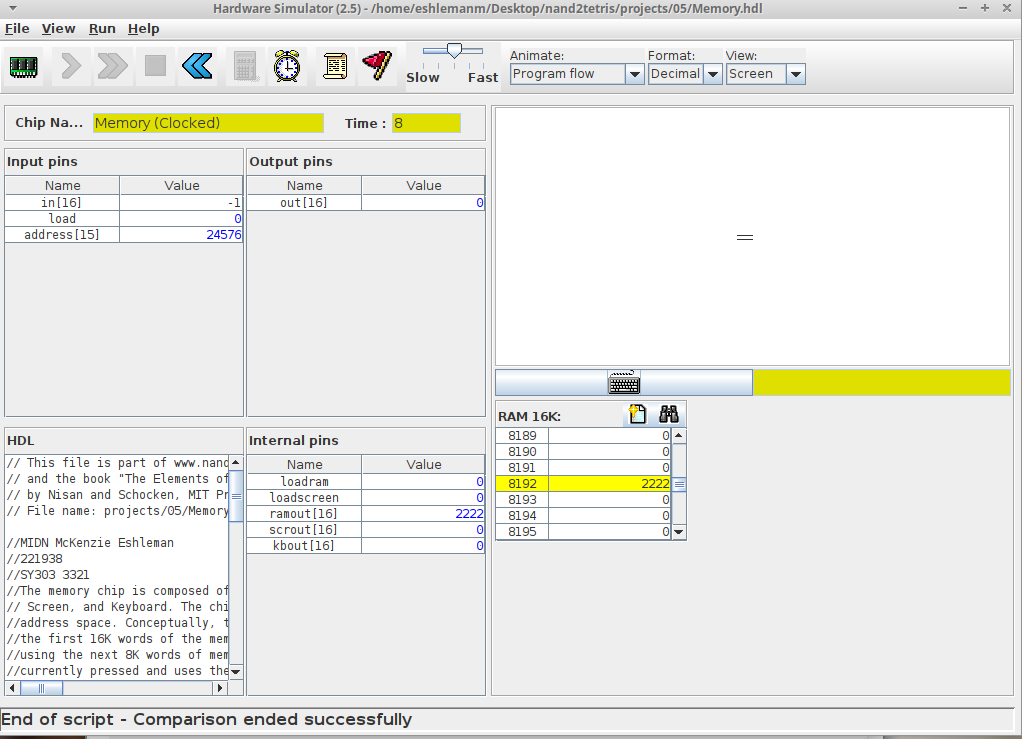
|  |
| --- |
|  |

**HDL (Screenshot of VM or Raw Text):** Enter your HDL below that describes your gate diagram above. (5 pts)

|  |
| --- |
| DMux(in=load, sel=address[14], a=loadram, b=loadscreen);  //takes a single inputs and loads them to a three outputs, we are  //creating the load for ram and screen. This allows us to send the  //variables for an appropriate register.    RAM16K(in=in, load=loadram, address = address[0..13], out=ramout);  //since the address is 15 bits we use the fist 14 bits for the RAM and the  //last bit is for kdb. We load the ram from the DMUX and it gives us the ram output.    Screen(in=in, load=loadscreen, address=address[0..12], out=scrout);  //the screen uses 13 bits of addresses, we load the screen from the DMux and then we output  Keyboard(out=kbout);  //last bit is the kbd address, only uses a single bit ,outputs when a key is pressed  Mux4Way16(a=ramout, b=ramout, c=scrout, d=kbout, sel=address[13..14], out=out);  //We use two Ram outs, the screen out, and the keyboard out and using addresses 13 & 14  //we get a single output. This is acting as almost a funnel taking in all the inputs and  //creating a single output. |
|  |

**Test in HW Simulator (Screenshot of VM):** (5 pts)

This is not automatic! You will need to perform some interaction with the keyboard as this test script executes.



**Additional Screenshots (as needed):**

1. **Read through the Wikipedia page on von Neumann architecture (https://en.wikipedia.org/wiki/Von\_Neumann\_architecture). It includes a list of early computers that did not employ the von Neumann model and another list of early computers that implemented the stored program model. Read about one of these computers and describe its architecture. How did it differ from a typical von Neumann machine? How did it store and execute programs? What instruction set did it use?** (10 pts)

**The IBM Selective Sequence Electronic Calculator was an electromechanical computer built in late 1944. It had many features of a stored-program computer and was the first operational machine able to treat its instructions as data. The SSEC used vacuum tubes that were used in the arithmetic unit, control, and its eight registers. The memory for this machine was organized as signed 19 bit digit decimal numbers. The data was stored at a capacity of 400,000 digits in the form of reels of punched tape. The difference between a typical von Neumann computer and the SSEC is that the SSEC was partially electromechanical, and not fully electronic. Instructions were read from paper tape due to its limited memory.**

1. **Read about the Apollo Guidance Computer (**<https://en.wikipedia.org/wiki/Apollo_Guidance_Computer>**). Compare and contrast it with the Hack computer specified in Chapter 5.** (10 pts)

**The Apollo Guidance Computer is a digital computer that was produced for the Apollo Program and was installed on board the command module and lunar module. The AGC has a 16-bit word length, with 15-data bits and one parity bit. This is similar to the hack computer, because the hack computer is a 16-bit von Neumann machine. The AGC and hack machine are also similar because they both have two-memory mapped I/O devices: a screen and a keyboard.**