TECH 3233

# Lab #1

# Memory

Online Ver 2.50 (with 6800 code)

Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Purpose:** To familiarize students with address and data bus architecture.

**Discussion:** As we discussed in class, a computer uses the address bus to send the address of the memory location to be ‘read from’ or ‘written to’ to the corresponding pins of the memory IC. The memory IC then either puts the data requested onto the data bus, in the case of a read function, or takes the data from the data bus and places it in that memory location in the case of a write function.

In this lab we will use a simulation of a 1K PPROM to gain first hand experience with the process of reading data from memory using the Address and Data Bus.

**Procedure:**

1. Copy the contents of the USB Provided by the instructor to your desktop. If Java is NOT installed, click on JavaSetup\_\_\_\_.exe to install (if asked for admin login,you’re your normal computer login – This will only work in ET 227). When the install is finished, erase the JavaSept\_\_\_\_.exe file from the desktop. Now, click on ‘hades.jar’. Once Hades is running, select FILE | Open and select rom.hds.

(ALT if you cannot install java on the pc) Refer to the last lecture notes on running simulations and load the simulation shown in Fig 10.

Select Layers | all of the above layers. This will display additional information such as pin names (eg A0..A9) and identify which hex inputs and 7-segment display outputs are the HIGH and LOW order nibbles.

Make the nOE (Output Enable) and nCS (Chip Enable) active (hint: the “n” is the same as a BAR in digital logic).

1. Place the value 0x100 on the hex inputs (the ADDRESS LINES)

You should see 0xFF on the OUTPUTS, if you do not, **STOP** and ask for help. This indicates that memory location (Address 0x100) contained a value of 0xFF and that this value is being placed on the DATA LINES.

1. Now place the value 0x101 on the Address Lines. Read the data lines as before. It should read 0x00, if it does not, **STOP** and ask for help.
2. Now place the value 0x110 on the address lines and complete the table below by reading each consecutive memory location. **Stop when the DATA returns a 0x0A**. Using an ASCII Table (found on the internet) convert the hex values to their ASCII equivalent and write them in the table below (NOTE: the **table is LARGER THAN NEEDED**).

|  |  |
| --- | --- |
| **Memory Address** | **Data Value** |
| Hex Value | Ascii Character |
| 0x110 |  |  |
| 0x111 |  |  |
| 0x112 |  |  |
| 0x113 |  |  |
| 0x114 |  |  |
| 0x115 |  |  |
| 0x116 |  |  |
| 0x117 |  |  |
| 0x118 |  |  |
| 0x119 |  |  |
| 0x11A |  |  |
| 0x11B |  |  |
| 0x11C |  |  |
| 0x11D |  |  |
| 0x11E |  |  |
| 0x11F |  |  |

1. Now place the value 0x130 on the address lines. What is the value in the memory location in hex? Use windows calculator (in programmer mode) convert this value to decimal.

|  |  |
| --- | --- |
| **Memory Address** | **Data Value** |
| Hex Value | Decimal Value |
| 0x130 |  |  |

1. Now place the value 0x138 on the address lines. What is the value in the memory location (in hex)? What is the value in decimal? What is the value assuming the original value was stored in TWO’S Complement (Complement (in Win Calculator, you if the size of the data is defined correctly, it will show the 2s comp value)?

|  |  |
| --- | --- |
| Memory Address | **Data Value** |
| Hex Value | Decimal Value | Signed Decimal Value |
| 0x138 |  |  |  |

1. Now read memory locations 0x140 AND 0x141 and record both values (in hex) side by side Convert that value to decimal and signed decimal. When two bytes are combined in this fashion they are said to be in Big Endian format (big value first)

|  |  |
| --- | --- |
| **Memory Address** | **Value** |
| 0x140 |  |
| 0x141 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Memory Address | Hex Value | Decimal Value | Signed Decimal Value |
| 0x140 | 0x141 |
|  |  |  |  |  |

1. Now read memory locations 0x145 AND 0x146 and record both values (in hex) and swap the two bytes. Convert that value to decimal and signed decimal. When two bytes are combined in this fashion they are said to be in Little Endian format (small value stored first)

|  |  |
| --- | --- |
| **Memory Address** | **Value** |
| 0x145 |  |
| 0x146 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Memory Address | Hex Value | Decimal Value | Signed Decimal Value |
| 0x146 | 0x145 |
|  |  |  |  |  |

1. Now read memory locations 0x150 and 0x151 and record the values in table two. Now convert it into binary then into the Date format as shown.

|  |  |
| --- | --- |
| Memory address  | Binary Value |
| 0x150 | 0x151 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | M | M | M | M | D | D | D | D | D | Y | Y | Y | Y | Y | Y | Y |
|  |  |  |  |  |

* 1. This represents what date?
	2. What is the significants of this date in Space Exploration History?
1. Now turn on the A9 input. What memory location is being addressed? What is in that location:

|  |  |
| --- | --- |
| Memory Address | Data Value |
| Hex Value | Decimal Value |
|  |  |  |

1. Now add 0x010 to the address above. Read a total of 4 memory locations (32 bits) and record the hex value below (assume Big-Engian). Now, using the website <https://www.binaryconvert.com/convert_float.html> convert the hex value to a decimal number and record below:

|  |  |
| --- | --- |
| Memory | Floating Point Equivalent |
| Address -> |  |  |  |  |
| Data -> |  |  |  |  |  |

1. Now select address 0x000 and record the data at each location in the list

|  |  |
| --- | --- |
| **Memory Address** | **Data** |
| 0x000 |  |
| 0x001 |  |
| 0x002 |  |
| 0x003 |  |
| 0x004 |  |
| 0x005 |  |
| 0x006 |  |
| 0x007 |  |
| 0x008 |  |
| 0x009 |  |

1. Assuming these are 6502 processor instructions, use the Op Code Set at <http://www.6502.org/tutorials/6502opcodes.html> to interpret the instructions as we did in class. Write the ASM code below:
2. What do you think the ASM Code will do?
3. Open the 6502 processor simulator found at <http://www.visual6502.org/JSSim/expert.html> and enter the values into the same memory location as the ROM (overwriting the existing values). Using the ⮙to rewind the program then 🢖 to step forward, run the code. You will need to step forward multiple times to run the code (yellow highlight shows program counter location). What did it do? Is it what you predicted above?

Going back to the ROM Simulation:

1. What happens when you set the $\overbar{OE}$ line inactive? What does the gold color on the data lines indicate?
2. Set $\overbar{OE}$ active once again and set $\overbar{CE} $ inactive. What happens?
3. Using what you have observed above, state how multiple ROMS could be connected to a computer with one additional address line. Keep in mind that both IC’s outputs (D7..D0) would be tied together.

**Questions:**

1. How many address lines does the IC have?
2. How many bytes of information can the IC hold (in 2x format)?
3. How many Kilobytes of memory is that?
4. A 27C64 IC hold 8Kbytes of memory. What does the “64” in the part number represent?
5. Is there a way to determine what a value in memory actually represents? If so explain how it can be determined, if not, state why this is the case.
6. What is the significance of the ASCII Values 0x0D and 0x0A? What would these characters do when printed to a monitor at the end of the message?

Submit the completed document file via the on line assignment submission link on the class website. (Due in one week)