CMPSC 210
Principles of Computer Organization
Fall 2016
John Wenskovitch
`http://cs.allegheny.edu/~jwenskovitch/teaching/CMPSC210`

Lab 4 – Hexadecimal, Arithmetic, and MIPS

Due (via Bitbucket) Wednesday, 28 September 2016
30 points

Lab Goals

- Convert some numbers between hexadecimal and decimal
- Practice some binary and hexadecimal addition and subtraction
- Explore hexadecimal data storage in MIPS/MARS

Assignment Details

Our course content for the past week has been a bit scattered yet again: we've covered converting numbers from hexadecimal to decimal and back again (along with hexadecimal to binary and back), took a break for a Q&A session, and then concluded with this morning's lectures about binary and hexadecimal addition and subtraction. Because of the diverse lecture content, this lab will also include a good deal of diverse content, though there is clearly a focus on conversions and mathematics in other bases.

Part 1: Number Conversions (10 points)

Please answer each of the following questions about change-of-base conversions (for one point each). Please show all of your work for partial credit.

1. Convert the unsigned hexadecimal number $(DECADE)_{16}$ to decimal.
2. Convert the unsigned hexadecimal number $(FADED)_{16}$ to decimal.
3. Convert the decimal number $(73)_{10}$ to hexadecimal.
4. Convert the decimal number $(1001)_{10}$ to hexadecimal.
5. Convert the binary number $(0001001010010)_2$ to hexadecimal.
6. Convert the hexadecimal number $(ACCEDE)_{16}$ to binary.
7. Convert the decimal number $(-4321)_{10}$ to two's complement hexadecimal. (Hint: convert to binary first)

8. Assuming that the hexadecimal number is unsigned, how many hexadecimal digits will it take to store the equivalent of the decimal number $(247)_{10}$? How do you know? (Hint: convert to binary first)
9. Assuming that the binary number is signed using the sign+magnitude method, how many hexadecimal digits will it take to store the equivalent of the decimal number $(247)_{10}$? How do you know? (Hint: convert to binary first)
10. Assuming that the binary number is signed using the twos-complement method, how many hexadecimal digits will it take to store the equivalent of the decimal number $(247)_{10}$? How do you know? (Hint: convert to binary first)

Part 2: Binary and Hexadecimal Addition and Subtraction (10 points)

Please answer each of the following questions about binary and hexadecimal addition and subtraction (for two points each). Again, please show all of your work for partial credit.

1. Add together $(1001010)_2$ and $(11111)_2$, both **unsigned** binary numbers, and report the sum in **binary** form.
2. Add together $(1001010)_2$ and $(11111)_2$, both **two's complement** binary numbers, and report the sum in **decimal** form.
3. Add together $(BEEF)_{16}$ and $(FEED)_{16}$, both **unsigned** hexadecimal numbers, and report the sum in **hexadecimal** form.
4. Subtract $(BEEF)_{16}$ from $(FEED)_{16}$, both **unsigned** hexadecimal numbers, and report the sum in **binary** form.
5. Add together $(1010101010)_2$, $(12345)_{10}$, and $(CAFE)_{16}$, all **unsigned** numbers, and report the sum in **hexadecimal** form.

Part 3: Exploring Hexadecimal in MIPS and MARS (10 points)

Now that you are (hopefully!) a bit more comfortable with seeing hexadecimal numbers, let's explore a bit further into MIPS and MARS to look at the way data is stored and loaded. In the /cmpsc210f2016-share/labs/lab4 directory is a partially complete `HexExploration.asm` file. Please open it in MARS and take a look at what is inside.

To begin, the `.asm` file contains our Hello World program from the first lab, but also has a few other values tossed in the `.data` section and a big comment block. These values have three different data types: `.word`, `.half`, and `.byte`. These datatypes map the values to different sizes in memory. A word takes up 4 bytes (32 bits) of memory, a halfword naturally takes up half of that (2 bytes or 16 bits of memory), and a byte takes up 1 byte (8 bits) of memory.

1. What are the biggest and smallest values (in decimal) that can be stored in a `.word`? How do you know?

2. What are the biggest and smallest values (in decimal) that can be stored in a `.half`? How do you know?

3. What are the biggest and smallest values (in decimal) that can be stored in a `.byte`? How do you know?

4. Will all of the values included in the `.data` section above fit into the memory sizes defined? If any do not, you can replace them with your choice of number.

If you assemble the program, you will see that these extra data values are taking up space in memory after the "Hello World" string, beginning at address 0x10010010. This data is stored inside each block from right-to-left, again because of reasons that we have not discussed yet. In general, you can think of the ordering of addresses within these blocks as follows:

```
###########################################################################
## 0x+03 # 0x+02 # 0x+01 # 0x+00 ### 0x+07 # 0x+06 # 0x+05 # 0x+04 ### ...
###########################################################################
```

As you can see, the memory block containing the four bytes from 0x...00 through 0x...03 is listed in reverse order of bytes, followed by the memory block containing the four bytes from 0x...04 through 0x...07 in reverse order, and so on.

5. Convert each of the integers from the data section into hexadecimal (going through binary first if you would like). Remember that MIPS/MARS uses the two's complement representation.

6. Fill in the table in the comment block of the `.data` section, listing which number is associated with each byte in memory. Some bytes may have more than one number, others may have no number at all. Some numbers may be spread across several bytes. Include a copy of this table in your submission document (Deliverable #4).

7. Did you notice anything interesting about the way the numbers are laid out in memory?

Now let's start to manipulate these numbers. Recall that a previous lab mentioned the commands to load words, halfwords, and bytes from memory are `lw`, `lh`, and `lb`. For example, you can load the word *number1* with the command "`lw $t1, number1`."

8. Write the instructions (and include comments) to load each of the numbers from memory into the `$t` registers using the appropriate load commands. What happens if you try to load *number4* using `lw`? What happens if you try to load *number5* with `lb`?

Assuming that you loaded each of the numbers into the `$t` registers sequentially (*number1* into `$t1`, *number2* into `$t2`, etc) then you should have a value of 1701 stored in `$t3` and a value of -1701 stored in `$t5`. Let's add them together! You already saw the `addi` instruction in the string length example. There is also an `add` command that allows you to add the values in two registers together and store them in a third register. The format for the instruction is: `add sum number1 number2`.

9. Write the instructions (and include comments) to add together the 1701 and -1701 values and store the sum in `$t8`. What happened? Did you get the value that you expected?

There is likewise a substraction instruction (`sub`) with a similar format: `sub difference number1 number2`.

10. Write the instructions (and include comments) to subtract -1701 from 1701 and store the difference in `$t9`. What happened? Did you get the value that you expected?

One last set of instructions for today. Just like we could load data from memory, we can also store it back into memory. The instructions for this functionality are `sw` to **s**tore a **w**ord, `sh` to **s**tore a **h**alfword, and `sb` to **s**tore a **b**yte. First, we should define a space in memory.

11. In the `.data` section, write `.word` placeholders for the sum and difference values currently in `$t8` and `$t9`. You can call them `sum` and `difference`. Since these are placeholders, you can give them a default value of 0.
12. In the `.text` section, write the instructions (and include comments) to store the sum value from `$t8` into `sum` and the difference value from `$t9` into `difference`. Use `sw` to store these words.
13. What happens if you try to store halfwords instead? Is this what you expected?
14. What happens if you try to store bytes instead? Is this what you expected?

Submission Details

For this assignment, your submission to your BitBucket repository should include the following:

1. Your responses to the questions in Part 1. [If you have hand-written the solutions to these exercises, please submit them to TA Victor Zheng instead of BitBucket.]
2. Your responses to the questions in Part 2. [If you have hand-written the solutions to these exercises, please submit them to TA Victor Zheng instead of BitBucket.]
3. A commented version of your `HexExploration.asm` code.
4. A document responding to the questions scattered throughout of Part 3. (Not all of the questions require responses, but most of them do.)

Before you turn in this assignment, you also must ensure that the course instructor has read access to your BitBucket repository that is named according to the convention `cs210f2016-<your user name>`.