Introduction

This assignment is a *group* assignment. If I, in the room, do not **hear** you interacting with the other students, you are doing it *wrong*!

Note

- By **design**, this assignment includes questions about topics that you have not yet seen in lecture to encourage *creative* exploration.
- Work together on *each* question. I **must** hear the group discussing the questions and the answers. All group members **must** participate.
- Constructing your own model is more important than the "right" answer. Read, think, discuss the material **together**. This process is **short-circuited** by looking the material up without engaging with it.
- Each student in the group has more or less experience with the topics in the assignment. If you have greater familiarity with the topics, *please* hold back a little bit so others can engage with building their own knowledge. You, too, must engage because there is always more to learn.

Assignment Goals

Learning Outcomes After completing this group assignment, each student is expected to be able to

- **Trace** a block of MIPS assembly, noting the value of the `PC`, other registers, and RAM locations at various points.
- Write MIPS code to *declare* global `int` variables, *load* variables into registers, *manipulate* the values in registers, and *store* values back into variables in RAM.
- Be able to interpret *hexadecimal* values as equivalent *binary* values.
- Be able to apply logical operations to *binary* values.

Procedure

Get out paper for a *single* turn-in at the end of class. Copy enough of each question so that the paper could stand alone as a study guide.

Assign Roles. Students should take roles they have not held recently (or, perhaps, ever):

> **Manager** Move discussion forward.
> **Recorder** Writes the report that will be turned in.
> **Reflector** Monitor that everyone gets heard and is caught up. (This is a **group** obligation, really.)
> **Speaker** (Combine w/ **Reflector** if there are not four group members.) Asks the facilitator questions and communicates what the team has done.

Answer these questions.

1. How many different values are there for each **b**nary dig**it** (bit) in a base-2 number? Per digit in a base-10 number? Per digit in a base-16 number?
2. How many bits are needed to store at least 4 distinct values? What about 8 values? 16?
3. If, instead of bits, you used base 10 digits, how many different values could you store with each width in the previous question?
4. If, instead of bits, you used base 16 digits, how many different values could you store with each width in the previous question?

5. Fill in the following table.

Dec (2)	Hex (1)	Bin (4)
00		
	1	
		0010
		0011
04		
	5	
06		
	7	
		1000
		1001
10		
	b	
12		
13		
14		
15		

6. A **directive** in MIPS assembly begins with a dot, *i.e.* **`.data`**, and commands the *assembler* during assembly-time rather than commanding the CPU during run-time

   A program has two *segments*: one for globally declared **data** and one for **instructions**, the **text** of the program. For example:

```
      .data
x:    .word    100

      .text
main:
      lw      $t0, x
```

   (a) If the **`.word`** directive sets aside a MIPS *word* of memory, how many **bytes** is that?

   (b) How much memory is set aside by the **`.byte`** directive?

   (c) What does a label, *e.g.* `x` or `main`, mean to the assembler?

   (d) Write a **`.data`** segment with room to store three (3) integer variables, `s`, `t`, and `u`. Set the first two to non-zero values of your choice and the last to zero.

   (e) Write a main program that uses **`lw`**, **`sw`**, **`add`**, and **`sub`** to evaluate the expression

   $$u = 3 \times s - 2 \times t$$

   (f) **`beq`** `$rX, $rY, label` is the *branch equal instruction*. The label is either an address or a label (that the assembler translates to an address). This is the basic *selection* operator in MIPS assembly.

address	*opcode*	*rX*	*rY*	*target*
`00400020`	`beq`	`$t3`	`$t5`	`00400084`

   i. What is the address of the **next** instruction after this one?

   ii. What is the *least-significant* nybble of the target in binary?

   iii. Why does it end with two zero bits?

   iv. What is the `pc` set to if **`$t3`** == **`$t5`**?

   v. What is the `pc` set to if **`$t3`** != **`$t5`**?

7. What is the value in **`$s0`** after the following command with the given initial values?

```
$s0 = 0xABCD0123
$s1 = 0x0123DCBA

and $s0, $s0, $s1
```