**TPS (Think-Pair-Share) activity 1 Paired with the classmate sitting next to you and do the following tasks (25 minutes):**

**1. Once fib.s is assembled, open the Execute tab (it should be opened by default after assembled). Two segments of memory are displayed here: Text Segment and Data Segment. What is the starting addresses of Text Segment and Data Segment? Give your answers in Hex format.**

The starting address of text segment is: 0x00400000

The starting address of data segment is: 0x10010000

**2. The Text Segment shows you how each actual machine code is stored in the memory (again it is simulated) and its corresponding MIPS code. Two types of MIPS code are shown here: Basic and Source. We call the Basic one True Assembly Language, and the Source one MIPS Assembly Language. From the display, what can you tell about the difference between the two in terms of their relationship with the machine code? We will cover this topic in future lectures**.

The source code is what we are using to get familiar with MIPS. The two differences I can spot right away is that Basic is using registers from $1-$32 and our source uses your typical MIPS syntax like $t0-$t7 or $s0-$s7. You have 1 to one relationships for some commands, however there are exceptions like in MIPS(**la)** that transfers in 2 commands in Basic(**lui** and **ori**) offset in basic is like calling a function kinda

**3. Now, let’s take a look at the Data Segment. How much difference in bytes are there between 2 address locations (again, the addresses are in Hex)**?

The difference between any two addresses is changed by an offset of 4 along the row:

0x10010000 so we have that 0x10 is 1\*161 + 0 \* 160 equal to 2 bytes or 16 bits

* 0x10010020 so we have that 0x20 is 2\*161 + 0 \* 160  equal to **4 bytes or 32 bits**

0x20

**4. For each address location, how many columns are there**?

So each address has 8 different columns that increase by 4 bytes values each time until it equals 24. (1c hex.)

**5. What can you tell about the relationship between the address difference and the number of columns at each address location**?

The address difference is 32 bits and the columns increase by four bits each.

**6. From the source code, how do you create a new variable/label named m and**

**set it to 20?**

m: .word 20

**7. Save and assemble your file. At what address is the value of m stored?**

0x10010004 going from left to right in Value(+4)

**8. Besides numbers, we can also initialize strings for the program to use. Search**

**from the Internet on how to declare a string named str1 and set it to “I love**

**CSE31!”**

str1: .asciiz “\nI love CSE31!\n”

**9. Insert the declaration of str1 in your code and assemble it. From the Data**

**Segment, we can see that the string is occupying 3 address locations. At what**

**addresses is str1 stored**?

It is occupying:

0x1001008

0x100100c

0x1001010

**10. str1 is stored as numerical values in the memory. Check the ASCII box and**

**observe how it is stored. Does the display of characters agree with what you**

**have learned from Lab #4 about how an array of characters is stored**?

Yes, each character is stored by one bit in our memory.

**11. In order to print str1, we will need to use syscall function. Search the**

**Internet to find out how to print str1.**

We utilize:

li $v0, 4

la $a0, str1

Syscall

**12. Now let’s go back to the program. Search from the Internet to find out what**

**“la $t3, n”(load address) does. What will be the value stored in $t3 after running this**

**instruction? From this we can see that we cannot use the initialized variables**

**(labels) directly in our MIPS program. We need to use la, then lw to save the**

**value into a register.**

It stores the address of the n into register $t3.

The value stored in 0x00000000

**TPS (Think-Pair-Share) activity 2 Paired with the same classmate and answer**

**the following questions (25 minutes):**

**1. From lectures, we have learned that we can perform different inequality comparisons (<, >, <=, >=) by using only slt(Set Less Than instruction), beq(branch on equal), and bne(Branch on not equal) instructions. Why not having one instruction for each inequality in MIPS?**

Due to the limitations in MIPS we can’t use these symbols it would require additional registers for these symbols. For Some reason it is SLT is easier than the symbols. Having bne and beq is much more convenient as we can account for two different instances.

**2. Declare a new variable/label (n) and set it to 25.**

n: .word 25

**3. Insert instructions to declare the following strings:**

**a. str1: "Less than\n"**

str1: .asciiz "Less than \n"

**b. str2: "Less than or equal to\n"**

str2: .asciiz "Less than or equal to\n"

**c. str3: "Greater than\n"**

str3: .asciiz "Greater than\n"

**d. str4: "Greater than or equal to\n"**

str4: .asciiz "Greater than or equal to\n"

**4. Insert instructions to read in an integer from users. Search from the Internet on how to use syscall to do it.**

main: li $v0, 5

syscall

move $t0, $v0

**5. Insert code so the program will compare if the user input is less than n. If it is, print “Less than”.**

slt $t2, $t1, $t0 # Storing the result of the comparison if user input is less than n

bne $t2, $0, L # If the $t2 is false then jump to "less than" function

**6. Insert code so the program will compare if the user input is greater than or equal to n. If it is, print “Greater than or equal to”.**

slt $t2, $t1, $t0 # Storing the result of the comparison if user input is less than n

beq $t2, $0, GE # If the $t2 is true then jump to "greater than or equal" function

**7. Now comment out your code from part 5 and 6. Insert code so the program will compare if the user input is greater than n. If it is, print “Greater than”.**

slt $t2, $t0, $t1 # Storing the result of the comparison if n is less than user input

bne $t2, $zero, G # If the $t2 is false then jump to "greater than" function

**8. Insert code so the program will compare if the user input is less than or equal to n. If it is, print “Less than or equal to”.**

slt $t2, $t0, $t1 # Storing the result of the comparison if n is less than user input

beq $t2, $zero, LE # If the $t2 is true then jump to "Less than or equal" function