Andre Martin

Lab 6 – CSE 31

***TPS: Activity 1***

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 are the starting addresses of Text Segment and Data Segment? Give your answers in Hex format.

**Text Segment: 0 x 00400000**

**Data Segment: 0 x 10010000**

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 cove this topic in future lectures.

**Basic one True Assembly Language: add $8, $0, $0**

**Source one MIPS Assembly Language: main: add $t0, $0, $zero**

**The difference between the two in terms of the relationship with the machine code is the $0 and the $zero.**

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)?

**Date Segment: The difference in bytes between 2 address locations is 0 x 10010000 and 0 x 10010020 so the difference is 00000020.**

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

**For each address location, there is eight columns for Data Segment.**

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

**The relationship between the address difference and the number of columns at each address location is that for Data Segment the address location changes but the columns remain the same.**

6, From the source code, how do you create a new variable/label named *m* and set it to 20?

**.data**

**m: .word 20**

**.text**

**lw $t0, m**

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

**The address that m is stored is 0 x 00400014**

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!"*

**.data**

**str1: . asciiz "I love CSE31!"**

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 stored in Data Segment 0 x 10010000**

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, it doesn't since the characters are stored Backswords.**

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

**We can print str1 by doing a system call code for printing string, load address of string to be printed, call operating system to perform operation, syscall takes in the argument and then terminate the program.**

12. Now let's go back to the program. Search from the Internet to find out what "*la $t3, n*" does. What will be the value stored in *$t3* after running this instruction? From this we can see that we cannot use the initialized variables (labels) directly in our MIPS program.

**la (loads the address), n = 13, to $t3 the value of $t3 will be 11 after running the program.**

***<u>TPS: Activity 2</u>***

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

**We can't have one instruction for each inequality in MIPS because MIPS Goals is Simpler is Better so by having only slt, beq, and bne instructions is simpler.**

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

**Ok**

3. Insert instructions to declare the following strings:

**Ok**

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

  **Ok**

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

  **Ok**

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

  **Ok**

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

  **Ok**

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

**Ok**

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

**Ok**

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*".

**Ok**

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*".

**Ok**

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*".

**Ok**