

**Course Name**: Computer Architecture Lab

**Course Number and Section**: **14:332:333:02**

**Experiment**: Lab 2 – Control Flow (loops and branches), Inputs & Outputs, and Making Decisions

**Lab Instructor**: Bangtian Liu

**Date Performed**: 2/7/18

**Date Submitted**: 2/21/18

**Submitted by**: Yuqing Feng (yf184)

Purpose

This laboratory exercise taught me how to use flow control in MIPS as well as input and output data to the user. Branching and loops were used extensively and are important for more complex problems. User input and output were extremely important in providing an interface to interact with the program. By providing instructions to the user and by getting the user inputs, the interaction between the user and the program is clear and successful. String manipulation is another part of this lab that requires getting the character from the string and knowing how to increment to the next element like with an array. Throughout all this, decision making based on the values (such as ending when the char is the end of the string, branching when a number is less than another number, etc), determined the branching and what actions to take to complete the program.

Approach

For reading and printing, “syscall” is used. For syscall to know what to do, $v0 must be loaded with a code. When reading values, the value the user inputs is put into $v0 and to put it into a register, one must use “move $t0, $v0” so the value is stored in $t0. The codes below specify what syscall will do:

|  |  |  |
| --- | --- | --- |
| $v0 code | What it does | Arguments needed |
| 1 | Print int | Value in $a0 |
| 2 | Print float | Value in $f12 |
| 3 | Print double | Value in $f12 |
| 4 | Print string | Value in $a0 |
| 5 | Read int | - |
| 6 | Read float | - |
| 7 | Read double | - |
| 8 | Read string | Addr. to store $a0  No. of char $a1 |
| 9 | Memory allocation | Bytes storage $a0 |
| 10 | Exit program |  |
| 11 | Print character | Integer in $a0 |
| 12 | Read character |  |

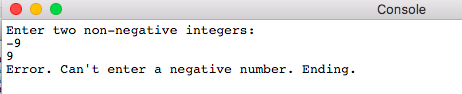
Other concepts used are branching, such as beq, beqz, bne, etc, which branch off into other sections. To return to the loops, using “j loop” with loop as the loop name returned it to the top of that loop. Loops were used extensively to continuously get user data, to iterate through all characters of a string, and to sort arrays, just to name a few.

Loops, array and string element retrieval, and conditionals were a huge part of these assignments as well as eliminating user inputs for being negative, being too large, or for not being made of just letters.

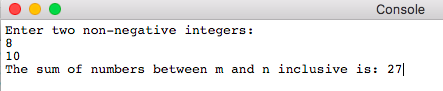
Results (and screenshots of program results)

Assignment 1 – code attached

Below: the program rejects any negative number



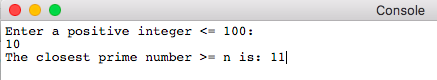
Below: Showing the added result (8+9+10 = 27)



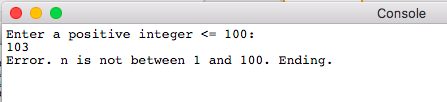
I tested the program for other values as well and the sum of the numbers between m and n inclusive is correct.

Assignment 2 – code attached

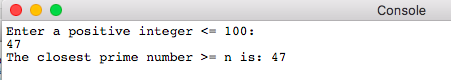
Below: example of program finding closest prime number

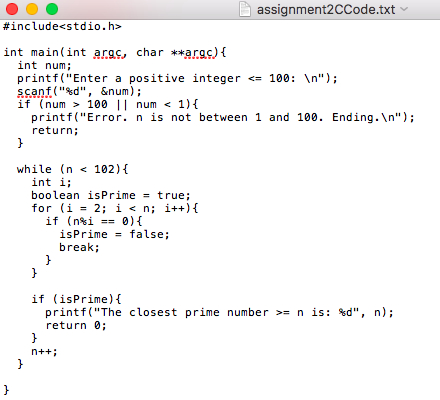


Below: example of program rejecting number above 100 (rejects below 1 as well)



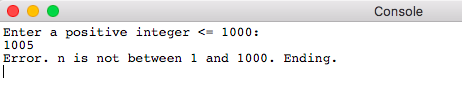
Below: example of program knowing number entered is prime



Assignment 2 C code/pseudocode:

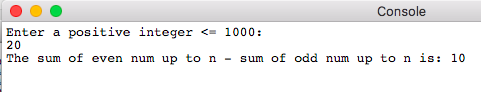
Assignment 3 – code attached

Below: program ending if not in the range



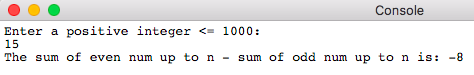
Below: Program running with n = 20

Check: (2+4+6+8+10+12+14+16+18+20)-(1+3+5+7+9+11+13+15+17+19) = 10



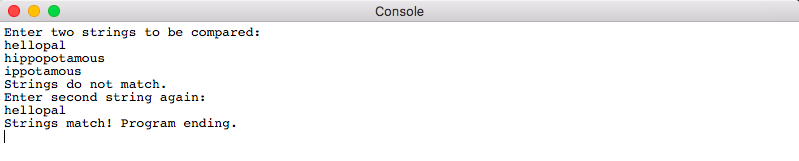
Below: Program running with n = 15

Check: (2+4+6+8+10+12+14)-(1+3+5+7+9+11+13+15) = -8

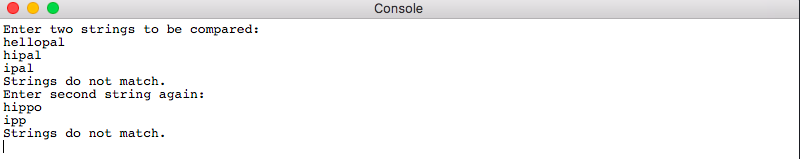


Assignment 4 – code attached

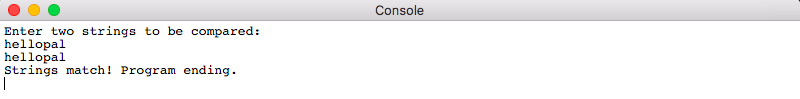
Below: not matching at first, then matches on second time



Below: not matching twice and program ending

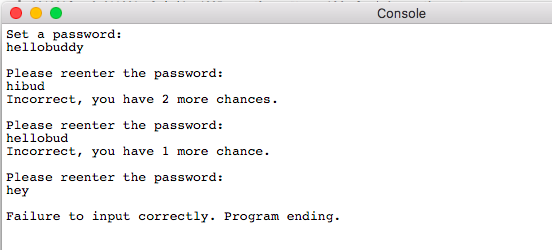


Below: two strings matching immediately

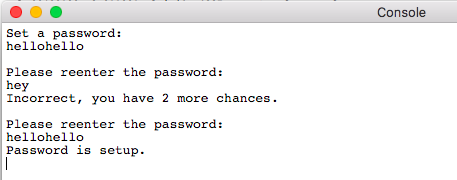


Assignment 5 – code attached

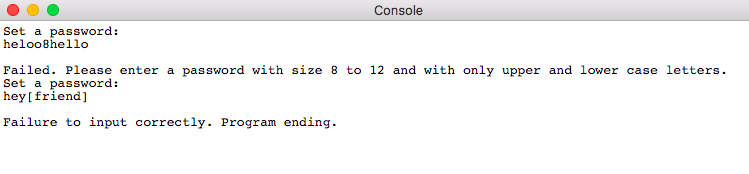
Below: First password entry fit criteria, and three failed reentries



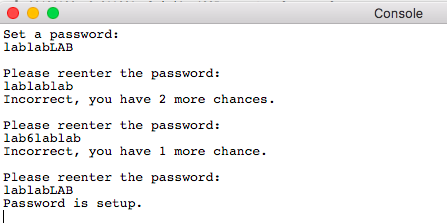
Below: Good first password setting, then one incorrect reentry before correctly entered



Below: failure twice to enter in good password because of non-letter entry, program ended



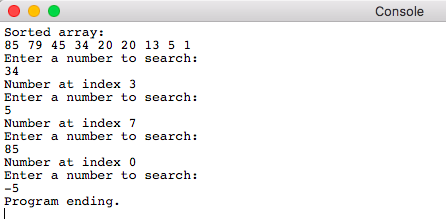
Below: Good password and two failed reentries before correct entry



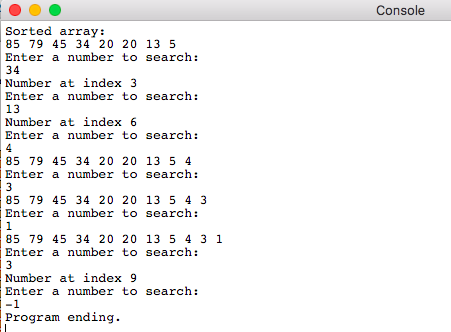
Assignment 6 – code attached

Original array 🡪 array: .word 45, 85, 34, 79, 13, 20, 20, 5, 0xF

Below: sorted array displayed and searching for numbers present working, ending at negative number entry



Below: Searching and inserting numbers



Conclusion

Assignment 1 required user input for the bounds of the sum and the use of loops to sum up all the numbers in between the bounds. Then, the result was outputted using syscall and $v0 as well. In Assignment 2, conditionals were used to make sure the number was within the specified range of 1-100 inclusive, and then the closest prime was found by looping from that number until 102 (because 101 is the nearest prime number to 100) and dividing the numbers by all numbers before to check if it was prime. Assignment 3 required using “jal” to go to loops and return using “jr $ra” because of the use of multiple loops, one for adding all even numbers and one for all odd numbers. Assignment 4 taught how to get specific characters from a string and how to traverse the string. Assignment 5 continued with string manipulation, but included criteria to check for other than mismatches characters. It required checking the char ascii ranges to make sure the characters in the password string were only letters. Then, the password has to be matched and three overall chances were given to let the user enter it again. Finally, Assignment 6 taught how to sort an array using two for loops and swapping with the sw command, how to search through an array, and how to insert into an array. Overall, I learned string manipulation, array searching and adding, and prime checking. I also interacted with the user by taking in user inputs and by outputting commands and status updates.