CS224 - Fall 2019 - Lab #3 (Version 3: October 31, 23:34)[1]

MIPS Assembly Language Programming
Floating Point Numbers, Recursion, Linked Lists

Dates: Section 1, Wednesday, 6 November, 13:40-17:30
Section 2, Friday, 8 November, 13:40-17:30
Section 3, Wednesday, 6 November, 8:40-12:30
Section 4, Thursday, 7 November, 13:40-17:30
Section 1, 2, 3, 4 Prelim. Report Deadline: Wednesday November 6, 10:40
Lab Location: EA-Z04

NO LATE SUBMISSION IS ACCEPTED

Purpose: More experience with MIPS assembly language programming using recursion, bit manipulation, and linked list operations. A small review for floating point number representation. Gaining experience in MIPS program analysis by adding more to the utilities/subprograms of an existing program.

You are obliged to read this document word by word and are responsible for the mistakes you make by not following the rules. Your programs should be reasonably documented and must have a neat syntax in terms of variable names and spacing. In all labs if it is not specified assume that inputs are always correct.

Summary
Part 1 (30 points): Learning the principles of recursive programming and the implementation of dynamic link list data structure by program analysis in MIPS assembly language. Some experience with floating point numbers.
Part 2 (70 points): More experience with linked lists and recursive programming.

DUE DATE OF PART 1: SAME FOR ALL SECTIONS:

a. Please bring and drop your hardcopy (printed copy) of the preliminary work into the box(es) provided in the lab by 10:40 on Wednesday November 6, 2019.

Please **upload your programs of Part 1 (PRELIMINARY WORK)** to the Unilica by 10:40 on Wednesday November 6. Use filename **StudentID_FirstName_LastName_SecNo_PRELIM_LabNo.txt** Upload only the programs. Only a NOTEPAD FILE (txt file) is accepted. Your paper submission for preliminary

work must match MOSS submission. Any format other than txt gets 0 points.

a. To get credit for preliminary work you have to submit its hard copy and upload its txt version to unilica. **No late submission will be accepted**.

DUE DATE PART 2: (different for each section) YOUR LAB DAY

You have to demonstrate your lab work to the TA for grade by **12:15** in the morning lab and by **17:15** in the afternoon lab. Your TAs may give further instructions on this. If you wait idly and show your work last minute, 20 points may be taken off from your grade.

At the conclusion of the demo for getting your grade, you will **upload your entire program work** to the Unilica Assignment, for similarity testing by MOSS. See Part 3 below for details.

If we suspect that there is cheating, we will send the work with the names of the students to the university disciplinary committee.

Information about Linked List implementation in MIPS

Linked lists are important dynamic data structures, useful to a variety of algorithms because of their ability to grow and shrink. Utilities / libraries for linked lists are therefore useful, so that common functions can be available to users, pre-written, tested and ready to go. In this lab, you will write the utility functions defined only in this assignment for linked lists, in MIPS assembly language. Your utilities will be combined with other utility programs such as create_list and display_list, and called by a main program. [Any code other than the you are asked to write in this lab will be provided (see the Unilica folder)—you don't need to write it].

In memory, the linked lists your utilities will work with are implemented as follows: each element consists of 2 parts: pointerToNext, and value. Each part is a 32-bit MIPS word in memory. The two parts are located in successive word addresses, with the pointerToNext being first. For example, if the byte address of the pointerToNext is 100, then the byte address of the value will be 104. Remember, MIPS memory is byte addressable.

In the last element of the linked list, the pointerToNext has value 0, in other words it is the null pointer. This means that there is not any next element. The last element still has a value.

Write only the linked list utility routines as defined in Part 1 and Part 2. Follow MIPS programming conventions in terms of using the stack and registers. Ignore the other methods not implemented. Modify the menu to support the new subprograms.

In both parts when needed you have to follow the conventions of professional MIPS programmers and use stack when needed.

For the linked list parts only upload the utilities that you have implemented do not upload the parts provided to you.

Part 1. Preliminary Work / Preliminary Design Report (30 points)

You have to provide a neat presentation prepared by Word or a word processor with similar output quality such as Latex as you were asked in Lab 1. At the top of the paper on left provide the following information and staple all papers. In this part provide the program listings with proper identification (please make sure that this info is there for proper grading of your work, otherwise some points will be taken off).

CS224
Section No.: Enter your section no. here
Fall 2019
Lab No.
Your Full Name/Bilkent ID

1. **(5 points: 2 + 2 + 1) Floating Point Numbers Problem Solving**:

   For each show your work briefly as suggested below.

a. Convert the number - 77.125 (do not miss the minus sign) to IEEE 754 standard. Show how you obtain mantissa and exponent. Give the final answer in hex.
   Single precision representation:
   Double precision representation:

b. Convert - 77.125 to a hypothetical floating point representation. Give the final answer in hex. For both cases show how you obtain mantissa and exponent.

   For single precision use bias of 120:
   For double precision use bias of 1020:

c. Consider the following 32 bit hexadecimal number 0xc1a00000. Assuming that it is a Floating point number give its decimal equivalent. Show how you obtain mantissa and exponent.

2. **(10 points) recursiveSummation**: Write a **recursive** MIPS subprograms to add the digits of an .asciiz string. Assume that the string contains an integer number. For example, for

   string: .asciiz "1204"

it returns 7.

Provide a main program to properly test the subprogram. Note that this program is not related to linked list processing.

3. **deleteAfter_x (15 points)**: Study the linked list program provided. Write a **non recursive** subprogram, i.e. extend the given linked list program. to delete the element from the linked list that follows the node that contains the data value x: the pointer to the linked list is passed in $a0, and the integer value of the element to be deleted is provided in $a1. Note that this operation must be done for all occurrences of the x value. Assume that no two or more x value can appear consecutively. Return the number of elements deleted from the linked list in $v0 and print it after the operation.

   For example for a linked list containing the values 2, 7, 4, 8, 4, 9, 10 if the x value is 4 the linked list becomes 2, 7, 4, 4, 10. For the same original input string if the value of x is 10, the linked list remains the same.
   Are you able to return the deleted node back to the heap? If not include a comment in the program to explain why.

Part 2. Lab Work: Writing MIPS assembly language programs (70 points)

1. (15 points) checkPattern: Write a **non recursive** subprogram to count the bit pattern stored in the rightmost n bits (window of size n bits) of a given input. Following are the inputs for the problem:

$a0 - pattern to search, for example 101 (stored as 0000000.....00101 in $a0)
$a1 - input to search, for example 10000111101101000101000101101000
$a2 – n, for example 3

Note that, there is no need to check the validity of n (i.e., assume that $a2 is a valid input between 1 and 32. The search in $a1 must be performed from right to left. During pattern matching, bit pattern windows **cannot** overlap. For example, for the above sample input your program will divide the bit representation in $a1 as follows:

10000111101101000101000101101000 -> 10 000 111 101 101 000 101 000 101 101 000

The number of windows matching the given pattern ($a0, 101 for this example) is 5. Starting from right (least significant bits),

window 1 - 000 - not matching
window 2 - 101 - matching
window 3 - 101 - matching
window 4 - 000 - not matching
window 5 - 101 - matching
window 6 - 000 - not matching
window 7 - 101 - matching
window 8 - 101 - matching
window 9 - 111 - not matching
window 10 - 000 - not matching
window 11 - 10 - cannot match by definition

For n=4 the maximum match possible is 8, since you would first check the rightmost four bits then four bits that come before rightmost four bits, etc. For n=2, maximum match can at most be 16, for n=3 it can be at most 10.

Provide a main program to properly test the subprogram and be sure it works for different patterns and inputs to search. Note that this program is not related to linked list processing.

2. (15 points) reverseString: Write a **recursive** subprogram, called reverseString, to copy an asciiz string pointed by $a0 into another asciiz string pointed by $a1 of the same size.

string1: .asciiz "123"
string2: .asciiz "abc"
When we invoke the subprogram after executing
la $a0, string1
la $a1, string2
jal reverseString
string2 contains "321"

Provide a main program to properly test the subprogram. Note that this program should also work with inputs with sizes different than 3. Also, not that this program is not related to linked list processing.

3. Insert_n (10 points): Study the linked list program provided. Write a **non recursive** program to insert an element to the linked list as the nth element: n= 1 means that the new item becomes the new list head, if n is higher than the list length the new element becomes the new last element of the linked list. The pointer to the linked list is passed in $a0, the integer value of the new element to be inserted is provided in $a1 and the position to be inserted is provided in $a2. This utility will request space in memory from the operating system, use it to create a new element, and then insert the new element correctly into the linked list. The returned value in $v1 contains the pointer to the head of the linked list. Also note that the list can be empty before the insertion.

4. duplicateListIterative (15 points): Study the linked list program provided. Write a **non recursive** subprogram to duplicate a linked list. When called $a0 points to the original list. It returns the new list

head in $v0. Program also should be printing the new list, including the information of pointerToNext address and value of each node, like the given display function.

5. duplicateListRecursive (15 points): Study the linked list program provided. Write a **recursive** subprogram to duplicate a linked list. When called $a0 points to the original list. It returns the new list head in $v0. Program also should be printing the new list, including the information of pointerToNext address and value of each node, like the given display function.

Part 3. Submit your code for MOSS similarity testing

1. Submit your MIPS codes for similarity testing to the Unilica > Assignment specific for your section.
2. You will upload one file. Use filename **StudentID_FirstName_LastName_SecNo_LAB_LabNo.txt**
3. Upload only the programs, the parts, you have written.
4. Only a NOTEPAD FILE (txt file) is accepted.
5. Be sure that the file contains exactly and only the codes which are specifically detailed Part 1 and Part 2, including Part 1 programs (your paper submission for preliminary work must match MOSS submission). Check the specifications! *Even if you didn't finish, or didn't get the MIPS codes working, you must submit your code to the Unilica Assignment for similarity checking.*
6. Your codes will be compared against all the other codes in the class, by the MOSS program, to determine how similar it is (as an indication of plagiarism). So be sure that the code you submit is code that you actually wrote yourself!
7. All students must upload their code to Unilica > Assignment while the TA watches.
8. Submissions made without the TA observing will be deleted, resulting in a lab score of 0.

Part 4. Cleanup

1. After saving any files that you might want to have in the future to your own storage device, erase all the files you created from the computer in the lab.
2. When applicable put back all the hardware, boards, wires, tools, etc. where they came from.
3. Clean up your lab desk, to leave it completely clean and ready for the next group who will come.