CSE 520

Project 1

Due Date: Wednesday 11, 2018, 11:59 PM

Objective: Learn to program in MIPS assembly language

In this project, you will be required to write assembly code to implement the following four problems. In all the programs, you are required to strictly adhere to the register conventions, as stipulated by the MIPS architecture. Please refer to the lecture slides, or the ISA reference manual, for the same. Please read the Submission Instructions carefully.

Problem 1: String handling [1 point]

Declare a string in the data section:

```
.data
        string:     .asciiz     "Welcome to Computer Architecture Class!"
```

Write a program that converts every other character in a string to upper case, starting with the first character. To convert a lowercase character to uppercase, you can subtract **0x20** from that character in the string, or vice-versa. Executing your program should display the converted string on the I/O console of the MARS simulator. Make sure that you ignore punctuation marks and spaces.

Although this problem asks you to evaluate this specific string, your program should be generic enough to convert any random string from upper-case to lower-case. For example, your assembly program would display the same string, if the defined string is already uppercase.

Problem 2: Arithmetic expressions [1.5 points]

Write a program to evaluate the following function of u and v:

$$\mathbf{5u^2 + 2uv - 2v^2 + 9}$$

Here, the variables u and v, are user inputs, and the program should receive them from the user at run-time. Then it should print the outcome, computed as per given arithmetic expression. [To understand how to take the user input or write on console, please refer to the sample MIPS example programs].

Your `main` function (or program) would operate on the values of u and v, computing outcome of above equation. To compute terms such as u^2 or `uv`, you would pass the necessary arguments to the corresponding subroutines. You will create and use two subroutines:

1) `int Square (int a):Return` a^2.

2) `int Multiply (int a, int b): Return a * b.`

These subroutines can be implemented using MIPS assembly instructions (e.g. MUL, ADD), any pseudo instructions supported by MARS, or their combinations, for computing the square and multiplication. Your `main` function should invoke these subroutines, with needed arguments. For the sake of simplicity, you can assume that the resultant value is always smaller than the largest 32-bit integer value.

Pay attention to the registers you use for passing arguments to the *subroutine*, and to the registers used for returning the output from a *subroutine*.

Problem 3: Pointers [2.5 points]

Write a program in MIPS assembly language that will compute the sum of all the elements in an array. Write this program using a function "updateSum," that takes two parameters -- a pointer to the running total, and a pointer to the current element. (To get help on how to deal with the pointers in MIPS assembly, refer to the sample MIPS examples.)

The "C" program looks like this:

```
int sum = 0; int *sumPtr = &sum;

int array[10];

void updateSum(int *total, int *element)

{

        *total += *element;

}

int main()

{

        for (int i=0; i<10; i++)

        {

                array[i] = 2*(i+1);

        }

        for (int i=0; i<10; i++)

        {

                updateSum(sumPtr, array+i);

        }

        printf("Sum = %d", sum);

}
```

Problem 4: Recursion [3.0 points]

Write a program in MIPS assembly language to find `compute(i, x)`, where `compute (i, x)` is defined recursively as:

```
int compute (int i, int x)
{
    if (x>0)
        return compute(i, x-1) + 1;
    else if (i>0)
        return compute(i-1, i-1) + 1;
    else
        return 1;
}
```

Note: Your program should print on the console the computed value `compute(i,x)`. The values for variables `i` and `x` are user inputs to the program. So, your MIPS program should allow the user to enter values for variables `i` and `x` at run-time. User can enter any integer values (32-bit arithmetic), including negative numbers.

Problem 5: Linked List Insertion [3.0 points] (For 520 students only)

Write a subroutine which inserts an element into a sorted linked list. The 1st parameter is the pointer to the element you want to insert, and the 2nd is a pointer to the head of the list. Code the subroutine based on this C structure(Note: for null pointer input i.e a 0-element array, have the subroutine return a pointer to the head element. You can assign the null pointer to any statically defined area allocated in the .data area). The following C code should provide a good enough template for you:

```
struct element{
    int num;
    struct element* next;
}
element* insert(element* head, element* value)
```

Submission Instructions

1) Please make one assembly file for each of the given problems. Save your program files with .asm extension.

2) Zip all the files into one file and submit it to the blackboard. Name the zip file Project1.zip. In case where you are doing project in a group, only one group member should make submission through blackboard.

3) There should be no name or ASU ID number of any of the team members in any part of the file or program for the project submission. If a team violates this policy, the score for the project will be ZERO. In short, the submission must be anonymous for a double blinded scoring process.