CS 270 Assignment-03
Fall 2024
Points: 20
Due: 11/06/2024

***Objectives*:** The objective of this assignment is to develop proficiency with functions, arrays, memory management, and the formatting of array contents for clear output. You will accomplish this by translating the provided C code, following the systematic approach of a compiler as discussed in class. Ensure that your translation aligns closely with compiler standards to reinforce your understanding of these foundational concepts.

***Deliverables*:** For this assignment, you are required to write MIPS programs for both problems listed below and submit each solution as an individual *.asm* file on Canvas. After executing and testing your code in MARS, please capture and include screenshots of the *.data* segment, *.text* segment, and I/O display, as done for Assignment 02. Ensure that all style guidelines are closely followed for clarity and consistency. Submit these screenshots in a PDF along with your *.asm* files to complete the assignment.

Problem-01 (*6 points*):

Write a MIPS assembly language program to calculate the product of two integers, `A` and `B`, that are stored in memory. The program should use a loop structure to achieve the multiplication through repeated addition, as multiplication instructions (such as `mult`, `multu`, `mul`, `mulo`, or `muluo`) are **not** permitted in this exercise. Instead, think of multiplication as the repeated addition of `A` for `B` times, like the mathematical example 2×3=2+2+2.

The result of this operation should be stored in memory at the label `C`. Document your code clearly to make each step understandable, as this exercise reinforces concepts around loops, memory access, and arithmetic operations in assembly language.

The following pseudocode demonstrates the logic for the multiplication process:

```
i = 0;
C = 0;
while (i != B) {
    C = C + A;
    i = i + 1;
}
```

In this code:

- A and B are the inputs.
- C is the variable that will store the product of A and B.

Problem-02 (*14 points*)

Write a MIPS assembly program that sorts a list of integers in ascending order using the ***bubble sort*** algorithm. This exercise will help you develop skills in array manipulation, loops, and conditional statements in assembly language.

Problem Description

The program should prompt the user to input a series of integers. The first input will specify the number of integers to be sorted (we'll call this `n`). This will be followed by `n` integers, each provided on a new line. The program must read these inputs, store them in an array, and then sort the array using bubble sort. Finally, the program should display the sorted array in ascending order.

Requirements

Input Format:

- The first input is an integer `n`, which represents the number of elements in the array.
- The next `n` inputs are the integers to be sorted, each separated by a newline.

Output Format:

- The program should display a single output line that lists the sorted integers in ascending order. The output should be formatted as follows:

  ```
  The elements sorted in ascending order are: -7, 1, 2, 2, 4, 5, 16
  ```

- Ensure that each element is separated by a comma, and there is no trailing comma at the end.

Constraints:

- You may assume there will be at least 1 and no more than 20 integers to be sorted.
- Each integer will fit within a signed 4-byte (32-bit) integer.
- Your program must be able to handle duplicate values in the array.

Sorting Algorithm:

- Implement the **bubble sort** algorithm as shown in the C code provided below. This means that:
  - You should repeatedly pass through the array, comparing each pair of adjacent elements.
  - Swap the elements if they are in the wrong order (i.e., if the left element is greater than the right element).
  - Continue this process until the array is fully sorted.
- You **must** create a function in MIPS that performs the bubble sort. This function should not call any other functions and must operate independently within the program.
- The array should be stored in the static data section.

Sample Input and Output

Sample Input:

```
7
2
5
1
-7
2
4
16
```

Sample Output:

```
The elements sorted in ascending order are: -7, 1, 2, 2, 4, 5, 16
```

Guidelines

Array Declaration:

- Declare space for the array in the static data section (`.data`) with enough room to store up to 20 integers.

Bubble Sort Function:

- Implement the bubble sort function as described in the C code provided below:

```
void bubble_sort(int arr[], int len) {
    int i, j, tmp;
    for (i = 0; i < len - 1; i++) {
        for (j = 0; j < len - 1 - i; j++) {
            if (arr[j] > arr[j + 1]) {
                tmp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = tmp;
            }
        }
    }
}
```

I/O Operations:

- Use appropriate syscalls to read integers from standard input and write the sorted array to standard output.
- Ensure the output format matches the requirements, with proper spacing and comma separation.

Program Flow:

- The program should follow these steps:
  - Read the integer `n` (number of elements).
  - Read `n` integers into an array.
  - Sort the array using bubble sort.
  - Display the sorted array in the specified format.

Additional Notes

- Make sure to document your code, explaining each step clearly for readability.
- Use meaningful register names or comments to indicate their purpose, such as using `$t0` for `i`, `$t1` for `j`, and `$t2` for `tmp` during swaps.
- Consider edge cases, such as arrays with only one element or arrays containing duplicate values.