Indian Institute of Technology Kharagpur

AUTUMN Semester, 2019
COMPUTER SCIENCE AND ENGINEERING

Computer Organization Laboratory

Assignment-5: MIPS-32 Assembly Language Programming

Full Marks: 25

Time allowed: 6 hours

INSTRUCTIONS: ATTEMPT BOTH PROBLEMS. Make one submission per group of your source code on Moodle. Name your submitted source files following the format `Assgn_5_Prob_1_Grp_<Group_no>.s` **(e.g.** `Assgn_5_Prob_1_Grp_25.s`**), etc. Inside each submitted file, there should be a clear header describing the assignment no., problem no., semester, group no., and names of group members. Liberally comment your code to improve its comprehensibility.**

1. [**Recursive GCD Calculation in MIPS-32**] Write a complete MIPS-32 program to calculate and display the GCD of two non-negative integers collected from the user, using a recursive function `find_gcd`. After the input numbers are collected from the user, there should be sanity checking to ensure that the integers are non-negative using a function `check_non_negative_values`; if they are not, print an error message and exit. Otherwise, print the GCD from inside the `main` function with a proper message. **Follow all usual register usage conventions for recursive and non-recursive MIPS-32 function calls**. (10 marks)

2. [**Quicksort in MIPS-32**] Write a complete MIPS-32 program to collect an array of eight integers by the user, and then perform `Quicksort` to sort the array. **Collect the numbers from the input console using a loop inside a function** `form_array`**, and store in memory in an array called "**`array`**". Do not store the numbers as scalars in eight different non-contiguous locations or in eight different registers.** In your code, have all the usual functions like the (recursive) `Quicksort`, the `Partition` routine, etc. After sorting, print the sorted array on the console with a proper message. **Follow all usual register usage conventions for recursive and non-recursive MIPS-32 function calls.** (15 marks)

   The original Quicksort algorithm is given here for your reference.

```
function QuickSort(A, n)
    Quicksort'(A, 1, n)
end function
```

```
function QuickSort'(A, p, r)
    if p ≥ r then
        return
    end if
    q = Partition(A, p, r)
    Quicksort'(A, p, q − 1)
    Quicksort'(A, q + 1, r)
end function
```

```
function Partition(A, p, r)
    x ← A[r]
    i ← p − 1
    for j ← p to r − 1 do
        if A[j] ≤ x then
            i ← i + 1
            Exchange A[i] and A[j]
        end if
    end for
    Exchange A[i + 1] and A[r]
    return i + 1
end function
```