ELL305 Computer Architecture
Assignment 1
Sorting in SimpleRisc

August 11, 2019

In this assignment you have to implement three sorting algorithms - bubble sort, quick-sort and merge-sort using the SimpleRisc assembly language. You have to sort in the **ascending order**. Please note that this is an automatically graded assignment, and failure to adhere to the instructions will result in zero marks.

1 Description

- Download the assignment and extract it.
- The folder "assignment1" contains two subfolders: "emulator" and "examples". The folder "emulator" has the source code for the emulator (interpreter.c), a Readme.txt and three skeleton SimpleRisc assembly files - *bubblesort.asm*, *quicksort.asm* and *mergesort.asm*. All your changes are to be limited to *these three files*. **DO NOT** change any other file. You have to replace "ADD YOUR CODE HERE" with your implementation of each of the three sorting algorithms. You have to sort in the **ascending order**. The folder "examples" contains a few sample assembly codes for your reference. Please go through the Readme.txt file before starting the assignment.
- Quick sort and merge sort have to be **recursive**. Any other implementation will not be accepted.
- The provided code loads a set of numbers into memory. You are free to place your own numbers instead of the ones we have provided, and add more numbers to contiguous memory locations. Each integer occupies four bytes, and hence whenever a new number is added, it has to be stored at the $<$ *previous memory location* $>$ $+4$. However, you should not change the memory locations where these numbers are stored. We will test your implementation with up to **100** numbers, i.e., N = 100.

- Do not leave any extra print commands in the final version of the code that you submit. Since the assignments are autograded, any print statements other than the ones we have provided will lead to the submission being marked incorrect.
- Please make sure that you do not add any label after the ".main" function. ".main" has to be the last function in your assembly file, else the emulator may show incorrect results. So, limit your code changes only in the region marked as "ADD YOUR CODE HERE".
- Run the following commands:
  - `gcc interpreter.c -o interpreter`: to compile the interpreter
  - `./interpreter <path_to_assembly_file>`: to run the emulator on your implementation. This will print the output on the terminal. Example: `./interpreter bubblesort.asm`

1.1 Sample File and Output

Here we describe a sample input file for quick-sort and its expected output. The sample input file looks like this. Let us assume we want to sort N numbers.

```
.quicksort:
     @ ADD YOUR CODE HERE
 .main:
  @ Loading the values as an array into the registers
  mov r0, 0
  mov r1, 12 @ replace 12 with the number to be sorted
  st r1, 0[r0]
  mov r1, 7  @ replace 7 with the number to be sorted
  st r1, 4[r0]
  mov r1, 11 @ replace 11 with the number to be sorted
  st r1, 8[r0]
  mov r1, 9  @ replace 9 with the number to be sorted
  st r1, 12[r0]
  mov r1, 3  @ replace 3 with the number to be sorted
  st r1, 16[r0]
  mov r1, 15 @ replace 15 with the number to be sorted
  st r1, 20[r0]

  @ EXTEND ON SIMILAR LINES FOR MORE NUMBERS

  mov r2, 0        @ Starting address of the array

  @ Retreive the end address of the array
  mov r3, 5  @ REPLACE 5 WITH N-1, where, N is the number of numbers
      being sorted
  mul r3, r3, 4
  add r4, r2, r3
```

```
@ ADD YOUR CODE HERE
call .quicksort

@ ADD YOUR CODE HERE

@ Print statements for the result
ld r1, 0[r0]
.print r1
ld r1, 4[r0]
.print r1
ld r1, 8[r0]
.print r1
ld r1, 12[r0]
.print r1
ld r1, 16[r0]
.print r1
ld r1, 20[r0]
.print r1
@ EXTEND ON SIMILAR LINES FOR MORE NUMBERS
```

In this example, you will place your implementation of quicksort in the function `.quicksort`. The main function calls the function `.quicksort`. The final set of print statements would print the sorted values on the terminal. You do not have to modify these lines. You may, however, add more lines depending on the number of numbers being sorted. In this case, your code should print:

```
r1: 3
r1: 7
r1: 9
r1: 11
r1: 12
r1: 15
```

Please make sure that you ***set up the stack properly***, before and after a function call, so that the return addresses are not overwritten.

1.2 Submitting Your Assignment

- We will be evaluating your submission with a different set of input numbers. So make sure your implementation works for all possible cases.
- Create a folder named as <EntryNumber> which contains the three assembly files. Compress the folder and submit only the <EntryNumber>.zip archive on Moodle. For example if your entry number is 2017EE51010, then submit as 2017EE51010.zip. No other format will be accepted.

  To summarize, you have to do the following steps on Windows:

  1. Create a folder named <EntryNumber>.

2. Copy bubblesort.asm, quicksort.asm and mergesort.asm in this folder.
3. Create the .zip file
   - *For Windows*: Select the three assembly files, right-click and point to "Send to" and then select Compressed (zipped) folder. Name the folder as <EntryNumber>.zip
   - *For Linux:* Select the three assembly files, right-click and point to "Compress" and then create the zipped folder with the name as <EntryNumber>.zip
4. Submit this zip file on Moodle.

- The deadline is **September 01, 2019, 23:59 hours**.

1.3 Grading Scheme

1. Marks division: Total = 50 marks
   Bubblesort: 14 marks
   Mergesort: 18 marks
   Quicksort: 18 marks

2. If your implementation is not working, you will get zero marks.

3. We will be taking a look at all the programs manually. The assembly codes will also be evaluated for style, indentation, and comments. If any code is found to be very messy, then we will **deduct 2 marks**.

4. Late policy: 20% penalty per day (rounded to the next day, i.e. 3 hours late implies 1 day late).

5. **Plagiarism**: We will run MOSS over all submissions. If any similarity to any other source is found, you will get a zero.

6. **Bonus Marks**: If anybody does this assignment in x86 or ARM assembly, then there will be a demo and a viva. Only if they can rigorously explain their implementation, they will be awarded marks for the assignment including a bonus (of +10 marks).

   However, if they are not able to answer the TAs questions, they will get a 0 overall (no chance of a resubmission).