ELL 782/ SIL 618 Computer Architecture
Assignment 1
August 31, 2021

In this assignment, you have to implement the binary search and quick-sort algorithms using the SimpleRisc Assembly language. In quick-sort, you have to sort the array in an **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 three subfolders: "emulator," "examples," and "tester". The folder "emulator" has the source code for the emulator (interpreter.c), a Readme.txt, and two skeleton SimpleRisc assembly files *binarysearch.asm* and *quicksort.asm*. All your changes are to be limited to these two files. DO NOT change any other file. You have to replace "ADD YOUR CODE HERE" with your implementation of each of the two algorithms. You have to sort in **ascending order**. Please go through the Readme.txt file before starting the assignment. We recommend that you first understand and implement some basic examples, and then you can start your implementation of binary search and quicksort. The tester folder contains the script for testing your code.
- Binary search has to be implemented on an **unsorted** array. You can use any sorting algorithm to sort the array.
- Quicksort has to be **recursive**.
- 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 auto-graded, 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 to 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 binarysearch.asm**
- **Emulator** There is a GUI-based simpleRisc emulator, written in Java, which can be downloaded here. You can use it for debugging purposes. Please note that the GUI version is to be used only for your reference. You should however, check that your **submitted code works correctly on the command-line version of the emulator** since the evaluations will be done automatically based on the command line version.

1.1 Sample File and Output

Here we describe a sample input file for binary search and its expected output. The sample input file looks like this. Let us assume we want to search an element in an array.

```
.binarysearch:
            @ 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

   @ Store the element to be searched in r0
```

```
mov r0,16

@Flag for storing the boolean result
mov r1, 0

mov r2, 0       @ Starting address of the array

mov r3, 6           @ REPLACE 6 WITH N, where N is the number of numbers being sorted

@ ADD YOUR CODE HERE

call .binarysearch

@ ADD YOUR CODE HERE

@ Print statement for the result
@ Boolean result is stored in r1
.print r1
```

In this example, you will place your implementation of binary search in the function .binarysearch. The main function calls the function .binarysearch. The final boolean result will be stored in register r1 and printed on the terminal. In this case, your code should print:

```
r1: 0
```

In the case of quicksort, your output 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 the possible cases.

- Create a folder named <EntryNumber> containing the two assembly files. Compress the folder and submit only the <EntryNumber>.zip archive on Moodle. For example, if your entry number is 2017EE51010, then submit 2017EE51010.zip. No other format will be accepted. There should be no directories in the folder.

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

1. Create a folder named <EntryNumber>.
2. Copy binarysearch.asm and quicksort.asm to this folder.
3. Create the .zip file
   - For Windows: Select the two 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 two 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 22, 2021, 23:59 hours**.

1.3. Grading Scheme

1. Marks division: Total = 25 marks

   Binary search: 10 marks

   Quicksort: 15 marks

2. If quicksort is not implemented recursively, you will be awarded zero marks for the whole assignment.
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: 33% 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 the submissions. If any similarity with any other source is found, you will get zero marks.