University of Nottingham Ningbo China

School of Computer Science

A Level 1 Module, Spring Semester 2020–2021

COMP1047: Systems and Architecture (AE1SYS)

Coursework (Assessment Weight: 50%)

Submission deadline: Wednesday 5^{th} May 2021, 23:59:59 GMT+8

---

1 Synopsis

This coursework is about MIPS programming, implementation and testing. You will implement and test your code which answers the **THREE (3)** questions below, before submitting them in the Moodle page. You should develop and test your code ONLY using the MIPS simulator adopted in the lab sessions in this course. The total mark of this coursework is [**50 MARKS**].

2 Deliverable

Submit the following **FOUR (4)** *uncompressed* files to COMP1047 Moodle page, where you will find a submission portal to be opened later.

1. `COMP1047CW-'YourName'-'YourID'-Q1.s` for the code corresponding to Question 1. Example: `COMP1047CW-JaneDoe-20219999-Q1.s`.

2. `COMP1047CW-'YourName'-'YourID'-Q2.s` for the code corresponding to Question 2.

3. `COMP1047CW-'YourName'-'YourID'-Q3.s` for the code corresponding to Question 3.

4. `COMP1047CW-'YourName'-'YourID'-readme.txt` for whatever you would like to tell the evaluator. Word count limit is 500, including all components. This item is *optional*.

Unable to follow the code naming convention would lead to mark deduction. Late submission rules apply, as indicated in the accompanying coursework issue sheet.

3 Plagiarism

You are gently reminded that we are at liberty to use plagiarism detection tools on your submission. Plagiarism will **absolutely** not be tolerated, and academic offenses will be dealt with in accordance with UNNC policy and as detailed in the student handbook. This means you may informally discuss the coursework with your classmates, but you must implement your own code and provide your own answers. **DO NOT copy and paste, or paraphrase from others.**

4 Assessment

1. This coursework has a total of 50 marks, which constitutes 50% of the module weight. Individual marks are shown along each questions.

2. We will assess your submissions largely based on your code execution results, among other criteria. Detailed evaluation rubrics for Questions 1 and 2 are provided in Appendices 1 and 2, respectively. Evaluation rubrics for Question 3 is provided in Question 3 problem description.

Question 1 [10 Marks]

Write a program in MIPS32 assembly language which takes three input arguments: register `A` will receive the initial address of a string, register `B` will receive a character, and register `C` will receive another character. Within the string `A`, your program will replace any occurrence of the character stored in register `B` by the character stored in register `C`. Finally, output the replaced string in register `A`. For example:

```
Input string in register A: common
Input character in register B: m
Input character in register C: t
Output string in register A: cotton
```


Question 2 [15 Marks]

Implement a MIPS32 program which prompts user two integer inputs x, y from the console and calculate the following expression in signed 32-bit arithmetic:

$$x^3 + 3x^2y + 3xy^2 + y^3$$

Note that you are NOT allowed to use pseudo-instructions with overflow checking for the calculation (e.g. you cannot use mulo). If an overflow occurs during any step of the calculation, you should print an error message instead, and stop the program.

Hint: You could simplify the expression before calculation. Please remember to test your program with a range of different inputs, for example: $x = 2$, $y = 3$; $x = -3$, $y = 4$; $x = 1000, y = 150000$; etc.

Question 3 [25 Marks]

Implement a MIPS-version of the `strncmp()` C function, as standardized by the ISO C standard.

Background

The format of `strncmp()` is defined below:

```
int strncmp(const char *s1, const char *s2, size_t n);
```

Quoted from the ISO description:

"The `strncmp()` function shall compare not more than n bytes (bytes that follow a NUL character are not compared) from the array pointed to by $s1$ to the array pointed to by $s2$. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type `unsigned char`) that differ in the strings being compared. Upon successful completion, `strncmp()` shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by s1 is greater than, equal to, or less than the possibly null-terminated array pointed to by s2 respectively."

Your Task

Given the following C code snippet:

```
1   #include <stdio.h>
2   #include <string.h>
3
4   int main() {
5
6     char s1[5] = STRING 1;
7     char s2[6] = STRING 2;
8     int n = N;
9
10    int result = strncmp(s1, s2, n);
11    printf("The result is %d.", result);
12
13    return 0;
14  }
```

Note: To execute the code snippet above, you may need to replace the RED colored content by syntactically correct C code. For example:

```
6       char s1[5] = "abcd";
7       char s2[6] = "abcde";
8       int n = 2;
```

Then the program should output

```
The result is 0.
```

Your are required to implement the MIPS assembly code that returns **EXACTLY** the same result as effectively calling the `strncmp()` function in line 10 of the code snippet above. You *should not* change any parts in the above code snippet, except the red colored content. No marks is awarded to this question if this requirement is not followed.

Input:
Once you execute your code, at the simulator console, by invoking the appropriate `syscall`, your program should intake a string in the following format (Px indicates Parameter x):


```
P1:STRING 1;P2:STRING 2;P3:N;
```

where the RED colored content should be replaced in the same way as the above C snippet. For example:


```
P1:"abcd";P2:"abcde";P3:2;
```

Output:
After your code is executed based on the above input, by invoking the appropriate `syscall`, at the simulator console, your program should output the corresponding `strncmp` result. For the example above, the output should be:

```
0
```

What happens if `strncmp()` *returns an error?*
At the simulator console, your program should output the following message below. Note that: you *do not* have to specify the error, but make sure your program generates *exactly* this error message whenever `strncmp()` generates any error message.

```
An error has occurred.
```

Resources

1. You may need to refer to the ASCII Table. Note: No "Extended ASCII Codes" need to be considered in this question.

2. Use `https://www.programiz.com/c-programming/online-compiler/` as the standard `strncmp()` behavior reference. You can copy and past the C code snippet in Page 4 into this online C gadget and run, to obtain the referential `strncmp()` results. Namely, your program should output *exactly* the same 'result' value (except errors) as the C code outputs in this online gadget.

3. Click here to refer to the formal definition of `strncmp()` function.

4. Feel free to post questions in Moodle Discussion Forum, or email us regarding your doubts for clarification. Especially, with doubts in this question, contact Dr. Heng directly.

Evaluation

1. Out of the 25 marks for this question, TEN (10) <input, output> test cases will be employed to evaluate the functionality of your program. Example test case:

   <P1:"abcd";P2:"abcde";P3:2;, 0>.

   Each correct answer is rewarded 2.5 marks. Each incorrect answer is rewarded 0 mark. Your output answer should follow exactly the output requirement given in this question, otherwise it would be deemed incorrect.

2. Unless otherwise specified, we will use `https://www.programiz.com/c-programming/online-compiler/` as the standard `strncmp()` behavior reference, to evaluate your program output.

3. Good luck! Enjoy MIPS Programming. Remember to submit your coursework on time.

APPENDIX 1

Rubrics Q1	Weight (100)	Zero (0%)	Poor (20%)	Pass (40%)	Good (60%)	Excellent (80%)	Outstanding (100%)
Basic Function	40	No answer or all normal test cases failed	Iterative implementation, errors in normal test cases	Iterative implementation, no error in normal test cases, errors in special test cases	Iterative implementation, no error in normal test cases, no error in special test cases, other MIPS usage errors	Iterative implementation, no error in normal test cases, no error in special test cases, no MIPS usage errors	Excellent level + special considerations in data/control hazards (Encourage self-explorations)
Prompt	20	No prompt	Basic prompt to allow user's input	Advanced prompt to guide user's input to ensure normal input cases	Advanced prompt to guide user's input to ensure normal input cases and special input cases	Advanced prompt to guide user's input and warn consequences of special input. Present informative message to let user know of special input results	Advanced prompt to guide user's input and warn consequences of special input. Present informative message to let user know of special input results. Ensure service availability
Documentation	20	No comment. Very poor coding style	Few comments. Poor coding style	Insufficient comments. Good coding style	Comments on key instructions. Good coding style	Clear comments to explain the logic flow. Good coding style	Professional comments explaining program information, input/output, design considerations, etc. Good coding style
Input Test	20	No input test	Evidence of attempted input test, but failed	Attempted input test, only ensuring normal input	Identify one type of special input, reply with corresponding prompt	Identify and handle two types of special input, reply with corresponding prompt	Identify and handle more than two types of special input. Reply with excellent prompt for user's next input. Exhibit intelligence

APPENDIX 2

Rubrics Q2	Weight (100)	Zero (0%)	Poor (20%)	Pass (40%)	Good (60%)	Excellent (80%)	Outstanding (100%)
Basic Function	40	No answer or all normal test cases failed	There is answer, but errors in normal test cases	Iterative implementation, no error in normal test cases, errors in abnormal test cases	There is answer, no error in normal test cases, no error in abnormal test cases, other MIPS usage errors	There is answer, no error in normal test cases, no error in abnormal test cases, no MIPS usage errors	Excellent level + special considerations in data/control hazards (Encourage self-explorations)
Prompt	20	No prompt	Basic prompt to allow user's input	Advanced prompt to guide user's input to ensure normal input cases	Advanced prompt to guide user's input to ensure normal input cases and abnormal input cases	Advanced prompt to guide user's input and warn consequences of abnormal input. Present informative message to let user know of abnormal results	Advanced prompt to guide user's input and warn consequences of abnormal input. Present informative message to let user know of abnormal results. Ensure service availability
Documentation	20	No comment. Very poor coding style	Few comments. Poor coding style	Insufficient comments. Good coding style	Comments on key instructions. Good coding style	Clear comments to explain the logic flow. Good coding style	Professional comments explaining program information, input/output, design considerations, etc. Good coding style
Input Test	20	No input test	Evidence of attempted input test, but failed	Attempted input test, only ensuring normal input	Identify one type of abnormal input, reply with corresponding prompt	Identify and handle two types of abnormal input, reply with corresponding prompt	Identify and handle more than two types of abnormal input. Reply with excellent prompt for user's next input. Exhibit intelligence