**Project 2 Report**

# Youssef Samwel

## [yo800238@ucf.edu](mailto:yo800238@ucf.edu)

EEL3801: Computer Organization

Due Date: 7/23/2023

# **Project Description**

# The goal of this project is to introduce students to algorithm development through string processing, which is one of the most common applications of algorithms and has a wide range of uses. Some of the applications of string processing include search indexing, text documents, regular expression (RegEx) operations, Unicode handling, HTTP protocol, terminal operations, and more.

# For Project 2, the technical objective was to count certain characters in a string based on the character set "KNIGHTS" (case insensitive). Initially, the students were required to write a program using MIP's assembly to achieve this objective. Subsequently, they were tasked with optimizing their program to reduce energy consumption and dynamic instruction count (execution time).

# **2.0 Program Design**

To simplify the testing process, this program has the input string inside the data section of memory instead of taking user’s input.

## Part A

## The execution begins at the start label, where we load the string address into $t0. Afterwards, we enter the "text loop," stepping through every character until NULL termination. For each character, we invoke the "count letter" subroutine to process the letter. The letter must be stored in $t1 for the subroutine.

## The "count letter" subroutine first converts the letter to lowercase and then loads the memory address of the character set "KNIGHTS" into $s0. We iterate through each character in the character set and compare the current letter to the letters in the character set. If the current letter matches any letter in the character set, we invoke the "valid charset letter" subroutine, where we increment the count of that specific letter in the counter integer array. This process repeats until the end of the string.

## Afterwards, we call the "print\_output" function with $t7 set to 1 to display the count of each letter in the console output. Then, we call the "print\_output" function again with $t7 set to 0 to display the count of each letter using a histogram format in the console output.

## Part B

Part B also achieves the same output as Part A but with a reduced dynamic instruction count and lower energy consumption. Execution begins at the start label, where we load the input string into $t7 and step through each character until NULL termination. Several optimizations are implemented in Part B.

The first optimization involves unrolling the character set loop. Since the "KNIGHTS" string is constant, we can unroll the charset loop, saving several instructions and branch instructions.

Secondly, we convert the character to lowercase to only perform half of the comparison instructions.

Thirdly, we optimize the lowercase conversion process. Typically, when converting a letter to lowercase, we must do a minimum of checks (two branches). However, we can reduce this to one branch because we only care if the character is equal to "knights." We do not care if the character value is erroneously converted to any other character, as long as it is not converted to any of the following characters: "knights." This lowercase procedure works by checking if the character is less than ASCII 'a'; if it is, we add 32 to the current character. This may produce incorrect results for characters with values less than ASCII 'A,' but that is irrelevant to our goal.

Finally, we no longer use the counter integer array to count the instances of each character; instead, we use registers $s0 to $s6 to count the letters. There was also a minor optimization in the display procedure by unrolling the loops.

A diagram of a computer program

Description automatically generated

Part A

A diagram of a process

Description automatically generated

Part B

# **3.0 Symbol Table**

### Part A

|  |  |
| --- | --- |
| Register | Purpose |
| $t0 | Hold memory address of sample\_text |
| $t1 | Hold characters in sample\_text |
| $t6 | Read current value in counter\_array when printing output |
| $t7 | As subroutine argument for print\_output |
| $v0 | For syscall id |
| $a0 | For syscall arguments |
| $s0 | Holds the character set address |
| $s1 | Hold the current character in character set |
| $s2 | As an index for counter\_array |
| $s7 | To load, add, and store values from counter\_array |

### Part B

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Register | Purpose | | $t0 | Used to hold memory address of sample\_text | | $t1 | Used to hold character's in sample\_text | | $t6 | Used to read current value in counter\_array when printing output | | $t7 | Used for as subroutine argument for print\_output | | $v0 | Used for syscall # invoke os to execute current function id | | $a0 | Used for syscall # invoke os to execute current function arguments | | $s0 | Count of letters K | | $s1 | Count of letters N | | $s2 | Count of letters I | | $s3 | Count of letters G | | $s4 | Count of letters H | | $s5 | Count of letters T | | $s6 | Count of letters S | | $t7 | Address of current character | | $s7 | Current character | |

# **4.0 Learning Coverage**

# **5.0 Prototype in C-Language**

# **6.0 Test Plan**

# **7.0 Test Results**

# **8.0 References**