Project 2 Report

Youssef Samwel

yo800238@ucf.edu

EEL3801: Computer Organization

Due Date: 7/23/2023

1.0 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

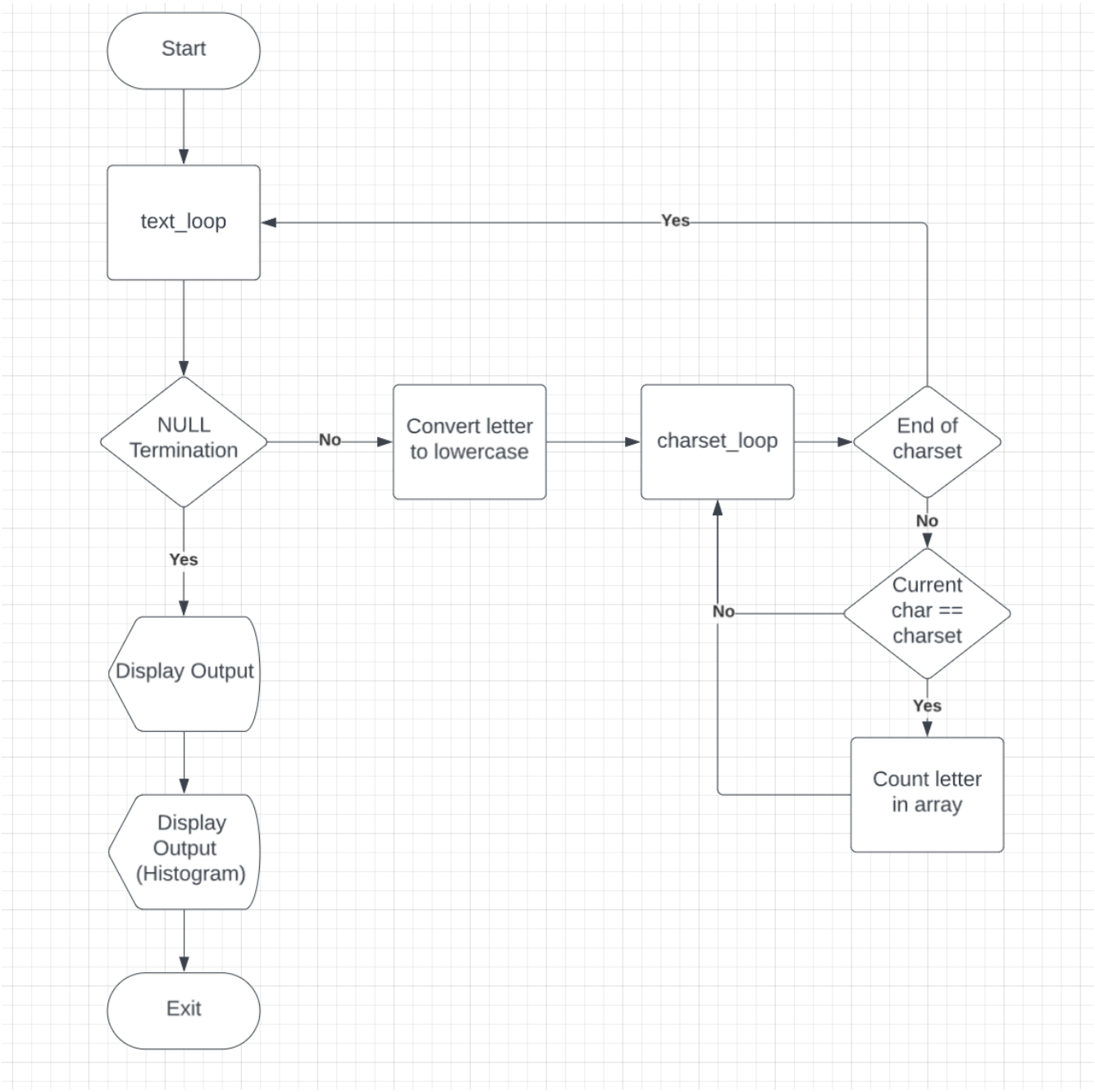
Part B also achieves the same output as Part A but with a reduced dynamic instruction count and lower energy consumption. Execution begins at the start label, where we load the input string into $t7 and step through each character until NULL termination. Several optimizations are implemented in Part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

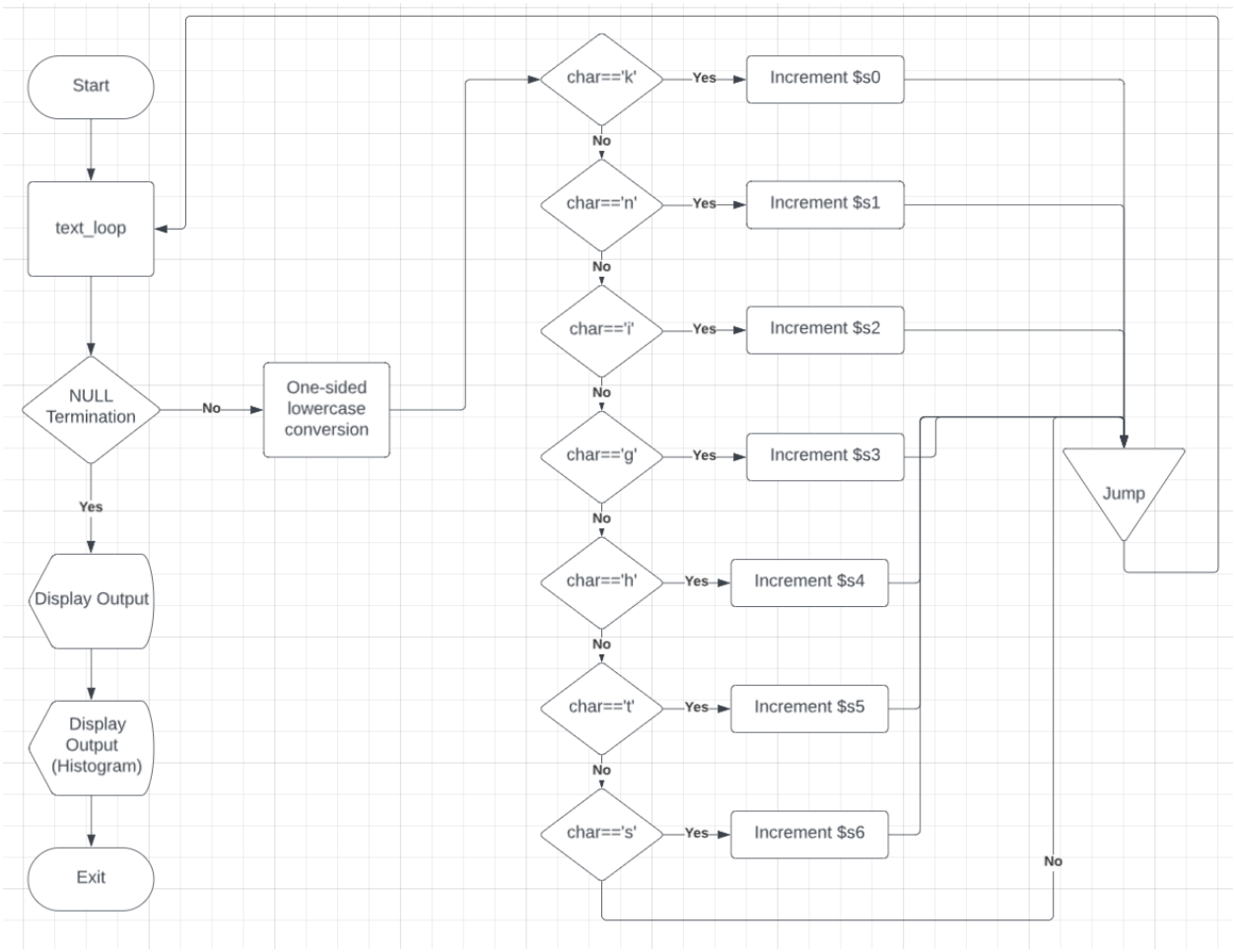
Secondly, we convert the character to lowercase to only perform half of the comparison instructions.

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.


Start
text_loop
Yes
NULL Termination
No
Convert letter to lowercase
charset_loop
End of charset
No
Yes
Current char == charset
No
Yes
Count letter in array
Display Output
Display Output (Histogram)
Exit

Part A



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

- String processing and String Manipulation
- MIPS Assembly Language
- Algorithm optimization
- Computing energy consumption
- Computing MIPS/mw
- Formatting program output
- Data and Storage
- Subroutine and Function Calls

5.0 Prototype in C-Language

Part A

```c
#include <stdio.h>

// Note: The MIPs assembly cannot be converted directly to C because the lack of access to
registers.
// however, this prototype attemps to be as close to the original MIPs code.

static const char* character_set = "knights";
int counter_array[7] = { 0 };

// count_letter:
void count_letter(char c) {
    if(c < 'a') {
        c += 32;
    }
    const char* address = character_set;
    char test_character = *address;
    while(test_character != '\0') {
        // valid_charset_letter:
        if(test_character == c) {
            counter_array[(int)(address - character_set)] += 1;
            break;
        }
        address = address + 1;
        test_character = *address;
    }
}
```

```c
// print_output:
void print_output(int histogram_mode) {
    if(histogram_mode == 0) {
        for(int i = 0; i < 7; i++) {
            printf("%c: %d\n", character_set[i] - 32, counter_array[i]);
        }
    } else {
        for(int i = 0; i < 7; i++) {
            printf("%c: ", character_set[i] - 32);
            int value = counter_array[i];
            for(int j = 0; j < value; j++) {
                putchar('#');
            }
            putchar('\n');
        }
    }
}

int main() {
    // const char* sample_text = "Lorem, ipsum.";
    // const char* sample_text = "Lorem ipsum dolor sit.";
    // const char* sample_text = "Lorem ipsum dolor sit amet, consectetur adipisicing elit.";
     const char* sample_text = "Lorem ipsum dolor sit amet consectetur adipisicing elit.
Harum maxime magni nostrum soluta est dolores ad?";
    // const char* sample_text = "Lorem, ipsum dolor sit amet consectetur adipisicing elit.
Modi quas, voluptatum laudantium incidunt voluptatibus blanditiis provident vitae voluptate
exercitationem recusandae ullam distinctio libero sit quos delectus vero nostrum ex
cupiditate ipsa atque.";

    const char* address = sample_text;
    char c = *address;
    while(c != '\0') {
        count_letter(c);
        address = address + 1;
        c = *address;
    }

    print_output(0);
    print_output(1);
}
```

Part B

```c
#include <stdio.h>

// Note: The MIPs assembly cannot be converted directly to C because the lack of access to
registers.
// however, this prototype attemps to be as close to the original MIPs code.
int register_file[7] = { 0 };
```

```
// count_letter:
void count_letter(char c) {
    if(c < 'a') {
        c += 32;
    }
    if(c == 'k') register_file[0]++;
    else if(c == 'n') register_file[1]++;
    else if(c == 'i') register_file[2]++;
    else if(c == 'g') register_file[3]++;
    else if(c == 'h') register_file[4]++;
    else if(c == 't') register_file[5]++;
    else if(c == 's') register_file[6]++;
}

// print_output:
void print_output(int histogram_mode) {
    if(histogram_mode == 0) {
        printf("%c: %d\n", 'K', register_file[0]);
        printf("%c: %d\n", 'N', register_file[1]);
        printf("%c: %d\n", 'I', register_file[2]);
        printf("%c: %d\n", 'G', register_file[3]);
        printf("%c: %d\n", 'H', register_file[4]);
        printf("%c: %d\n", 'T', register_file[5]);
        printf("%c: %d\n", 'S', register_file[6]);
    } else {
        printf("%c: ", 'K');
        for(int i = 0; i < register_file[0]; i++) putchar('#');
        printf("\n%c: ", 'N');
        for(int i = 0; i < register_file[1]; i++) putchar('#');
        printf("\n%c: ", 'I');
        for(int i = 0; i < register_file[2]; i++) putchar('#');
        printf("\n%c: ", 'G');
        for(int i = 0; i < register_file[3]; i++) putchar('#');
        printf("\n%c: ", 'H');
        for(int i = 0; i < register_file[4]; i++) putchar('#');
        printf("\n%c: ", 'T');
        for(int i = 0; i < register_file[5]; i++) putchar('#');
        printf("\n%c: ", 'S');
        for(int i = 0; i < register_file[6]; i++) putchar('#');
        putchar('\n');
    }
}

int main() {
    // const char* sample_text = "Lorem, ipsum.";
    // const char* sample_text = "Lorem ipsum dolor sit.";
    // const char* sample_text = "Lorem ipsum dolor sit amet, consectetur adipisicing elit.";
     const char* sample_text = "Lorem ipsum dolor sit amet consectetur adipisicing elit.
Harum maxime magni nostrum soluta est dolores ad?";
```

```
    // const char* sample_text = "Lorem, ipsum dolor sit amet consectetur adipisicing elit.
Modi quas, voluptatum laudantium incidunt voluptatibus blanditiis provident vitae voluptate
exercitationem recusandae ullam distinctio libero sit quos delectus vero nostrum ex
cupiditate ipsa atque.";

    const char* address = sample_text;
    char c = *address;
    while(c != '\0') {
        count_letter(c);
        address = address + 1;
        c = *address;
    }

    print_output(0);
    print_output(1);
}
```

6.0 Test Plan

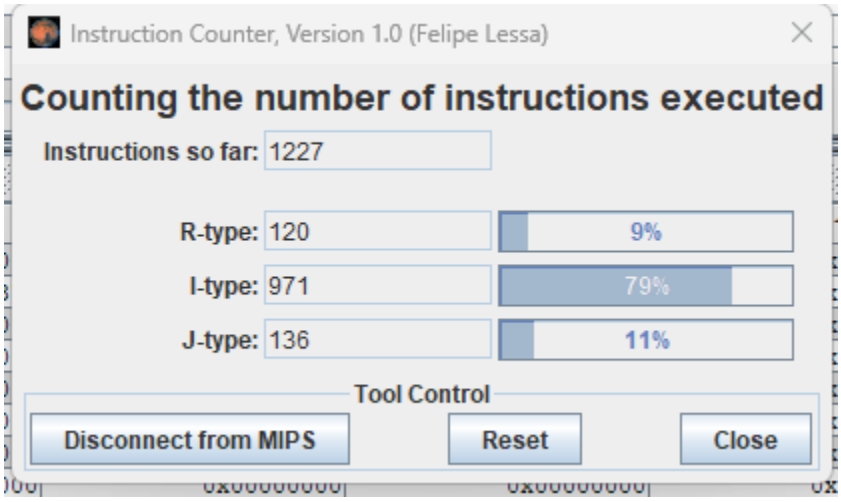
To test the performance uplift in Part B we must test the exact same string sequences in both versions. For an optimal test plan, we should be using a standardized text document such as a common book but for this lab, I used lorem ipsum generated text (2, 4, 8, 16, 32 words for each sentence respectively). We must compare the dynamic instruction count and energy consumption for Part A and Part B code.

Note the data collected in the test results is from MAR's MIPs emulator.

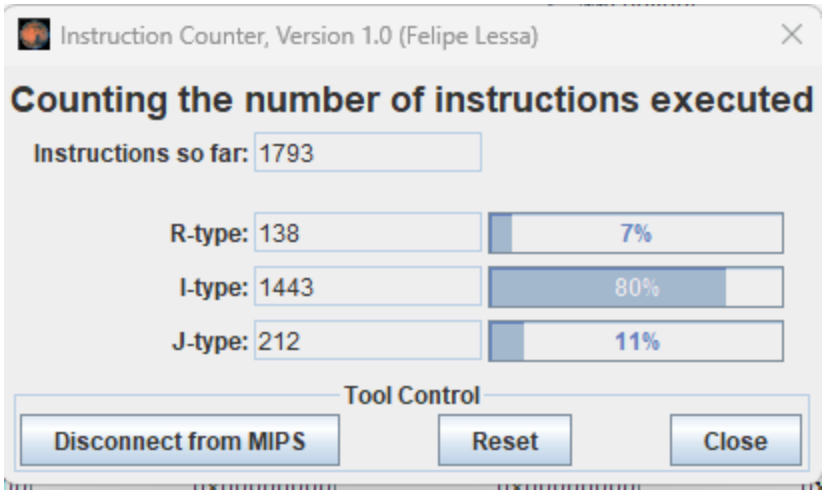
7.0 Test Results

Part A (Unoptimized)

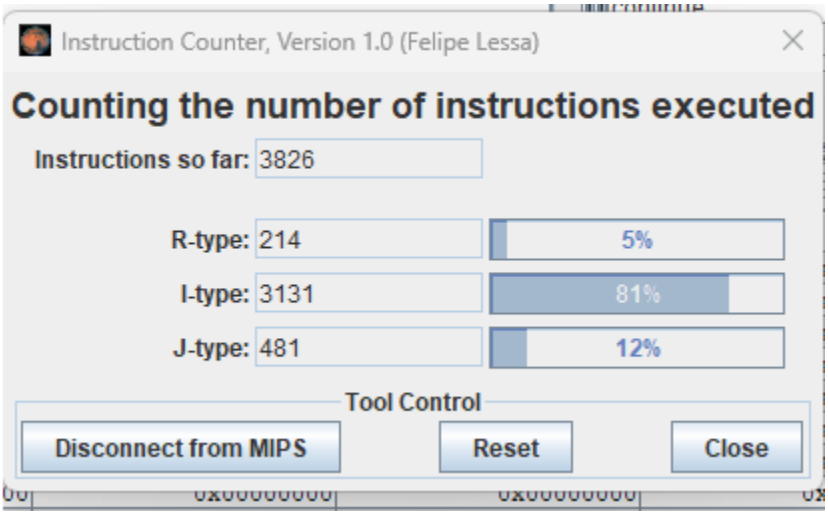
a. Lorem, ipsum.



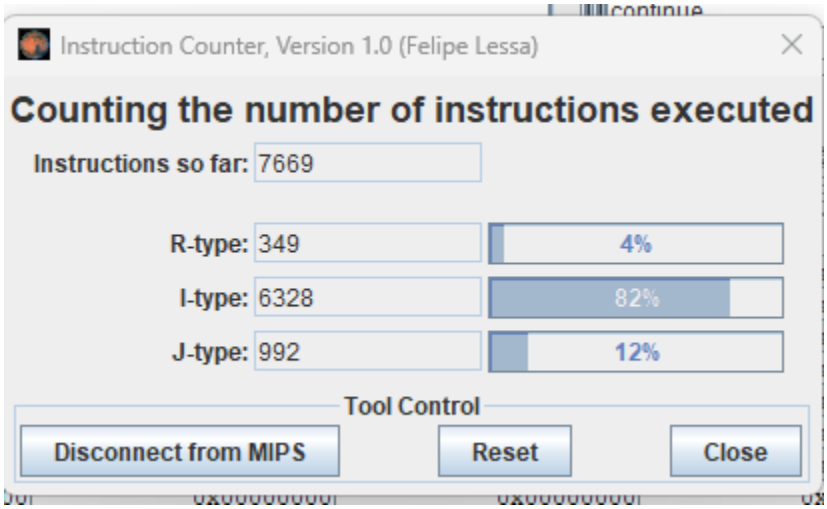
b. Lorem ipsum dolor sit.



c. Lorem ipsum, dolor sit amet consectetur adipisicing elit.

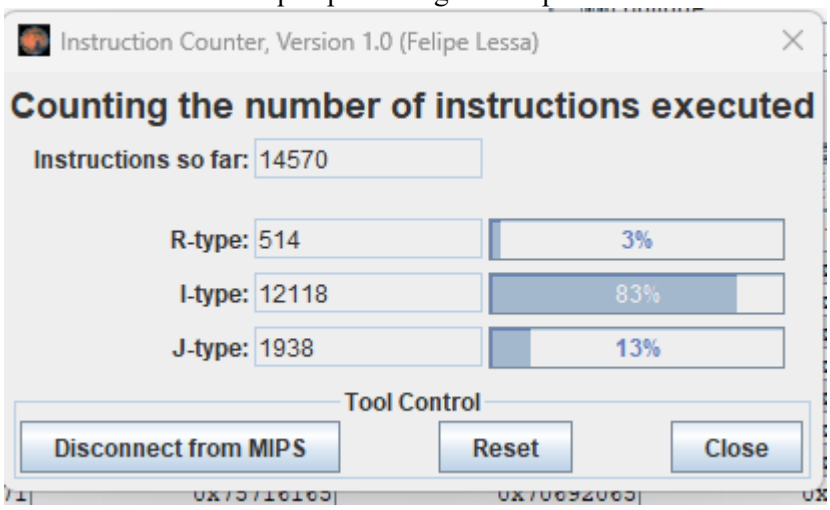


d. Lorem ipsum dolor sit, amet consectetur adipisicing elit. Debitis eveniet est aliquid ipsum sed dignissimos laboriosam.

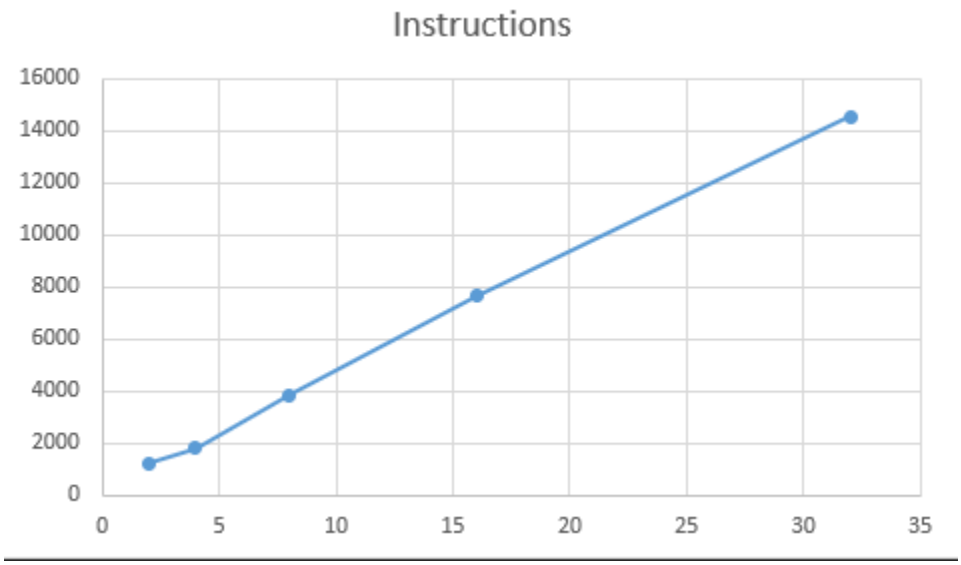


e. Lorem ipsum dolor sit amet consectetur adipisicing elit. Explicabo dolor quod optio totam quam, eaque accusantium deleniti tenetur ipsam vel. Similique, quae? Molestias fuga incidunt saepe sed? Similique

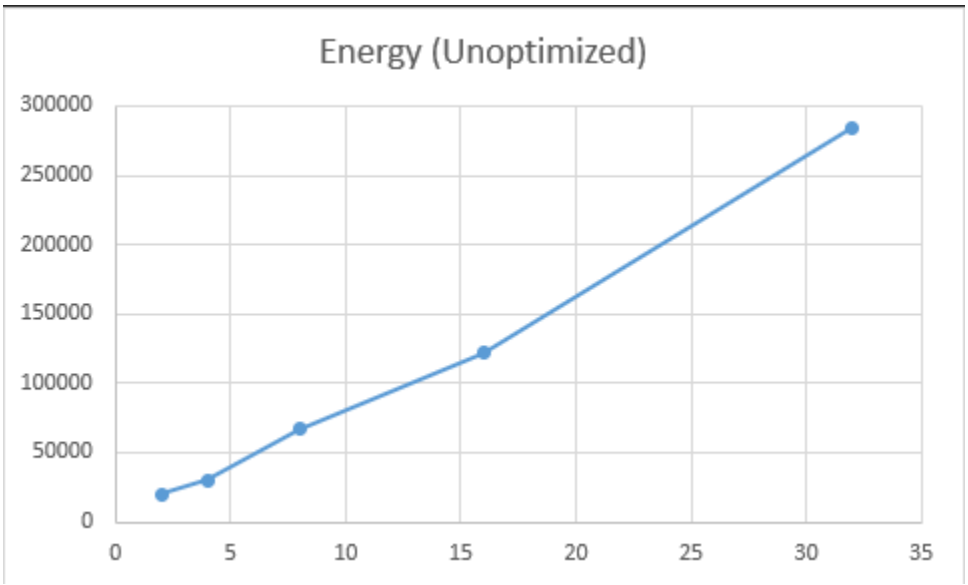
eaque ipsum magnam culpa!.



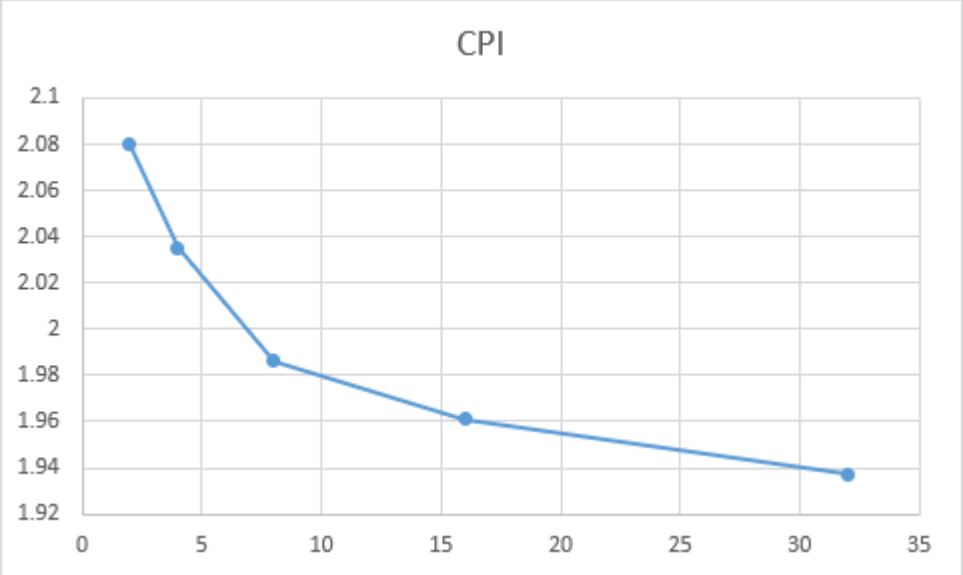
Graph of dynamic instruction count for sentences of 2, 4, 8, 16, and 32 characters in length.



Graph of Energy Consumption for sentences of 2, 4, 8, 16, and 32 characters in length. Energy in Femtojoules

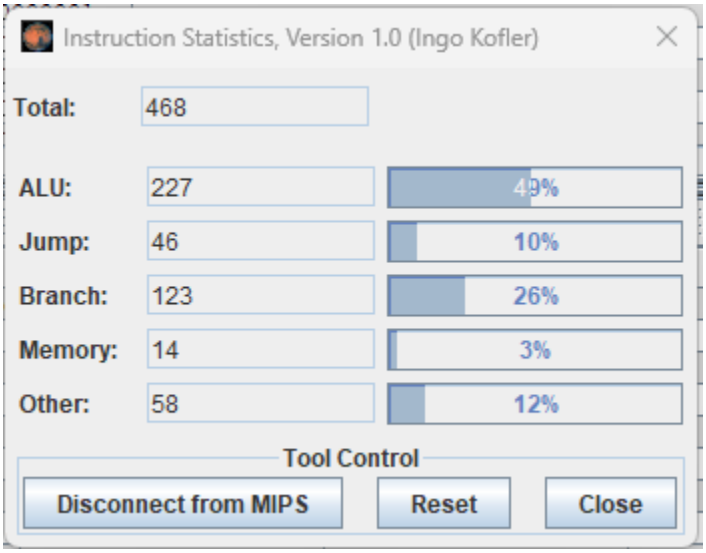


CPI Calculations for sentences of 2, 4, 8, 16, and 32 characters in length.

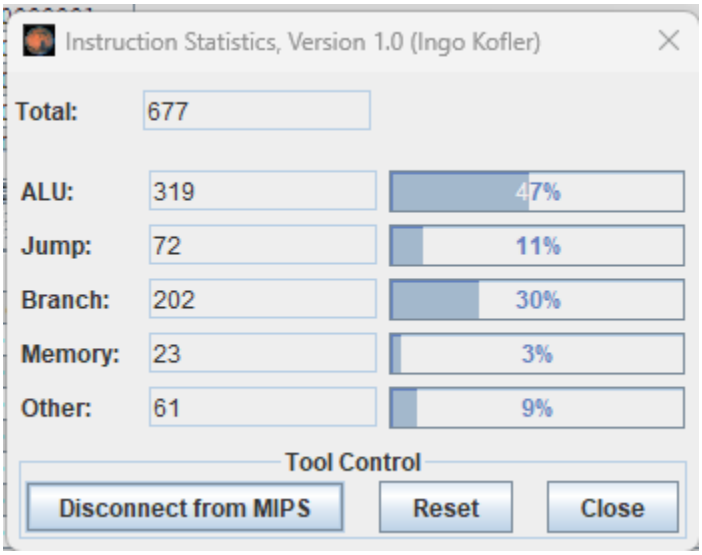


Part B (Optimized)

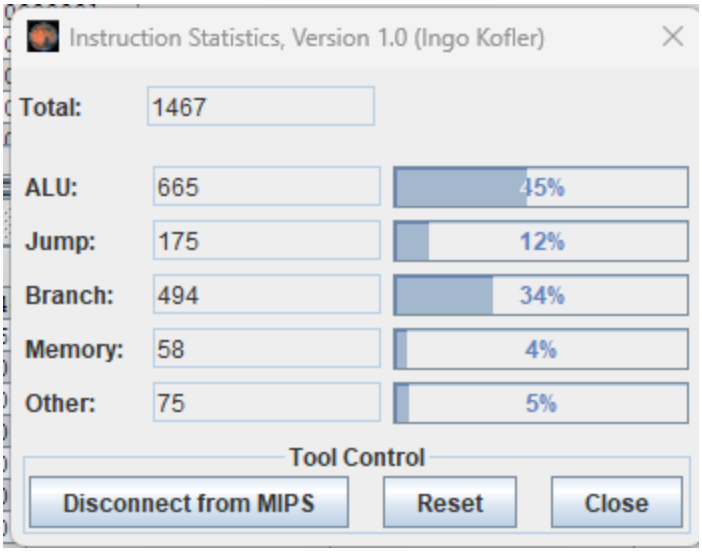
a. Sentence 1



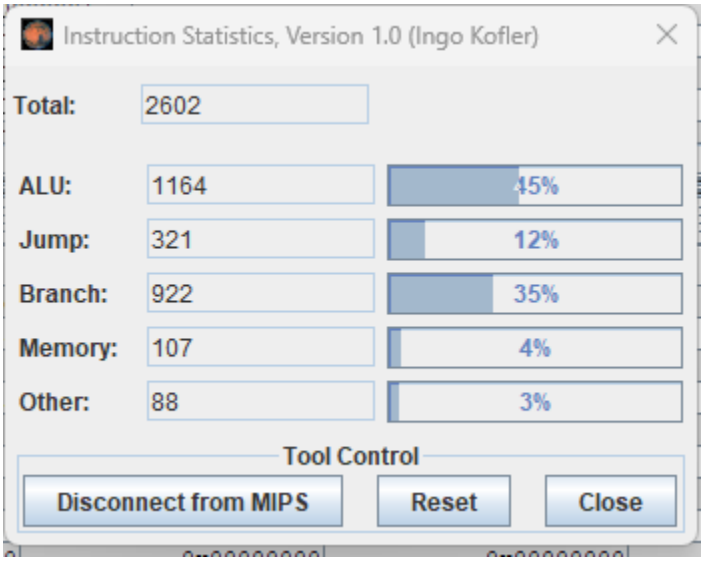
b. Sentence 2



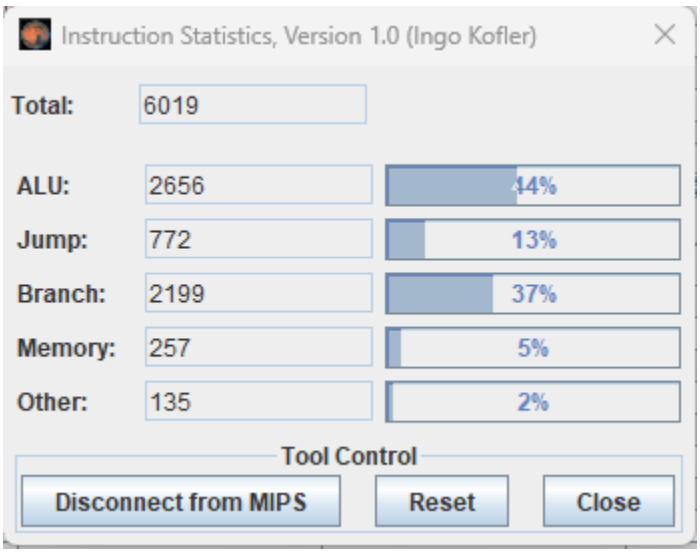
c. Sentence 3

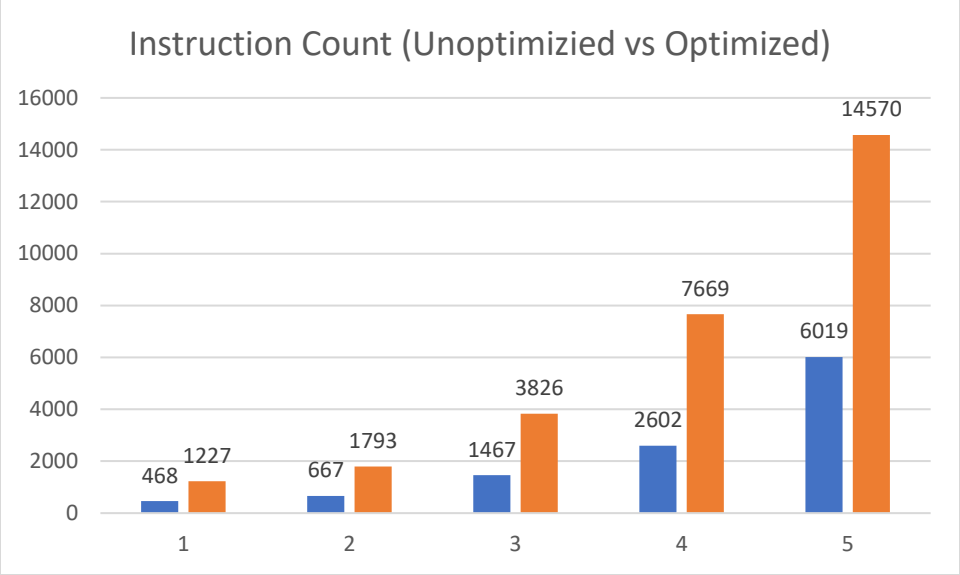


d. Sentence 4

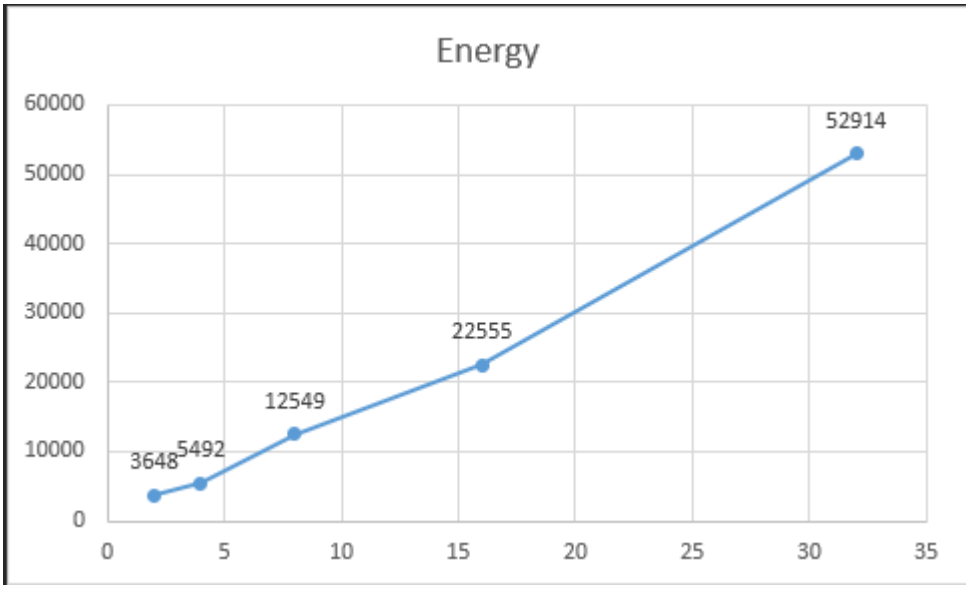


e. Sentence 5





Energy in Femtojoules



8.0 References

8.1 Textbook

Chapter 3 of Patterson and Hennessy Textbook which is identified in the Course Syllabus.

8.2 MARS Simulator

The MARS Simulator for MIPS processors, available at: http://courses.missouristate.edu/kenvollmar/mars/ and MARS syscall functions listed at: http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html were used