User Statement & Input Automated Word Design

Schnieder A. Maxime
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816-2362

***Abstract*—**In this paper, I will be sharing some observations and results I have analyzed from my Project 3 Capstone Report. The task from the project was to develop a program that finds how many times a word is used in a given statement and the index of each word found. Thus for the title of this paper came, User Statement & Input Automated Word Design. Test cases for this project was simple to choose. By creating sentences of different character sizes I was able to test my code. First test case came from the GTA with a character count of 790, then I came up with two more test cases of character count 176 and 71. Given an output to generate made the code simple, prompting for a statement and word to find, displaying how many times the word was found and lastly the index of each word found.

***Keywords*—**data, arrays, address, memory contents, string, indexes, null, and loops

I. Introduction

Tasked to develop a program that finds how many times a word is used in a given statement and the index of each word found, I used MIPS assembly language to do such that.

A. Project Design

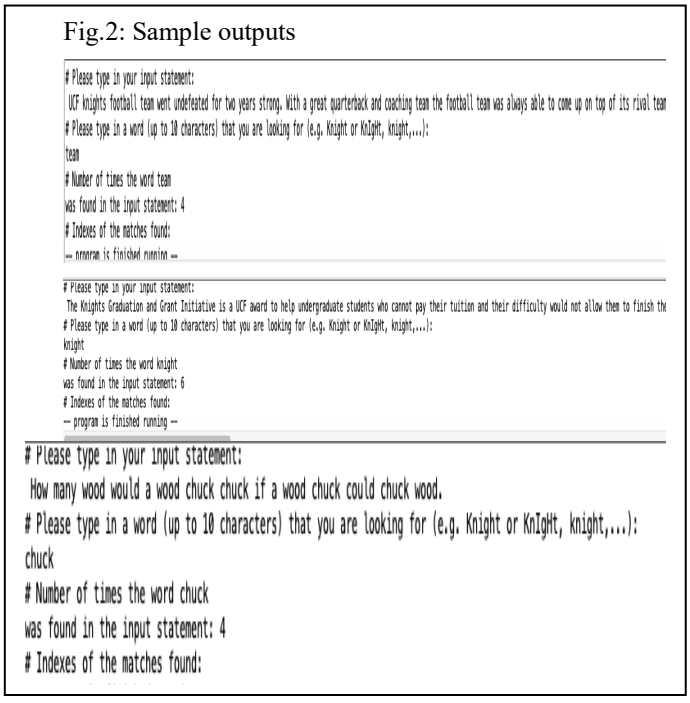
To complete such a task, first look at the information given. Having a predetermined input and output we know exactly how we went the code to begin and end. First, prompt the user to input a statement. Here we capture the user input after allocating enough space for the user and saving it in a register. Next, we prompt for a word the user would like to find in the statement of up to 10 characters, which would be captured and also stored in another register. This is when the real gears of the code begin. It takes the loaded registers and goes to each bit address to check not only if the word is found in the statement, but how manytimes it is found and at what index it was found at. Thankfully this program account for all cases, upper case and lower case, so feel free to test out any "wOrD". It will take the funky word and return back to the user in the correct form. It is able to let the user know how many times a word is found because after each times the word is found the counter would increment by 1. This is where we can also save an index of where that word was found in an array. Finally, the program will terminate when it finds its null character, and print out to the user how many times the word was seen and the index, thus completely our task.

B. Test Cases

Having already a sample input and output given, this is first tested. After getting the first test complete we can now try other test cases of different characters and word sizes as seen in figure 2. Using words Knight, team, and chuck passed through, we now check if these selected words would output the correct found word and index.

Fig.1: See Below

Fig.2: Sample outputs



II. Reliability Bit-Cells

Triple modular redundancy (TMR) is a form of N-modular redundancy. Which there are three systems that are tasked to process a majority-voting system to a single vote output. There are three systems because if one fails the other two will pick up with the task. Reliability with the TMR is if the none of the three modules fail or if only one fails. Since two event are mutually exclusive the system is equal to the sum of probabilities of the two events. When compared to other systems such as dual module redundancy the energy consumption can be high. When handled correctly TMR can be at a much smaller fraction energy usage then duplex and even hamming error correction.

III. Results and Discussion

Using keywords Knight, team and chuck these are the energy consumption through MIPS.

Result for Test Case 1

$$(1 \times 6963) + (3 \times 3548) + (2 \times 6207) + (.88 \times 1431) + (5 \times 5541) = 183654\ fJ$$

Result for Test Case 2

$$(1 \times 1844) + (3 \times 957) + (2 \times 1671) + (.88 \times 397) + (5 \times 1515) = 15981\ fJ$$

Result for Test Case 3

$$(1 \times 739) + (3 \times 388) + (2 \times 652) + (.88 \times 164) + (5 \times 602) = 6361\ fJ$$

Table I: Energy consumption for a single bit-cell memory in the designs provided in [1-3].

Design	Energy consumption of a Single Bit-Cell Memory
SEU-Latch [1]	0.88 fJ
DNU-Latch [1]	0.28 fJ
[2]	6.96 fJ
[3]	1.51 fJ

Table II: Total Energy consumption for the assembly program using designs provided in [1-3].

Design	Total Energy Consumption
SEU-Latch [1]	6361
DNU-Latch [1]	6262
[2]	7358
[3]	6464

IV. Conclusion

In a nutshell, from the results seen in section III, as the character count gets smaller for the word being looked in a statement the less energy is consumed. This is because the code is being loop through less times since it does not have to go through a long character count. Extra test cases were made to see what would happen if using the same statement and passing smaller, larger or no word to search for. It was shown that the smaller the word or if there wasn't a word to search (e.g. not a word in the statement) then the program would run the fastest. My best design happens to be the result for test case 3 with an energy consumption of 6361 fJ. Using the DNU latch appeared to have a much faster energy consumption over SEU.

Topics learned:

- Using loops in MIPS assembly
- Calculation of energy consumed by a program using dynamic instruction count
- Using Instruction Statistics and Instruction Counter tools in MARS
- Converting letters to uppercase/lower case using ASCII values
- Learning of TMR and other voting type systems

References

[1] Berg, M., & Label, K. (2015, February 9). "Verification of Triple Modular Redundancy (TMR) Insertion for Reliable and Trusted Systems." Retrieved April82020,from https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20160001756.pdf

[2] F. S. Alghareb, R. Zand and R. F. Demara, "Non-Volatile Spintronic Flip-Flop Design for Energy-Efficient SEU and DNU Resilience," in IEEE Transactions on Magnetics, vol. 55, no. 3, pp. 1-11, March 2019, Art no. 3400611.

[3] H. Pourmeidani and M. Habibi, "Hierarchical defect tolerance technique for NRAM repairing with range matching CAM," 2013 21st Iranian Conference on Electrical Engineering (ICEE), Mashhad, 2013, pp. 1-6.

[4] K. Katsarou and Y. Tsiatouhas, "Double node charge sharing SEU tolerant latch design," 2014 IEEE 20th International On-Line Testing Symposium (IOLTS), Platja d'Aro, Girona, 2014, pp. 1

[5] R. E. Lyons and W. Vanderkulk. "The Use of Triple-Modular Redundancy to Improve Computer Reliability".IBM Journal.1962

[6] Sheble, N. (2003, October 1). "More is always better when it's critical". Retrieved April 8, 2020, from https://www.isa.org/standards-and-publications/isa-publications/intech-magazine/2003/october/more-is-always-better-when-its-critical/