Reliable Nanometer Tech with Addresses and Index

Tommy Rodriguez
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816-2362

***Abstract*— This essay report goes over an assembly program created to find a user's word inside of a given phrase. The program reports back to the user how many times it appeared in the user and the indexes that it appeared in using the help of arrays. In this project we used the words Knight, UCF and grants. We did this to have some difference in the amount of characters. We will later figure out that the more characters a word has the more it makes the design use less memory consumption. We use the DNU (Double Node Upset) to show that it is the best in memory consumption just as it is in speed. This project was made to find a 'Reliable Nanometer Tech with Addresses and Index" using the assembly code and the given individual energy consumptions from sources [1-3]. Showing how the DNU indeed beats the designs presents in the report in section III.**

Keywords*— *Triple Modular Redundancy, addresses, arrays, null, index, Nonometer Technology, Double Node Upset (DNU), Reliability.

I. Introduction

Using MARS/MIPS coding we are able to achieve a goal of having the user input a statement, and then asked for a word to be found in that statement. That word is counted and shown to the user how many times it appears. The code also tells the user what index this word appeared in the statement.

A. Project Design

In order to have the project to function we must first give enough space for the user to input their statement. We also need a space for the word that the user will input to find within their given statement to the program. The user will start off by being asked for their statement, following after the user inputs the statement the program will then ask for the word, where the coder must specify how many characters it can be. In our case we specify to the user to input a word like 'KnIgHt' being maximum 10 characters. The program will then load the statement into a registers address. The program goes to each bit of the address to check if the word matches. The program accounts for capital and lower-case letters by converting it to the given word and returning it back to normal for the user. If the word is a match, then a counter will increment to tell the user at the end how many times the word appeared. At the same time, the program has a counter for the indexes. There is a counter increasing for each index, and when the word is a match the current count of the index is inputted into an array. Once the program finds the null, the end of the statement, it will output to the user the number of times a word appeared and what index the word appeared by the help of out putting the array to the user.

B. Test Cases

To test out our project we are recommended by our graduate teaching assistant (GTA) to input the following statement: "The Knights Graduation and Grant Initiative is a UCF award to help undergraduate students who cannot pay their tuition and their difficulty would not allow them to finish their degree. The Knights Success Grant is the most well-known program inside the Knights Graduation and Grant Initiative. In order to be awarded the Knights Success Grant, you need to be referred but it does not mean that all students who are referred will be awarded the grant. The students who want to apply for the Knights Success Grant need to submit a required application and complete the Knights Success Grant web course. For more information, you can stop by their office in the Registrar's Office on the main campus of UCF." To test our input, we give the program another input, which is the word. We use the following words to test the project, (Knights, UCF, and Grants) all in 3 different runs of the project. The following output that came from the written MIPS coding is displayed out in Fig.2.

Fig.1: Flowchart of the assembly program.
FOUND ON PAGE 3 BELOW

Fig.2: Sample outputs of the program.
FOUND ON PAGE 4 BELOW

II. Reliability Bit-Cells

Nanometer technology is a developing technology that is promising us an increase in device density. However, with such great development there is a great defect on nanometer technology and that is the reliability decrease using nanometer technology. The implementation of triple module redundancy (TMR) to help with the reliability levels of nanometer technology. This implements three identical latches and a voter circuit. With this, the majority of the outputs return correctly even if there is a failure in one of the latches. But this also comes with a cost, this method will impose significant areas, and also power consumption overheads.

With the TMR we also have the voter circuit. The voter is typically more reliable than most components in TMR. There usually is just one voter circuit at the end for the outputs, just as a simple circuit. There are cases where there are multiple voter circuits throughout the TMR where in this case has a higher probability of failing because we are adding more reliability to the voters. Voters are crucial parts of the TMR system. If there is a failure of a voter, and this TMR only has one voter at the end, then more than likely then the entire system will fail as it is no longer storing the outputs for us. However, adding more voters, even though more complicated, it does raise its reliability because if the error detection and automatic switching mechanism, if one fails, switching over to the others to back up the outputs kept on the other voter, making this system more reliable.

III. Results and Discussion

After testing my program using the energy consumption per instruction values, my program gave me the following results below. The results are displayed using the key word Knight, UCF, and Grant:

1) ALU = 7086 fJ (30%)
2) Branch = 6310 fJ (26%)
3) Jump = 3611fJ (15%)
4) Memory = 1456 fJ (6%) Refer to Table I
5) Other =5633 fJ (24%)
Energy Consumption: 239428fJ

1) ALU = 8686 fJ (34%)
2) Branch = 6455 fJ (24%)
3) Jump = 3695fJ (14%)
4) Memory = 1482 fJ (6%) Refer to Table I
5) Other =5758 fJ (22%)
Energy Consumption: 248471fJ

1) ALU = 7123 fJ (30%)
2) Branch = 6384 fJ (26%)
3) Jump = 3650fJ (15%)
4) Memory = 1478 fJ (6%) Refer to Table I
5) Other =5713 fJ (24%)
Energy Consumption: 242667fJ

Using Table I we can find the total energy consumption for the designs that were used in references [1-3]. We will use the best energy consumption calculated for my code to test it along with the designs of references [1-3]. You can find these results in Table II.

IV. Conclusion

In conclusion, from our results in section III we can observe that more energy consumed as the word we are looking for has less characters in it. It seems like the memory for each test subject we used appeared different and increased as our word decreased. More than likely this came from having to jump less, holding less memory, and having to branch less contributed to this. With memory making the biggest impact to energy consumption, even with a few digits changes it will give us a drastic change in energy consumption. This is the reason in Table II we observed such high changes in energy consumption among the designs. With Table II we can assess that the DNU-Latch was the most reliable in energy consumption. This is expected as the DNU has a remarkable high-speed application, but this does come with a cost of higher power consumptions.

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]	221928 fJ
DNU-Latch [1]	134528 fJ
[2]	1107228 fJ
[3]	313728 fJ

References

[1] 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.

[2] 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.

[3] 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. 122-127.

[4] M. Radu, D. Pitica and C. Posteuca, "Reliability and failure analysis of voting circuits in hardware redundant design," International Symposium on Electronic Materials and Packaging (EMAP2000) (Cat. No.00EX458), Hong Kong, China, 2000, pp. 421-423.

[5] S. Mitra and E. J. McCluskey, "Word-voter: a new voter design for triple modular redundant systems," Proceedings 18th IEEE VLSI Test Symposium, Montreal, Quebec, Canada, 2000, pp. 465-470.

[6] J. F. Wakerly, "Microcomputer reliability improvement using triple-modular redundancy," in Proceedings of the IEEE, vol. 64, no. 6, pp. 889-895, June 1976.

Figure. 1

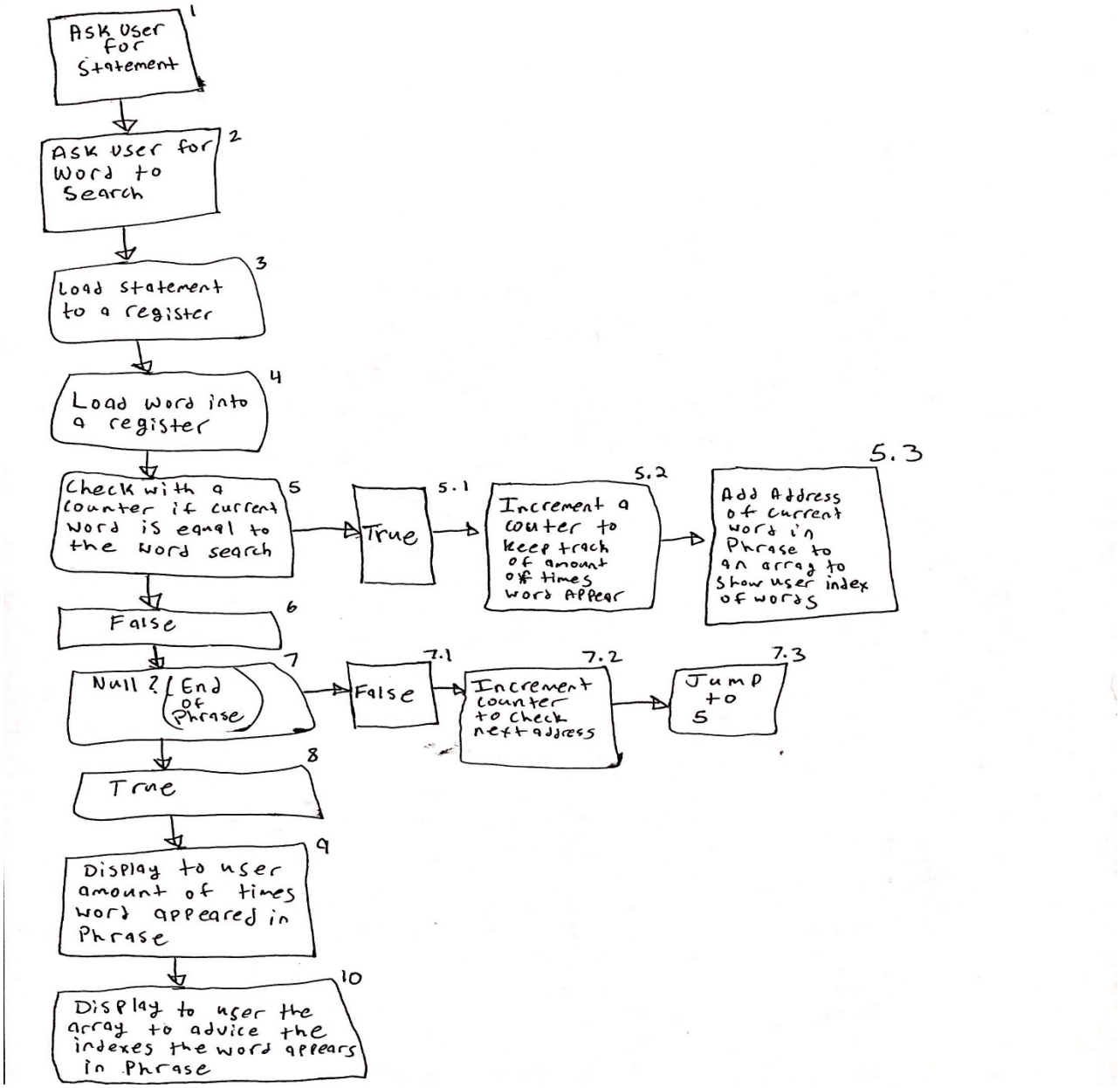

Figure 2

```
# Please type in your input statement: The Knights Graduation and Grant Initiative is a UCF award to help undergraduate s

# Please type in a word (up to 10 characters) that you are looking for (e.g. Knight or KnIgHt, knight,...): knight

 # Number of times the word 'knight' was found in the input statement:  6
# Indexes of the matches found:
-- program is finished running --
```

```
# Please type in your input statement: The Knights Graduation and Grant Initiative is a UCF award to help undergraduate s

# Please type in a word (up to 10 characters) that you are looking for (e.g. Knight or KnIgHt, knight,...): UCF

 # Number of times the word 'UCF' was found in the input statement:  2
# Indexes of the matches found:
-- program is finished running --
```

```
# Please type in your input statement: The Knights Graduation and Grant Initiative is a UCF award to help undergraduate s

# Please type in a word (up to 10 characters) that you are looking for (e.g. Knight or KnIgHt, knight,...): grant

 # Number of times the word 'grant' was found in the input statement:  7
# Indexes of the matches found:
-- program is finished running --
```