

# COMPUTER SCIENCE COURSEWORK

1. What my project is about?
2. What is little man computer?
3. Who is my client and what are he's needs?
4. What is my stakeholder's role and how will it help with the development?
5. What should my program so and what should it feature?
6. Hardware and software requirements for my program.
  7. Justification of hardware requirements.
  8. Justification of software requirements.
  9. Limitations of my program.
10. How is my program solvable by computational methods and is it necessary during the development.
  11. Existing software similar to my idea.
12. What ideas did my stake holder propose for my program?
13. Success criteria for my program to be successful.
  14. Problem break down and algorithms.
  15. Recognising the structure of LMC and its syntax.
  16. Creating a lexical analysis algorithm.
  17. Lexical analysis flow chart.
  18. Creating a variable handling algorithm.
  19. Variable handling algorithms flowcharts.
  20. Interpreting algorithm.
21. Algorithms for different buttons of the user form.
22. Validation necessary for the program to be redundant to crashing.
23. Methods for preventing the program from crashing due to ambiguous inputs/overflow.
  24. Necessary data structures.
  25. Necessary variables I will use.
  26. Structure diagram.
  27. Development.
  28. Initial design.
  29. Second design.
  30. Final design.
31. Beginning the development of the final prototype.
  32. Testing the final prototype.
  33. White box testing for added validation.
34. Beginning the development of the second prototype.
  35. Algorithm for the step button.
  36. LMC language interpreter optimisation.
37. Optimisation techniques (explanation of my algorithm).
38. Example of how my interpreter reads a Fibonacci sequence code.
  39. Full system test.
  40. Testing to inform evaluation.
  41. User acceptance testing.
42. Looking if success criteria has been satisfied.
  43. Evaluation.
  44. Future improvements.
  45. Code listing.

## 46. Bibliography.

## What is my project about?

My goal of the project is to create a program which works on similar foundations like Little Man Computer, "LMC". The aim is to create an interpreter (a dedicated program to be able to understand a specific language) which will allow a user enter LMC based code and be able to execute it. The program must be able to perform operations such as running the code at variable speeds or even be able to step through it line by line (more detailed functional requirements will be mentioned later on once I start thinking more about further details of the project). The goal is also to make a more customised version of little man computer which will be more feature rich which will be based on my stakeholder's needs. I believe that my implementation of little man computer will make understanding the concepts of basic low-level programming and understanding the von Neumann architecture easier and more enjoyable due to a more intuitive design.

The reason why my project is about little man computer is because my stakeholder wants to do some low level programming as revision for his A-level exams and also to become more aware of how computers work on a lower level, but feels like all of the little Man computer editions that he has tried are lacking functionality and an intuitive design. Little man computer also has a very short and a simple instruction set which is very easy to learn in comparison to instruction sets such as : x86 which is still low level but hugely more complex and not suitable for beginner programmers. Since the instruction set is very short and simple this is a perfect choice for my project since I have a fairly limited time to debug and develop the code which will probably still be time consuming even with a short instruction set, this makes LMC a perfect option.

My stakeholder wasn't satisfied with the layout of the different attributes within the LMC emulators that he has tried which was not very efficient to use and didn't really paint a good picture of how the von Neumann cycle works. There aren't many good little man computer programs and there are in total very few little man computer programs, this gives me an opportunity to create a more unique program with exclusive features. Me and my stakeholder saw a room of improvement that could be fulfilled so I decided to create a program that will fill in the gaps to make the user experience of coding in Little man computer better and overall less confusing.

My project is also about helping younger people who never programmed before to get into programming through using my program due to its simple nature friendly for beginners. Caleb is going to be teaching a couple of computer science lessons to the lower school as part of his work experience this is an opportunity to familiarise the lower year group with basic programming using the LMC instruction set. This further reinforces the need for the program to be simple and intuitive, yet be still feature rich, for someone like a year 9 student who never programmed before.

## What is little man computer?

Little man computer (LMC) is a simple model of a computer that is based on the concepts of the von Neumann architecture. It was created by Dr. Stuart Madnick in 1965 purely for educational purposes and its. It has a very simple instruction set and its very low level meaning its commands can only perform very basic operations such as adding or storing a value. Little man computer was created to teach how the von Neumann model works, it's also very good for teaching beginners how to program since it doesn't require remembering any complex syntax or many complex instructions, the way each function operates is also very simple since each instruction corresponds to a single operation.

Little Man Computer is used by schools to teach people how to program and it also gives an insight on how the von Neumann structure works, it's also a part of the computer science syllabus for GCSE and A-Level. Little man computer is not a recognised programming language in the industry, it's purely made to educate beginners due to its incredible simplicity however its mechanics are similar to more complex languages which are used in the industry for real hardware such as IBM basic or intel assembly language. Little man computer doesn't feature an official compiler which means it's an interpreted programming language so it runs on a layer of a different programming languages (this is how it will also work in my program, it will run on vb.net).

## Who is my client and what are his needs?

My client is Caleb winter-tear, he is an also a computer science student that is in my computing class. He enjoys computer science and he's main interest is hardware and programming. He is also interested in being a teacher and he happens to teach some lower school students computer science in his free time which serves as work experience for him. Caleb enjoys learning spoken as well as programming languages, he also likes product design so design of the program will be highly important and has to meet his needs from a functionality and visual stand point. Caleb is very strict about how he wants the layout to look as well as the overall design, this is mainly because he got really frustrated with the existing solutions that he tried, so I will need to ensure that this criteria is met very well. I meet Caleb almost every lesson and outside of school so consulting him about the project shouldn't be an issue.

As I Mentioned before, Caleb asked me if I could make a program to help him understand LMC to a greater level and how the von Neumann architecture functions. His understanding will allow Caleb to teach lower school students LMC and lower school students would also benefit from using the program themselves. Being able to write your own code and then being able to see how the CPU (central Processing unit) handles each instruction using visualization will greatly help increase understanding of low level programming and all the physical components of the von Newman architecture work in an organised way which is simplified and easy to follow step by step.

Caleb really desires to expand his knowledge about little man computer since he wishes to study computer science at university level where little man computer is used to help university students to develop their own simple programming languages. Caleb had a plan to become an expert in this programming language now so later on he will find using LMC easy and he will be

quick a doing tasks involving little man computer. This means that this project can be still user full and relevant to Caleb in the longer terms (when he goes to university).

The computers in our school have little man computer installed. When Caleb tried to use the program, he didn't really enjoy using it. Caleb found that program to be hard to use and understand, also it was limited in terms of visualizing the von Neumann architecture. my program will try to fix those issues that Caleb came across, improve the already existing solution and add extra useful functionality.

## What is my stake holder's role and how will his role help development?

Caleb will be mostly responsible for the design and he will help me identify what he wants in the program so I can then implement that. He will choose the layout and colour scheme of the user form. Caleb will also help me in testing the program after I think it's mostly finished (white box testing) this will let me do final refinements and also help me find issues that I haven't individually seen if there are any.

Caleb will be a very important figure during the development of my program since the success criteria is based on how Caleb is satisfied with the final product. He will help me find problems with the program as well as help me develop the design of the program. He will share his ideas about the user experience and he will be involved in Black box testing for my final prototype. Caleb will be involved in testing all of my prototypes including the most important final version of my program. Caleb's feedback will help me improve the program and fix errors if there are any.

Before I begin programming I will design the user form first since I will have a template to help me program and also meet all the expectations of the functionality of the program.

## What should my program do and what should it feature?

My program needs to fulfil my stakeholder's expectations. It has to be easy to use which means it has to have an intuitive Layout of input and output sections which means it has to be fairly easy to operate as soon as the user sees it the first time, the buttons and input boxes have to be placed appropriately so it's fast and easy to operate.

The program has to be visually pleasing with an appropriate colour scheme that will decided by Caleb since design is a personal preference. I will also implement the ability to customise the colour scheme by allowing the user to change the background colour/fount colour of the program however the default colour scheme will be whatever Caleb desires to have since he is my main customer for this project.

The program has to be able to accept a low-level language which in this case is Little man computer assembly language (LMC), it has to be able to interpret the code and execute it in real time(as fast as the hardware and software allows it to run) or allow the user to step through it(run code line by line every time a "STEP" button is pressed) so it's easier to keep a track of what's going on, also stepping helps to fix logical errors since it gives time to think how each line is affecting the ram and accumulator contents which are a key part of LMC functionality. Another piece of functionality that I want to implement that's not native to original little man computer is commenting, I started using commenting in vb.net and I found it very useful of keeping track what each line does so I think it will be a highly valuable feature to implement.

It is also essential that my program visualizes a CPU and allows the user to see what's going on inside it during stepping through their own code, it will be a highly simplified model which will contain the things that are present in the von Neumann model which A-level students have to learn.

My program should be able to save and load LMC code from (.txt) files, this will make easier to save any work so the user can preserve the work they have done and code can be easily loaded back into the text box on the user form. This saves time copying and pasting code all the time.

If I have any time left I will try to implement extra useful features such as sample LMC programs that can be loaded in into the input box just by selecting them with some button or drop-down menu. This will be very useful to demonstrate the syntactical layout of little man computer assembly language and can help to demonstrate how it works to someone not familiar with it.

Caleb also asked if I could show which line is being run so if he steps thru the code it will be easy to see which line he is on. This is why I decided to use text highlighting so when the program is run it will highlight the line it's currently dealing with.

Lots of LMC implementations do not allow the user to run the program at full computers speed which can make more complex little man computer programs take a long time to complete. This is a big disadvantage therefore my program will allow to run program very quickly without delays. This will be implemented in such a way that the user will have the ability to select the speed at which the program runs at.

## Hardware and software requirements for my program:

My programs hardware requirements will be very low meaning that if a machine can run windows 7 or higher, it will be able to run my program.

The minimum hardware requirements for my program are:

- A single core 1 GHz processor with an x86-64 architecture.
- 1 GB of RAM.
- 21 GB hard disk space (secondary storage) with an extra 500 kb for the program.
- A dx9 compatible Graphics processing unit (to allow windows to function properly).
- Basic input/output devices (keyboard, mouse, monitor).
- Monitor with at least a resolution 800x800 pixels.

## Justification of hardware requirements:

My program will not require a processor with multiple threads therefore a single core CPU will be able to handle the workload. my program will also be able to be run on a CPU with fairly low frequency since it won't put much stress on the CPUs pipeline due to not needing to iterate as fast as the CPU would allow to (there may be an exception when the user runs the program in "as fast as possible" type of mode that I will implement, however 1 GHz will certainly be plenty for basic little man computer programs. my program won't take much space and will be definitely smaller than 1 GB which means that 21 GB of hard disk space will be enough since Microsoft windows requires 20GB as a minimum. The program will also require very little resources from the main

memory(RAM) since the arrays and variables will only have hold a very small fraction of 1GB of ram so 1GB will be definitely enough just to run my program.

The user has to have some way of seeing as well as interacting with the program so a graphics processing unit and a display is necessary to display my program (the use form) and keyboard and mouse is necessary to have the ability to use the program.

The software requirements for my program are:

- .NET Framework 4.5 or later is necessary.
- Microsoft Windows 7 operating system or any later versions such as 8, 8.1 or 10.
- Microsoft basic display driver or NVIDIA/AMD drivers.

## Justification of software requirements:

Since my program was developed alongside .NET Framework 4.5, to ensure compatibility with my program, .NET Framework 4.5 or higher has to be installed which is supported on windows 7 and up.

To ensure stability and compatibility, windows 7 or higher has to be installed. Windows is also necessary because .exe files can only be executed in a windows environment unless they are run via a virtual machine.

A basic graphics driver has to be installed (windows 10 automatically configures this via network), this is so an image can be displayed on a monitor and also it ensures the full resolution of the monitor is used which means the program will look better and be more convenient to use (especially if the monitor resolution is small).no/incorrect display driver can cause the program to look out of proportions since windows won run at default resolution of the monitor.

## Limitations of my program:

As I mentioned before my program will only be able to run on windows-based machines since vb.net programming language is compiled into an (.exe) file which can only run on windows. If Caleb decided to use this program on his second laptop which is running a mackintosh operating system or on a mobile phone, it wouldn't run unless its runs on a virtual machine.

The user form will be certain number of pixels wide and tall, due to timing constraints I won't be able to make it scalable for low resolution displays/ very high-resolution displays. This make it less convenient to use on a low/very high-resolution display however most school computers as well as Caleb's main windows machine has an appropriate resolution.

One of the major limitations of my program is that it won't be able to predict infinite loops (whether the program will Halt), this is because there are a huge number of programs that can be inserted into my interpreter (theoretically infinite if memory was infinite), this means it's impossible to write an algorithm to predict if a different algorithm will HALT. This can be an issue when the program is running at full speed since it could make it crash, infinite loops won't be an issue when the program isn't running at full speed since it can be stopped.

My program most likely won't be able to do one the functions that little man computer has which is assigning a variable during runtime. This is because the lexical analysis function that I will talk about later would have to be more complex and since I have limited time I doubt I will implement that feature, this feature is also not very important since the same thing can be

done using a combination of other functions so it doesn't limit the power of this programming language.

I will probably discover more limitations during the development of the program which I will talk about in the development section.

## How is my program solvable using computational methods and why are they necessary during the development?

The development of my program will require lots of computational thinking as well as computational methods to be completed successfully.

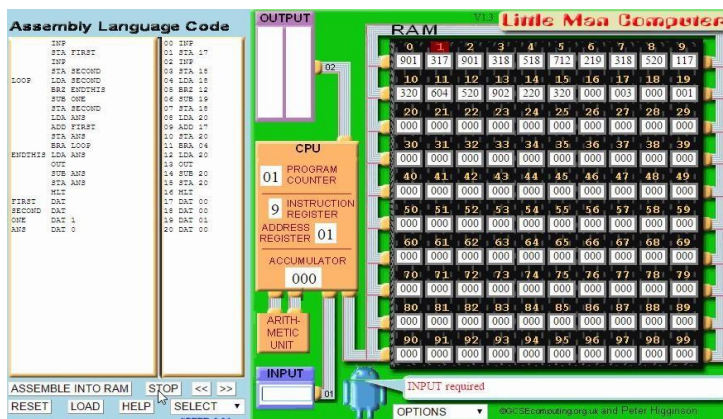
I will have to think ahead by planning what my programs functional requirements are so I will have something to work towards and be able to work on certain parts of the program in a logical order so it's easier for me to create the final solution. For example, I will design the user form first so it will be easier for me to program it since I will have the outline of the program which will make coding also a little easier. This is known as "thinking ahead" in the computational thinking terminology. "Thinking ahead" also involves breaking down a problem into smaller pieces and determining the potential problems that might arise which can be helpful in determining how the problem will be approached to be solved successfully.

The program will have to be able to interpret the code that has been entered and then process it so it displays correct outputs given the inputs. This will require lexical analysis which means the program will have to search for key words within the code such as ("ADD"). This means I will have to use selection ("if") statements and iteration (loops). This is an example of logical thinking because I will have to make the program perform decisions depending on different permutations of data that is passed through the code that's will trigger certain "if" statements if a condition is met, this means it will handle text in a specific logical order and the code entered perform logical decisions based on the logic I applied in the first place.

The part of my program that will display all of the registers and how the work together when LMC is run will require visualization. I will be able to visualize it by updating labels that show what each memory cell is holding. This will require a layer of abstraction that will translate what's going on behind the scenes (in my code) into a visualized format by breaking down the problem into sections. This is obtainable simplifying/braking down the von Neumann model into separate sections such as one part of the form will display the ram and another part of the form will display the ALU (arithmetic and logical unit). This will allow me to code for individual components of the program and then I will try to make these modules communicate with each other by using global variables. This is known as "abstraction by generalization".

"thinking procedurally" is when order of steps needed to solve a problem are considered by looking at the component of the problem and also looking at sub procedures which will help solving parts of the problem to eventually successfully solve the problem. When I will be planning out how my interpreter will handle each line of low level code, I will have to use procedural thinking. I will have to determine how each instruction affects the contents of the accumulator as well how variables are handled in arrays each time an instruction is executed.

## Existing software that is similar to my idea:

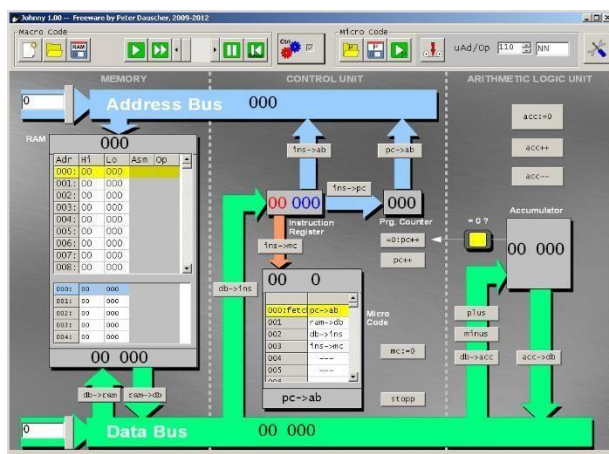


This is little man computer (also known as LMC), created by Dr. Stuart Madnick in 1965 for the purpose to teach others how assembly language is handled by a computer. It's composed of an input box, output box, ram (the large matrix consisting of 100 slots, each can store a value from 0 to 999).

It has the ability to store the code that has been assembled and it can also run it instantly or the code can be stepped through. This program can run on windows machines as an executable. This is the program that our school computers have and this is the program that Caleb didn't particularly enjoy to use.

My program will also feature a place to insert code and main buttons such as start/stop. it will also have ram that's displaying its contents however my solution will have a ram displayed as a linear structure so it will only be one column with lots of rows going down. This piece of software was my main inspiration to create my own version of it.

Another thing Caleb really disliked about the program was that the colour scheme made the experience bad for him, the program has a retro look and he suggested that I will use a more modern design in terms of colour choice and overall structure of the program.

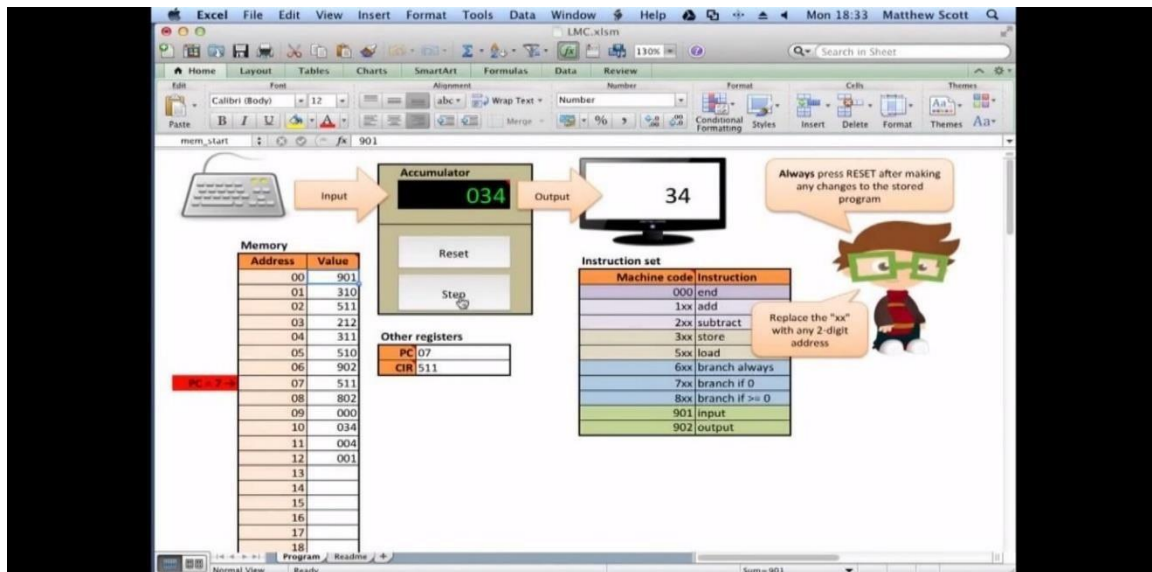


This is a von Neumann model simulator, it has lots of detail that clearly shows how this model functions.

This is the type of thing I want to create for program however it won't have such high level of complexity since this particular program has an information on it that's not even in A-level specification. I want my program not to be intimidating so the complexity level has to be well balanced.

Here's another implementation that has been created using excel and visual basic programming language that can be accessed using excel developer options. This implementation is an even more basic version of the one above it features even less memory cells and it only displays a





Program counter and "CIR" which is the current instruction register.

It shows what number corresponds to what instruction and it allows the user to input the code into the ram (memory) that is visible on the left part of the screenshot. Also, it doesn't feature a Place where code can be written in instead the user has to manually put the numerical values of the code they want to run in the ram that's located on the left side of the screen shot. This definitely makes it harder to see how the code works because it's not displayed using commands made up of words/phrases.

This program also features a small helper that guides you how to use LMC which definitely helps beginners know what to do.

This lacks some features that my program will have however I do like how it's very minimalistic which can be a good thing. It has a pretty vague representation how the ALU and ram works (very little behind the scenes). It also features ram which has 100 slots for storing values which is similar to what my program will feature, this piece of the software was something I would like to implement in my program since ram is a linear structure so it would be more accurate to represent it like this program is representing.

In Conclusion, all of the implementations I have analysed have their draw back and merits. I have shown my research to Caleb and we decided what my program should inherit from other implementations from a design/functionality and usability standpoint. Caleb told me what he liked and disliked about each implementation as well as things he would like to see that didn't exist in the implementations I have shown him.

## What ideas did my stake holder propose for my program?

Caleb said that the way ram is presented in most little man computer implementations is very confusing since the variables are represented as numbers. This is something I will take into account when planning out how I am going to visualize each component of the von Neumann architecture. I will show Caleb few options to choose from and ensure they are as intuitive as they can be for him.

## Success criteria for my program to be successful:

- My program has to be effective in explaining how all the registers, Ram and ALU work together in a computer system, the effectiveness of that part of the program will be determined by white box testing.
- My program has to be able to understand LMC code. It will have to understand any combination of code that is error free and has to be able to run on my program in the same way that original little man computer runs given the same piece of code.
- My program has to have consistent results so every time a user inputs new code or they run the program with different inputs they will receive the correct output every time.
- The solution has to be able to deal with incorrect inputs by the user which means it has to have appropriate validation.
- The program has to be able to load and save ".txt" files.
- My program has to be able to give appropriate error messages so the user knows how to fix their problem if there is one.
- My program has to be visually appealing (more modern) and intuitive to use.
- My program will allow the user to step through the program so the user has time to see what is happening.
- My program will allow the user to run their programs at different speeds if they desire to run the code quickly.
- My program will feature sample codes so the user can run pre-made programs so they can get used with the syntax and the general structure of little man computer assembly language.

## Problem break down and Algorithms:

### Recognising the structure of LMC and its syntax:

LMC features simple command set featuring 12 low level basic commands (lexemes):

Command:	Description of functionality:
LDA	Loads a value from a specific memory location (variable).
ADD	Adds a value from a specific memory location (variable) to the accumulator.
SUB	Subtracts a value from a specific memory location (variable) from the accumulator.
STA	Stores the contents of the accumulator into a specific memory location (variable).
OUT	Either outputs the contents of the accumulator or a value from a specific memory location (variable).
HLT/COB	Stops the program.
DAT	Used to declare variables.
BRZ	Branches if accumulator is equal to zero.
BRA	Branches unconditionally.
BRP	Branches if the accumulator is positive (above or equal to 0).
INP	Allows the user to input a value.

This is a typical structure of LMC:

Loop tags are located on the left side of the command, they are used for branching.

```

INF
STA FIRST
INF
STA SECOND
LDA SECOND
BRZ ENDTHIS
SUB ONE
STA SECOND
LDA ANS
ADD FIRST
STA ANS
BRA LOOP
ENDTHIS LDA ANS
OUT
SUB ANS
STA ANS
HLT
FIRST DAT
SECOND DAT
ONE DAT 1
ANS DAT 0
  
```

All LMC commands are 3 letters long which is a key pattern that is important to consider during lexical analysis

The right side of the command is the variable (operand).

This portion of the LMC code is the part that is going to execute until "HLT" command is hit.

Variables are declared from the bottom in LMC and all declarations end after the HLT command, this is inherent in the LMC structure.

Another important part of the code is what is present on the left side of the command and what is present on the right side of the command. The left side of the command is always either the loop tag or a variable name in the variable declaration section at the bottom. A loop tag is a tag which tells the program where to branch to which can be used anywhere in the assembly code except below the "HLT" command. The right side features the operands (the data that the operation code uses), also the right side of the code is used to assign values to variables during declaration. By knowing those attributes of LMC I can start creating a lexical analysis algorithm.

## Creating a lexical analysis algorithm:

The key thing is that all instructions are the same length which makes lexical analysis a lot easier, another pattern that is present is that all instructions have to be in capital letters so ASCII value ranging from 65 to 90. This means I can easily search for each instruction by searching for one of the 12 instructions that LMC instruction set has. I can do that by doing a comparison between current 3 characters read and the 12 commands available.

Here's a visualized how a section of an array, containing the single line of LMC code (declaration section):

A		D	A	T		2
1	2	3	4	5	6	7

Each character can be analysed one by one, since every command is 3 letters long I can search the array for a 3 characters long string and increment the starting position by one until the end of array is hit by the last position of the character. While searching for the command the left side of the command can be built by concatenating strings together until command is hit, spaces will be ignored which will allow the code to be less strict with spacing. After finding the command the right side of the string can be built into a sub string that will be whatever is on the right side of the command.

Here are few iterations explaining the process visually:

No command is found within this location of the array, position is increased by one:

A		D	A	T		2

Program iterates until it can recognise an op-code:

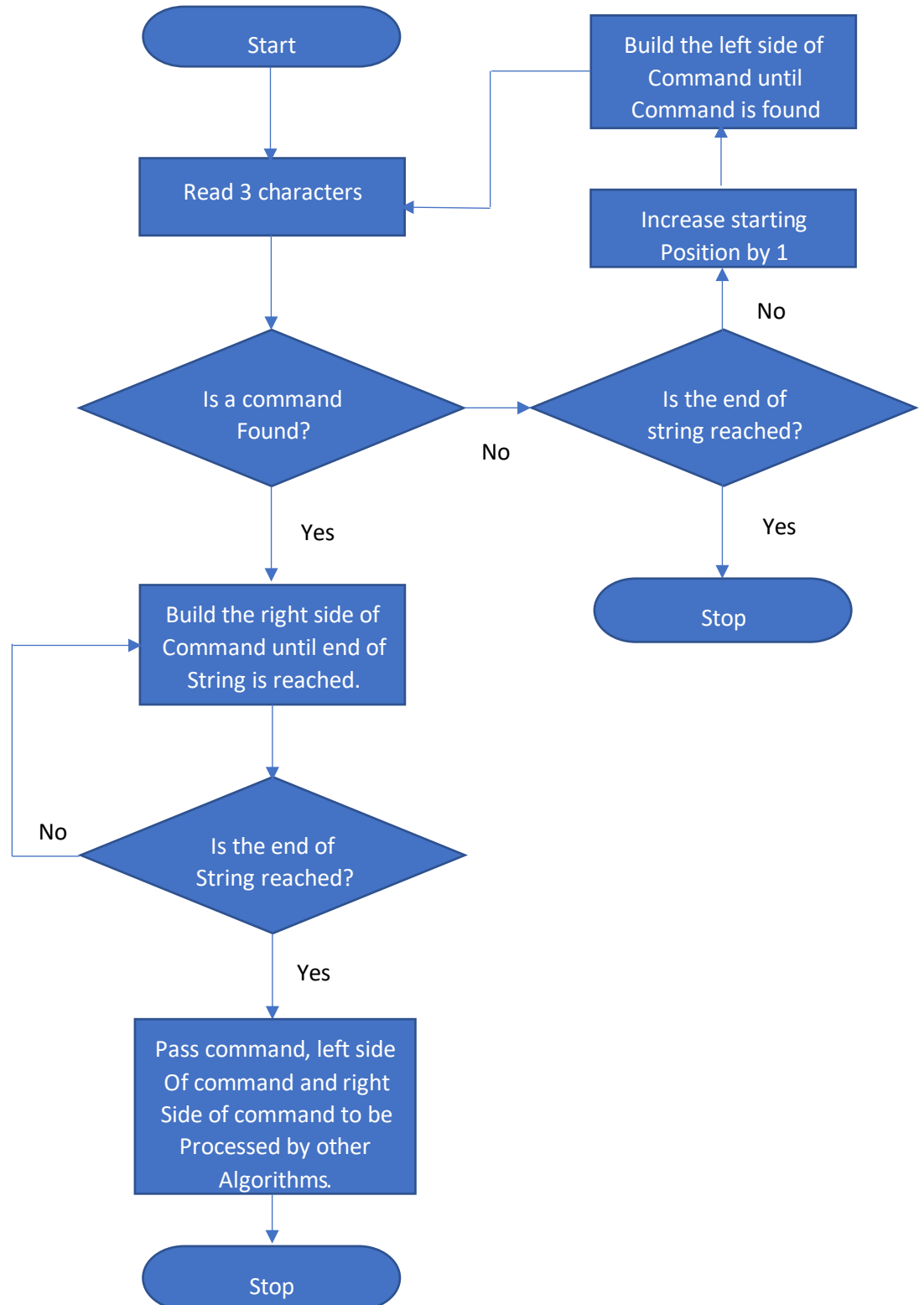
A			D	A	T		2

Command is found: "DAT", the algorithm can also save "A" as the string found on the left side which is this case is the variable name since "DAT" is used for declaring variables:

A			D	A	T		2

Result: Right side of command = "A", Left side of command = "2", command="DAT"

## Lexical analysis algorithm flowchart:



This is a simplified outline of the Algorithm, there will be more validation required in the actual code such as ignoring spacing and stopping reading the line once "/" is reached.

## Creating variable handling algorithm:

Here is an example what the user might write when declaring their variables:

```
ans DAT
base DAT
shift DAT
one DAT 1
number DAT
```

Some variables may not necessarily be assigned with a value and some may, for example in the example above “one” is assigned with 1 and “ans” doesn’t have any value assigned to it. This is important because my algorithm will have to be able to recognise this, otherwise there may be a type mismatch error since “” is a string.

The first thing that will have to occur is lexical analysis so the program can recognise the command. After the command has been recognised the data can be passed to procedure that will handle the data.

The data will be stored in a 2-d array, the variable and its corresponding value will be stored like such:

Ans	0
Base	0
shift	0
one	1
number	0

Variables won’t be sorted, they will be placed in the order the program reads them (at least for initial prototype).

When the LMC code will be executed the program will request to save/load values into/from the array which will have to involve searching the array.

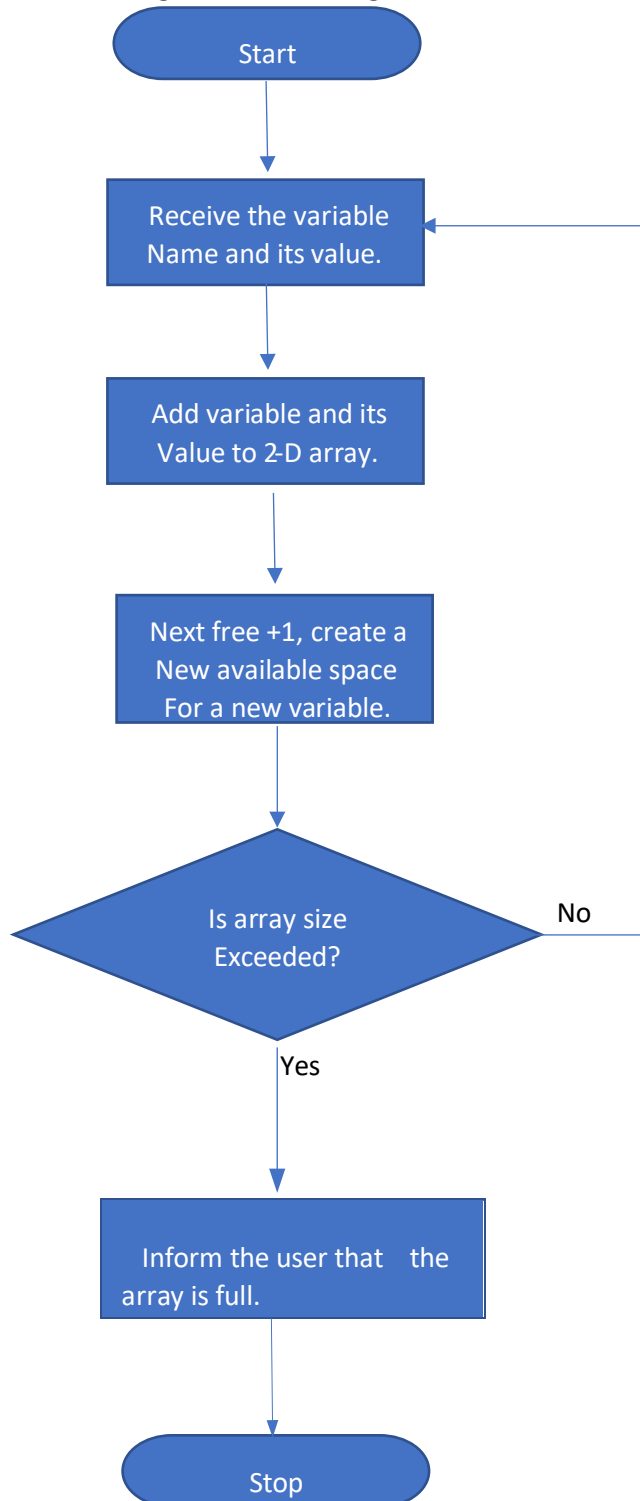
For example, if the one of the lines of assembly code say: “STA shift” and if the accumulator is equal to 5 then “shift” will have to be located and the value will have to be stored, here is an example of what “STA shift” would cause:

Ans	0	Ans	0	Ans	0
Base	0	Base	0	Base	0
shift	5	shift	5	shift	5
one	1	one	1	one	1
number	0	number	0	number	0

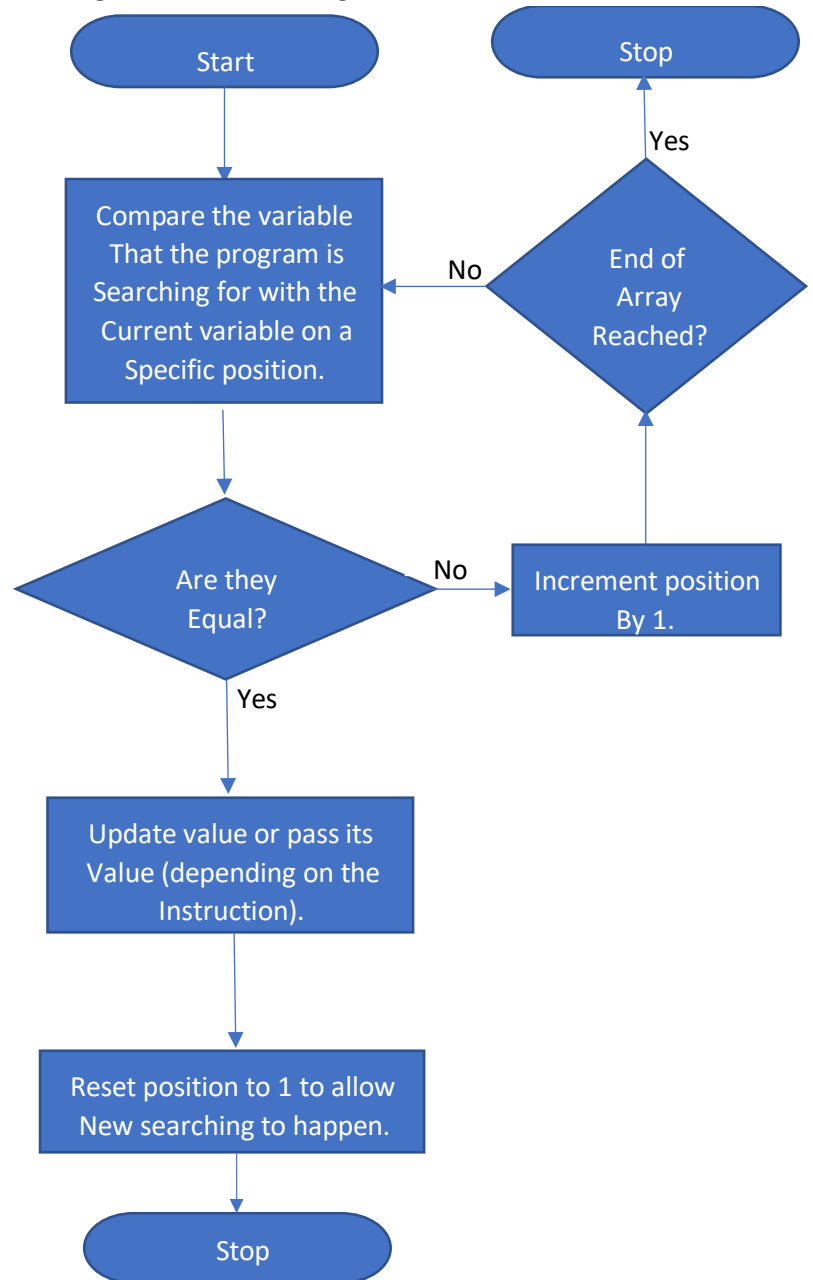
The variable will be found using a linear search, once it’s located its value would be updated.

## Variable handling algorithms flowchart:

Algorithm for adding variables:



Algorithm for searching variables:



## Interpreting algorithm:

The interpreting algorithm is the main algorithm which puts all of the algorithms together. It will decide what operation to perform based on the current command that is executed and it will perform most of the validation. It will read code line by line until "HLT" or "COB" is detected.

The interpreter has to perform specific operation in a specific order for a specific command, here is an outline of the operation that have to perform for a given command:

### "ADD":

- Locate the variable in the array that stores variables.
- Copy the value that is assigned to that variable.
- Add the value to the accumulator (adding to the accumulator is cumulative therefore its existing value + the value that's added on).

### "SUB":

- Locate the variable in the array that stores variables.
- Copy the value that is assigned to that variable.
- Subtract the value from the accumulator (subtracting from the accumulator is cumulative therefore its existing value of the accumulator - the value that's added on).

### "LDA":

- Locate the variable in the array that stores variables.
- Copy the value that is assigned to that variable.
- Overwrite the accumulator with the value that has been copied.

### "INP":

- Allow the user to input a value.
- Overwrite the accumulator with the value that the user has inputted.

### "BRZ":

- Check the value of accumulator, if it's not zero then do nothing.
- Find the loop tag that's associated with the command.
- Determine the loop tags position by looking at its line position
- Branch to line the tag is on.

### "BRP":

- Check if the value of the accumulator is above 0, if not then do nothing.
- Find the loop tag that's associated with the command.
- Determine the loop tags position by looking at its line position
- Branch to line the tag is on.



**“BRA”:**

- Find the loop tag that's associated with the command (BRA is unconditional so it doesn't have to check the state of the accumulator).
- Determine the loop tags position by looking at its line position
- Branch to line the tag is on.

**“STA”:**

- Locate the variable in the array that stores variables.
- Copy the value from the accumulator to the memory location that represents the specific variable. (The value in the accumulator doesn't get deleted).

**“OUT”:**

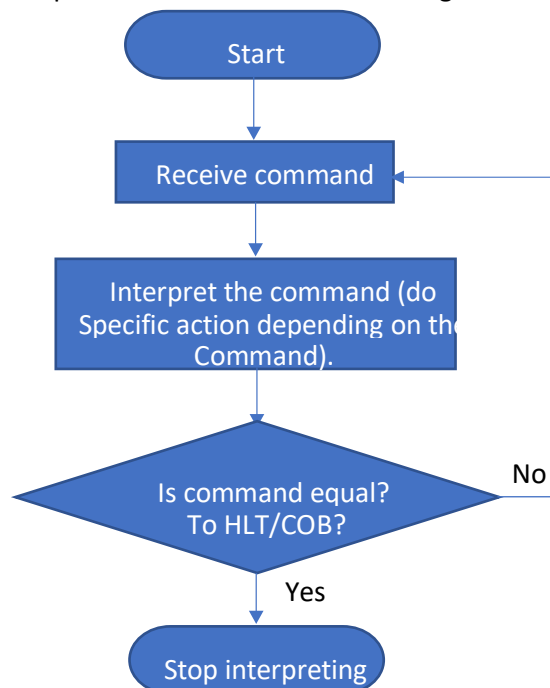
- If no variable is written after the command then output the contents of the accumulator.
- If there is a variable that's written after the command then:
  - Locate the variable in the array that stores variables.
  - Copy the value that is assigned to that variable.
  - Output the value.

**“HLT/COB”:**

- Stop interpreting the code.

To allow branching and reading preceding lines of code all of the interpreting code has to be encapsulated in a large loop so each line can be interpreted in order from top of the code to bottom.

Here's a flowchart which presents the mechanics of the algorithm:



## Algorithms for different buttons of the user form:

My program will feature buttons on its user form so the user is able to interact with the program. There will be five main buttons on the form, the algorithms for them are shown below:

### “RUN” button:

- Call the procedure responsible for executing the code

### “STOP” button:

- If the program is running, send an instruction to the procedure that's running the code (most likely a Boolean) which will terminate execution.
- Reset all variables to their original state.

### “STEP” button:

- If pressed during the runtime of the code, prevent the program from executing next line until “STEP” is hit again.
- If pressed before “RUN” is pressed, call the procedure responsible for executing the code and then perform step 1 (bullet point above).

### “CLEAR OUTPUTS” button:

- Clear the contents of the list box.

### “EXTENDED VIEW” button:

- Open a second user form that displays the registers, ram etc...

The buttons above are the essential buttons which will enable the desired functionality, as development progresses there might be more added, in a drop-down menu for example.

## Validation necessary for the program to be redundant to crashing:

My program should only accept lower case and upper-case letters as well as “/” for commenting the code. I can disable any other characters from being entered using keypress which basically doesn’t allow any other ascii to be entered except the ascii that has been allowed. Another alternative would be to detect an invalid character during lexical analysis and then inform the user that there’s a syntax error on a specific line.

Also, when a user writes an “INP” command an input box will appear which will allow to user to input data into it. This required validation. The input box will only have to accept numbers that are 4 digits long and there can’t be any other characters within it therefore a ascii check has to be done vb.net has a isnumeric () function which allows to check if a string is only made of integers. Type mismatches can also occur when declaring variables so my program will have to be able to cope with that as well.

My program also has to have validation during the interpreting process such as checking if there’s “HLT” command present in the code or if a variable exists is a command requests to load it. If there’s something wrong, using selection the program will inform the user that there’s is something wrong and it will specify what is wrong. For example, is there’s no variable found on a specific line it will say: “no variable found on line (the number that the error occurred on)”.

Another thing my program needs to be able to handle is infinite loops which happen often especially when learning how to use them. I have couple of ideas how to tackle that by creating some form of validation which will inform the user that there might be an infinite loop, this will prevent the user form freezing and crashing.

The program has to be able to handle overflow errors. Overflow errors can occur from arithmetic operations when the sum of the two integers is either too small or too big to be represented on a computer, there’s a theoretical integer size limit for a computer, depending on its architecture (64 bit or 32 bit). However, vb.net has its own limit when overflow happens, this limit is: 2,147,483,648 to - 2,147,483,647. This means my program has to be able to determine whether an overflow will happen before two integers are added or subtracted from each other

## Methods for preventing the program crashing from ambiguous inputs/overflow:

There are simple checks that can be applied to my solution which will prevent my program from running into an error.

When the user inputs data into the input box the program has to check if it’s an integer, vb.net features a function which performs a check and returns a Boolean (true for an integer), if false is retuned the user can be informed that the data they entered is not an integer. Verifying whether it’s an integer is only half of the problem because the integer has to have an appropriate magnitude, it cannot be too negative or too positive.my program checks the integers length since the magnitude is the problem, not the actual value (positive or negative).it will set the length to be long (about 8 place values big).this is still below the vb.net limit and at the same time it seems to be very substantial especially compared to official LMC (where max value is 999).

As I mentioned before, overflow can occur during an arithmetic operation. to prevent this my program has to have a threshold when either the value added to the accumulator or the accumulator is too large the sum of the two thresholds have to lower than the limit of vb.NET. The threshold will be +-99999999 for both the size of accumulator and the temporary variable that will be used for adding/subtracting to/from the accumulator. Once the number exceeds those values an error will be displayed to the user and no overflow error will actually happen.

## Necessary data structures:

Every program has to use some form of a data structure to function, a data structure is a model of how data is stored and handled. There are many formats of storing data such as linked lists or binary trees. However certain data structures are better suited to certain tasks so they all have drawn backs and advantages.

I have chosen the data types which I think are most suited and most efficient for my project, some data types are necessary otherwise it would be impossible to create my code without them.

Here is a description of what each of them are:

1-dimensional array:

A one-dimensional array features only one column of a given length, a programmatic example of a 1D array would be: Array(x). Here is a visual example of how the data might be stored in a 1D array:

hello
world
22
1

2-dimensional array:

A 2-dimensional array, when displayed in a visual format it would look like a table with row and columns. It's a two dimensional since it has x and y position therefore its referred as 2D, an example of how the array would be used in code would be: Array (x, y). Here is a representation of this data type:

A	7	1
B	2	5

My program will only require those two data structures, my program will store the code in a 1D array since the array will only need to store each line of the code entered by the user, each line then could be retrieved from the array using a pointer that will allow to access a specific location of array. For example, if line 6 of the code want to be accessed then the code for this

would look like this: Array (5) =variable..., the reason why 5 is in the brackets is because arrays in vb.net are zero based so the actual position is always one less in an array.

A 2D array is very useful when storing different data that's is related in some way. For example, my little man computer interpreter will have to store variables that the user declared in a 2-d array, this means that there will have to be two different pieces of data stored to represent one thing. The variable name and its corresponding value will have to be stored so the interpreter can save and load values from a given location of the ram array and that location will be searched by looking at the variable name which will be placed in one of the columns of the array.

Here is an example of how I will use a 2d array data structure to store variables:

COUNTER	12
VALUE	33

Arrays are generally a fast and efficient store of data and data can be retrieved out of an array in O (1) time.

## Necessary data types:

### Integer:

An integer is a whole number than can be zero, negative or positive. It cannot be a decimal since it's not a floating-point data type. I will use this data type very often since arrays and everything else that requires some form of a pointer will required to be controlled by an integer value.

### Long:

long is an extension to the integer long is a value that has exactly the same properties as an integer however in vb.net a long variable is allowed to hold values of a greater magnitude.

### String:

A string is a sequence of characters which are a part of Ascii code or the Unicode, this data type will be used to handle any data that has been inputted by the user. Strings can be manipulated by the vb.net code which is very important otherwise I wouldn't be able to create my implementation VB.NET.

### Boolean:

A Boolean is a data type that can represent only two values (usually true or false), it's very useful in selection. Its state can affect how the program works if a program relies on a specific Boolean condition.

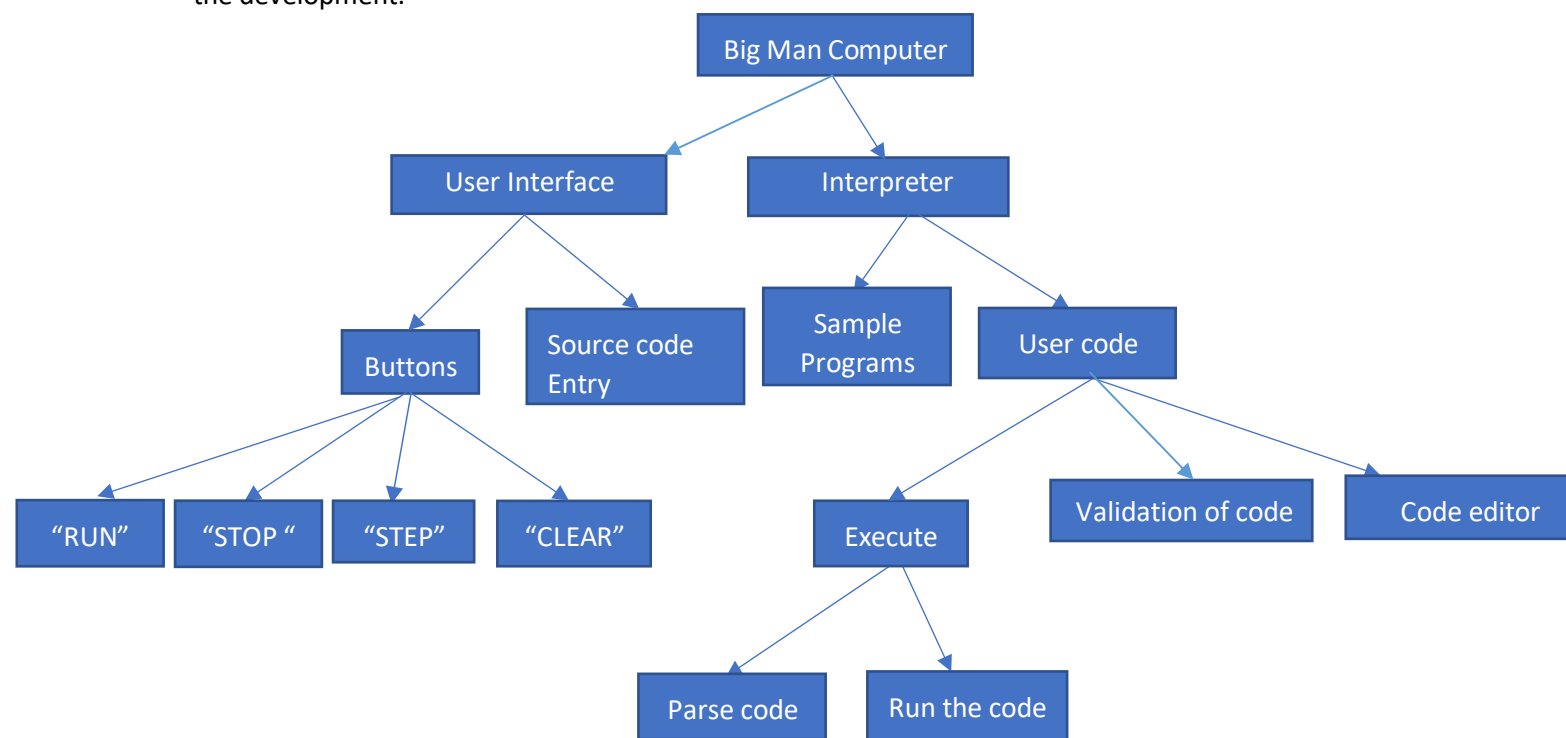
## Necessary data types /variables I will use:

variable	What is its purpose?	Data type/structure:
"Linecount"	To tell the program what line of code its reading.	integer
"RAMarray(100,1)"	To store the variables and their values so the interpreter can access them during runtime.	String(array)
"Mainarray(100)"	To store the code user has inputted.	String(array)
"Command"	Used to store the command that is currently being processed.	string
"Bdecide"	decides how "RAM Array ()" is used, if it's reading or updating data.	Boolean
"looparray(100, 1)"	Holds the tags for loops so the program can branch.	String(array)
"PLACE"	(used in lexical analysis), position within the text	integer
"HASBEEN"	Tells the logic of the program if a command is found.	Boolean
"position"	(used in lexical analysis), used in searching the array	Integer
"TEMPWORD"	Temporary word used to build strings individual characters.	String
"TEMPVAR"	Holds data of the left side of the command.	String
"VAR"	String that's on the left side of command.	String
"VARVALUE"	string that's on the right side of command	String
"FOUNDlocation"	Allows to modify ram's contents.	Integer
"RAMlocation"	Position value within the ram.	Integer

The table of variables are the variables I could think of, during the development the total number of variables could increase slightly since it's difficult to determine all variables I will require before experimenting with code. However, this table will become helpful during the development, this is an example of computational thinking since it involves planning ahead which helps to predict problems with the development as well as organise the program well.

## Structure diagram:

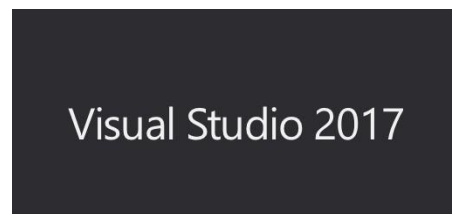
The structure diagram will help me plan out the features of my programs as well as its functionality. This is an example of computational thinking since I'm using abstraction to atomise the problem into small pieces to make the problem more digestible when I will start the development.



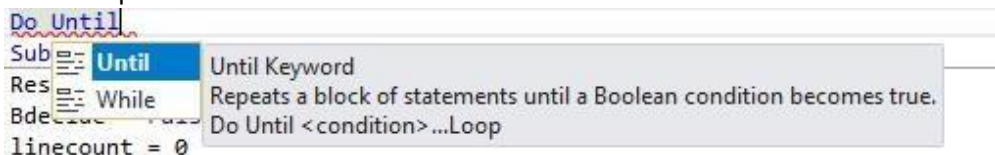
One of the main aspects of my solution is the user interface and the functionality both of those aspects have to satisfy the success criteria that I have developed with my stakeholder.

## Development:

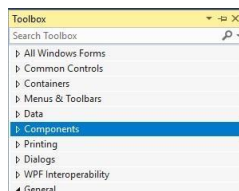
To create the code and the user form for my stakeholder I'm going to use visual studio and the programming language will be vb.net.



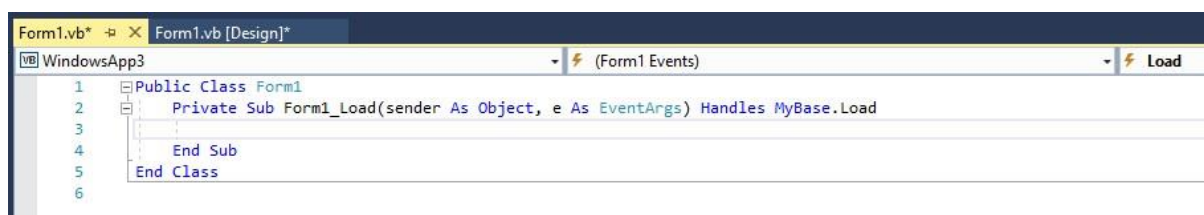
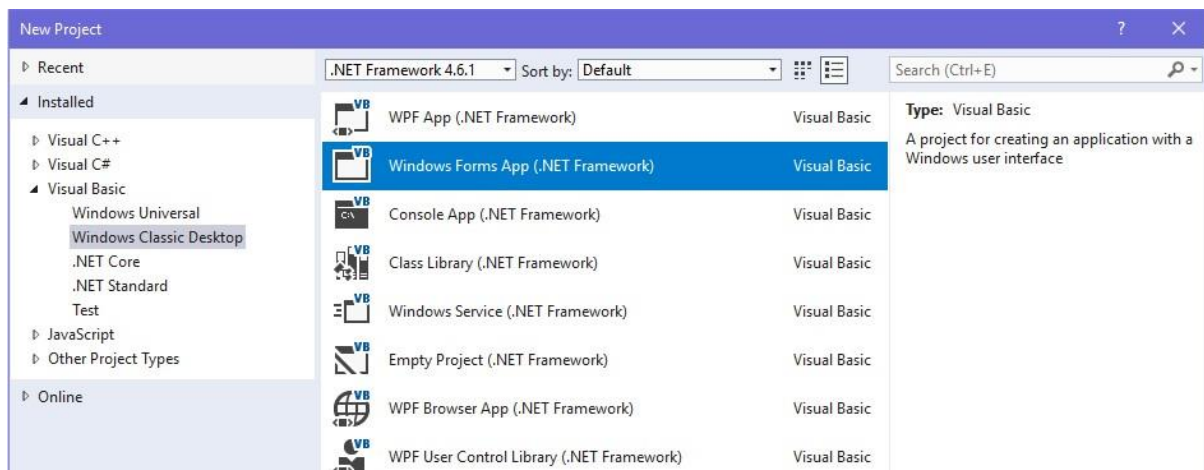
Visual studio and vb.net is packed with features that will help me be more efficient in creating the solution for my stakeholder. This IDE (integrated development environment) has many powerful features such as intellisense that helps to write vb.net code syntactically correct by showing syntax suggestions. This feature definitely helps me be a more efficient programmer. Here is an example of intellisense:



Another useful feature of visual studio is a "toolbox" that allows to create a urform really easily by using drag and drop, for example if I want to put a new button into the form I go to common controls and then I drag a and drop a button onto the user form.

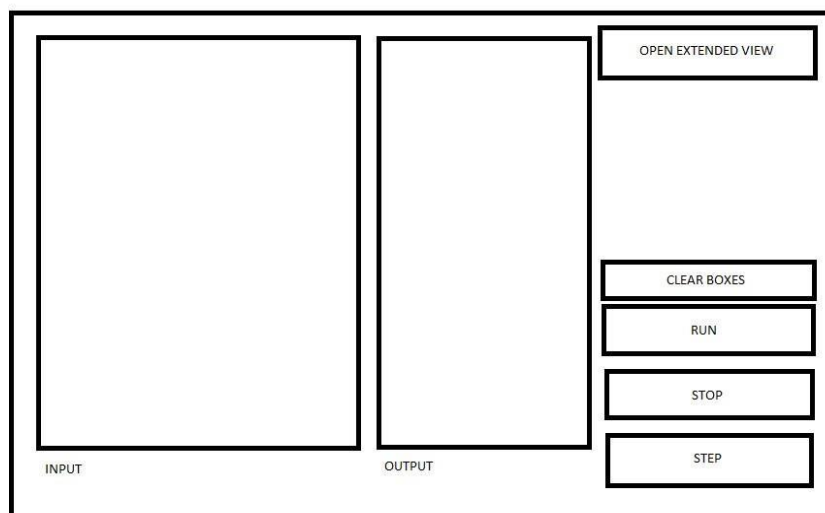






## Initial Design:

For my initial designs of the user interface I decided to use Microsoft paint since it's just a rough outline of what will be required and the layout of different things on the user form. Paint is also easy and quick to use.



The user form will feature a place to insert code and a place where it's going to generate outputs. I made the coding space fairly wide which can seem unusual because its designed for low level command set that utilizes commands that are 3 characters long. I did this because it will allow the user to use long variable names and I'm also going to implement commenting

which will allow the user to comment on their code. This is very useful because you can describe code to others more easily and also it helps to remind yourself what each line does.

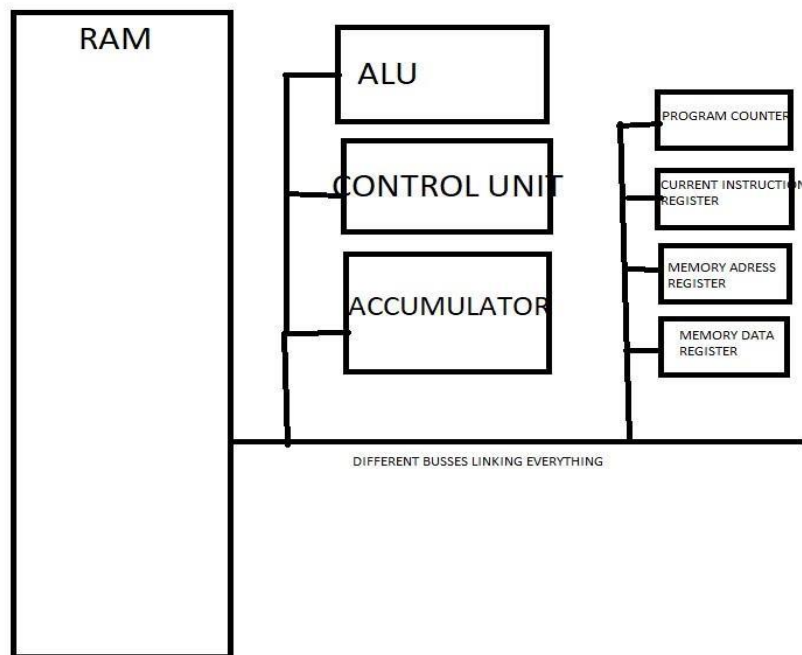
My user-form will feature a “RUN”, “STEP”, “STOP” button which will allow the user to control the program.

The “CLEAR BOXES” button will help the user clear the input/output section of the program quickly so highlighting and the deleting won’t be necessary which saves time.

“OPEN EXTENDED VIEW” will open another form that will contain all of the parts of the basic von Newman architecture so it’s not in the way if someone wants to just play around with the programming language.

I presented the initial design to my stake holder and he liked the general lay out however he recommended me to have a drop-down menu at the top of the user form which will allow to fit more buttons for more operations such as saving/loading. it will also make the program look more minimalistic since there will be less buttons on use form at once. This is a very good way of making the program simpler since some buttons won’t be used as often as others such as save and load button won’t be used as often (on average) as a “run “button or a “step” button.

I decided to only have “run”, “stop”, “step” and “clear” button on the user from and rest will be accessible using a drop-down menu that will be at the top of the user form.



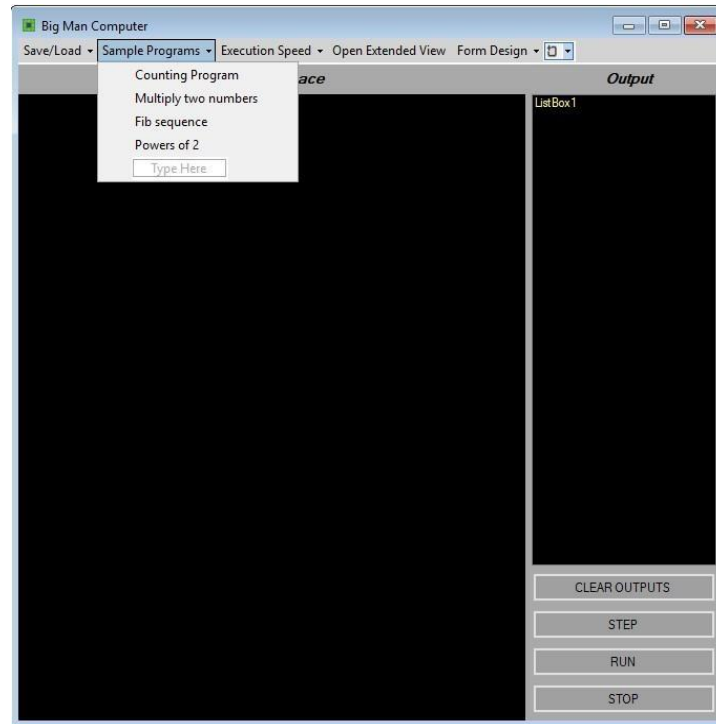
This is a prototype of the extended view that will appear once the user has pressed “EXTENDED VIEW” button.

It will feature different busses that link all of the components of the CPU together (this prototype only features a simplified version of the representation which will feature more busses), all of the registers, arithmetic and logic unit, accumulator and RAM will feature a space for numbers to appear in them as the program runs. My plan is also to show which bus is used by making it highlighted if data is passed through it. The RAM will be a linear structure since that’s how it is in a real computer. The diagram will update every time the user steps in the code. This will allow to see step by step what is going on.

This second user form will still be operated through the first user form and it won't have any buttons on it, this second form that will appear will just be used for a visualisation purpose. The design will be more refined in the final design, this is just the outline of the expected features of the form.

## Second Design:

My second design was done in a visual studio which is an IDE that I will also use to create code due to its large number of useful features.



Me and my stake holder decided that extra functionality that isn't as commonly used should be used in the top bar, it's a very space saving approach and it makes the program look more minimalistic since the drop-down menu only shows its contents when hovered over.

The form is 682 X 684 pixels, I tried to make it small enough so the extended view and the user form can be seen fully side by side on most displays.

I named the program "Big man computer" since its more advanced than the original version, it has more sophisticated features and the name doesn't differ much from the original name "little man computer" so it's recognisable.

I made the coding space very wide on my form ,this is because I will allow the users to have long variable names and also make commenting possible using the "/" key which is unique because it won't be used in my interpreter for anything else so the program won't care what's written after the "/" character .this will also allow to comment out code that the user doesn't want to execute which is also very useful because they won't need to delete a line in case they want the code to be back in that same location later on.

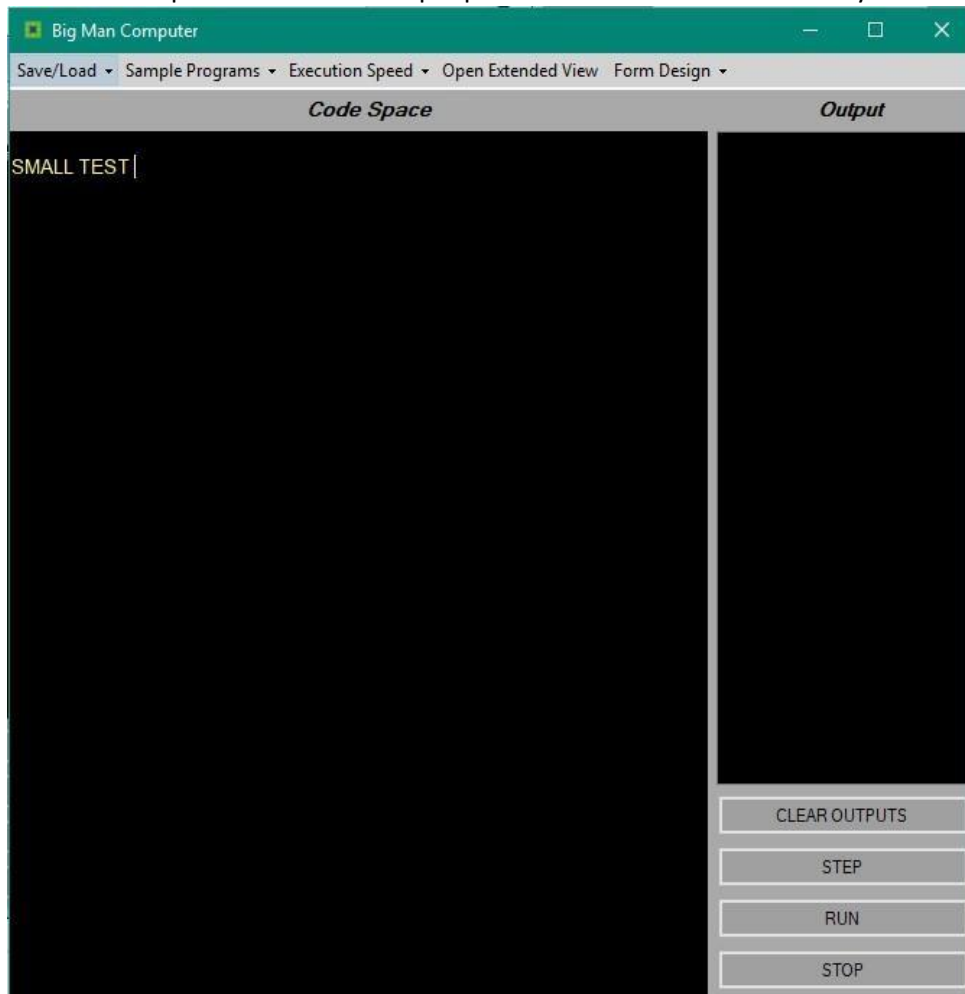
All of the less commonly used buttons have been put at the top of the form in a drop-down menu, as I said before this make the form look more simplistic and compact. It eliminates having lots of buttons on the form at once. "Save/load" to save and load programs into the user form. "Sample programs" to give the ability to load a ready-made program into LMC. This will be good for beginners to learn the code. "Execution speed" will allow the user speed up or slow

down the program so while the user views the extended view they can follow through what is going on. "Form design" will allow to customise the user form in terms of colour scheme.

Caleb wanted to see the code an output box easily and have told me that he likes having bright text on black background which was what I did in my final design. The background colour is grey and the buttons are grey except the stop button which Caleb wanted to be red.

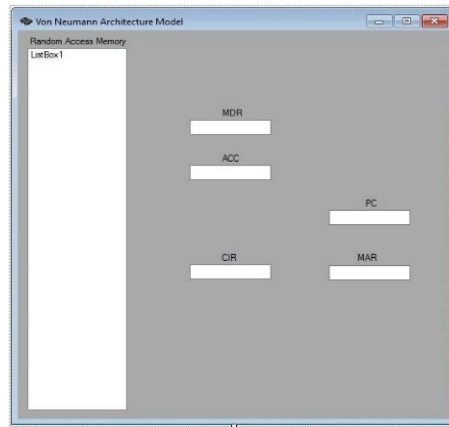
I used a list box as my output box since a list box crates a new line each time it's fed with new data which is exactly how I wanted to display outputs of the program.

I made the output and input space very clear by putting "Output" above the output box and "input" above the input section this is so people do not think it's the other way.



Visual studio allows to run the program while it's still in development stage, I run the program just to see how the user form functions and how the drop-down menu behaves, it all seemed to work as intended and no crashing so far.

After making my main form I started to create the final design for the "extended view" form which will be used to visualize the CPU functionality. I have presented this design to my stake holder and he was satisfied with it therefore we settled on this design. The design can change slightly during the development, if new ideas arise.

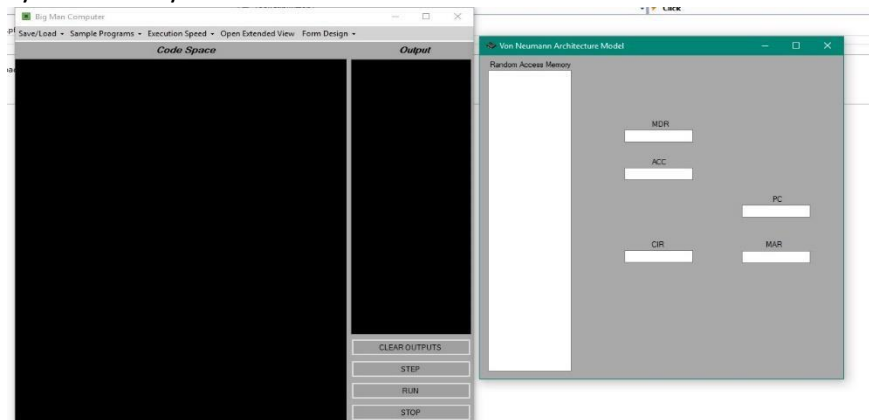


The form has text at the bottom which helps to explain what each register does. The registers don't look connected in any way because I am going to implement code that will draw lines on the form that will allow w to join the ram and registers together and also, I'll be able to display which bus is being used during execution. I will be able to do that by changing colour of each line when it's in use, for example if data is flowing from ram to MDR, the line joining the ram and MDR will change colour. Ram and all of the registers are text boxes with disabled input so the user can't input data into them since it's used for visualization purposes only.

I have used text boxes to display the contents of the registers since only one line of data needs to be displayed per register, I have used a list box for the RAM since it will be easier to add data to it because a list acts as a stack where it creates a new line when new data is inputted to it.

This form is 572x810 pixels big and it fits nicely with the other screen without obstructing the view.

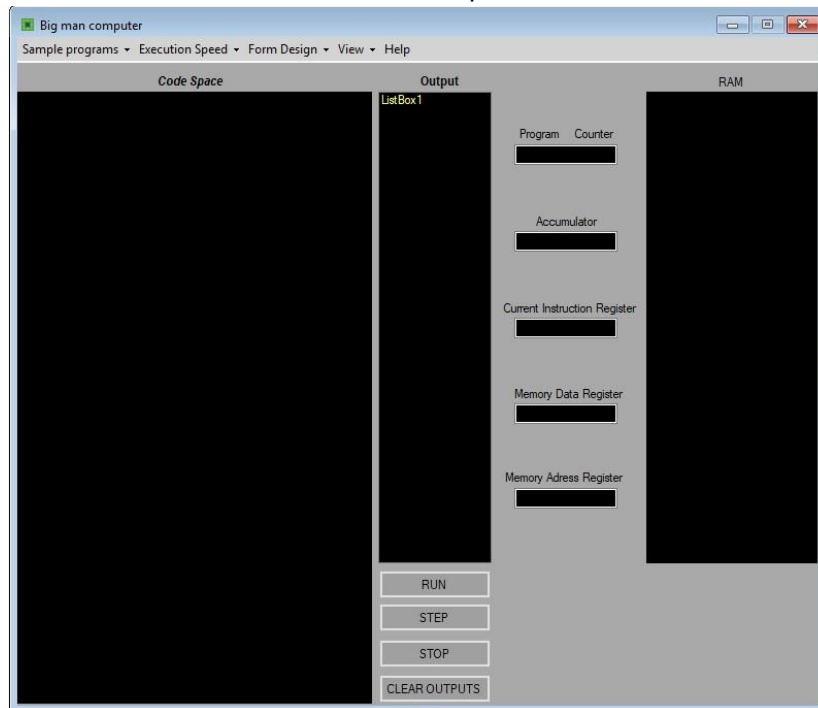
This is how they look side by side:



There is plenty of space to have the forms side by side so the user can look at the code and the "CPU visualizer" at once.

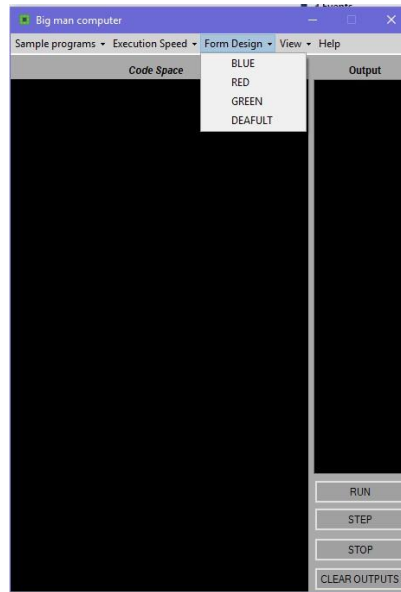
## Final Design:

After speaking with my stake holder, we decided that the program will be more user friendly if everything is integrated to the same user-form so I move the part that visualizes the von Neumann architecture on the same form, it will still be able to be hidden and opened by making the user form slide out to reveal the extended part.



This design has some minor usability tweaks that Caleb proposed I should change in my initial prototype, for example the order how the buttons are placed is different. I have placed buttons in such order so the user will have move their mouse less, if someone runs the code then they are most likely step through or stop it so those buttons are closest to run, after stopping the program the user will most likely wan to clear the outputs so the clear outputs button is places right below the stop button. Also the code font has been changed so it's a mono spaced font which makes coding better because all characters are aligned in an invisible grid and every character has the same spacing making reading code slightly better.

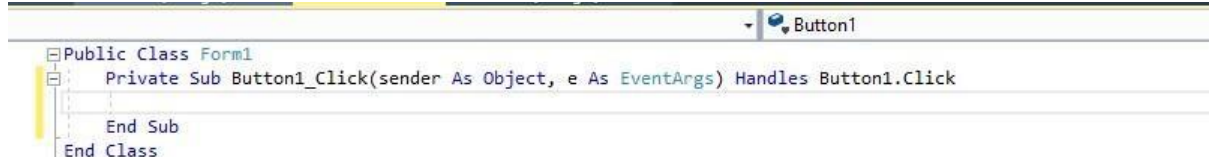
This is how the minimalistic view looks like, it has the part which shows what's happening behind the scenes hidden if someone decides just to experiment with the code.



## Beginning the development of the first prototype:

First, I decided to determine what variables I will require for this solution:

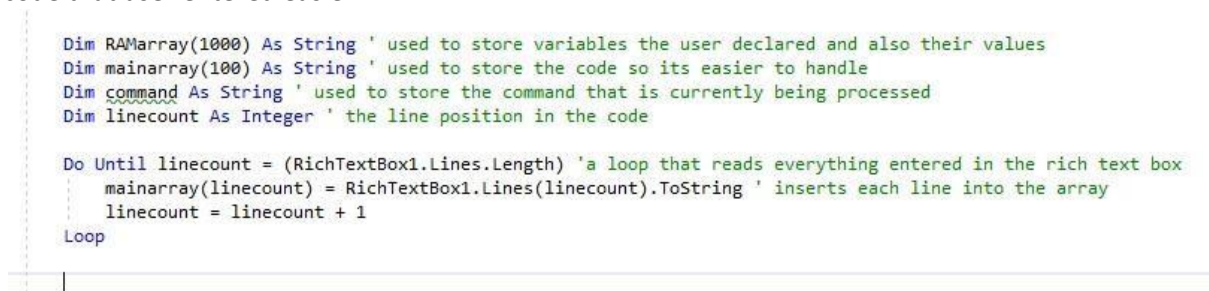
To access code space, I double clicked on the button I want to code for:



In the design section I double clicked the “RUN” button to code for that button. This is the code that appeared. The class is called Form1 and will contain all of the code that will be present in this particular form, the code can be for individual buttons and generally anything within the form.

“private sub Button1\_click....” Is a procedure that runs on click event (meaning when the button is pressed).

I started writing some code that will populate the “mainarray()” which will make handling the code that user entered easier:



I also declared some variables which will be used later on in other parts of the code.

```

Dim RAMarray(1000) As String ' used to store variables the user declared and also their values
Dim mainarray(100) As String ' used to store the code so its easier to handle
Dim command As String ' used to store the command that is currently being processed
Dim linecount As Integer ' the line position in the code

Do Until linecount = (RichTextBox1.Lines.Length) 'a loop that reads everything entered in the ric
    mainarray(linecount) = RichTextBox1.Lines(linecount).ToString ' inserts each line into the ar
    linecount = linecount + 1
Loop

End Sub

Sub lexicalAnalisys() 'used analyse the string on each line,pick up any commands and integers

End Sub

Sub RamHandler() ' handles the RAMarray,used to update the ram's contents

End Sub

Sub Interpreter() ' the part which performs a spspecific operation given the command.

End Sub
End Class

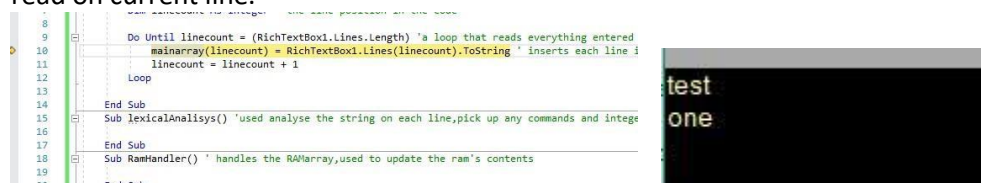
```

I added all the procedure's that I currently think will be essential.

One of the procedures "LexicalAnalysis" will be responsible for understanding each line of code, it will search for key strings(commands).it will also be responsible to find the value and loops tags on each line.

"RamHandler" will be used to populate the virtual Ram's contents (RAMarray), it will erase and update contents of the ram which will be governed by the commands that the user uses in their assembly code.

"Interpreter" will be the main part of the program that will join the functionality of all of the procedures.it will have all necessary "if" blocks that will be triggered by the command that is read on current line.



Name	Value	Type
RichTextBox1	(Text = "test" & vbCrLf & "one" & vbCrLf)	System.Windows.Forms.RichTextBox
RichTextBox1.Lines	(Length=3)	String()
linecount	2	Integer
RAMarray	(Length=101)	String()
(0)	"test"	String
(1)	"one"	String
(2)	Nothing	String
(3)	Nothing	String
(4)	Nothing	String
(5)	Nothing	String

: Test data.



Visual studio allows to step through the code which is incredibly useful for debugging. I run the code to see if the array will get populated with some data I entered. The test was successful since during the step process the contents of the array can be monitored and the data I have entered was there.

Afterwards I have started coding the part of the code that will analyse the code:

```
Sub lexicalAnalysis() 'used analyse the string on each line,pick up any commands and integers
Dim PLACE As Integer
Dim HASBEEN As Boolean
Dim TEMPWORD As String
Dim TEMPVAR As String
Dim VARVALUE As String
Dim VAR As String
PLACE = 1
If mainarray(linecount) <> "" Then 'this statment cchecks if the line is blank
Do Until PLACE = Len(mainarray(linecount)) + 1 ' this loop will iterate until variable "PLACE" equal to the length of the array
TEMPWORD = Mid(mainarray(linecount), PLACE, 3) 'this variable holds 3 letters of the
If Asc(Mid(mainarray(linecount), PLACE, 1)) <> 32 And TEMPWORD <> "DAT" And TEMPWORD <> "LDA" And TEMPWORD <> "STA" And TEMPWORD <> "SUB" And TEMPWORD <> "INP" And TEMPWORD <> "SUB" And TEMPWORD <> "I"
If Mid(mainarray(linecount), PLACE, 1) = "/" Then ' if this symbol is present then the do loop stops since this symbol allows th user to comment
Exit Do
Else
TEMPVAR = TEMPVAR & Mid(mainarray(linecount), PLACE, 1) 'builds the string that is on th right side of a LMC command
End If
End If
If TEMPWORD = "ADD" Or TEMPWORD = "SUB" Or TEMPWORD = "BRZ" Or TEMPWORD = "STA" Or TEMPWORD = "BRP" Or TEMPWORD = "OUT" Or TEMPWORD = "INP" Or TEMPWORD = "LDA" Or TEMPWORD = "DAT" Or TEMPWORD = "HLT"
VAR = TEMPVAR 'if the command is found then left side of th ecommand is saved so it can be used to represent a loop tag or a variable
TEMPVAR = ""
PLACE = PLACE + 2 ' place is incremented by 2 because the command is a 3 letter string therefore by moving 2 places will allow to shift by one place
HASBEEN = True ' this boolean tells the program if the command was found so the right side of the command can be found
command = TEMPWORD ' this passes the contents of TEMPword into command variable so the other procedure knows which command was read
End If
If HASBEEN = True Then
VARVALUE = TEMPVAR 'this stores the right side of the command
End If
PLACE = PLACE + 1 ' position in the string is increased by 1 each iteration
Loop
ElseIf mainarray(linecount) = "" Then
VAR = ""
VARVALUE = ""
command = ""
End If
HASBEEN = False 'some variables are set to default so the pcedure is ready to be fed with new data
TEMPWORD = ""
TEMPVAR = ""
End Sub
```

This part of the program finds 3 things: It finds the command such as "ADD", finds the string on the right side of the command and also it finds the string that's on the left side. Usually the string on the right side is either the loop tag or variable name during declaration. Right side is for integers and strings that are variables used by the code. The program is coded in such a way that it will recognise the code no matter how big the spaces are between the words which makes it more forgiving for beginners. When "/" is detected the programs stops any more interpreting since I decided to use this character to represent commenting. Commenting is a very powerful feature that allows to explain what each line does.

I didn't encounter any major error during the development of this piece of code, I had couple of syntactical errors such as setting "type mismatch" which means I have tried to use an integer as a string. This was easily fixed by setting it as a string since string can hold number and letters.

I temporarily inserted a call command so I can test the lexical analyser:

```
Do Until linecount = (RichTextBox1.Lines.Length) 'a loop tha
mainarray(linecount) = RichTextBox1.Lines(linecount).ToS
linecount = linecount + 1
Loop
linecount = linecount - 1
Call lexicalAnalysis()
End Sub
Sub lexicalAnalysis() 'used analyse the string on each line,pick
Dim PLACE As Integer
Dim HASBEEN As Boolean
Dim TEMPWORD As String
Dim TEMPVAR As String
Dim VARVALUE As String
```

When I stepped through the code there was a problem with the value of “line count” which was too large by one, I fixed this by subtracting 1 from its value before anything else is done. After the fix the code worked.

It was able to detect the Command, operand (the value that comes after the command) and variable name:



- The command detected was “DAT”
- The variable name detected was “HELLO”
- The value of that command detected was “10”

```
Public Class Form1
    Dim linecount As Integer ' the line position in the
    Dim RAMarray(100, 1) As String ' used to store varia
    Dim mainarray(100) As String ' used to store the cod
    Dim command As String ' used to store the command th
    Dim Bdecide As Boolean
    Private Sub Button1_Click(sender As Object, e As Eve
```

I had to set certain variables so they are global because they are used across different procedures.

Also, I have added “Bdecide”, this is a Boolean which will control whether the ram is being only read from or is it’s being populated with a new variable.

Next, I have added the code which will initialize the program one “RUN” button is clicked by the user.

```
Dim VARVALUE As String 'string thats on the right side of command
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Do Until linecount = (RichTextBox1.Lines.Length) 'a loop that reads everything entered in the rich text box
        mainarray(linecount) = RichTextBox1.Lines(linecount).ToString ' inserts each line into the array
        linecount = linecount + 1
    Loop
    linecount = linecount - 1
    Do Until command = "HLT" ' this loop will repeat until it finds "HLT" command
        Call lexicalAnalysis()
        If linecount = 0 And command <> "HLT" Then ' validation to inform the user that they need to insert a "HLT" command
            MsgBox("NO HALTING COMMAND FOUND")
            Call Reset() 'procedure that resets all variables to default state
            Exit Sub
        ElseIf command <> "" And command <> "HLT" Then ' if "HLT" inst reached then ramhandler can populate the array with data
            Call RamHandler()
        End If
    Loop
    linecount = 0
    Do Until linecount = (RichTextBox1.Lines.Length) 'this loop looks for any loops in th ecode
        Call lexicalAnalysis()
        looparray(linecount, 0) = VAR
        linecount = linecount + 1
    Loop
    Bdecide = True
    command = ""
    linecount = 0
    Call Interpreter()
End Sub
Sub lexicalAnalysis() 'used analyse the string on each line,pick up any commands and integers
```

The first loop is the loop I have already discussed, the second loop allows to place the declared variables into the “RAMarray”. It also has validation that ensures a “HLT” command is present. The third loop is responsible for finding all of the loop tags that may be within the code that has been inserted by the user.

While creating the code I have been doing lots of “thinking ahead” which is one of the types of computational thinking. This is because I had to write code that can handle a large number of permutations of code which can contain syntax errors, this means the code I’m making has to be able to recognise those errors and be also flexible enough not to crash.

One of the things I took into consideration was vertical and lateral spacing, my program is able to ignore spacing no matter how much separated the words are. Also, it can detect if a “HLT” command is entered which is very important that this check is done otherwise my program would crash if the code proceeded to run any further.

```

End Sub
Sub RamHandler() ' handles the RAMarray, used to update the ram's contents
    Dim RAMlocation As Integer ' position value within the ram
    Dim position As Integer ' used in searching the array
    If Bdecide = False Then 'this part is used for searching for data in the ram
        Do Until VAR = RAMarray(RAMlocation - position, 1) 'this part compares the thing its looking for to the rams content in that location
            position = position + 1
            If position > RAMlocation Then 'validation so the array doesnt get searched in -1 position which is invalid
                MsgBox("Could not locate variable")
                Exit Sub
                Call Reset()
            End If
        Loop
        position = 0
    ElseIf Bdecide = True Then ' this part is used for inserting new data into the ram
        RAMarray(RAMlocation, 1) = VARVALUE 'inserts the variable
        RAMarray(RAMlocation, 0) = VAR ' inserts the variables value
        RAMlocation = RAMlocation + 1
        If RAMlocation = 100 Then 'limited ram space to 100 slots
            MsgBox("RAM CAPACITY EXCEEDED")
            Call Reset()
            Exit Sub
        End If
    End If
End Sub

```

I started coding the part of the program which will handle the “RAMarray”, I added a new variable called “RAMlocation” which allows to view different locations of the array as well as modify them.

After some time, I managed to create some code that allows the ram to be searched as well as allow data to be inserted to it.

```

End Sub
Sub RamHandler() ' handles the RAMarray, used to update the ram's contents
    Dim RAMlocation As Integer ' position value within the ram
    Dim position As Integer ' used in searching the array
    If Bdecide = False Then 'this part is used for searching for data in the ram
        Do Until VAR = RAMarray(RAMlocation - position, 1) 'this part compares the thing its looking for to the rams content in that location
            position = position + 1
            If position > RAMlocation Then 'validation so the array doesnt get searched in -1 position which is invalid
                MsgBox("Could not locate variable")
                Exit Sub
                Call Reset()
            End If
        Loop
        position = 0
    ElseIf Bdecide = True Then ' this part is used for inserting new data into the ram
        RAMarray(RAMlocation, 1) = VARVALUE 'inserts the variable
        RAMarray(RAMlocation, 0) = VAR ' inserts the variables value
        RAMlocation = RAMlocation + 1
        If RAMlocation = 100 Then 'limited ram space to 100 slots
            MsgBox("RAM CAPACITY EXCEEDED")
            Call Reset()
            Exit Sub
        End If
    End If
End Sub

```

“Bdecide” is the Boolean I mentioned before that governs the decision whether the “RAMarray” should be written to or read from. It uses a linear search for now but I’m planning to implement an optimization that will allow the retrieval of data to be  $O(1)$  instead of  $O(n)$  in terms of time complexity.

I will be able to increase performance a lot by using direct addressing. I will implement direct addressing by analysing each line and checking what memory location each line of code is needing then the memory location for that line will be saved. This means that if there is a loop in the LMC code, next time the same line is visited, the required location that locates the variable and its value will be already known by a simple pointer.

This approach will be efficient to use if a user is using a loop but otherwise it can reduce efficiency resulting in a negative impact on performance. This means I will have to implement some selection that will decide when to use pointer method I mentioned or whether just do a linear search.

Example of the pointer implementation:

ADD A	1
LDA C	4
ADD B	3
STA D	2

The left side is an example of how the contents of the “mainarray()” could look, the right side is the implementation of pointers that tell the program where each variable is without doing any searching. The search will only happen once when the code is run first time, if there is a loop then the next time the line would be visited the location of the variable can be found instantly.

This is the part of the code that will receive commands and run appropriate blocks of code given the command:

```

Sub Interpreter() ' the part which performs a spspecific operation given the command.
    linecount = 0
    Do Until command = "HLT"

        If command = "ADD" Then

        ElseIf command = "SUB" Then

        ElseIf command = "LDA" Then

        ElseIf command = "INP" Then

        ElseIf command = "BRZ" Then

        ElseIf command = "BRP" Then

        ElseIf command = "BRA" Then

        ElseIf command = "STA" Then

        ElseIf command = "OUT" Then

        ElseIf command = "DAT" Then

        End If
        linecount = linecount + 1
    Loop
    Call Reset()
End Sub

```

There is an "if" statement for every command that is present in the little man computer command set. This large if statement is in a loop that will run until the program reads the "HLT" command, After the loop exits everything will be reset so modified/new code can be inserted by the user. by having the code in the loop, it will make coding branching commands neater since to allow branching all that has to be done is the line count has to be set to a value that points to the loop tag.

```

Do Until command = "HLT" ' runs loop until HLT is found
    If command = "ADD" Then 'adds a value to accumulator
        If IsNumeric(VARVALUE) = True Then ' checks if a avriable is added or a integer
            temp = VARVALUE ' allows to value to be added to accumulator
        Else
            Call RamHandler()
            temp = RAMarray(FOUNDlocation, 1)
        End If
        accumulator = accumulator + temp ' adds the value to the acumulator
    ElseIf command = "SUB" Then 'subtracts a value from the accumulator
        If IsNumeric(VARVALUE) = True Then ' checks if a avriable is added or a integer
            temp = VARVALUE ' allows to value to be subtracted from accumulator
        Else
            Call RamHandler()
            temp = RAMarray(FOUNDlocation, 1)
        End If
        accumulator = accumulator - temp ' subtracts the value to the acumulator
    ElseIf command = "LDA" Then 'loads a value into the accumulator

```

I started adding code to the different if blocks, as shown above, the "if" statement: "if command = ADD then", adds the value to the accumulator. The value is either stored by a variable or is directly written in code such as "ADD 20". The same rules apply for the "SUB" command.



```

        MsgBox("COULD NOT FIND TAG")
        Call Reset()
        Exit Sub
    End If
    Loop
    If linecount = loopcounter Then 'this could result in a infinite loop if it was allowed
        MsgBox("ILLEGAL OPERATION")
        Call Reset()
        Exit Sub
    End If
    linecount = loopcounter - 1
    loopcounter = 0
End If
ElseIf command = "BRP" Then ' Branches if accumulator is positive
    If accumulator > 0 Then
        Do Until looparray(loopcounter, 0) = VARVALUE 'finds the location of the tag
            loopcounter = loopcounter + 1
            If loopcounter = RichTextBox1.Lines.Length Then ' determines if the tag exists
                MsgBox("COULD NOT FIND TAG")
                Call Reset()
                Exit Sub
            End If
        Loop
        If linecount = loopcounter Then 'this could result in a infinite loop if it was allowed
            MsgBox("ILLEGAL OPERATION")
            Call Reset()
            Exit Sub
        End If
        linecount = loopcounter - 1
        loopcounter = 0
    End If
ElseIf command = "BRA" Then ' Branches unconditionally
    Do Until looparray(loopcounter, 0) = VARVALUE 'finds the location of the tag
        loopcounter = loopcounter + 1
        If loopcounter = RichTextBox1.Lines.Length Then ' determines if the tag exists
            MsgBox("COULD NOT FIND TAG")
            Call Reset()
            Exit Sub
        End If
    Loop
    If linecount = loopcounter Then 'this could result in a infinite loop if it was allowed
        MsgBox("ILLEGAL OPERATION")
        Call Reset()
        Exit Sub
    End If
    linecount = loopcounter - 1
    loopcounter = 0
ElseIf command = "STA" Then 'stores the value of an accumulator into a variable
    Call RamHandler()
    RAMarray(FOUNDlocation, 1) = accumulator
ElseIf command = "OUT" Then ' outputs the contents of the accumulator
    If IsNumeric(VARVALUE) = True Then
        temp = VARVALUE
    ElseIf VARVALUE = "" Then
        ListBox1.Items.Add(accumulator)
    Else
        Call RamHandler()
        temp = RAMarray(FOUNDlocation, 1)
        ListBox1.Items.Add(temp)
    End If
End If
linecount = linecount + 1

```

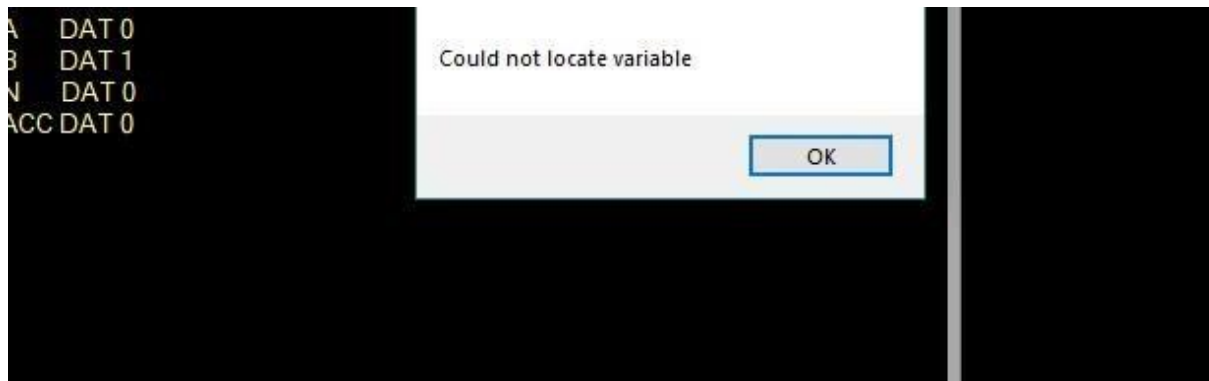
This is the rest of the code I have added, this procedure relies on many global variables that are used in other procedures, for example a very common global variable is “linecount” that allows other procedures such as “lexical analysis” to analyse the given line number.

```

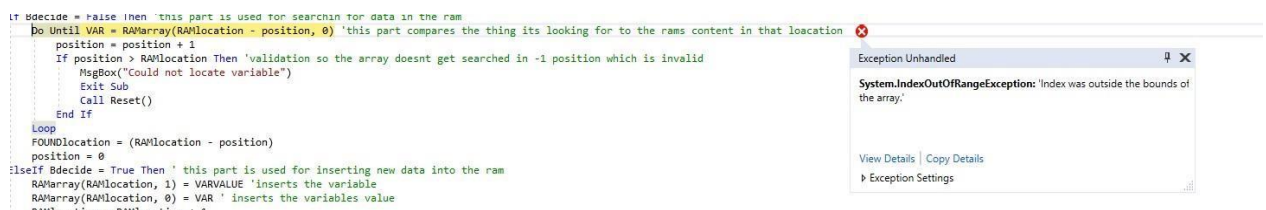
101 End Sub
102
103 Sub Interpreter() ' the part which performs a spspecific operation given the command.
104     linecount = 0
105     Dim accumulator As Integer ' stores the running value
106     Dim temp As Integer ' used to add subtract a value from accumulator
107     Dim loopcounter As Integer ' allows to search for a tag

```

After I have finished coding every “If” block I did I small test with some test data that was a Fibonacci sequence program:

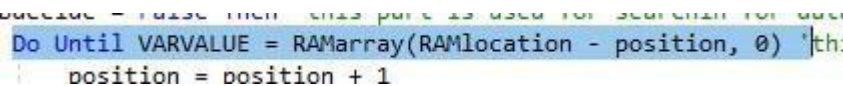


I stepped through the code and I noticed a logical error that was present in the code, I first changed the value of the 2-dimensional array from 1 to 0 since anything (x, 1) holds the variable value and anything (x, 0) is the variable name therefore I knew that this was one of the issues.

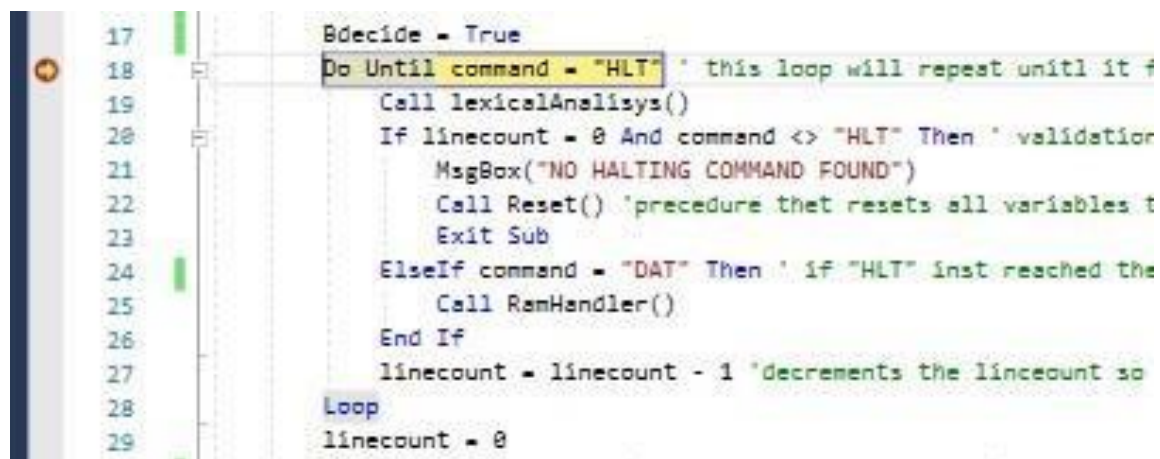


After running the code again, I noticed that another issue occurred. This problem was related to trying to read location of value of -1 from the array which is impossible. I managed to fix the issue by exiting the loop and it seemed to work. Another thin that was suspicious to me was that “variable not found” should not be displaying since all variables are declared properly.

Then I discovered that there was a major logical mistake I have made:



This line of code had to have “VARVALUE” instead of “VAR” since in this case “VARVALUE” represents the variable not its value due to the way LMC code is laid out. This is because LMC first has its variables on the right when they are declared but when other commands are used they are placed on the right.

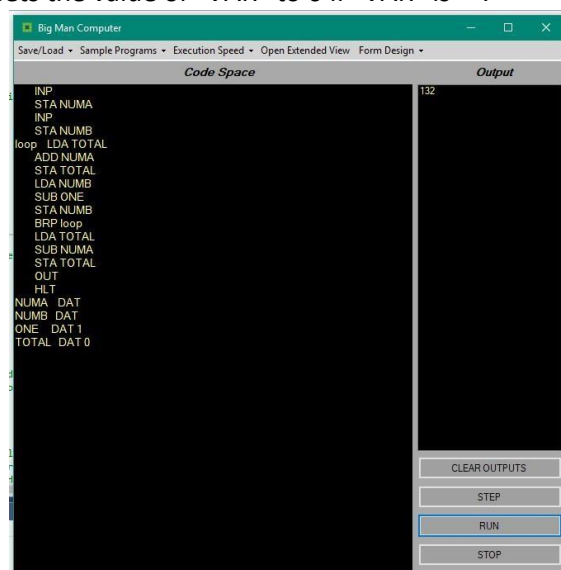


After doing those changes I noticed that the program functioned as intended when correct/acceptable data is inserted.

I have noticed that LMC also allows to declare a variable without assigning an initial value to it, even zero. This is problematic since it would cause a type mismatch since "" is not classed as an integer in vb.net. When the "RAMarray" is populated, there cannot be "" inserted into the array so I made a simple piece of code that solves the problem:

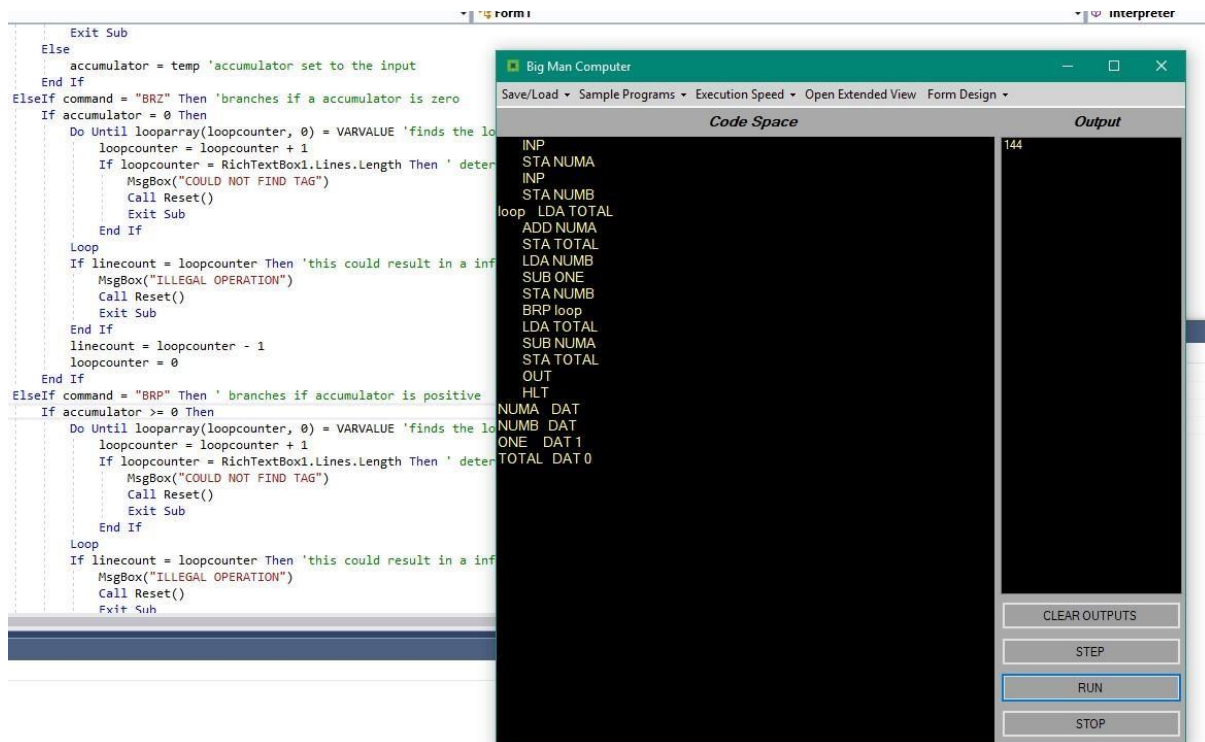
```
VAR = TEMPVAR ' If the command is found then left side of the command is saved
If VAR = "" Then ' this simple validation prevents type mismatches
    VAR = 0
End If
TEMPVAR = ""
PLACE = PLACE + 2 ' place is incremented by 2 because the command is a 3 lett
HASRFFN = ' program if the command was found so t
```

This small statement sets the value of "VAR" to 0 if "VAR" is "".



I run a multiplication program and I have inputted the data 12 x 12 so the product of those values should be 144 instead I got 132. since 132 is in the 12 times table, in fact its 12x 11, I thought that it must be something with the loop exiting too soon.





The issue was in the "interpreter" procedure:

The problem was an inaccurate condition that when:  $\text{accumulator} > 0$  then the rest of the functionality of the "BRP" command is executed. This statement excludes the value 0 which LMC happens to include for its "BRP" command. I fixed this by simply changing the symbol from  $>$  to  $\geq$  to include 0 as a valid value for the if statement to be triggered.

As shown in the picture above, this made the program work and the result was 144 which is correct.

```

End Sub
Sub Reset()
    PLACE = 0
    HASBEEN = False
    TEMPWORD = ""
    TEMPVAR = ""
    linecount = 0
    command = ""
    Bdecide = False
    VAR = ""
    VARVALUE = 0
    FOUNDlocation = 0
    RAMlocation = 0
    position = 0
    PLACE = 0
    HASBEEN = False
    TEMPWORD = ""
    TEMPVAR = ""
    accumulator = 0
    temp = 0
    loopcounter = 0
    Array.Clear(mainarray, 0, mainarray.Length)
    Array.Clear(RAMarray, 0, RAMarray.Length)
    Array.Clear(looparray, 0, looparray.Length)
End Sub
End Class

```

This procedure resets/changes everything to default so new data can be inserted and there isn't conflict with old data that could have stayed in the program. It gets used every time an error is detected and every time the code finished running.

```

Private Sub wait(ByVal interval As Integer)
    Dim sw As New Stopwatch ' allows to delay the program
    sw.Start()
    Do While sw.ElapsedMilliseconds < interval
        ' Allows UI to remain responsive
        Application.DoEvents()
    Loop
    sw.Stop()
End Sub
End Class

```

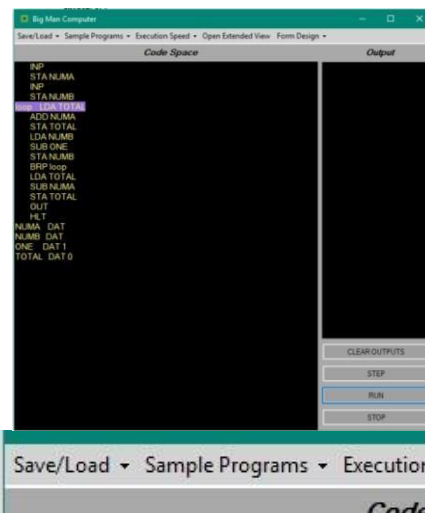
This function allows to slow down the program so it doesn't run in real time and also doesn't freeze the software which is convenient. I will implement it in different parts of the program that would benefit from being slowed down to let the user see what it's going on if they want to see the program un almost in slow motion. This can be used for my step button that requires a delay so the user can have time to see what's going on.

```

End If
linecount = linecount + 1
wait(wait1)
RichTextBox1.Select(RichTextBox1.GetFirstCharIndexFromLine(linecount), Len(mainarray(linecount)))
RichTextBox1.SelectionBackColor = Color.MediumPurple
wait(wait1)
RichTextBox1.Select(RichTextBox1.GetFirstCharIndexFromLine(linecount), Len(mainarray(linecount)))
RichTextBox1.SelectionBackColor = Color.Black
Loop
Call Reset()
End Sub

```

At the bottom of the "interpreter" procedure I have added the delays that will be controlled by a variable called "wait1" since the name "wait" is ambiguous. The rest of the code is responsible for highlighting each line that is visited so the user can see which line of code is executed.



This is a small demonstration it working, the line is that is Being currently read is highlighted in blue.

Since I got the hardest part of the code working with correct data I will now try to finish simpler parts such as the drop-down menu that is on the user form.

```

3 Private Sub LineSecondToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles LineSecondToolStripMenuItem.Click
    wait1 = 140
End Sub

3 Private Sub LinessecondToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles LinessecondToolStripMenuItem.Click
    wait1 = 30
End Sub

3 Private Sub RealTimeToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles RealTimeToolStripMenuItem.Click
    wait1 = 0
End Sub

3 Private Sub ToolStripButton1_Click(sender As Object, e As EventArgs) Handles ToolStripButton1.Click
    Form3.Show()
End Sub

3 Private Sub WhiteToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles WhiteToolStripMenuItem.Click
    RichTextBox1.SelectionBackColor = Color.White
End Sub

3 Private Sub BlackdefaultToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles BlackdefaultToolStripMenuItem.Click
    RichTextBox1. = Color.White
End Sub

3 Private Sub GreenToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles GreenToolStripMenuItem.Click
    Me.BackColor = Color.Green
End Sub

3 Private Sub BlueToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles BlueToolStripMenuItem.Click
    Me.BackColor = Color.LightBlue
End Sub

3 Private Sub DeafulToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles DeafulToolStripMenuItem.Click
    Me.BackColor = Color.DarkGray
End Sub
End Class

```

I have the started adding functionality to the drop-down menu by double clicking on each button in designer section which opened up procedures that run when specific button is pressed.

This prototype still doesn't feature certain functionalities, the main focus for prototype is to finish the interpreter part since that's the most complex part of the project.

## Testing the first prototype:

To test the first prototype, I will use some little man computer code that works in original LMC and see if it works in mine, after that I will test my program with invalid data. Caleb will also be involved in testing since he could think of tests that I haven't thought of, also he then will be able to share his initial impression of the program so I will know better what to improve in my second iteration of the development of the program.

## Testing strategy:

I will try to first test every validation I have implemented to see if the safe guards that I made to prevent crashing work. After that I will ask my stake holder to test my first prototype and submit any errors /problems/improvements in a txt file so then I can use that data to improve the prototype and effectively create a second prototype. This will also tell me how usable the program is and whether the layout is convenient for Caleb.

I also asked my stakeholder to try inputting and combination which might crash the program, this is a type of black box testing since my stake holder will be doing the test, by having another person to test my program I will be able to find more problems since the programs might be used in ways not thought of before by me.

## White box testing of added validation:

<u>Test data</u>	<u>outcome</u>	<u>How is the data deliberately incorrect?</u>	<u>Pass(y/n)</u>
INP STA N loop LDA A SUB N BRP e LDA A OUT LDA B ADD A STA ACC LDA B STA A LDA ACC STA B	Program outputs fib sequence: 0 1 1 2 3 5 8..	No deliberate errors made.	Yes.






BRA loop e HLT A DAT 0 B DAT 1 N DAT 0 ACC DAT 0			
---	--	--	--

INP STA N loop LDA A SUB N BRP e LDA A OUT LDA B ADD A STA ACC LDA B STA A LDA ACC STA B BRA loop HLT A DAT 0 B DAT 1 N DAT 0 ACC DAT 0	The program display an error message "loop tag not found".	Removed one of the tags.	Yes.
INP STA NUMA INP STA NUMB loop LDA TOTAL ADD NUMA STA TOTAL LDA NUMB SUB ONE STA NUMB BRP loop LDA TOTAL SUB NUMA STA TOTAL	Program displays an error message: "cannot locate a variable"	One of the variables was renamed.	Yes.

OUT HLT NUM DAT NUMB DAT ONE DAT 1 TOTAL DAT 0			
INP STA A loop LDA A SUB 1 STA A OUT A LDA A BRP loop HLT A DAT 0	Program runs Into an infinite loop and crashes when "real time" is selected.	-1 was inserted as an input	NO.
INP STA A/TEST loop LDA A SUB 1 STA A /HELLO /COMMENT 1 OUT A LDA A BRP loop HLT A DAT 0	The program counts down as it should.	Commenting added using the backslash character.	Yes.
INP STA A loop LDA A HLT SUB 1 STA A OUT A LDA A BRP loop HLT A DAT 0	The program exits early.	A "HLT" command has been added on line 4.	Yes.

INP STA A loop LDA A SUB 1 STA A OUT A LDA A BRP loop HLT A DAT A	The program displays an error that says: "fatal error "	A string has been inserted where an integer should be.	Yes.
INP	The program crashes due to an	Two large integers where multiplied in this	No.
STA NUMA INP STA NUMB loop LDA TOTAL ADD NUMA STA TOTAL LDA NUMB SUB ONE STA NUMB BRP loop LDA TOTAL SUB NUMA STA TOTAL OUT HLT NUM DAT NUMB DAT ONE DAT 1 TOTAL DAT 0	Exception error due to an overflow.	program : 9999 x 9999	

INPUT into input box: ABDWAA	The program displays an error message which tell the user that only integers can be entered.	String was inserted into input box instead of an integer.	Yes.
------------------------------	--	---	------

Validation that is currently implemented:	Did it prevent the program from crashing/error?	Screen shot of outcome:
Validation to inform the user there's nothing entered.	Yes, pass.	
Validation to prevent strings being entered where number should be.	Yes, pass.	
Validation to inform user that a loop tag is not found.	Yes, pass.	
Validation for invalid data type input when declaring variable :( A DAT W).	Yes, pass.	
Validation that prevents entering more data than the array can hold.	Yes, pass.	
Multiplying two 6-digit integers using the multiplication program, value placed into input box.	No, fail	Overflow error in vb.net

The testing revealed that I will have to make my program be able to handle infinite loops in such a way that it doesn't make it crash. I can either slow down the program so when the real time button is selected, it is slightly slower than the fastest speed it can do it in and I can then display a message every 10,000 loops whether the user want to continue or not. I also could create a program that will detect whether infinite loop will happen by checking the contents of the accumulator and whether it's approaching the exit condition or whether it's staying the same or even diverging from the exit condition. the only problem with this approach is that this implementation would only work on the "BRP" command because "BRA" is unconditional and "BRZ" needs to have zero in the accumulator to work.

To prevent overflow errors, I will add validation that will prevent adding a large number to the accumulator if the accumulator is already storing a large number, this will apply to negative



numbers as well by implementing "math.abs" which will look at the absolute values of the numbers no matter if they are negative or positive.

The variation of inputs that the user can enter is almost endless so it will be impossible for me to test every program/combinations of code. I can however test it in general which means I can test each aspect of the code one by one individually as I did above. If all of the commands work and loops work then it's safe to assume that my solution should work with any "LMC" code that would work in original little man computer.

My personal testing didn't reveal any issues apart from overflow error, however Caleb's testing revealed some flaws in the code, here is the txt file Caleb has written for me to tell me what problems he came across:

Caleb's comments:

*When selecting new sample programs, the program pasted on top what was already in the box which was inconvenient.*

*When hovering over drop down menu it didn't drop down which was not intuitive.*

*When running the re running the program couple of times the program displayed: "could not display variable".*

*When I selected the faster mode, the highlighting became "glitchy" and some lines didn't appear to be highlighted.*

*The location of error could be specified using the line number.*

I have to go back to code development and tweak my program to solve those issues.

## Beginning the development of the second prototype:

The first prototype was very promising and has an almost fully functional interpreter. However, there was still few issues and there are few functionalities I have to implement such as the "STEP" button.

Firstly I wanted to address the problem involving a glitch which occurs when faster mode is selected. I realized the faster mode is running faster than the code is able to highlight text which means the fix wasn't complicated, all I had to do is tweak the delay so the code can have time to highlight each line. Here is the solution to the problem:

```
Private Sub LineSecondToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles LineSecondToolStripMenuItem.Click
    wait1 = 250
End Sub
Private Sub LinesSecondToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles LinesSecondToolStripMenuItem.Click
    wait1 = 50
End Sub
Private Sub RealTimeToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles RealTimeToolStripMenuItem.Click
    wait1 = 0
End Sub
Private Sub Button5_Click(sender As Object, e As EventArgs) Handles Button5.Click
```

In the second procedure (responsible for the second fastest mode), I increased the delay from 20 to 50 which is over double the delay than previously, this eliminated the glitch.

Then I considered how my program will deal with overflows, the first prototype was only able to capture an overflow if a number entered was too large however an overflow can also occur during an arithmetic operation which adding or subtracting two integers.

```

If accumulator < -99999999 Or temp > 99999999 Then
    MsgBox("Overflow Error")
    Call Reset()
    Exit Sub
End If

```

This piece of code allows to detect an overflow before a physical overflow occurs due to the limitations of the CPU architecture or the limitations of the vb.net programming language. I set a threshold value to be very high (much higher than most implementations of little man computer) that is still below the physical limit of the system. It's a basic if statement that comes before every arithmetic operation that LMC can perform (which is addition and subtraction).

```

End Sub
Sub Interpreter() ' the part which performs a specific operation given
    Dim temp As String ' used to add subtract a value from accumulator
    Dim loopcounter As Integer ' allows to search for a tag
    Dim accumulator As Integer ' stores the running value
    Do While stopcode = False ' runs loop while the user has not pressed
        command = executionpaths(linecount, 0)
        If wait1 <> 0 Then
            TextBox1.Text = accumulator
            Call graphics()
        End If
    End While
End Sub

```

Sometimes when the "STOP" button has been pressed and error occurred due to a logical error involving the code being out of sync, I implemented a do while loop which will exit if the condition is met, the condition is when "stop code" is equal to true, The "STOP" button will be responsible for changing the Boolean to true.

## Algorithm for the step button:

The step button has to be able to stop the interpreter at a given line and when pressed it has to allow the interpreter to read next line and stop again.

```

Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
    If start = True Then
        beenpressed = True
        Num = Num + 1
    Else
        beenpressed = True
        Num = Num + 1
        Call Initialazation()
    End If
    wait1 = 250
End Sub

```

This procedure is executed every time the user clicks the "STEP" button, if the "RUN" button wasn't pressed beforehand then the step button can also act as a run button initially however it stops the code until its pressed again(just like step button should works)."Num" is a running total and it gets updated every time the button is pressed this allows other procedure determine if it was pressed by updating the value of "Num".

```

If beenpressed = True Then 'this if block allows the program to be stepped through
    Do While stopcode = False
        If Num Mod 2 = 0 Then 'checks if num1 is a multiple fo 2
            Num = Num + 1 'makes the value odd again
            Exit Do
        End If
        wait(8)
    Loop
End If

```

This part of the code is inside the “interpreter ()” procedure and it allows the interpreter to stop on a specific line. it is a constantly running loop which is checking if the step button has been pressed, this is determined by the condition of “num” being an even number which can be determined using the modulus function. If “num” is a multiple of 2 then there will be no remainders meaning the modulus of those two values will be equal to 0.

If the condition is met then the loop can exit and “num” is changed to an odd number again, this allows the interpreter to execute a new line and enter this loop again until the condition is met again (when the button is pressed again). I implemented a small delay “wait (8)” this puts less stress on the CPU since the loop isn’t running at full speed at all times.

```

position > RAMlocation Then 'validation so the array doesnt get searched
    MsgBox("Could not locate variable at line : " & linecount + 1)
    notfound = True

```

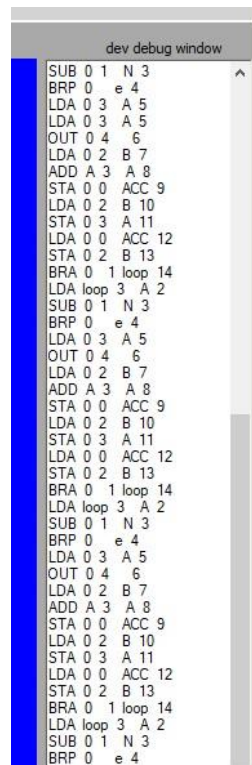
This portion of the code is located in the “RAMhandler()” procedure and if the variable is not found in that array, the code is able to inform the user on which line the error occurred by specifying the line number (linecount+1, arrays are zero based as I mentioned before).

```

End Sub
Private Sub ToolStripDropDownButton1_MouseEnter(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ToolStripDropDownButton1.MouseEnter
    Me.ToolStripDropDownButton1.ShowDropDown()
End Sub
Private Sub ToolStripDropDownButton4_MouseEnter(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ToolStripDropDownButton4.MouseEnter
    Me.ToolStripDropDownButton4.ShowDropDown()
End Sub
Private Sub ToolStripDropDownButton3_MouseEnter(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ToolStripDropDownButton3.MouseEnter
    Me.ToolStripDropDownButton3.ShowDropDown()
End Sub
Private Sub ToolStripSplitButton1_MouseEnter(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ToolStripSplitButton1.MouseEnter
    Me.ToolStripSplitButton1.ShowDropDown()
End Sub
End Class

```

This piece of code was added for better convenience, it allows the drop down menu to drop down automatically when the mouse is hovered over it.



The screenshot shows a 'dev debug window' with a list of assembly instructions. Each line is preceded by a blue vertical bar, indicating it has been executed. The instructions are as follows:

```

SUB 0 1 N 3
BRP 0 e 4
LDA 0 3 A 5
LDA 0 3 A 5
OUT 0 4 6
LDA 0 2 B 7
ADD A 3 A 8
STA 0 0 ACC 9
LDA 0 2 B 10
STA 0 3 A 11
LDA 0 0 ACC 12
STA 0 2 B 13
BRA 0 1 loop 14
LDA loop 3 A 2
SUB 0 1 N 3
BRP 0 e 4
LDA 0 3 A 5
OUT 0 4 6
LDA 0 2 B 7
ADD A 3 A 8
STA 0 0 ACC 9
LDA 0 2 B 10
STA 0 3 A 11
LDA 0 0 ACC 12
STA 0 2 B 13
BRA 0 1 loop 14
LDA loop 3 A 2
SUB 0 1 N 3
BRP 0 e 4
LDA 0 3 A 5
OUT 0 4 6
LDA 0 2 B 7
ADD A 3 A 8
STA 0 0 ACC 9
LDA 0 2 B 10
STA 0 3 A 11
LDA 0 0 ACC 12
STA 0 2 B 13
BRA 0 1 loop 14
LDA loop 3 A 2
SUB 0 1 N 3
BRP 0 e 4

```

I added a temporary window which allows me to see whether each line is executed properly by displaying the values of all variables involved with execution, I made this to accelerate the development of my project. I made finding bugs and logical errors much easier. This will not be in the final version, it's just there to help me (the developer) fix any issues. Each line had all data that was present during execution of each line so in case there was an error related to my program and not the actual code I was able to see what was abnormal and then I could trace the symptom of the problem to the location of code that is not functioning correctly.

# LMC assembly language interpreter optimization:

After creating my 2<sup>nd</sup> prototype I realized that I can make my program much more efficient in handling and executing code, I noticed that especially when I created a prime finding program using my interpreter, Finding first 20 primes didn't take a long time however finding first 100 took an incredibly long time (which indicated above linear scaling), This wasn't completely the fault of my interpreter but the actual efficiency of the algorithm I created to find primes. Because I still had some time left I was able to improve the efficiency of the interpreter so even less efficient LMC code won't run as slow as it is now (without optimizations). This motivated me to implement some optimizations which should greatly improve the efficiency of my interpreter.

LMC is a very basic programming language and is very weak compared to higher level languages like "c++" or "vb.net". It is mostly used for education due of its simplicity therefore it's hard to imagine someone using it for real world number crunching and also Its hard to make a very complex program in LMC, this means efficiency wasn't my main concern. This is why I left this issue as my lowest priority however my 2<sup>nd</sup> prototype has so many optimization opportunities that I felt like it would be an interesting topic to include in my write up and also it would greatly speed up some of my LMC programs that I have written.

## Theory behind my optimizations:

```

loop      INP
          STA N
          LDA A
          SUB N
          BRP end
          LDA A
          OUT
          LDA B
          ADD A
          STA ACC
          LDA B
          STA A
          LDA ACC
          STA B
          BRA loop
end        HLT
A          DAT 0
B          DAT 1
N          DAT 0
ACC DAT 0

```

This is a piece of code that generates a fib sequence up to a given value that the user inputs. as seen, the code features a main loop which repeats the code which runs from line 3 to line 15 until a condition is met where it exits the program using "BRZ" (branch if zero).

The code has to perform lots of data retrieval each time the loop repeats. Each time a variable is loaded into the accumulator the program has to search for the variable in an array which can have the worst time complexity of  $O(n)$  since my interpreter does a linear search. This means each time branching occurs and code is repeated, the program has to look for the variables all over again, for example "A" has to be located 4 times between line 3 and 15, meaning that it will take 4x4 array accesses each iteration. similarly "B" has to be accessed 3 times between line 3 and 15 meaning it will take 3x3 array accesses since B will always be found in position 3 in the "RAMarray()". in total there will be 25 array accesses per iteration for "A"

And "B" alone.

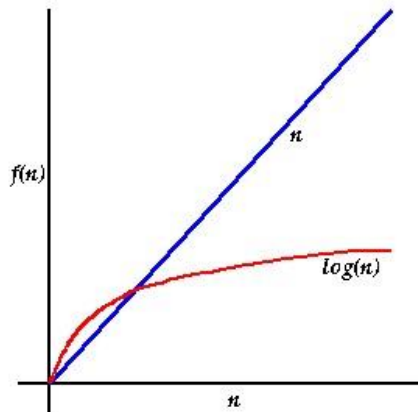
The same issue is with each single command that is on each line, my program doesn't know what command is located on each line which means each time it visits new line it has to perform lexical analysis to determine the command (this also has linear complexity).

Another problem with my prototype is that each branch is found using linear search as well which inherits the same issues as looking for variables in an array. Each time there is a branch command

the program has to search for the loop tag which can take total number of lines – number of lines dedicated for declaring variables, since branching cannot be done there.

My initial idea is to sort the declared variables in an array using merge sort or a quick sort and then perform a binary search each time the program wants to access a variable. This would improve the efficiency of my program and have the worst-case search complexity of  $O(\log n)$ , however initially sorting the data would be computationally costly but would become less and less significant if the program has to iterate many times.

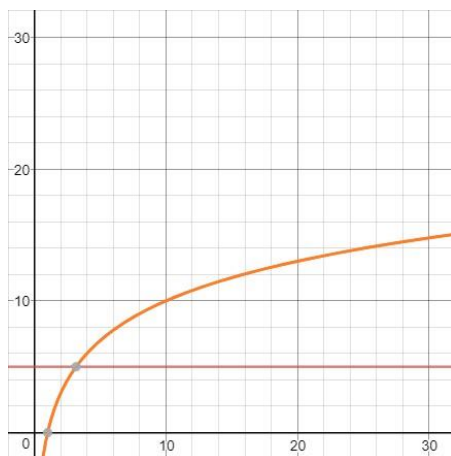
This graph shows  $O(\log n)$  vs  $O(n)$ :



As seen above, as data increases the  $\Delta$  in time complexities becomes large.

This however would not solve the problem related to searching loop tags since they cannot be put in order because of their nature. This is because a tag has to be in a specified order and place which is unique to a specific algorithm that is written. This means my program would still have to just perform a linear search each time to locate a tag.

This graph shows  $O(\log n)$  vs  $O(1)$ :



The biggest difference between  $O(1)$  and any other time complexity is that it is independent of the amount data that is present which means it has much better scaling than even  $O(\log)$  which has second best time complexity.

Hashing and which involves pointers leading directly to data have that trait of being  $O(1)$ .

One of the fastest way to retrieve data is to use hashing, hashing has the time complexity to  $O(1)$  which is the best time complexity for an algorithm on a classical computer.

There are many opportunities to make my program faster and I came up with a method that resolves all of those problems using the same strategy.

## Optimization techniques (explanation of my algorithm):

Since LMC code isn't dynamic, like most code it cannot be changed during run time, this means that line "n" will always feature the same command/data on it during runtime. This attribute is key since it means each line can be remembered once it has been read only once. This is known as: "static code analysis".

This implementation is similar to hashing because it generates a pointer that directly points to a memory location, however it doesn't perform an arithmetic operation each time it wants to find the variable which is what hashing would involve.

Firstly, my algorithm interprets all of the code as it would normally do however it does it before actually running it so it doesn't interpret during runtime (which was what my old prototype did).

Before, I had a loop array that stored all of the loop tags so the program can search for each time. I modified the array so it's a 2-d array which will contain all of the necessary information in each row and column.

Here's what the array features in each column (visualized with example data):

line position	command	Data location	Loop tag	Loop tag position
23	LDA	12	""	""
24	BRA	""	repeat	46

To generate this array my program has to do lexical analysis on each line and then locate the location of a variable (that the program picks up on each line) in RAM array which initially is still a linear search. On the same line my algorithm checks for any loop tags that are present always on the left side of a command. This allows to save the location of the loop tag later once the code is run. This means that finding the loop tag for the first time on a particular line will also require a linear search initially. However, once the location is found it's saved in the array so it can be found instantly next time.

This process is known as "*initialization*", it's when the code is set up for most efficient execution by the compiler before it's executed.

Once initialization is done the code can be run really quickly since the code will know all of the execution paths it requires to know on each line such as the command and the location of data that is being used. This means the code already has every necessary data to execute the data before it's executed.

Also, when speed is important the user selects "real time" mode which allows the code to run at full computer speed. Since my program has visualization such as highlighting there's performance degradation. When run at full speed the visualization is useless because it updates too quick to see anything. This means that it can be disabled when "real time" is selected to also increase the efficiency of the code so the program can complete the task as quickly as possible. I found that applying this technique also improved performance.



## Issues with this specific approach:

If a user creates a program which is very linear (doesn't feature any branching) then my implementation will have a negative impact on performance since it will have to perform each command only once so it will be not appropriate to perform this optimization technique.

Another limitation is when a specific little man computer code only uses a single variable (powers of 2 program):

```
top LDA A
    ADD A
    STA A
    OUT A
    BRA top
    HLT
A DAT 1
```

This is a very simple program I created to output powers of 2, this is an example of scenario when only one variable is used which means my optimization won't help since best case for a linear search is  $O(1)$ . The variable will always be found instantly since it's at the start of the array because it's the only item.

Another drawback is that more space is required compared to my previous solution since more data has to be stored in an array to perform execution. This is because as I mentioned before, my previous program didn't remember its lines as it executed them so it only had to store the code entered by the user.

## Investigating performance improvements after modifying the 2<sup>nd</sup> prototype:

```
loop INP
    STA N
    LDA A
    SUB N
    BRP end
    LDA A
    OUT
    LDA B
    ADD A
    STA ACC
    LDA B
    STA A
    LDA ACC
    STA B
    BRA loop
end   HLT
A DAT 0
B DAT 1
N DAT 0
ACC DAT 0
```

As mentioned above "A" and "B" would require 25 array accesses and if we combine that with "N" and "ACC" this will give a total amount of array accesses in the branch block. The total is  $25+2+2=29$  array accesses each loop, by applying my optimization the number would drop to 7 which is approximately 75% more efficient (this is excluding loop tags as well). The performance improvement would become more and more significant with longer code and the code that needs to iterate more times.



```

INP      ADD ONE
number   STA TARGET
        LDA CURRENT
        STA A
        LDA N
        STA A
        LDA N
        LDA N
        SUB DIVISOR
        LDA N
        BRZ not
        BRZ DIV
        LDA DIVISOR
        ADD ONE
        STA DIVISOR
        LDA N
        SUB N
        ADD A
        STA N
        LDA N
        SUB DIVISOR
        BRZ exit
        BRA loop
exit     LDA COUNT
        ADD ONE
        STA COUNT
        LDA CURRENT
        ADD ONE
        STA CURRENT
        LDA DIVISOR
        SUB DIVISOR
        ADD ONE
        STA DIVISOR
        LDA TARGET
        SUB CURRENT
        BRZ finished
        LDA N
        SUB N
        STA N
        BRA number
not      LDA CURRENT
        ADD ONE
        STA CURRENT
        LDA DIVISOR
        SUB DIVISOR
        ADD ONE
        STA DIVISOR
        LDA TARGET
        SUB CURRENT
        BRZ finished
        LDA N
        SUB N
        STA N
        BRA number
finished LDA COUNT
        ADD ONE
        OUT COUNT
        HLT
N        DAT 0
DIVISOR  DAT 2
A        DAT 0
TARGET  DAT 0
CURRENT  DAT 2
COUNT  DAT 0
ONE      DAT 1

```

Here is a slightly more complex code I created, it counts the number of primes up to a given value so if the user enters 100, and the output is 25.

This program features many branching and many different variables which allows the optimization to have a positive impact on the performance and widen the time  $\Delta$  between my old implementation.

When I run the old version of my program it took 11 seconds to find primes up to 100 and 31 seconds to find 200 primes. On the same machine I run my new program and it took 0.5 seconds to find 100 primes and 1.2 seconds to find 200 primes. The time reduction is very significant and very noticeable.

By applying those optimizations I'm enabling the user to create more interesting programs and have them run more quickly (compared to previous prototype).

To implement those algorithms into my project I had to redesign some part of my code:

This section of the code is responsible for initialising the interpreter so it's ready to execute code, it places all necessary pointers and data in to an array (algorithm is mentioned above).

```

Do Until linecount = (RichTextBox1.Lines.Length) 'this loop looks for any loops in the code
    Call lexicalAnalysis()
    If command = "" And Len(mainarray(linecount)) <> 0 And Mid(mainarray(linecount), 1, 1) <> "/" Then 'verfiys whether correct syntax been used.
        MsgBox("Error on line: " & linecount + 1) 'Arrays are zero base so +1 allows to show "one based" loacation of the code
        Call Reset()
        Exit Sub
    Else 'assign values to execution paths array on specific linecount value
        executionpaths(linecount, 0) = command
        executionpaths(linecount, 1) = VAR
        executionpaths(linecount, 4) = VARVALUE
    End If
    If command = "ADD" Or command = "SUB" Or command = "LDA" Or command = "STA" Or command = "OUT" Then
        Call RamHandler()
        If notfound = True Then
            Call Reset() 'procedure thet resets all variables to deafult state
            Exit Sub
        End If
        executionpaths(linecount, 2) = FOUNDlocation 'adds a direct address so the variable can be founfd in a array in O(1) time
    End If
    linecount = linecount + 1
    command = ""
Loop

```

This is a 2-d array which hold all necessary data for each line of code that the user has entered.

```

Dim executionpaths(100, 4) As String 'Array to store all necessary pointers once the code is compiled
Dim ramcounter As Integer 'used to determine the allocation size of the RAMarray (depending on the amount

```

To allow the new implementation to work I had to modify how loops function in my program. if a branch command is read by the interpreter for the first time, a linear search will be performed to find a loop tag. once it's found it will be saved to the "execution paths()" A array so next iteration the program will know straight away where to branch to. Here is an "if" block responsible for handling the "BRZ" command.

```

ElseIf command = "BRZ" Then 'branches if a accumulator is zero
  If accumulator = 0 Then
    If executionpaths(linecount, 3) = "" Then
      Do Until executionpaths(loopcounter, 1) = executionpaths(linecount, 4) 'finds the location of the tag
        loopcounter = loopcounter + 1
        If loopcounter = RichTextBox1.Lines.Length Then ' determines if the tag exists
          MsgBox("Could not find the tag: " & executionpaths(linecount, 4))
          Call Reset()
          Exit Sub
        End If
      Loop
      executionpaths(linecount, 3) = loopcounter - 1
      linecount = executionpaths(linecount, 3)
      loopcounter = 0
    Else
      linecount = executionpaths(linecount, 3)
    End If
  End If
End If

```

During the development of my second prototype I didn't have any interesting errors to report, mostly small syntax errors were made.

At the end I have added couple of sample programs which can be used to help the user get more familiarised with syntax and structure:

I have added:

- Multiplying program
- Fibonacci sequence program
- Powers of 2 program

## Walkthrough of my Fibonacci program and how my interpreter reads it and interprets each line:

My fib program is 20 lines long. On line 1 the op code "INP" is interpreted by my program and it launches an input box that allows to update the contents of the accumulator which is variable in my interpreter (hard coded within my program).

The op code "STA" is the read which tells my interpreter to store the entered value (value saved in the accumulator in location "N" which is initially found using a simple linear search but then its hashed for O(1) retrieval.

Line 3 is loading "A" from the array of variables into the accumulator that is just a variable in my interpreter called an "accumulator".

On line 4, "SUB N" is executed so the contents found in the location represented by the variable "N" are subtracted from the accumulator.

"BRP" is the branching op code that checks whether the contents of the accumulator satisfy the condition for branching. The condition for branching for "BRP" is when the accumulator is positive (this includes zero). Inside my interpreter this decision is carried out by "if" blocks.

```

Dim sb1 As New System.Text.StringBuilder
RichTextBox1.Clear()
sb1.AppendLine("    INP / This a fibonacci sequence generator")
sb1.AppendLine("    STA N")
sb1.AppendLine("loop LDA A")
sb1.AppendLine("    SUB N")
sb1.AppendLine("    BRP tag")
sb1.AppendLine("    LDA A")
sb1.AppendLine("    OUT")
sb1.AppendLine("    LDA B")
sb1.AppendLine("    ADD A")
sb1.AppendLine("    STA ACC")
sb1.AppendLine("    LDA B")
sb1.AppendLine("    STA A")
sb1.AppendLine("    LDA ACC")
sb1.AppendLine("    STA B")
sb1.AppendLine("    BRA loop")
sb1.AppendLine("tag HLT")
sb1.AppendLine("A    DAT ")
sb1.AppendLine("B    DAT 1")
sb1.AppendLine("N    DAT ")
sb1.AppendLine("ACC DAT 0")
RichTextBox1.AppendText(sb1.ToString)
End Sub

```

0 references

```

Private Sub MultiplicationToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles
Dim sb2 As New System.Text.StringBuilder
RichTextBox1.Clear()
sb2.AppendLine("    INP/will multiply two positive integers")
sb2.AppendLine("    STA NUMA")
sb2.AppendLine("    INP ")
sb2.AppendLine("    STA NUMB")
sb2.AppendLine("loop LDA TOTAL")
sb2.AppendLine("    ADD NUMA")
sb2.AppendLine("    STA TOTAL")
sb2.AppendLine("    LDA NUMB")
sb2.AppendLine("    SUB ONE")
sb2.AppendLine("    STA NUMB")
sb2.AppendLine("    BRP loop")
sb2.AppendLine("    LDA TOTAL")
sb2.AppendLine("    SUB NUMA")
sb2.AppendLine("    STA TOTAL")
sb2.AppendLine("    OUT")
sb2.AppendLine("    HLT")
sb2.AppendLine("NUMA DAT")
sb2.AppendLine("NUMB DAT")
sb2.AppendLine("ONE  DAT 1")
sb2.AppendLine("TOTAL DAT 0")
RichTextBox1.AppendText(sb2.ToString)
End Sub

```

0 references

```

Private Sub PrimeFinderexpertToolStripMenuItem_Click(sender As Object, e As EventArgs) Hand
Dim sb3 As New System.Text.StringBuilder
RichTextBox1.Clear()
sb3.AppendLine("    INP /enter a positive integer")
sb3.AppendLine("    STA B")
sb3.AppendLine("loop LDA A")
sb3.AppendLine("    OUT A")
sb3.AppendLine("    ADD A")
sb3.AppendLine("    STA A")
sb3.AppendLine("    LDA B")
sb3.AppendLine("    SUB A")
sb3.AppendLine("    BRP loop")
sb3.AppendLine("    COB")
sb3.AppendLine("A    DAT 1")
sb3.AppendLine("B    DAT")
RichTextBox1.AppendText(sb3.ToString)

```

This is some of the hard coded example code that the user can load up and run/modify.

## Testing to inform evaluation:

### Full system test (glass box test):

My program allows an infinite amount of possible inputs (a very large amount).it is impossible to test my program with every possible algorithm that could potentially be functional in my program. This means I have to test each execution path and if all execution paths function correctly then it will mean my interpreter should function as planned with any input, whether it's correct or not (if invalid then my program should behave accordingly to the error).

Test data:	Expected outcome:	Outcome:	Pass/fail:
Standard multiplication program that works with original LMC.	No errors (The output is the product of two numbers entered)	No errors (The output is the product of two numbers entered)	pass
A little man computer code that uses all available commands (prime finder)	Output of 25 when 100 is entered.	No errors, worked as intended, output: 25.	pass
No code entered.	Error message to enter code.	Error message to enter code.	pass
Inserted a multiplication Program, deleted one of the declarations.	Error message indicting that a variable is not found.	Error message indicting that a variable is not found.	pass
Inserted a multiplication Program, deleted a loop tag.	Error message that a loop tag cannot be found.	Error message that a loop tag cannot be found.	pass
Inserted a functioning program, misspelled on of the commands.	An error message indicating what line the error occurred on.	An error message indicating what line the error occurred on.	pass
Inserted a functioning program, commented few lines using the "/" character.	No errors.	No errors.	pass

Inserted a multiplication Program, entered two large inputs which should trigger the overflow mechanism. (99999x99999)	Overflow error message, no crash.	Overflow error message, no crash.	pass
Executed a program that required user input, inserted a string.	A message that tells the user that an invalid input has been entered.	A message that tells the user that an invalid input has been entered.	pass
Entered a piece of code that is longer than 100 lines.	A message that too many lines have been entered.	A message that too many lines have been entered.	pass
Declared same variable twice.	Program should choose the top Variable as its initial value.	Program should choose the top Variable as its initial value.	pass
Pressed the run button twice	Program should ignore that until stop is pressed.	No problems.	pass
Used the step button while code was running.	Program should stop on the line it was currently executing.	Program stopped on the line it was executing.	pass
A sample program is loaded after another sample program was loaded into the code space.	The program should erase the data and paste the sample program into the code space.	Worked as intended.	pass
Used same loop tag names for two loop tags.	Program should branch to the loop tag that's closest to the top.	Program selects the loop tag that's closest to the top.	pass
Pressed the clear button when there where outputs left behind from previous code.	Outputs should be erased.	Outputs erased.	pass
Stop pressed when a program is running.	Program should stop.	Program stops.	pass
Running a program with many variables and checking the visualisation part of it.	The ram and resisters should update appropriately.	Graphics work as intended.	pass

Setting program to slowest speed to check how highlighting of code works.	Each line executed should be highlighted.	Each line is highlighted when it should be.	pass
Used direct addresses in code instead of their corresponding variable names.	Program should be able to understand it.	Program worked as intended.	pass
Not using a COB/HLT command.	Program should give an error message that not HLT or COB was found.	Program gives an appropriate message.	pass

## Screenshots of the program running:

## User acceptance testing:

### TEST PLAN:

I will want to test the entire functionality of my program which will require my stake holder testing each part of my program. The success criteria for this test is as follows:

- Every function of the program has a very low difficulty of operation
- The program is intuitive from the go without any learning required beforehand.

## What needs to be tested?

- Usability of the “Step” button.
- Usability of the “clear”, “stop”, “run” buttons.
- Usability of all of the functions available in the top bar.
- Asses the difficulty of understanding the extended view
- Asses the effectiveness of the “help” section.

After developing the final program I checked my success criteria to make sure most/all of the criteria is met to finalise my prototype and turn it into a final product eventually. My second prototype was handed to Caleb to verify he was happy with it. The success criteria can be Broken down into 3 main parts:

- Usability
- Functionality
- Ergonomics/design

These parts are the main parts I'm interested to hear about from Caleb's feedback. I am going to tell Caleb what I want him to concentrate on to give me a feedback on those points, also if he has any good ideas he could add that information on top.

I have gave Caleb specific task to do in the program to see how easy it was for him to do it. After each task was completed I asked Caleb for a short comment about how usable each section of the program is. I also asked Caleb to quantify the difficulty of operating each section of the program from 1-10. 1 being the easiest and 10 being incredibly confusing.

Here is the usability test I have performed:

Task:	How easy was it(1-10):	Caleb's comment:
<ul style="list-style-type: none"> <li>Load the fib sequence program.</li> <li>Run the program.</li> <li>Input 900 into the input box.</li> </ul>	2	<i>"I found this task to be very simple, It was incredibly easy to find the sample program on the user form and perform further tasks necessary."</i>
<ul style="list-style-type: none"> <li>Load the multiplication program.</li> <li>Select real time execution speed.</li> <li>Input 12 and the 12 again</li> <li>Clear output.</li> </ul>	3	<i>"No real issues here either. The option bar at the top makes it very easy to select the necessary operation I want to perform."</i>
<p>Input this code into the "big man computer" :</p> <pre>       INP       STA N loop  LDA A       SUB N       BRP tag       LDA A       OUT       LD  B       ADD A       STA ACC       LDA B       STA A       LDA ACC       STA B       BRA loop tag   HLT A     DAT B     DAT 1 N     DAT </pre>	4	<i>"It was slightly harder because the lines aren't numbered so i had to count each line to see which line number gave me the error. Other than that it was simple to perform."</i>

<p>ACC DAT □</p> <ul style="list-style-type: none"> <li>• Run the program</li> <li>• Locate the error using the error message.</li> <li>• Go to help section to see how to fix the error.</li> <li>• Fix error and run code, input 12.</li> </ul>		
<ul style="list-style-type: none"> <li>• Select the “powers of 2” program.</li> <li>• Run the program.</li> <li>• Input 300</li> <li>• Click “step” and then stop until the accumulator has the value of 2048.</li> </ul>	3	<i>“Fairly simple to do, I didn’t understand what the accumulator is and where I can find it. I clicked on the help section and it told me that I can see more things by clicking “extended view”. This allowed me to solve this task.”</i>
<ul style="list-style-type: none"> <li>• Create your own program to count up in and number that the user inputs until 100 is reached.</li> </ul>	5	<i>“This task was the hardest but using this interpreter and its features has greatly helped me save time coding up this solution. Commenting has helped me remember what I was thinking.”</i>
<ul style="list-style-type: none"> <li>• Change form colour.</li> </ul>	1	<i>“Very simple since all options are found at the top in the task bar.”</i>

Looking at the test results, the program passed the tests and satisfied the success criteria for usability testing.



## Caleb's comments about the final product:

*From a usability standpoint the program was fantastic, the layout was exactly how I desired it to be and consulting with my developer during development really helped to refine the program so it's as good as I can imagine it to be. The buttons were laid out properly, the user form was simplistic and very simple to use. The coding space is large so I have plenty of space to indent and comment the code I wanted to try. Every button functions and it is very intuitive, it didn't require much time for me to get used to. I also really appreciate the "HELP" section that was implemented after my recommendation.*

*Another great part about the program was the part in which I was able to view how the registers and the ram works, it really helped me give an insight on how it works.*

## Reviewing the success criteria to confirm the success of my program:

- My program has to be effective in explaining how all the registers, Ram and ALU work together in a computer system, the effectiveness of that part of the program will be determined by white box testing.

The program was able to inform my stakeholder on how the von Neumann architecture functions and he reported that it really helped him gain more knowledge about this topic.

- My program has to be able to understand LMC code. It will have to understand any combination of code that is error free and has to be able to run on my program in the same way that original little man computer runs given the same piece of code.

Looking at the test results, my program was able to interpret LMC code and the only limitation was interpreting "DAT" during runtime this is a minor limitation that wasn't noticed by my stakeholder when he used it.

- My program has to have consistent results so every time a user inputs new code or they run the program with different inputs they will receive the correct output every time.

In my final program all programs that I could find that work in original little man computer have worked properly and my code was able to execute a variety of different algorithms successfully. Caleb had a go at creating a program which found prime numbers and his implementation was different compared to my implementation of primes, however both functioned which means the interpreter is consistent with different data.

- The solution has to be able to deal with incorrect inputs by the user which means it has to have appropriate validation.

Caleb didn't always make perfect code, he had many syntax errors at the beginning however the error messages informed him and guided him how to fix his code, he found that very useful and it really helped him figure out problems.

- My program has to be visually appealing (more modern) and intuitive to use.

Caleb really enjoyed the design and found the user from very intuitive. He also appreciated the ability to customise it .

- My program will allow the user to step through the program so the user has time to see what is happening.

Caleb was able to step through his code which helped him create some programs as well as follow what is happening inside each register (when extended view was open).

- My program will allow the user to run their programs at different speeds if they desire to run the code quickly.

This function worked appropriately which gave my stake holder the freedom to choose how fast the program runs.

- My program will feature sample codes so the user can run pre-made programs so they can get used with the syntax and the general structure of little man computer assembly language.

All sample codes worked as intended.

The success criteria has been fulfilled which indicates that this piece of software is mostly a success in terms of functionality and stability.

## Evaluation:

## What went well:

After the program had been refined so the problems that were present were fixed, the program ran as intended. The program could handle any input by the user and if the code entered was wrong syntactically then my solution is able to inform the user about a problem. It was able to handle every little man command including branching.

At the start when Caleb tried to create his own assembly program, Caleb did mistakes when making his program. My solution instructed him on how he can fix the problem. For example when Caleb forgot to input a "HLT" command my solution was able to return a message to inform him that there was an issue and the "HLT" command had to be inserted.

Caleb used the program and found it very useful and easy to use, he really found using the program very intuitive and easy. Caleb really liked the "sample programs" tab that allowed him to input a program that I have pre-installed in my solution. This was useful because before he had to go online to find code for little man computer which was time consuming for him. Caleb also found the extended view to be very useful so he had something visualized in front of him.

After we tested the program with Caleb, we decided it's ready to be used by the lower school students that Caleb will be teaching as his work experience. By using my solution as a tool for learning Caleb was able to successfully teach basics of little man computer, including the syntax of the language as well as a couple of basic commands that LMC contains.

After looking at my success criteria that I made with collaboration with my stakeholder, my final product seems to tick all of the goals I have set with Caleb. My program is able to interpret code, does not crash and is able to visualize the parts of the von Neumann architecture during run time of the code. I asked year 9 students what they thought about the program since it was their first time dealing with little man computer assembly language and my program and they said that it was easy to use and very forgiving in terms of syntax.

## What didn't go as well:

One of the students that had a go on my software said that he would like to see a little tutorial when the program is launched which I found very interesting and I think that this would be a good idea since it will allow a beginner to self-teach LMC without a kick-start of a teacher/helper. Without this the students had to ask for help to understand the main foundation before they could use the program easily.

Caleb has noticed that he wasn't able to run the program at normal speed after the step button has been pressed. He had to stop the program and re-run it. This wasn't a big flaw however if the program needs to execute many operations it's time consuming to stop and re-run the program all over again.

Caleb and others that used the program really wanted to have the ability to change the font and the background colour inside the code space. Some found it hard to read the code in this format and some liked the font/colour scheme. Adding the ability to customise this would make my solution more easily usable to more people.

## Future improvements based on ideas that I came up with during the late stage of development:

There is one combination of commands that my code is not able to cope with which is re declaring a variable during run time, this is not very commonly used in little man computer and can be done with the existing command set however it take sorter lines of code. The line of code looks like this: "Loop A DAT 1", this would cause an error to show up "loop tag not found" since it would read "loop A "as "Loop A" this is not a big problem but I didn't have time to address it and also I didn't know this feature is possible in little man computer before I started the development. What I would have to modify is the way lexical analysis would have to work.

My program could be further optimized by hashing every line of code in such a way so it doesn't have to be analysed each time my program looks at it. This would reduce the time it takes to execute each line, this is not a very important thing to do since modern computers are plenty fast and my program won't be running anything complex such as graph processing.

As I mentioned above, I could add a tutorial at the start of the program so people can familiarize themselves, also I would add a "do not show" later option so the tutorial doesn't have to be launched each time he user launches the program. This could be does easily by saving user data/preferences into a text file.

I also wasn't able to implement a save load feature since it I had many problems with it and ended up running out of time. This would have been a good feature to have and is definitely something I would improve on if I had time.it would allow the user to save their work for later instead of copy/pasting into a text file each time.

Another useful feature would be automatic colour coding which was suggested by my stake holder, it would allow reading the code even easier and it would help to show its structure more clearly. For example, visual studio IDE has colour coding which allows the code to be more legible since each function is colour coded which make it stand out better.

A very simple improvement that could have been implemented would be numbering each line of code so when an error occurs it would be very easy to locate the faulty line of code. Caleb has suggested that improvement during user acceptance testing. Alternatively, the faulty line could be highlighted in red to inform where the error occurred. In the future I could also integrate some code which will handle a DLL (dynamic link library) file which could be responsible for defining the design of the user form and just by changing the DLL file the look and feel of the program could be changed. This would increase it modularity which would also make customisation easier. A same principle can be applied to the functionality of the program, more commands could be added to the command list by integrating a dedicated DLL file, although LMC is a low-level language it can still have more low-level commands that are present in the x86 instruction set. For example: bit-wise shifting.

Another key application of a DLL file could be using it for extending the list of sample commands that my program offers, my program currently offers 3 sample program and that number could be increased in the future to allow the user to have a greater selection of different algorithms to choose from. A DLL file would eliminate "hard coding" sample program into my solution.

I could add a mode where even more behind the scenes is shown, users with already good knowledge of the von Neumann cycle could deepen their knowledge and understanding even more, I could do that by including more components of a CPU.

Another improvement I could add is make arrays dynamic so more data could be entered, rather than having a fixed size of the array (in my case 100 lines of code can be entered since I set the array to hold 100 pieces of data).

My program could also have colour coding implemented that is commonly seen in modern integrated development environments. Colour coding would allow to colour code the op code, operand and the tags so it would make it even easier to distinguish them one from another. This could have been implemented in my lexical analysis section of my program that recognises sub components of the code such as the instructions.

## Code listing:

Public Class Form1

Dim linecount As Integer ' the line position in the code

Dim RAMArray(100, 1) As String ' used to store variables the user declared and also their values

Dim mainarray(100) As String ' used to store the code so it's easier to handle

Dim command As String ' used to store the command that is currently being processed

Dim Bdecide As Boolean ' decides how RAMArray is used

Dim VAR As String 'string thats on the left side of command

Dim VARVALUE As String 'string thats on the right side of command

Dim FOUNDlocation As Integer 'allows to modify rams contents

Dim RAMlocation As Integer ' position value within the ram

Dim position As Integer 'used in searching the array

Dim accumulator As Integer ' stores the running value  
Dim temp As String ' used to add subtract a value from accumulator  
Dim loopcounter As Integer ' allows to search for a tag  
Dim PLACE As Integer ' position wuithin the text  
Dim HASBEEN As Boolean ' boolean that tells the logic of the program if a command has been detected  
Dim TEMPWORD As String ' temporary word used to build strings individual characters  
Dim TEMPVAR As String 'holds data of the left side of the command  
Dim wait1 As Integer ' delays the program by a certain amount  
Dim executionpaths(100, 4) As String 'Array to store all necessary pointers once the code is compiled  
Dim ramcounter As Integer ' used to determine the allocation size of the RAMarray (depending on the amount of varibales that the user has declared  
Dim notfound As Boolean ' boolaen that is passed so the program can verify the existence of a decalared variable  
Dim Num As Integer  
Dim beenpressed As Integer  
Dim start As Boolean  
Dim stopcode As Boolean  
Private Sub Button1\_Click(sender As Object, e As EventArgs) Handles Button1.Click 'Initializes the code  
    Call Initialazation()  
End Sub  
Sub Initialazation()  
    If start = False Then  
        Do Until linecount = (RichTextBox1.Lines.Length) 'a loop that reads everything entered in the rich text box  
            mainarray(linecount) = RichTextBox1.Lines(linecount).ToString ' inserts each line into the array  
        If linecount = 100 Then  
            MsgBox("code length exceeded")

```
Exit Sub
End If
linecount = linecount + 1
Loop
Me.Text = "Big man computer (Running code....)"
If linecount = 0 Then
    MsgBox("Enter some code")
    Call Reset()
    Exit Sub
End If
linecount = linecount - 1
Bdecide = True
Do Until command = "HLT" Or command = "COB" ' this loop will repeat
until it finds "HLT" command
    Call lexicalAnalysis()
    If linecount = 0 Then
        If command <> "HLT" Or command <> "COB" Then ' validation to
inform the user that they need to insert a "HLT" command
            MsgBox("No HLT or COB command found")
            Call Reset() 'precedure that resets all variables to deafult state
            Exit Sub
        End If
        ElseIf command = "DAT" Then ' if "HLT" inst reached then ramhandler
can populate the array with data
            Call RamHandler()
            If notfound = True Then
                Call Reset() 'precedure that resets all variables to deafult state
                Exit Sub
            End If
            VAR = ""
        End If
    End If
```

linecount = linecount - 1 'decrements the linecount so variables can be read from bottom to top

Loop

linecount = 0

Bdecide = False

command = ""

Do Until linecount = (RichTextBox1.Lines.Length) 'this loop looks for any loops in the code

Call lexicalAnalysis()

If command = "" And Len(mainarray(linecount)) <> 0 And Mid(mainarray(linecount), 1, 1) <> "/" Then 'verfiys whether correct syntax been used.

MsgBox("Error on line: " & linecount + 1) 'Arrays are zero base so +1 allows to show "one based" loacation of the code

Call Reset()

Exit Sub

Else 'assign values to execution paths array on specific linecount value

executionpaths(linecount, 0) = command

executionpaths(linecount, 1) = VAR

executionpaths(linecount, 4) = VARVALUE

End If

If command = "ADD" Or command = "SUB" Or command = "LDA" Or command = "STA" Or command = "OUT" Then

Call RamHandler()

If notfound = True Then

Call Reset() 'procedure thet resets all variables to deafult state

Exit Sub

End If

executionpaths(linecount, 2) = FOUNDlocation 'adds a direct adress so the variable can be founfd in a array in O(1) time

End If

linecount = linecount + 1



```

        command = ""
    Loop
    command = ""
    linecount = 0
    start = True 'validation to prevent the program crashing from start button
being pressed twice
    Call Interpreter()
End If
End Sub
Sub lexicalAnalysis() 'used analyse the string on each line,pick up any
commands and integers.
    PLACE = 1
    If mainarray(linecount) <> "" Then 'this statment cchecks if the line is blank
        Do Until PLACE = Len(mainarray(linecount)) + 1 ' this loop will iterate until
variable "PLACE" equal to the length of the array
            TEMPWORD = Mid(mainarray(linecount), PLACE, 3) 'this variable holds
3 letters of the
            If Asc(Mid(mainarray(linecount), PLACE, 1)) <> 32 And TEMPWORD <>
"DAT" And TEMPWORD <> "LDA" And TEMPWORD <> "STA" And TEMPWORD
<> "SUB" And TEMPWORD <> "INP" And TEMPWORD <> "SUB" And TEMPWORD
<> "BRP" And TEMPWORD <> "BRZ" And TEMPWORD <> "OUT" And
TEMPWORD <> "HLT" And TEMPWORD <> "BRA" And TEMPWORD <> "COB"
Then
                If Mid(mainarray(linecount), PLACE, 1) = "/" Then ' if this symbbol is
present then the do loop stops since this symbol allows th user to comment
                    Exit Do
                Else
                    TEMPVAR = TEMPVAR & Mid(mainarray(linecount), PLACE, 1)
'builds the string that is on th right side of a LMC command
                End If
            End If
        End If
    End If

```

```
If TEMPWORD = "ADD" Or TEMPWORD = "SUB" Or TEMPWORD =  
"BRZ" Or TEMPWORD = "STA" Or TEMPWORD = "BRP" Or TEMPWORD = "OUT"  
Or TEMPWORD = "INP" Or TEMPWORD = "LDA" Or TEMPWORD = "DAT" Or  
TEMPWORD = "HLT" Or TEMPWORD = "BRA" Or TEMPWORD = "COB" Then  
    VAR = TEMPVAR 'if the command is found then left side of the  
command is saved so it can be used to represent a loop tag or a variable  
    If VAR = "" Then ' this simple validation prevents type mismatches  
        VAR = 0  
    End If  
    TEMPVAR = ""  
    PLACE = PLACE + 2 ' place is incremented by 2 because the command  
is a 3 letter string therefore by moving 2 places will allow to shift by one place  
    HASBEEN = True ' this boolean tells the program if the command was  
found so the right side of the command can be found  
    command = TEMPWORD ' this passes the contents of TEMPword into  
command variable so the other procedure knows which command was read  
    End If  
    If HASBEEN = True Then  
        VARVALUE = TEMPVAR 'this stores the right side of the command  
    End If  
    PLACE = PLACE + 1 ' position in the string is increased by 1 each  
iteration  
    Loop  
ElseIf mainarray(linecount) = "" Then  
    VAR = ""  
    VARVALUE = ""  
    command = ""  
End If  
HASBEEN = False 'some variables are set to default so the procedure is ready  
to be fed with new data  
TEMPWORD = ""  
TEMPVAR = ""
```

```
End Sub
Sub RamHandler() ' handles the RAMArray,used to update the ram's contents
    If Bdecide = False Then 'this part is used for searching for data in the ram
        Do Until VARVALUE = RAMArray(RAMlocation - position, 0) 'this part
compares the thing its looking for to the rams content in that loacation
            position = position + 1
        If position > RAMlocation Then 'validation so the array doesnt get
searched in -1 position which is invalid
            MsgBox("Could not locate variable at line : " & linecount + 1)
            notfound = True
            Exit Sub
        End If
    Loop
    FOUNDlocation = (RAMlocation - position)
    position = 0
    ElseIf Bdecide = True Then ' this part is used for inserting new data into the
ram
        If IsNumeric(VARVALUE) Then
            RAMArray(RAMlocation, 1) = VARVALUE 'inserts the variable
        Else
            RAMArray(RAMlocation, 1) = 0 'validation for assigning variables
        End If
        If RAMlocation = 100 Then 'limited ram space to 100 slots
            MsgBox("RAM capacity exceeded!!!")
            Call Reset()
            notfound = True 'this boolean can be also used to exit the program if
ram capacity is ecxceeded
        Else
            RAMArray(RAMlocation, 0) = VAR ' inserts the variables value
            RAMlocation = RAMlocation + 1
        End If
    End If
End If
```

End Sub

Sub Interpreter() ' the part which performs a specific operation given the command.

Dim temp As String ' used to add subtract a value from accumulator

Dim loopcounter As Integer ' allows to search for a tag

Dim accumulator As Integer ' stores the running value

Do While stopcode = False ' runs loop while the user has not pressed the stop button is found

command = executionpaths(linecount, 0)

If wait1 <> 0 Then

TextBox1.Text = accumulator

Call graphics()

End If

If command = "ADD" Then 'adds a value to accumulator

If IsNumeric(executionpaths(linecount, 4)) = True Then ' checks if a variable is added or a integer

temp = RAMarray(executionpaths(linecount, 1), 1)

Else

temp = RAMarray(executionpaths(linecount, 2), 1)

End If

If accumulator > 99999999 Or temp > 99999999 Then

MsgBox("Overflow Error")

Call Reset()

Exit Sub

End If

accumulator = accumulator + temp ' adds the value to the accumulator

Elseif command = "SUB" Then 'subtracts a value from the accumulator

If IsNumeric(executionpaths(linecount, 4)) = True Then ' checks if a variable is added or a integer

temp = RAMarray(executionpaths(linecount, 1), 1) ' allows to value to be subtracted from accumulator

Else

```
temp = RAMArray(executionpaths(linecount, 2), 1)
End If
If accumulator < -99999999 Or temp > 99999999 Then
    MsgBox("Overflow Error")
    Call Reset()
    Exit Sub
End If
accumulator = accumulator - temp ' subtracts the value to the
accumulator
Elseif command = "LDA" Then 'loads a value into the accumulator
    accumulator = RAMArray(executionpaths(linecount, 2), 1) ' sets
accumulator to the value that was loaded
Elseif command = "INP" Then ' allows the user to input a valaue
    temp = InputBox("INPUT")
    If IsNumeric(temp) = False Or Len(temp) > 8 Then ' validation to let only
integers to be entered
        MsgBox("Invalid input,number is too big or type error")
        Call Reset()
        Exit Sub
    Else
        accumulator = temp 'accumulator set to the input
    End If
Elseif command = "BRZ" Then 'branches if a accumulator is zero
    If accumulator = 0 Then
        If executionpaths(linecount, 3) = "" Then
            Do Until executionpaths(loopcounter, 1) =
executionpaths(linecount, 4) 'finds the location of the tag
                loopcounter = loopcounter + 1
            If loopcounter = RichTextBox1.Lines.Length Then ' determines if
the tag exists
                MsgBox("Could not find the tag: " & executionpaths(linecount,
4))
```

```
        Call Reset()
        Exit Sub
    End If
Loop
    executionpaths(linecount, 3) = loopcounter - 1
    linecount = executionpaths(linecount, 3)
    loopcounter = 0
Else
    linecount = executionpaths(linecount, 3)
End If
End If
Elseif command = "BRP" Then ' branches if accumulator is positive
    If accumulator >= 0 Then
        If executionpaths(linecount, 3) = "" Then
            Do Until executionpaths(loopcounter, 1) =
executionpaths(linecount, 4) 'finds the location of the tag
                loopcounter = loopcounter + 1
                If loopcounter = RichTextBox1.Lines.Length Then ' determines if
the tag exists
                    MsgBox("Could not find the tag: " & executionpaths(linecount,
4))
                End If
            Loop
            executionpaths(linecount, 3) = loopcounter - 1
            linecount = executionpaths(linecount, 3)
            loopcounter = 0
        Else
            linecount = executionpaths(linecount, 3)
        End If
    End If
End If
```

```
Elseif command = "BRA" Then ' branches unconditionally
    If executionpaths(linecount, 3) = "" Then
        Do Until executionpaths(loopcounter, 1) = executionpaths(linecount,
4) 'finds the location of the tag
            loopcounter = loopcounter + 1
            If loopcounter = RichTextBox1.Lines.Length Then ' determines if
the tag exists
                MsgBox("Could not find the tag: " & executionpaths(linecount,
4))

                Call Reset()
                Exit Sub
            End If
        Loop
        executionpaths(linecount, 3) = loopcounter - 1
        linecount = executionpaths(linecount, 3)
        loopcounter = 0
    Else
        linecount = executionpaths(linecount, 3)
    End If
Elseif command = "STA" Then 'stores the value of an accumulator into a
variabe
    RAMarray(executionpaths(linecount, 2), 1) = accumulator
Elseif command = "OUT" Then 'outputs the contents of the accumulator
    If IsNumeric(executionpaths(linecount, 4)) = True Then
        ListBox1.Items.Add(RAMarray(executionpaths(linecount, 4), 1))
    Elseif executionpaths(linecount, 4) = "" Then
        ListBox1.Items.Add(accumulator)
    Else
        temp = RAMarray(executionpaths(linecount, 2), 1)
        ListBox1.Items.Add(temp)
    End If
Elseif command = "HLT" Or command = "COB" Then
```

```
        ListBox1.Items.Add("Execution complete.")
        Call Reset()
        Exit Sub
    End If
    If beenpressed = True Then 'this if block allows the program to be
stepped through
        Do While stopcode = False
            If Num Mod 2 = 0 Then
                Num = Num + 1
                Exit Do
            End If
            wait(8)
        Loop
    End If
    linecount = linecount + 1
    Loop
    Call Reset()
    Exit Sub
End Sub
Sub graphics()
    RichTextBox2.Clear()
    Do Until ramcounter = RAMllocation
        RichTextBox2.Text = RichTextBox2.Text & (RAMarray(ramcounter, 0) & "
" & RAMarray(ramcounter, 1)) & vbNewLine
        ramcounter = ramcounter + 1
    Loop
    ramcounter = 0
    If beenpressed = True Then
        End If
        wait(wait1)
        RichTextBox1.Select(RichTextBox1.GetFirstCharIndexFromLine(linecount),
Len(mainarray(linecount)))
```



```
RichTextBox1.SelectionBackColor = Color.Blue
wait(wait1)
RichTextBox1.Select(RichTextBox1.GetFirstCharIndexFromLine(linecount),
Len(mainarray(linecount)))
RichTextBox1.SelectionBackColor = Color.Black
wait(wait1)

RichTextBox2.Select(RichTextBox2.GetFirstCharIndexFromLine(executionpaths(li
necount, 2)), Len(RAMarray(executionpaths(linecount, 2), 0)))
RichTextBox2.SelectionBackColor = Color.Blue
wait(wait1)

RichTextBox2.Select(RichTextBox2.GetFirstCharIndexFromLine(executionpaths(li
necount, 2)), Len(RAMarray(executionpaths(linecount, 2), 0)))
RichTextBox2.SelectionBackColor = Color.Black
TextBox5.Text = command
TextBox3.Text = executionpaths(linecount, 2) + 1
TextBox4.Text = RAMarray(executionpaths(linecount, 2), 1)
TextBox2.Text = linecount + 1
End Sub
Sub Reset()
PLACE = 0
HASBEEN = False
TEMPWORD = ""
TEMPVAR = ""
linecount = 0
command = ""
VAR = ""
VARVALUE = 0
FOUNDlocation = 0
RAMlocation = 0
position = 0
```

```
PLACE = 0
HASBEEN = False
TEMPWORD = ""
TEMPVAR = ""
accumulator = 0
temp = 0
loopcounter = 0
notfound = False
Array.Clear(mainarray, 0, mainarray.Length)
Array.Clear(RAMarray, 0, RAMarray.Length)
Array.Clear(executionpaths, 0, executionpaths.Length)
TextBox1.Text = ""
TextBox2.Text = ""
TextBox3.Text = ""
TextBox4.Text = ""
TextBox5.Text = ""
RichTextBox2.Clear()
wait1 = 30
Me.Text = "Big Man Computer"
Num = 0
beenpressed = False
start = False
stopcode = False
End Sub
```

```
Private Sub wait(ByVal interval As Integer)
    Dim sw As New Stopwatch ' allows to delay the program
    sw.Start()
    Do While sw.ElapsedMilliseconds < interval
        ' Allows UI to remain responsive
        Application.DoEvents()
    Loop
```

```
        sw.Stop()
    End Sub

    Private Sub LineSecondToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles LineSecondToolStripMenuItem.Click
        wait1 = 250
    End Sub

    Private Sub LinessecondToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles LinessecondToolStripMenuItem.Click
        wait1 = 50
    End Sub

    Private Sub RealTimeToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles RealTimeToolStripMenuItem.Click
        wait1 = 0
    End Sub

    Private Sub Button5_Click(sender As Object, e As EventArgs) Handles Button5.Click
        ListBox1.Items.Clear()
    End Sub

    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
        stopcode = True
        RichTextBox1.SelectionBackColor = Color.Black
    End Sub

    Private Sub BLUEToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles BLUEToolStripMenuItem.Click
        Me.BackColor = Color.LightBlue
    End Sub

    Private Sub REDToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles REDToolStripMenuItem.Click
        Me.BackColor = Color.Red
    End Sub
```

```
Private Sub GREENToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles GREENToolStripMenuItem.Click
```

```
    Me.BackColor = Color.Green
```

```
End Sub
```

```
Private Sub DEAFULTToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles DEAFULTToolStripMenuItem.Click
```

```
    Me.BackColor = Color.DarkGray
```

```
End Sub
```

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
```

```
    Me.Width = 479
```

```
    wait1 = 50
```

```
End Sub
```

```
Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
```

```
    If start = True Then
```

```
        beenpressed = True
```

```
        Num = Num + 1
```

```
    Else
```

```
        beenpressed = True
```

```
        Num = Num + 1
```

```
        Call Initialazation()
```

```
    End If
```

```
    wait1 = 250
```

```
End Sub
```

```
Private Sub ExtendedToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles ExtendedToolStripMenuItem.Click
```

```
    Me.Width = 796
```

```
End Sub
```

```
Private Sub DeafultToolStripMenuItem1_Click(sender As Object, e As EventArgs) Handles DeafultToolStripMenuItem1.Click
```

```
    Me.Width = 479
```

End Sub

Private Sub ToolStripButton1\_Click(sender As Object, e As EventArgs) Handles ToolStripButton1.Click

Form2.Show()

End Sub

Private Sub FibSequenceToolStripMenuItem\_Click(sender As Object, e As EventArgs) Handles FibSequenceToolStripMenuItem.Click

Dim sb1 As New System.Text.StringBuilder

RichTextBox1.Clear()

sb1.AppendLine(" INP / This a fibonacci sequence generator")

sb1.AppendLine(" STA N")

sb1.AppendLine("loop LDA A")

sb1.AppendLine(" SUB N")

sb1.AppendLine(" BRP tag")

sb1.AppendLine(" LDA A")

sb1.AppendLine(" OUT")

sb1.AppendLine(" LDA B")

sb1.AppendLine(" ADD A")

sb1.AppendLine(" STA ACC")

sb1.AppendLine(" LDA B")

sb1.AppendLine(" STA A")

sb1.AppendLine(" LDA ACC")

sb1.AppendLine(" STA B")

sb1.AppendLine(" BRA loop")

sb1.AppendLine(" tag HLT")

sb1.AppendLine("A DAT ")

sb1.AppendLine("B DAT 1")

sb1.AppendLine("N DAT ")

sb1.AppendLine("ACC DAT 0")

RichTextBox1.AppendText(sb1.ToString)

End Sub

Private Sub MultiplicationToolStripMenuItem\_Click(sender As Object, e As EventArgs) Handles MultiplicationToolStripMenuItem.Click

```
Dim sb2 As New System.Text.StringBuilder
RichTextBox1.Clear()
sb2.AppendLine("    INP/will multiply two positive integers")
sb2.AppendLine("    STA NUMA")
sb2.AppendLine("    INP ")
sb2.AppendLine("    STA NUMB")
sb2.AppendLine("loop  LDA TOTAL")
sb2.AppendLine("    ADD NUMA")
sb2.AppendLine("    STA TOTAL")
sb2.AppendLine("    LDA NUMB")
sb2.AppendLine("    SUB ONE")
sb2.AppendLine("    STA NUMB")
sb2.AppendLine("    BRP loop")
sb2.AppendLine("    LDA TOTAL")
sb2.AppendLine("    SUB NUMA")
sb2.AppendLine("    STA TOTAL")
sb2.AppendLine("    OUT")
sb2.AppendLine("    HLT")
sb2.AppendLine("NUMA  DAT")
sb2.AppendLine("NUMB  DAT")
sb2.AppendLine("ONE   DAT 1")
sb2.AppendLine("TOTAL DAT 0")
RichTextBox1.AppendText(sb2.ToString)
```

End Sub

Private Sub PrimeFinderexpertToolStripMenuItem\_Click(sender As Object, e As EventArgs) Handles PrimeFinderexpertToolStripMenuItem.Click

```
Dim sb3 As New System.Text.StringBuilder
RichTextBox1.Clear()
sb3.AppendLine("    INP /enter a positive integer")
sb3.AppendLine("    STA B")
```

```

sb3.AppendLine("loop LDA A")
sb3.AppendLine("  OUT A")
sb3.AppendLine("  ADD A")
sb3.AppendLine("  STA A")
sb3.AppendLine("  LDA B")
sb3.AppendLine("  SUB A")
sb3.AppendLine("  BRP loop")
sb3.AppendLine("  COB")
sb3.AppendLine("A  DAT 1")
sb3.AppendLine("B  DAT")
RichTextBox1.AppendText(sb3.ToString)

End Sub
End Class

```

### Bibliography:

Source name:	Source URL:	Description:	Last updated	Time accessed
Gcse computing	<a href="https://gcsecomputing.org.uk/theory/lmcinstructionset/">https://gcsecomputing.org.uk/theory/lmcinstructionset/</a>		Na	October 23rd 2017
vivaxsolutions	<a href="http://www.vivaxsolutions.com/web/lmc.aspx">http://www.vivaxsolutions.com/web/lmc.aspx</a>	This website was used for understanding LMC code and all of its commands, it made me understand which command erases the value Of the accumulator which is important for my solution to be compatible with LMC code.	Na	October 30th 2017

find icons	<a href="https://findicons.com/search/ico/2">https://findicons.com/search/ico/2</a>	This website was used to find an icon for my program.	Na	November r 12th 2017
source forge	<a href="https://sourceforge.net/projects/johnnysimulator/">https://sourceforge.net/projects/johnnysimulator/</a>	I Used this website To look at Some implementations of von Neumann architecture.	Na	October 23rd 2017
Wikipedia	<a href="https://en.wikipedia.org/wiki/Compiler">https://en.wikipedia.org/wiki/Compiler</a>	Used to help me understand what happens when a code is read by an interpreter/compiler	Na	December r 1st 2017
Stack overflow	<a href="https://stackoverflow.com/questions/17707795/passingvariables-between-windows-forms-in-vs-2010">https://stackoverflow.com/questions/17707795/passingvariables-between-windows-forms-in-vs-2010</a>	Learned how to pass values to a form.	Na	December r 12th 2017
Wikipedia	<a href="https://en.wikipedia.org/wiki/Halting_problem">https://en.wikipedia.org/wiki/Halting_problem</a>	Learned that one of my problem I came across is insolvable.		December r 16th 2017