|  |
| --- |
|  |
| A452 LMC CA |
| By: Manav Parikh |

# Contents Page

Table of Contents (Use CTRL+Click to go to a page)

[Contents Page 1](#_Toc414008657)

[*Hardware*: 3](#_Toc414008658)

[Software: 4](#_Toc414008659)

[Task 1a 4](#_Toc414008660)

[Introduction to task 4](#_Toc414008661)

[Research of the task 4](#_Toc414008662)

[Actual task 5](#_Toc414008663)

[Task 1b 6](#_Toc414008664)

[Introduction to task 6](#_Toc414008665)

[Research of the task 6](#_Toc414008666)

[Actual task 6](#_Toc414008667)

[Version 1 6](#_Toc414008668)

[Version 2 7](#_Toc414008669)

[Task 1c 7](#_Toc414008670)

[Introduction to task 7](#_Toc414008671)

[Research of the task 7](#_Toc414008672)

[Actual task 8](#_Toc414008673)

[Version 1 8](#_Toc414008674)

[Version 2 8](#_Toc414008675)

[Task 1d 8](#_Toc414008676)

[Introduction to task 8](#_Toc414008677)

[Research of the task 8](#_Toc414008678)

[Actual task 9](#_Toc414008679)

[Version 1 9](#_Toc414008680)

[Version 2 9](#_Toc414008681)

[Version 3 11](#_Toc414008682)

[Code 12](#_Toc414008683)

[Dry Run Table 12](#_Toc414008684)

[Task 1e 22](#_Toc414008685)

[Introduction to task 22](#_Toc414008686)

[Research of the task 22](#_Toc414008687)

[Actual task 22](#_Toc414008688)

[Task 1f 24](#_Toc414008689)

[Introduction to task 24](#_Toc414008690)

[Research of the task 24](#_Toc414008691)

[Actual task – Version 1 24](#_Toc414008692)

[Version 2 25](#_Toc414008693)

[Task 1g 26](#_Toc414008694)

[Introduction to task 26](#_Toc414008695)

[Test Plan V1 26](#_Toc414008696)

[Test plan V2 27](#_Toc414008697)

[Task 2a 29](#_Toc414008698)

[Introduction to task 29](#_Toc414008699)

[Research of the task 29](#_Toc414008700)

[Actual task 29](#_Toc414008701)

[Task 2b 29](#_Toc414008702)

[Introduction to task 29](#_Toc414008703)

[Actual task 29](#_Toc414008704)

[Pseudo code: 29](#_Toc414008705)

[Flowchart: 30](#_Toc414008706)

[Task 2c 30](#_Toc414008707)

[Introduction to task 30](#_Toc414008708)

[Actual task 30](#_Toc414008709)

[Version 1 30](#_Toc414008710)

[Version 2 32](#_Toc414008711)

[Task 3a 34](#_Toc414008712)

[Introduction to task 34](#_Toc414008713)

[Research of the Task 34](#_Toc414008714)

[Actual task 35](#_Toc414008715)

[Task 3b 36](#_Toc414008716)

[Introduction to task 36](#_Toc414008717)

[Actual task 36](#_Toc414008718)

[Task 3c 37](#_Toc414008719)

[Introduction to task 37](#_Toc414008720)

[Actual Task 37](#_Toc414008721)

[LMC code: 37](#_Toc414008722)

[Version 1 37](#_Toc414008723)

[Version 2 38](#_Toc414008724)

[Task 4 40](#_Toc414008725)

[Introduction to task 40](#_Toc414008726)

[Actual task 40](#_Toc414008727)

[Task 1a evaluation 40](#_Toc414008728)

[Task 1b evaluation 40](#_Toc414008729)

[Task 1c evaluation 40](#_Toc414008730)

[Task 1d evaluation 41](#_Toc414008731)

[Task 1e evaluation 41](#_Toc414008732)

[Task 1f evaluation 41](#_Toc414008733)

[Task 1g evaluation 42](#_Toc414008734)

[Task 1 overall evaluation 42](#_Toc414008735)

[Task 2a evaluation 42](#_Toc414008736)

[Task 2b evaluation 42](#_Toc414008737)

[Task 2c evaluation 43](#_Toc414008738)

[Task 2 overall evaluation 43](#_Toc414008739)

[Task 3a evaluation 43](#_Toc414008740)

[Task 3b evaluation 43](#_Toc414008741)

[Task 3c evaluation 43](#_Toc414008742)

[Task 3 overall evaluation 44](#_Toc414008743)

[Task 5 45](#_Toc414008744)

[Introduction to task 45](#_Toc414008745)

[Research of the task 45](#_Toc414008746)

[Actual task 46](#_Toc414008747)

# *Hardware*:

* Genius XScroll Mouse
* Genius K639 Keyboard
* CTX Monitor
* Intel Core 2 Duo vPro CPU
* 2GB of RAM

# Software:

* Microsoft Word 2010 – A word Processor which is part of Microsoft Office.
* Python 3.3.2 – A programming language which was required by some tasks.
* Little Man Computer – A program which was needed to do some of the tasks.
* Google Chrome – A very good web browser.
* VBE – Was required to run LMC.
* [Hosted software] Gliffy – Used to create flowcharts.
* Java Programming (JDK)
* Notepad – Used to convert the code in task 3 into a HTML file. It was also needed to edit the code.
* Windows 7 Professional 32-bit Service pack 1 – The OS I used.
* Camstudio – Used to record videos.
* Adobe Reader XI – used to view the .pdf file which contained the tasks.
* Snipping tool – Used to take screenshots of a specify area of the screen.

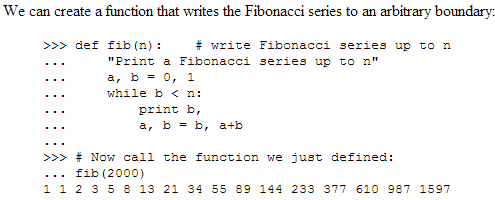
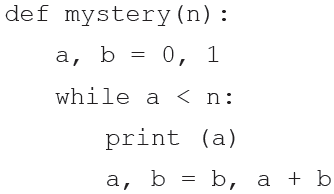
# Task 1a

## Introduction to task

This task requires me to explain the purpose of the code and it requires me to analyse and explain the code line-by-line.

## Research of the task

While I knew what the ‘*while*’ and ‘*print*’ commands do (by learning python on websites such as [codeacademy.org](http://www.codeacademy.org)), I was unsure of what the ‘*def*’ function does. So I searched ‘*python def*’ (on the Google search engine) and clicked on the first result which led me to this website - https://docs.python.org/release/1.5.1p1/tut/functions.html. I searched ‘python def’ instead of just ‘def’ to increase the relevancy of my results. I knew that the website was reliable because it was the official python website and it used HTTPS which meant that it was much more secure than most websites which don’t use SSL or TLS encryption. According to the webpage, the ‘*def*’ function allows users to create their own functions and then use them. E.g. you could create a function (like quadruple(n)) which would quadruple a number and output it, then you could type quadruple(4) and it would then quadruple 4 (4 \* 4) and output 16. The ‘def’ function defines a function. This allows for the creation of new, useful functions.

The website also yielded some unexpected, but very useful, information.  

The code on the website was extremely similar to the code in the controlled assessment! I looked at the code and I understood its purpose – **it writes out the Fibonacci sequence until it gets to the number of your choosing**. E.g. If you did ‘mystery(1000)’ the computer will *output* the sequence up to 987.

## Actual task

Now I will explain the code,

***‘def mystery (n):’***

This line begins the creation of the function called ‘mystery’. The function also accepts an argument – ‘n’. The colon at the end allows additional (indented) code to be entered for the function.

***‘a, b = 0, 1’***

This creates two variables, ‘*a’* and ‘*b’*. It assigns them with the values 0 and 1 respectively. This begins the Fibonacci sequence at 0.

***‘while a < n:’***

This creates a while loop which keeps executing the code within it as long as ‘*a’* is less than ‘*n’*. This means that the code will stop when ‘*a*’ is greater than ‘*n*’. The colon at the end allows additional (indented) code to be entered for the loop. This loop makes sure that the Fibonacci sequence only runs until it gets to the number in the argument - n.

***‘print a’***

This outputs the variable ‘*a*’ so that the user can see it.

***‘a, b = b, a + b’***

This line is main part of the code. This changes the value of the variable ‘*a*’ to the previous value of b. Also, it adds the previous values of ‘a’ and ‘b’ and it sets that as the value of b. This ends up creating a sequence where a number is equal to the value of the previous two numbers added together – the Fibonacci sequence.

# Task 1b

## Introduction to task

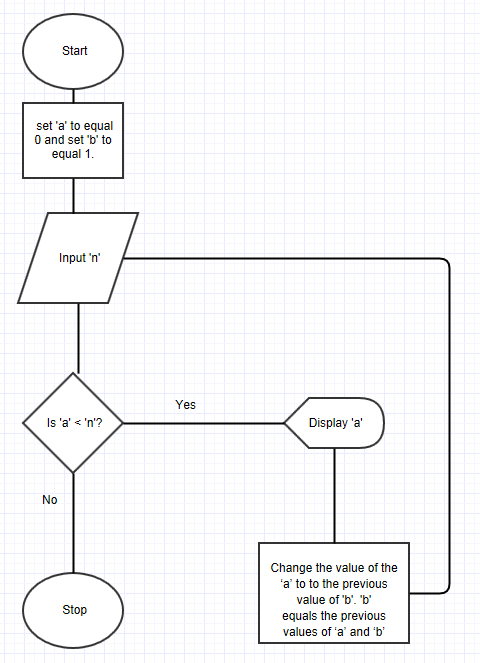
This task requires me to create a flowchart depicting the code in task 1.

## Research of the task

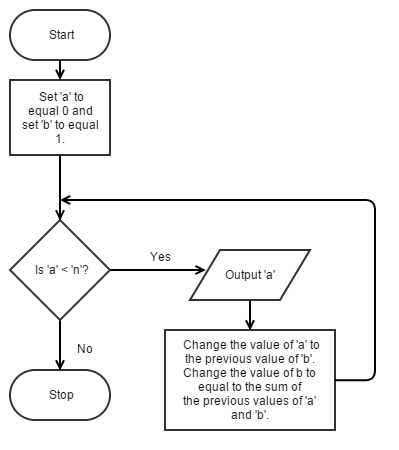
This task required me to make flowcharts, so I found a website ([gliffy.com](http://www.gliffy.com)) which makes creating flowcharts easy and efficient. I also looked up (using Google’s search engine) the various flowchart symbols to find out what they meant. I found an image (<http://www.conceptdraw.com/How-To-Guide/picture/Design-elements-Flowcharts.png>) which explained all of the symbols to me.

## Actual task

### Version 1



### Version 2



This version of the flowchart more accurately represents the python function. It fixes some of the problems of the previous version: there is an output box instead of the incorrect display box; there are arrows to guide users on how to read the flowchart.

# Task 1c

## Introduction to task

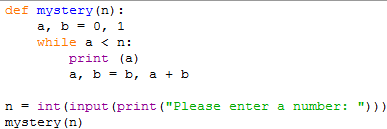
This task requires me to add some code to the existing code to make it run and be useful.

## Research of the task

I knew that for the code to be useful, it needed an input. I can add an input by simply using ‘*n = input()*’. However, I cannot simply leave it at that. There are 2 problems with this. Firstly, this does not give the end user a prompt to enter a number. Someone using this program might not know that they need to type a number in. I can inform the user by *outputting* a string of text saying “Please enter a number:”. Secondly, it allows the user to enter anything, so I need to use ‘*n = int(input())*’ to prevent the user from entering anything else.

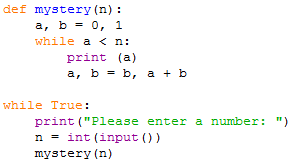
## Actual task

### Version 1



My first version of the program had some issues –it only worked once and a ‘None’ error kept appearing.

### Version 2



I fixed that in my second version by separating the *print* and the *input*. I also added a permanent *while* loop to make it run indefinitely. Without the while loop, it would only function correctly once and it would give error messages if the user inputted anything after that.

# Task 1d

## Introduction to task

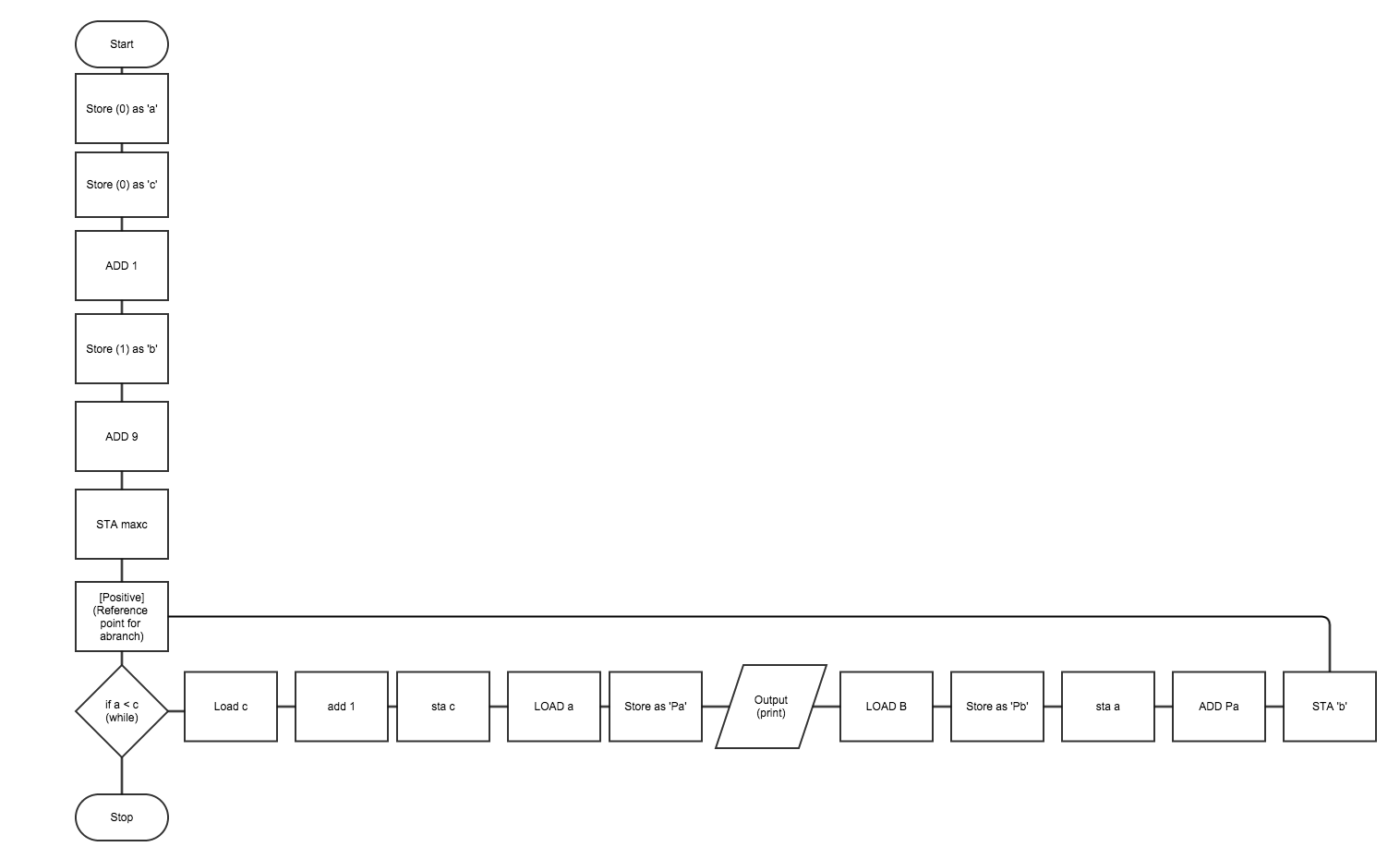
This task requires me to essentially convert my code for use in LMC. I need to plan, design, and dry run the code. However, an input is not necessary here, for this task ‘n’ is taken as 10. I need to calculate the first 10 numbers of Fibonacci sequence. In 1d, I simply need to plan the code

## Research of the task

The best way to plan this is to use a flowchart. I went back on to [gliffy.com](http://www.gliffy.com) and used the flowchart tool to design a flowchart. This flowchart will make it a lot easier for me to produce the LMC code. I had to review my knowledge of LMC by looking at the LMC Instruction set.

## Actual task

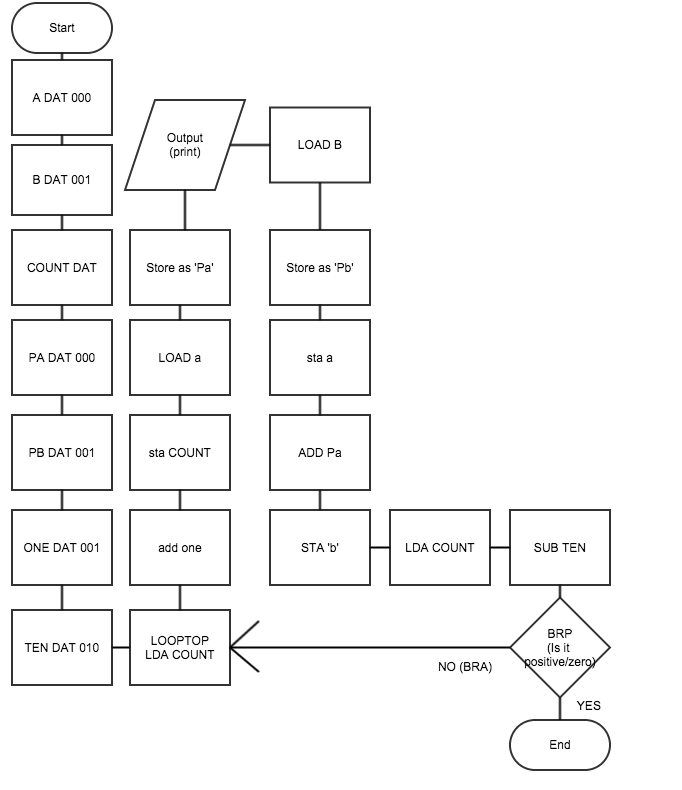
### Version 1

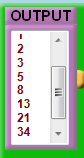


My first version of the flowchart is displayed here. It has a few issues. Firstly, I did not illustrate the decision in LMC code i.e. I didn’t mention what branches I had to use and where I had to use them. Secondly, in LMC you can give a value to a variable when you declare it, so some of my processes were unnecessary.

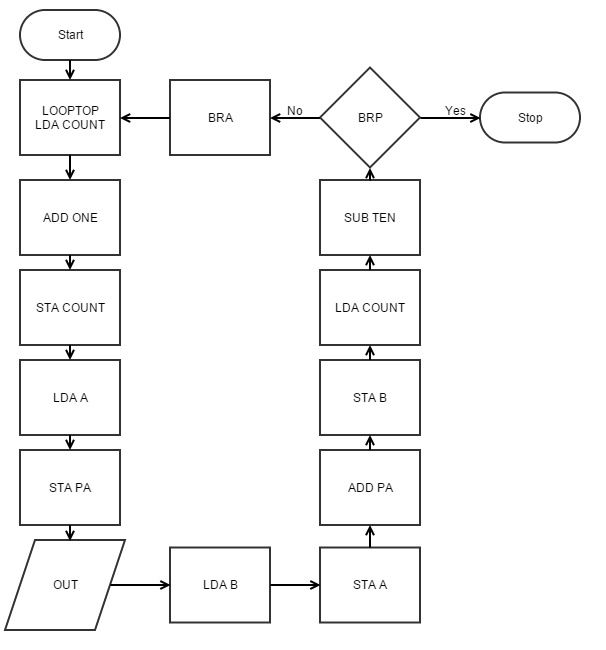
### Version 2

The version below fixes the aforementioned issues. It uses some LMC code to make it easy for me to use it when I actually write the code. However, it uses some Standard English to make it simple to read. This flowchart maps out the code I am constructing and it will help me when I actually code it.



This flowchart was great! Or so it seemed. When I did a dry run of the code in LMC the code did not work. This prevented me from taking any further steps in the dry run. Upon further investigation, this was because some of the code (where it actually calculates the Fibonacci sequence) was incorrect. I had one unnecessary variable ‘PB’ which created errors in my code. I then made a second, much more efficient, version of the code. When I did another dry run, this code was fully functional! My flowchart illustrates the code well if just one process is removed. The ‘*DAT*’s were at the beginning to make the flowchart look neater, in the actual code they would go at the end – after the ‘*HLT*’.

### Version 3



|  |  |
| --- | --- |
|  | Ensures that the code only runs for 10 iterations by keeping a count of how many times it has executed the code. |
|  | Stores the value of A as PB and outputs it |
|  | Stores the value of B as A and it adds it to the value of PA. The number (in the accumulator) is then stored as B. |

This version of my flowchart is error-free. This is the plan that I based my code on in 1e. I did not declare any ‘DATs’ in this flowchart because when I searched ‘LMC flowchart’ (on Google’s search engine) none of the flowcharts that appeared had any ‘DATs’ which led me to believe that it was wrong to have any ‘DATs’ in a flowchart. Additionally, I used arrows to direct the reader on how to read the flowchart and I phrased the contents of the flowchart like LMC code to make it easier for me to code. Additionally, I have labelled segments of the LMC code in the flowchart with their purpose.

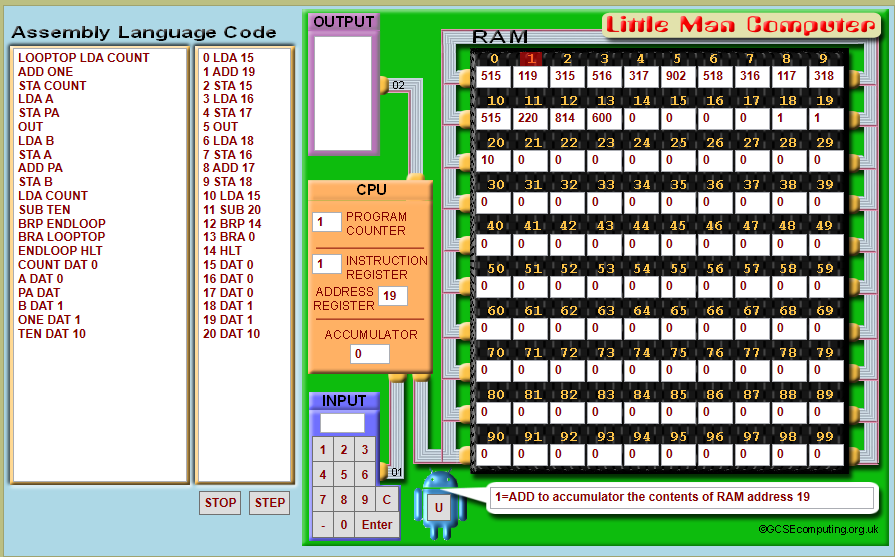
### Code

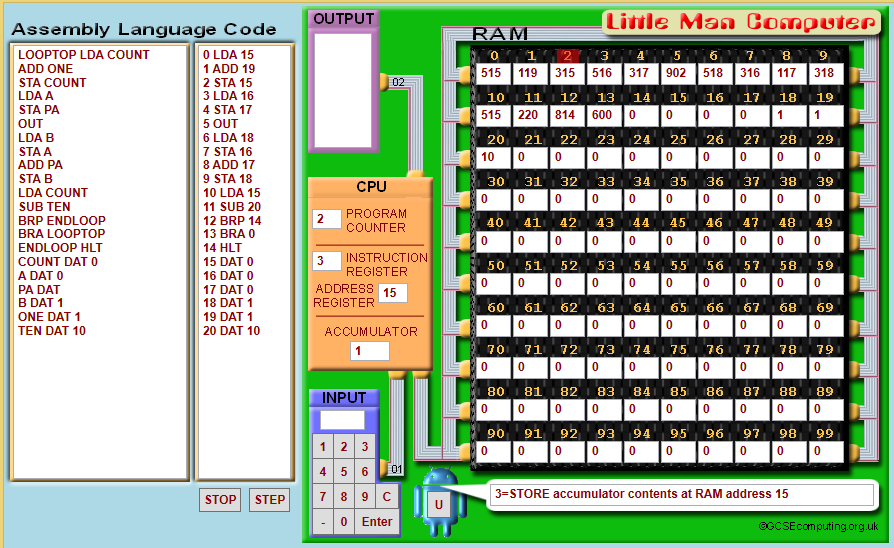
|  |  |
| --- | --- |
| 0 | LDA 15 |
| 1 | ADD 19 |
| 2 | STA 15 |
| 3 | LDA 16 |
| 4 | STA 17 |
| 5 | OUT |
| 6 | LDA 18 |
| 7 | STA 16 |
| 8 | ADD 17 |
| 9 | STA 18 |
| 10 | LDA 15 |
| 11 | SUB 20 |
| 12 | BRP 14 |
| 13 | BRA 0 |
| 14 | HLT |
| 15 | DAT 0 |
| 16 | DAT 0 |
| 17 | DAT 0 |
| 18 | DAT 1 |
| 19 | DAT 1 |
| 20 | DAT 10 |

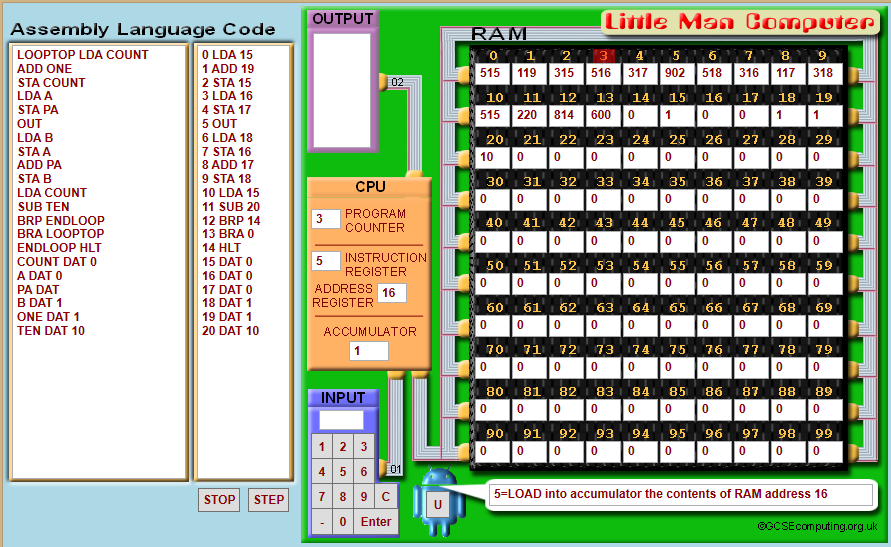
### Dry Run Table

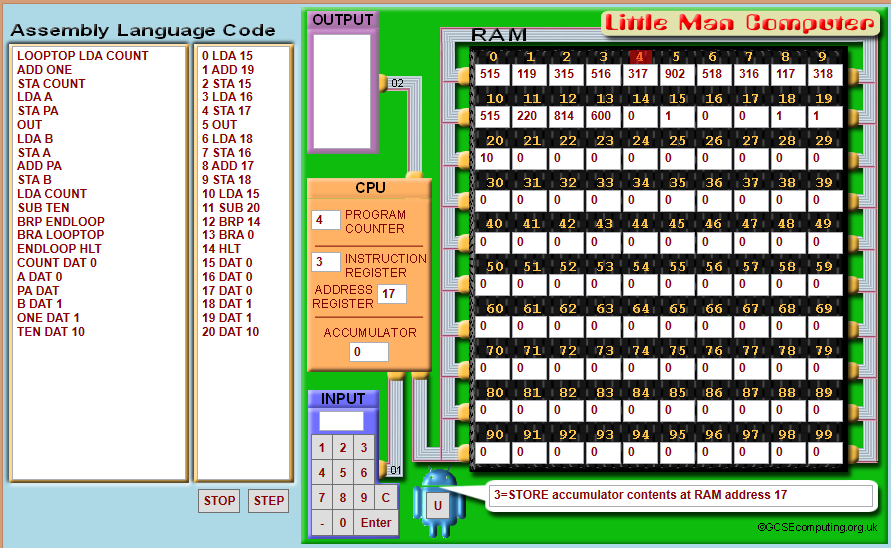
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Line | Count | A | PA | B | ONE | TEN | OUT |
| 0 |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 | 1 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  | 0 |  |  |  |  |
| 5 |  |  |  |  |  |  | 0 |
| 6 |  |  |  |  |  |  |  |
| 7 |  | 0 |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |
| 9 |  |  |  | 1 |  |  |  |
| 10 |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 | 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  | 1 |  |  |  |  |
| 5 |  |  |  |  |  |  | 1 |
| 6 |  |  |  |  |  |  |  |
| 7 |  | 1 |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |
| 9 |  |  |  | 2 |  |  |  |
| 10 |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 | 3 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  | 1 |  |  |  |  |
| 5 |  |  |  |  |  |  | 1 |
| 6 |  |  |  |  |  |  |  |
| 7 |  | 2 |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |
| 9 |  |  |  | 3 |  |  |  |
| 10 |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 | 4 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  | 2 |  |  |  |  |
| 5 |  |  |  |  |  |  | 2 |
| 6 |  |  |  |  |  |  |  |
| 7 |  | 3 |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |
| 9 |  |  |  | 5 |  |  |  |
| 10 |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 | 5 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  | 3 |  |  |  |  |
| 5 |  |  |  |  |  |  | 3 |
| 6 |  |  |  |  |  |  |  |
| 7 |  | 5 |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |
| 9 |  |  |  | 8 |  |  |  |
| 10 |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 | 6 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  | 5 |  |  |  |  |
| 5 |  |  |  |  |  |  | 5 |
| 6 |  |  |  |  |  |  |  |
| 7 |  | 8 |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |
| 9 |  |  |  | 13 |  |  |  |
| 10 |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 | 7 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  | 8 |  |  |  |  |
| 5 |  |  |  |  |  |  | 8 |
| 6 |  |  |  |  |  |  |  |
| 7 |  | 13 |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |
| 9 |  |  |  | 21 |  |  |  |
| 10 |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 | 8 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  | 13 |  |  |  |  |
| 5 |  |  |  |  |  |  | 13 |
| 6 |  |  |  |  |  |  |  |
| 7 |  | 21 |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |
| 9 |  |  |  | 34 |  |  |  |
| 10 |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 | 9 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  | 21 |  |  |  |  |
| 5 |  |  |  |  |  |  | 21 |
| 6 |  |  |  |  |  |  |  |
| 7 |  | 34 |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |
| 9 |  |  |  | 55 |  |  |  |
| 10 |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 | 10 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  | 34 |  |  |  |  |
| 5 |  |  |  |  |  |  | 34 |
| 6 |  |  |  |  |  |  |  |
| 7 |  | 55 |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |
| 9 |  |  |  | 89 |  |  |  |
| 10 |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |
| 15 |  |  |  |  |  |  |  |
| 16 |  | 0 |  |  |  |  |  |
| 17 |  |  | 0 |  |  |  |  |
| 18 |  |  |  | 1 |  |  |  |
| 19 |  |  |  |  | 1 |  |  |
| 20 |  |  |  |  |  | 10 |  |

The screenshots below show me performing a dry run of the code in my third flowchart.

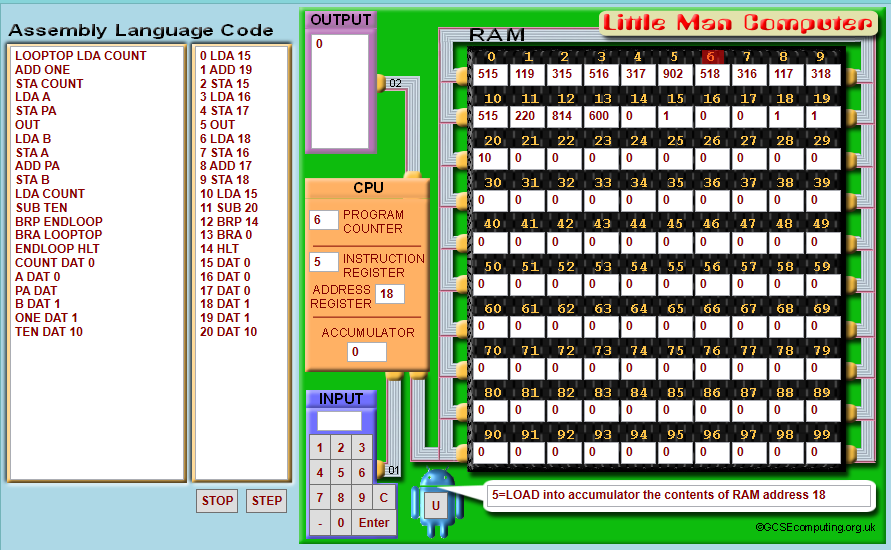


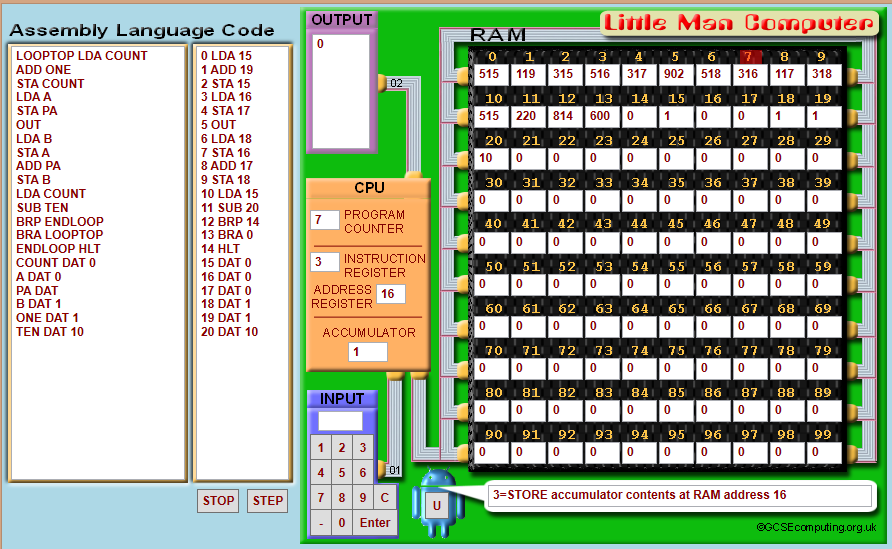


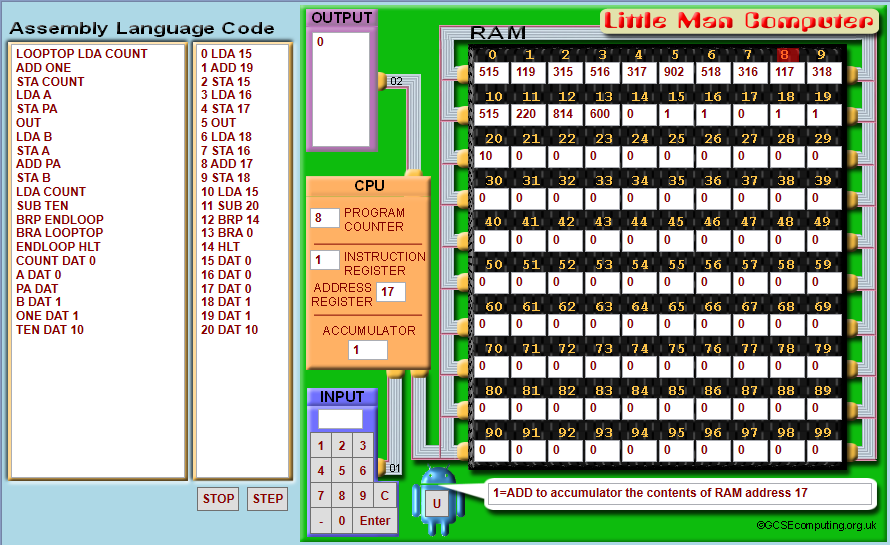


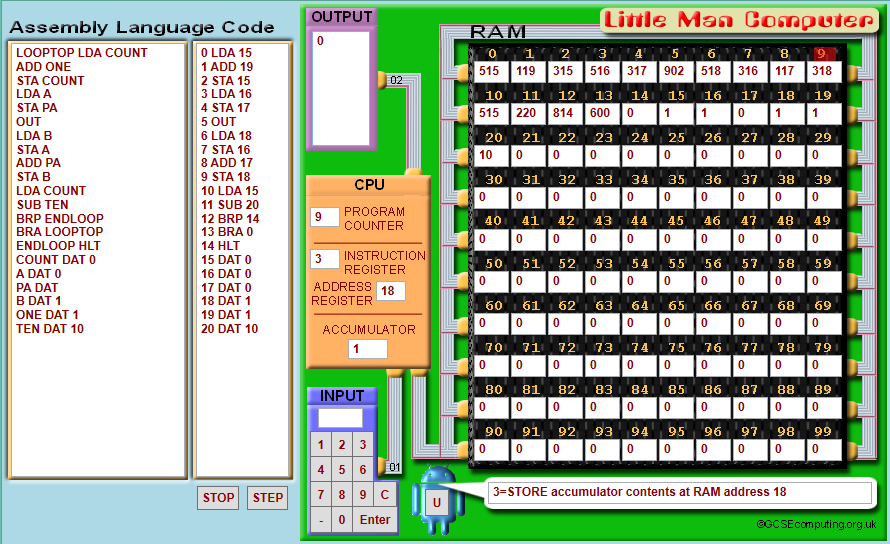




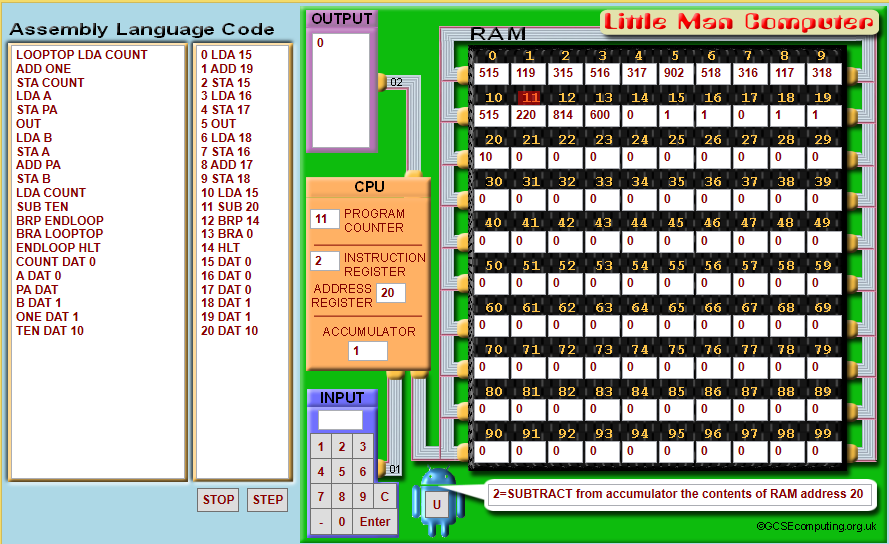


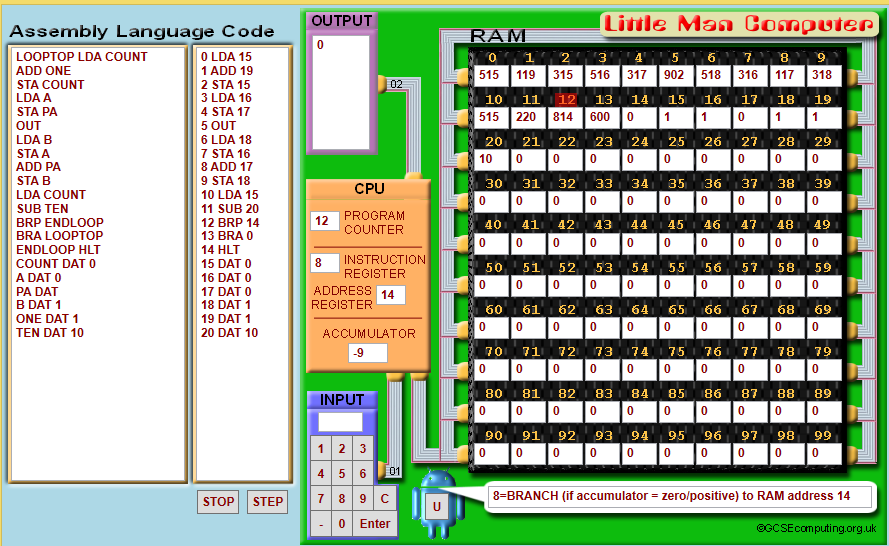


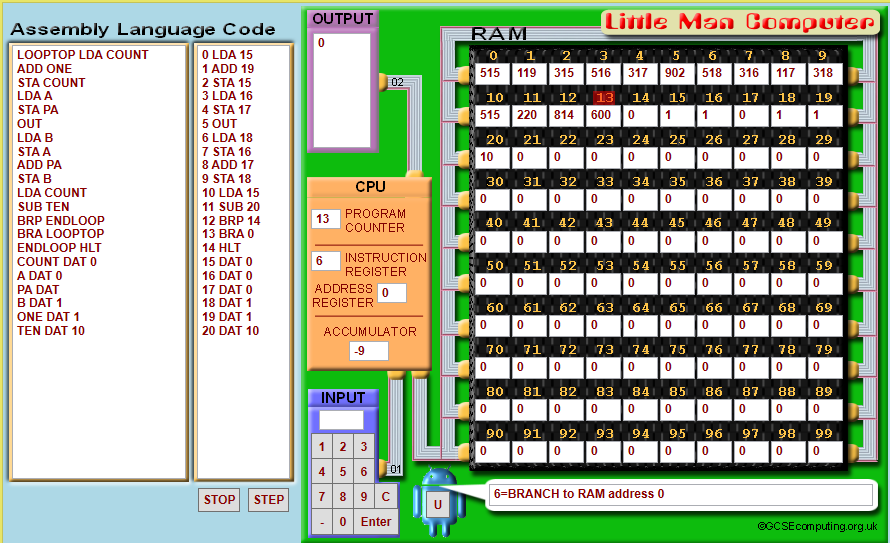












This dry run tests if the code works using the STEP function of LMC. These screenshots show the code’s actions in the RAM up to ‘BRA LOOPTOP’.

# Task 1e

## Introduction to task

This task requires me to code what I designed and planned.

## Research of the task

I reviewed my knowledge of LMC by using online resources like - <http://www.yorku.ca/sychen/research/LMC/>. The most important thing that I needed to review was loops. To make this program give me the results that I want, I needed to loop my code 10 times. The website told me how to achieve that; I need to have a variable which counted the amount of times I had done the sequence. I named this variable ‘*COUNT’*. Every time my program calculated the next number in the sequence, it incremented ‘*COUNT*’ by one. When count equals ten, the program should stop. To find out if ‘*COUNT*’ is 10, the program loads ‘*COUNT*’ and subtracts ‘*TEN*’ from it. If the answer is positive or zero (‘*BRP*’), it ends the program. If the answer is not positive or zero, it goes back to the beginning of the loop because of the ‘*BRA*’ command.

## Actual task

Code:

LOOPTOP LDA COUNT

ADD ONE

STA COUNT

LDA A

STA PA

OUT

LDA B

STA A

ADD PA

STA B

LDA COUNT

SUB TEN

BRP ENDLOOP

BRA LOOPTOP

ENDLOOP HLT

COUNT DAT 0

A DAT 0

PA DAT

B DAT 1

ONE DAT 1

TEN DAT 10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **What am I testing** | **Test description** | **Expected outcome** | **Success/Fail** | **Evidence given** |
| Compilation | This test will check if the code runs normally – using RUN instead of STEP. | The code should be transferred to memory without any errors. | Success |  |
| Output | The program should output the expected values – the first 10 numbers of the Fibonacci sequence (starting from 0 and 1). | The numbers in the output box should be: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34. | Success |  |

# Task 1f

## Introduction to task

This task requires me to just allow the user to input a number which affects the number of iterations the code runs for.

## Research of the task

From prior knowledge, I knew how to do this task. I need replace the variable ‘TEN’ with one which changes based on user input. This changes how many times it calculates the Fibonacci sequence. Due to the fact that I wrote my original code so that it would be easy to change, this task was simple.

## Actual task – Version 1

INP

STA MAXCOUNT

LOOPTOP LDA COUNT

ADD ONE

STA COUNT

LDA A

STA PA

OUT

LDA B

STA A

ADD PA

STA B

LDA COUNT

SUB MAXCOUNT

BRP ENDLOOP

BRA LOOPTOP

ENDLOOP HLT

COUNT DAT 0

A DAT 0

PA DAT

B DAT 1

ONE DAT 1

This is my planned code for the program. I will test it in task 1g.

## Version 2

This version of my code was made after the compilation failure in task 1g’s test plan. This version of the code should compile correctly. I will test it in another test plan in task 1g.

INP

STA MAXCOUNT

LOOPTOP LDA COUNT

ADD ONE

STA COUNT

LDA A

STA PA

OUT

LDA B

STA A

ADD PA

STA B

LDA COUNT

SUB MAXCOUNT

BRP ENDLOOP

BRA LOOPTOP

ENDLOOP HLT

COUNT DAT 0

A DAT 0

PA DAT

B DAT 1

ONE DAT 1

MAXCOUNT DAT

# Task 1g

## Introduction to task

This task requires me to test my code. For this task, I will utilise a test plan which tests my code.

## Test Plan V1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **What am I testing** | **Test description** | **Expected outcome** | **Success/Fail** | **Evidence given** |
| [Useful message output] STEP – Prompt for user input | This test will check (using STEP) if the code asks for an input and it will check to see if the code (mostly the branches) malfunctions. | LMC should prompt me to input a value into the accumulator and the code should not stop until it gets to ‘*HLT*’. | Success |  |
| Compilation | This test will check if the code runs normally – using RUN instead of STEP. | The program should serve its intended function. It should ask for an input (n) and use that input to calculate the Fibonacci sequence ‘n’ times. | Fail |  |

These tests existed to check if the code functioned properly. The first test checked to see if the code asked for an input and to see loops/branches functioned correctly. I used the ‘*STEP*’ button to **specifically** check if just **those** things worked. I expected the program to ask me to input a value into the accumulator. Additionally, the code should cycle through the loop and return to the beginning of the loop if it hasn’t reached ‘n’. The test was a success. The second test checked to see if the code functioned using the RUN button. This test checked to see if the average user of this program could easily use it and get the outcome that they want. I expected the program to ask me for an input and calculate the sequence ‘n’ times and output it. Unfortunately, this was a failure. Upon reviewing my code again, I discovered that I did not define one of my variables – MAXCOUNT. I fixed my code and I saved the updated version of the code as Task 1f – V2. I will also test this new version of my code with a test plan.

## Test plan V2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **What am I testing** | **Test description** | **Expected outcome** | **Success/Fail** | **Evidence given** |
| [Useful message output] STEP – Prompt for user input | This test will check (using STEP) if the code asks for an input and it will check to see if the code (mostly the branches) malfunctions. | LMC should prompt me to input a value into the accumulator and the code should not stop until it gets to ‘*HLT*’. | Success |  |
| Compile | This test will check if the code runs normally – using RUN instead of STEP. | The program should serve its intended function. It should ask for an input (n) and use that input to calculate the Fibonacci sequence ‘n’ times. | Success |  |
| User input – valid/valid extreme | What is ‘valid’ is determined by the input of the user. E.g. if the user inputs 6, the valid extreme would be 6. | The program should only output valid values, which are the Fibonacci sequence n times. E.g. the program should not output more than n values. N is the valid extreme – in this case 6. | Success |  |
| User input –invalid extreme | Continuing the previous example, 7 would be an invalid example as it exceeds the number 6. It is also just on the edge of being valid which also makes it the invalid extreme value. | The program should still output values n times. In this case n is the invalid extreme. E.g. the program should not output values more than 7 times | Success |  |
| User input – invalid | One limitation of LMC is that it cannot handle 3 digit values. This test will check to see what happens when an incredibly large value is inputted. | The program should not output valid values as it (LMC) was not designed to handle numbers this large. | Success |  |
| User input – erroneous | What happens if the values inputted are not integers? | The program should not accept the input and it should ask the user to input a value into the accumulator. | Success |  |

These new tests existed to test my new, improved code. These could not have been done on the other version as there was a compilation error.

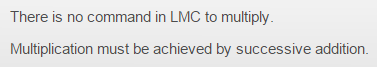
# Task 2a

## Introduction to task

This task requires me to figure out how to multiply two numbers in LMC.

## Research of the task

Whenever I encounter a problem which leaves me puzzled for more than a minute, I search it using Google’s search engine. So for this task I searched ‘multiplication in LMC command’. This led me to this reliable website (<http://www.ict4ocr.com/computing/lmc_blocks.html>) which stated:



## Actual task

According to the aforementioned website, I need to add one thing to itself a number of times.

E.g. 6 \* 4 = 24 can be simplified to 6 + 6 + 6 + 6 = 24.

# Task 2b

## Introduction to task

This task requires me to plan an LMC program which can multiply. I already did most of the research required for this task in the previous task. The rest of the task, I can do from prior knowledge. I have used a flowchart, as it is a user friendly (easy to read) method for displaying an algorithm. I created this flowchart in a way that makes it easy for other people to create the LMC code using this – e.g. I mention how you define the variables in the flowchart.

## Actual task

### Pseudo code:

Input a number

Store it as a

Input a number

Store it as b

Looptop Load the value of count

Add one to it

Store the number as count

Load the value of ans

Add a to it

Load the value of count

Subtract b from it

If value in the accumulator is positive:

Load ans

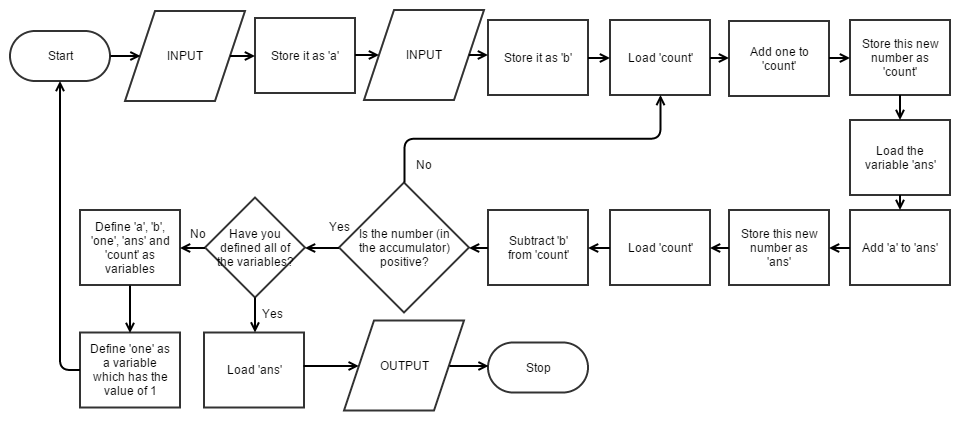
Output it

End program

Else:

Go to Looptop

### Flowchart:



# Task 2c

## Introduction to task

This task requires me to write the program I planned in LMC. I also need to demonstrate it. At this point, I am quite experienced in LMC (from doing the previous tasks) so; I did not need to do any extra research.

## Actual task

### Version 1

INP

STA a

INP

STA b

loop LDA count

ADD one

STA count

LDA ans

ADD a

STA ans

LDA count

SUB b

BRP end

BRA loop

end LDA ans

OUT

HLT

a DAT

b DAT

count DAT 0

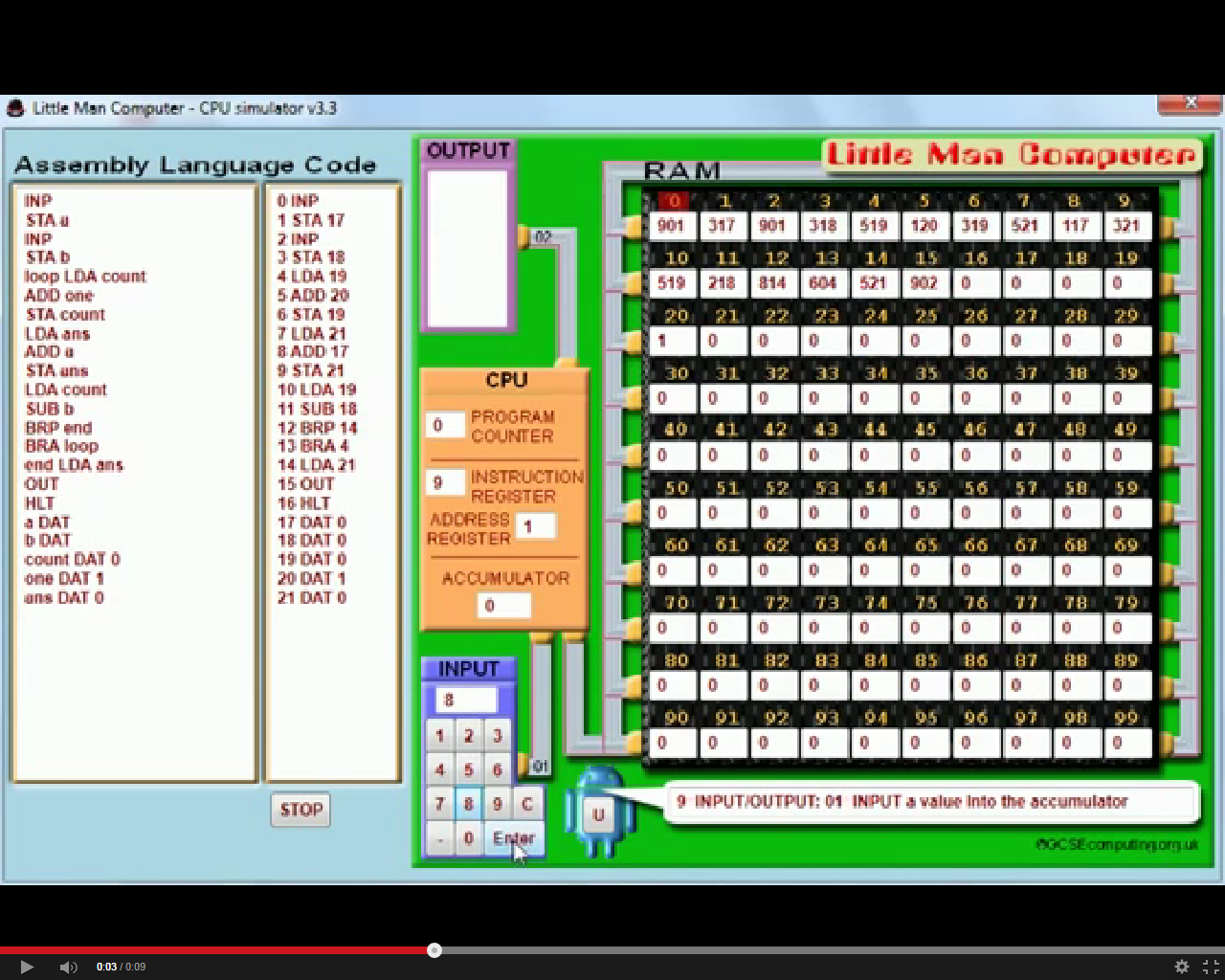
one DAT 1

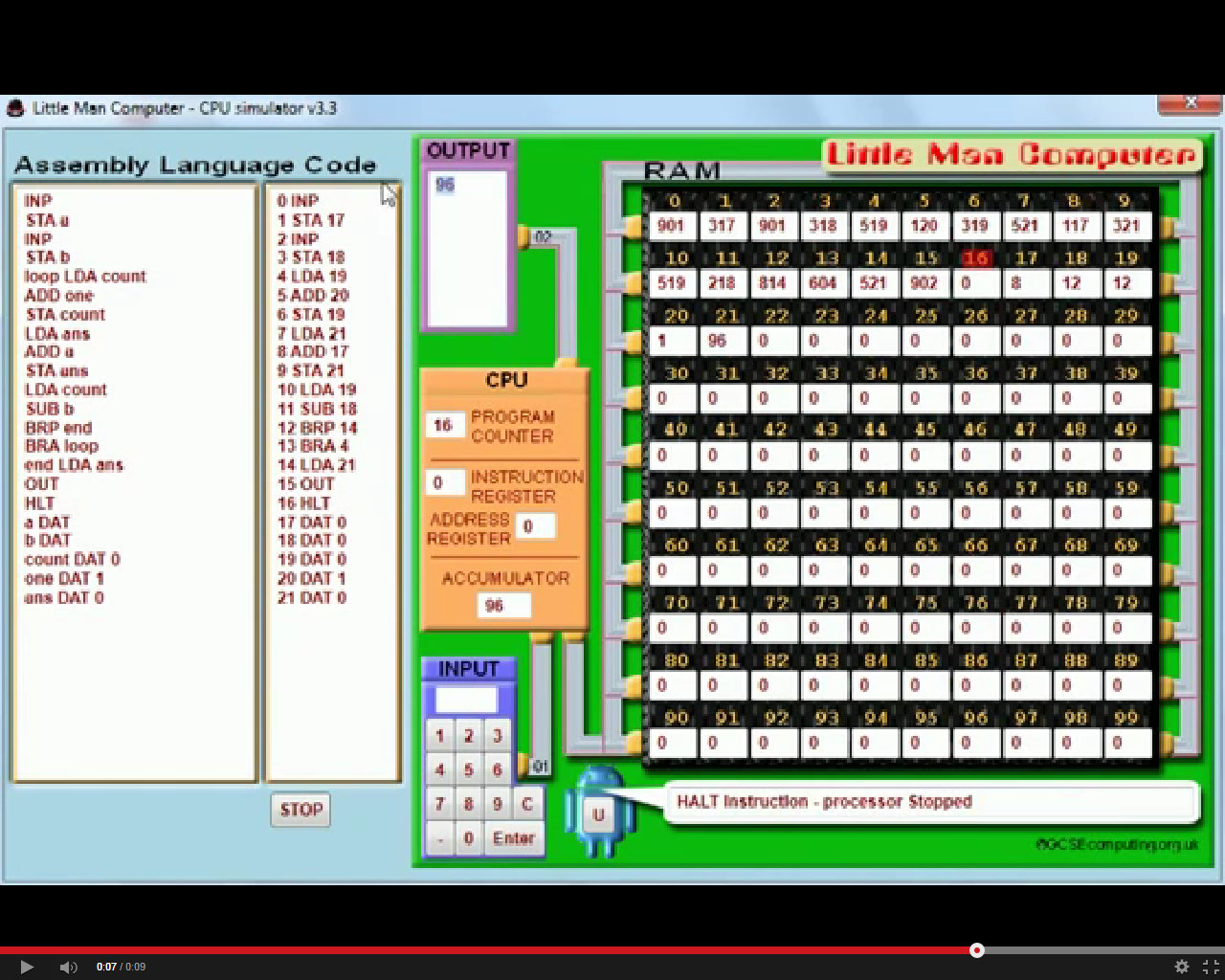
ans DAT 0

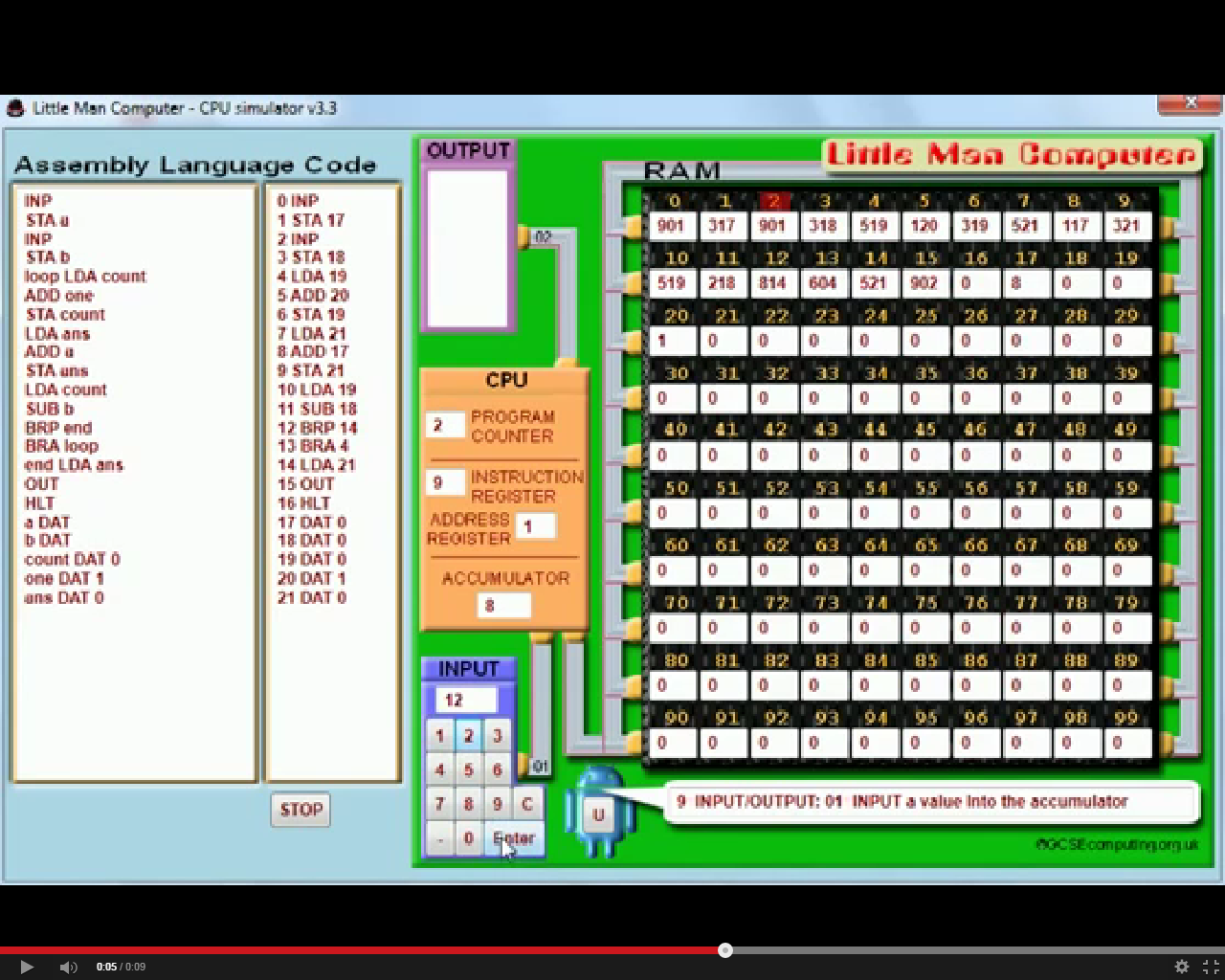
Video of program in action: **(**<https://drive.google.com/file/d/0ByLW4IVm4aZqNWZVQ0tHdHJOVE0/view?usp=sharing>**)**

(Ctrl+Click – Opens in browser – My video on Google Drive)

#### Screenshots from video:







### Version 2

INP

STA a

INP

STA b

SUB a

BRP loop\_b

loop\_a LDA count\_a

ADD one

STA count\_a

LDA ans

ADD a

STA ans

LDA count\_a

SUB b

BRP end

BRA loop\_a

loop\_b LDA count\_b

ADD one

STA count\_b

LDA ans

ADD b

STA ans

LDA count\_b

SUB a

BRP end

BRA loop\_b

end LDA ans

OUT

HLT

a DAT

b DAT

count\_a DAT 0

count\_b DAT 0

one DAT 1

ans DAT 0

This version of the code is far more efficient as it chooses the shorter of two methods to multiply in LMC.

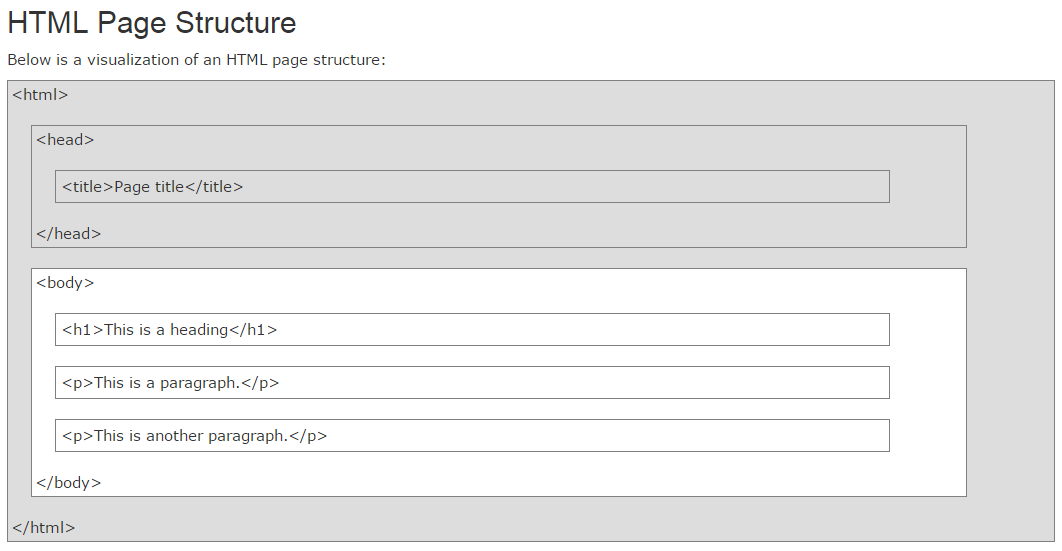
# Task 3a

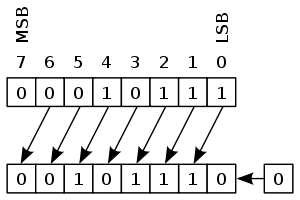
## Introduction to task

This task requires me to understand the code and explain what it does. I started off by copying and pasting the code into a text document, which I then saved as an HTML file.

## Research of the Task

Although JavaScript is quite alien to me, I can mostly understand the code by drawing upon previous research and knowledge. I knew what html was and how to use it for simple things because of research I did outside of school. To confirm my knowledge of html, I looked over the website that I used to first learn html - <http://www.w3schools.com/html/html_intro.asp>. This website helped me to understand the code in task 3. I searched ‘JavaScript operators’ (on Google’s search engine) and found this page (<http://www.w3schools.com/js/js_operators.asp>) on the same website which explained the ‘<<=’ operator in the JavaScript segment. It ‘Left shifts’ a number. I did not know what ‘left shift’ was so I searched it (on Google’s search engine) and found this Wikipedia article (<http://en.wikipedia.org/wiki/Arithmetic_shift>) on it. This particular article was simple to understand so I used it.





[Source: <http://www.w3schools.com/html/html_intro.asp>, <http://en.wikipedia.org/wiki/Arithmetic_shift>]

Left shift:

Left shift is essentially doubling. The 1s in binary are moved to the left.

**256 128 64 32 16 8 4 2 1**

0 0 0 0 0 0 1 1 0 = 6 (6 normally)

0 0 0 0 0 1 1 0 0 = 12 (6 left shifted once)

0 0 0 0 1 1 0 0 0 = 24 (6 left shifted twice)

0 0 0 1 1 0 0 0 0 = 48 (6 left shifted three times)

0 0 1 1 0 0 0 0 0 = 96 (6 left shifted four times)

## Actual task

The first line declares this document as a HTML document.

The <html> tag describes a HTML document.

The <body> tag describes whatever content is visible to the user.

These two tags usually begin most HTML documents. These documents also have to end with </html> and </body>. A ‘/’ in </html> ‘ends’ a particular tag. Everything between the <html> and the </html> is considered as HTML code.

The <script> tag defines a ‘client-side’ script – like JavaScript.

Everything beyond <script> and before </script> in this document is not HTML. It is code from JavaScript. Despite the fact that I don’t know JavaScript, I can still identify what some of the code does by making careful observations and using previous knowledge.

‘var temp = 14;’ simply creates a variable called ‘temp’ and it assigns it a value of 14.

‘var y = 2;’ simply creates a variables called ‘y’ and it assigns it a value of 2.

‘temp <<=y;’ left shifts ‘temp’ ‘y’ number of times.

‘document.write(temp);’ outputs the variable ‘temp’ onto the page.

# Task 3b

## Introduction to task

This task just requires me to change the value of ‘y’ and observe the change to the number displayed on the screen. In addition, I need to explain how the number changes. Furthermore, I do not need any research for a task like this. According to my previous research, the number displayed on the screen should be ‘temp’ doubled ‘y’ times.

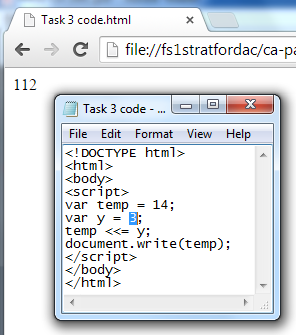
## Actual task

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Value of temp | Value of y | Number displayed on webpage | What the number can be broken down to | What the number can be simplified to |
| 14 | 1 | 28 | 14+14 | 14\*21 |
| 14 | 2 | 56 | 14+14+14+14 | 14\*22 |
| 14 | 3 | 112 | 14+14+14+14+14+14+14+14 | 14\*23 |
| 14 | 4 | 224 | 14+14+14+14+14+14+14+14+14+14+14+14+14+14+14+14 | 14\*24 |

If I were to break this down into a formula, it would look something like this:

Temp\*2y = Number displayed on webpage

That is what the line ‘temp <<= y;’ does, it doubles ‘temp’ ‘y’ times. However, it does not output the number to the webpage – which is what ‘document.write(temp);’ does.

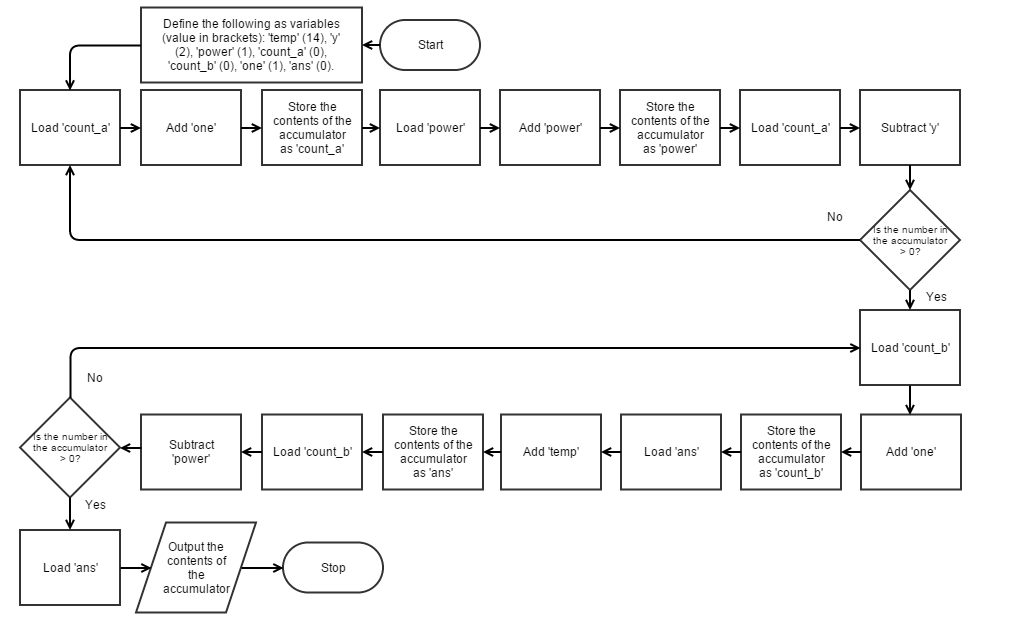
[Screenshot of me changing the value of y]

# Task 3c

## Introduction to task

For this task I need to plan and code a program for LMC that replicates the function of the JavaScript code. Having done something similar a few tasks ago, I did not need to do any research for this task. I will plan the program using the hosted software Gliffy.

## Actual Task



Because I designed my flowchart to be very similar to LMC, coding the program was not a difficult feat to accomplish. The flowchart also shows the values of the variables as it is essential to ensure that the code has the same values for the variables.

### LMC code:

### Version 1

loop\_a LDA count\_a

ADD one

STA count\_a

LDA power

ADD power

STA power

LDA count\_a

SUB y

BRP done\_a

BRA loop\_a

done\_a LDA count\_b

ADD one

STA count\_b

LDA ans

ADD temp

STA ans

LDA count\_b

SUB power

BRP done\_b

BRA done\_a

done\_b LDA ans

document.write(temp);

</script>

var temp = 14;

var y = 2;

OUT

HLT

temp DAT 14

y DAT 2

ans DAT 0

power DAT 1

count\_a DAT 0

count\_b DAT 0

one DAT 1

This code functions identically to the JavaScript code. To change the value of ‘temp’ you change this number and to change the value of ‘y’ you change this number. As there is no left shift function in LMC, I had to think of another method. I looked back at my formula and I utilised that. Additionally, I also used my code from task 2c in order to multiply. I split my code into two section – A and B. Section A focused on figuring out what power of two I needed. My formula (Temp\*2y) shows that y is equal to the power of two, so the code multiplies 2 by itself ‘Y’ times. After that, section B of the code just multiplies the ‘ans’ from section A by ‘Temp’.

### Version 2

INP

STA a

INP

STA b

loop\_a LDA count\_a

ADD one

STA count\_a

LDA power

ADD power

STA power

LDA count\_a

SUB b

BRP done\_a

BRA loop\_a

done\_a LDA count\_b

ADD one

STA count\_b

LDA ans

ADD a

STA ans

LDA count\_b

SUB power

BRP done\_b

BRA done\_a

done\_b LDA ans

document.write(temp);

</script>

var temp = 14;

var y = 2;

OUT

HLT

a DAT

b DAT

ans DAT 0

power DAT 1

count\_a DAT 0

count\_b DAT 0

one DAT 1

This version of my code allows the user to input values which is more user friendly than needing to change the values of the data in the code. The rest of the code is nearly identical.

# Task 4

## Introduction to task

This task requires me to look over my work and evaluate how successful it was. I also need to explain how I overcame any difficulties that were present.

## Actual task

### Task 1a evaluation

This task required me to explain the purpose of the code. I also needed to analyse and explain the code line-by-line.

As the beginning of the task, I ran into a very minor dilemma. I was unsure of what the ‘def’ function did. As a result, I searched ‘python def’ (on Google’s search engine) to find out what it did – it is used to create functions. I knew what the other commands did and as a result, I was able to easily explain what the separate lines in the code did and what they accomplished on the whole. I explained (the code) to a level anyone could understand.

### Task 1b evaluation

This task required me to create a flowchart depicting the code shown in task 1.

[Gliffy.com](http://www.gliffy.com) was a very user-friendly website which made creating a flowchart very simple. I understood the code so it was easy for me to convert the code into a flowchart. However, I ran into two roadblocks. Firstly, I was unsure of what some of the flowchart symbols meant so I looked up flowchart symbols and found an image which explained the major flowchart symbols to me. Secondly, I did not loop my flowchart correctly. I initially (in version 1) had my flowchart loop into an input. This was wrong for three reasons: (1) you are not allowed to loop into an input in a flowchart; (2) this would make it so that the program would keep asking me to enter an input; (3) the original code did not even have an input. I managed to fix these issues in version 2 by removing the input and adding a reference point for the loop. On the whole, however, the task was quite successful.

### Task 1c evaluation

This task required me to add some code to the existing code to make it run and be useful.

I identified what I needed to do – add an input. I also needed to output some text to prompt the user to enter a number. From prior computing lessons, I knew how to add an input. I also made it so that only integers could be entered into the program. However, there were some issues. The program only worked once and a ‘none’ error kept appearing. I hypothesised that the error kept appearing because I had a really long and confusing line in my code – ‘n = int(input(print(“Please enter a number: “)))’. I was able to simplify this line into two separate lines (in version 2) and the code worked without any errors. I was able to fix the issue of the code only functioning once be adding a while loop to the code. This allowed a user to enter values for n into the program indefinitely.

### Task 1d evaluation

The next task (1e) required me to convert my python code into assembly code for use in LMC. An input was not required for this task as ‘n’ was taken as 10. In essence, I needed to calculate the first 10 numbers of the Fibonacci sequence in LMC. For this task, I simply needed to plan the code.

I decided to create a flowchart to plan my code because flowcharts are one of the best methods of planning. Like before, I used [www.gliffy.com](http://www.gliffy.com) to code my program. The first version of the flowchart I created had a few issues. Firstly, I did not illustrate the decision in LMC code i.e. I didn’t mention what branches I had to use and where I had to use them. Secondly, in LMC you can give a value to a variable when you declare it, so some of my processes were unnecessary. My second version of the flowchart fixed the aforementioned issues but when I used it to write and perform a dry run of the code, there were some errors. These errors prevented me from taking any steps in my dry run. To combat these issues, I created a third version of the flowchart which I used to create my code in the next task. This flowchart did not have any ‘DATs’ because every other LMC flowchart I saw (on the internet by googling ‘LMC flowcharts’) did not have any. I also labelled the processes so that readers could see their purpose. Finally, I did a dry run with the third version of my flowchart which showed that the code ran without errors.

### Task 1e evaluation

As I stated in the previous task (1d), this task required me to convert my python code into assembly code for use in LMC.

After my planning in task 1d, this task did not take long. I simply reviewed my knowledge of LMC and, in particular, flowcharts relating to LMC using this website - <http://www.yorku.ca/sychen/research/LMC>. I was able to write the code without any issues. I then tested the program by using a test plan. Both of the tests I conducted were successful. I made the next task easier for myself by writing code that could be easily modified.

### Task 1f evaluation

This task required me to just allow the user to input a number to change the number of iterations the code ran for.

Because I wrote my code in 1e so that it could easily be changed, I only had to: have an input, store it in a mailbox, and substitute it for the ‘ten’ in original code. I did not, however, test this code. That had to be postponed until the next task.

Halfway through task 1g, I realised I had to make a second version of my code which addressed the compilation issue that my test plan revealed. I checked over my code and I identified that I did not define one of my variables – MAXCOUNT. I defined it in version 2 and I then tested the code in task 1g.

### Task 1g evaluation

This task required me test my code.

I utilised a test plan to make it easier for me to test my code. The test plan enabled me to conduct two tests on my code in detail. I included all of the essentials of a test plan in my one. It had: a test description, an expected outcome, a success/fail box, and a box which contained evidence for the test. I also explained the tests I conducted in detail underneath the table. My first test (useful message output) was successful but my second test (compilation) was not. The second test’s failure made it pointless for me to conduct any more tests on the first version of my code as they would all fail because the code would not compile. With that in mind, I went back to task 1f and created a new version of my code which I then tested in a second test plan. This new test plan tested many types of user inputs such as: valid/valid extreme, invalid extreme, invalid, erroneous. These tests were all successful.

### Task 1 overall evaluation

On the whole, this task was successful. I ran into a few errors on the way, but I managed to fix them. For example, I had numerous problems and road blocks with the flowcharts in task 1d. In addition to that, I ran into compilation issues in my code which made it difficult to complete tasks. I managed to persevere and learn from my mistakes which ensured that I wouldn’t run into the same issues later on. Throughout the task, I was able to use and refine my various skills in flowchart-making and coding in LMC.

### Task 2a evaluation

This task simply required me to explain how to multiply two numbers in LMC.

With the help of Google’s search engine, I was able to figure this out quickly. Multiplication could only be achieved in LMC by successive addition.

### Task 2b evaluation

This task required me to apply what I had learned in the previous task and plan an LMC program which can multiply.

I first wrote down my idea in pseudo code form. This made it easy for me to then turn it into real LMC code. I then used a flowchart (which I created using [www.gliffy.com](http://www.gliffy.com)) to plan my work. Because I have created flowcharts numerous times in this controlled assessment, I could make a good, effective flowchart quickly. As such, I was able to complete this task in a short period of time which let me spend more time on the other tasks. Compared to task 1d, this flowchart took me significantly less time because of the experience I had acquired from using (hosted) software like Gliffy to complete the previous tasks.

### Task 2c evaluation

This task required me to write the program I planned in LMC. I was able to successfully write the program but I noticed one issue with it. If someone input 3 and 99, it would keep adding 3 until it reached 297 instead of doing the much faster 99+99+99. Because of that, I created a second version of the program which addressed this issue. I did not run into any issues with either program.

### Task 2 overall evaluation

All in all, this task was very successful. I ran into very few errors on the way and I was able to use and refine my various LMC related skills. I was also able to identify the drawbacks of my code (e.g. speed) and change my program in order to compensate. Compared to task 1, this task took a lot less time to complete because of how much more experienced I was at the time.

### Task 3a evaluation

This task required me understand the HTML/JavaScript code and explain what it does.

Due to the fact that I had some prior experience in HTML, most of the research in this task was done so that I could review my knowledge of HTML. The website I used ([w3schools.com](http://www.w3schools.com)) also helped me to understand the operators used in the JavaScript segment of the code. I was able to successfully identify and explain, line-by-line, what the code did. I did not initially know what ‘left shift’ was but I was able to work it out by googling it and clicking on the Wikipedia link.

### Task 3b evaluation

This task just required me to change the value of ‘y’ and observe the changes to the number displayed on the webpage.

This task was very successful for me. By drawing upon previous knowledge, I was able to edit the code in a text editor – in this case it was Notepad. I was able to observe the changes and make a table showcasing these changes and how they fit in with my earlier hypothesis – about ‘<<=’ and left shift. In addition, I also converted what was foreign to me (‘left shift’) to something more familiar – a mathematical formula. This made it a lot easier for me to understand the concept of left shift. I also took a screenshot while I changed the value of ‘y’ to showcase how I did it.

### Task 3c evaluation

For this task I needed to plan and write a program (in LMC) that replicates the function of the JavaScript code.

I did not need to do any research for this task as I did similar things a few tasks ago. I planned the program in the form of a flowchart using the hosted software Gliffy. I designed my flowchart to be very similar to LMC in order to simplify the process of coding the program. This time, I declared the variables in the flowchart because I needed to know the values of the variables when I was planning the program. I was able to write the program without any issues. In the first version of my program, the user needed to change ‘temp DAT 14’ and ‘y DAT 2’ to change the values of ‘temp’ and ‘y’. This felt slightly tedious, thus I created version 2 of my program which asks the user for the values of ‘temp’ and ‘y’. I also explained what the code did for both versions.

### Task 3 overall evaluation

All things considered, this task was successful. I was able to find explanations for concepts I knew nothing about and I was then able to put them in forms where they were easier to understand. Additionally, I did not run into any errors on my code but I did find ways to improve it. In contrast to the previous tasks, this work in this task was new for me.

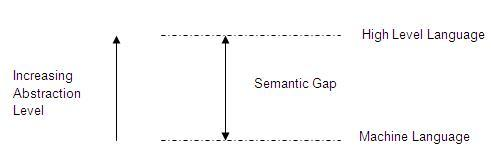
# Task 5

## Introduction to task

This task requires me to conduct research on the CISC and RISC architectures. I then need to explain the benefits and drawbacks of these two architectures in an unbiased form. Finally, I need to decide which architecture is better for a chip manufacturer today.

## Research of the task

I searched ‘CISC architecture’ instead of just ‘CISC’ to increase the relevancy of the results given by Google’s search engine. Numerous links came up on the search engine. The top link was a Wikipedia link - http://en.wikipedia.org/wiki/Complex\_instruction\_set\_computing. While Wikipedia is a good resource for discovering information, it is hard to understand some of it because most of the articles are written to appeal to professionals and the reliability of the information is questionable. The second link was this - <http://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/>. This website provided me with a lot of information on RISC as well as CISC. These two architectures were both developed in order to cover the semantic gap – illustrated below.



[Source: <http://www.engineersgarage.com/articles/risc-and-cisc-architecture>]

**CISC:**

The main goal of the CISC (Complex instruction Set Computers) architecture is to complete a task in the shortest amount of lines of assembly code. This goal is achieved by building processor hardware that is able to understand and execute a series of operations. This means that:

* There is a greater emphasis on hardware
* It includes multi-clock complex instructions
* The sizes of the code are smaller and there are a high number of cycles per second
* The transistors are used for storing complex instructions

The main advantage of this architecture is that the compiler does not need to do much work to convert high level language into assembly. Additionally, the short length of the code means that little RAM is needed to store instructions. This consequently reduces the cost of the RAM needed for a machine running on the CISC architecture. However, one drawback of CISC is that data must be reloaded from the memory bank into a register after a computation.

**RISC:**

The main goal of the RISC (Reduced instruction Set Computing) architecture is to use simple instructions that can be executed in one clock cycle. Complex instructions are divided into separate simple commands. This means that:

* There is a greater emphasis on software
* There are only single-clock instructions
* The sizes of the code are larger and there are a lower number of cycles per second
* More transistors are used on memory registers

The main advantage of this architecture is that each instruction requires only one clock cycle to execute, the entire program will run in roughly the same amount of time as a CISC complex instruction. Running in one clock cycle also allows for pipelining. In contrast with CISC, registers do not need to reload their data from the memory bank after a computation using RISC. Finally, the “Reduced Instructions” in RISC need less transistors of hardware space which leaves more room for general purpose registers. However, one drawback of RISC is that there is less software support which causes and results in it having a small hold of the commercial world. Because these two things directly impact each other, the RISC architecture should provide a significant speed advantage in order to compete with the other major architectures – like CISC.

**The performance of a computer is usually calculated using this equation:**



*[Image source:* [*http://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/*](http://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/)*]*

The CISC approach is to minimize the number of instructions per program, sacrificing the number of cycles per instruction. RISC attempts to do the opposite, reducing the number of cycles per instruction at the cost of the number of instructions per program. Both methods have their merits and they yield similar times for the execution of programs.

## Actual task

**Which one is better for a chip manufacturer today?**

Both chips have their advantages and disadvantages; there is no clear superior architecture. One relies on hardware for speed and the other relies on software. The time it takes to run a program varies based on the program itself and, overall, CISC and RISC both provide similar speeds. Even if RISC is sometimes faster, this is balanced by the added difficulty in programming for it. The main issue for these chips, in reality, is compatibility and therefore, I believe that the architecture that a chip manufacturer makes their chip for should be different based on the purpose of the chip.

If the chips are being built for smartphones or tablets, they should use the RISC architecture, specifically the ARM architecture – which is based off the RISC architecture. This architecture is used by the major smartphone OSs (Operating Systems) i.e. iOS, Android, Windows Phone etc. Most, if not all, smartphones in the world use the ARM architecture so any smartphone without it would not be able to use these OSs and they would suffer from a range of issues - in particular, regarding applications. It is a lot more difficult to port apps to a completely different architecture. Additionally, most programmers in the field are trained in the programing languages and SDKs (Software Development Kits) used for iOS and Android so it would be more difficult to find app developers who are willing to learn a completely different architecture. A non-ARM smartphone would find it extremely difficult to compete on the market.

If the chips are being built for PCs and laptops, they should use the CISC architecture – specifically, the x86 architecture. This architecture is compatible with the major OS families – Windows, Linux, OS X. This provides an incredible amount of compatibility with software, even allowing the use of decade-old programs – which a few companies are still reliant on. The large amount of compatibility with software is necessary as most consumers or businesses have software which they rely on (like Adobe PhotoShop, Microsoft Word etc.) and they would be extremely hesitant to buy something which does not allow them to use the programs they have been using for years.

All in all, the architecture of a chip should be decided by its purpose. A smartphone chip should use the ARM architecture (which is based off RISC) and a desktop computer chip should have an x86 architecture – which is based off CISC.

[Sources: <http://www.hitequest.com/Kiss/risc_cisc.htm>, <http://en.wikipedia.org/wiki/X86>, <http://en.wikipedia.org/wiki/ARM_architecture>]