Computer Architecture Assignment

Husnain Ahmed

21308666

2021

Contents

[Introduction 1](#_Toc87776421)

[Logisim CPU Simulation 2](#_Toc87776422)

[Testing the CPU 2](#_Toc87776423)

[Data loaded 2](#_Toc87776424)

[End of Program 2](#_Toc87776425)

[Amendments to the CPU 3](#_Toc87776426)

[Implementing a reset button 3](#_Toc87776427)

[Adding the halt instruction 4](#_Toc87776428)

[Adding functionality to the ALU 4](#_Toc87776429)

[Adding functionality for branch instructions 8](#_Toc87776430)

[Incorporating Input and Output devices 10](#_Toc87776431)

[MIPS Assembly Language Programming 17](#_Toc87776432)

[Testing the existing program 17](#_Toc87776433)

[Amendments to the program 17](#_Toc87776434)

[Improvements to messages output 17](#_Toc87776435)

[Using a subroutine 18](#_Toc87776436)

[Adding functionality 22](#_Toc87776437)

[Adding a loop 24](#_Toc87776438)

[Using an array 25](#_Toc87776439)

[Discussion 31](#_Toc87776440)

[The Logisim CPU 31](#_Toc87776441)

[MIPS Assembly Language compared to the Logisim CPU 31](#_Toc87776442)

[Links to programming in a high-level language 32](#_Toc87776443)

[Conclusion 33](#_Toc87776444)

[References 34](#_Toc87776445)

[Appendix A: The Logisim Code 35](#_Toc87776446)

[Appendix B: The MIPS Code 43](#_Toc87776447)

[Appendix C: Completed Marking Grid 46](#_Toc87776448)

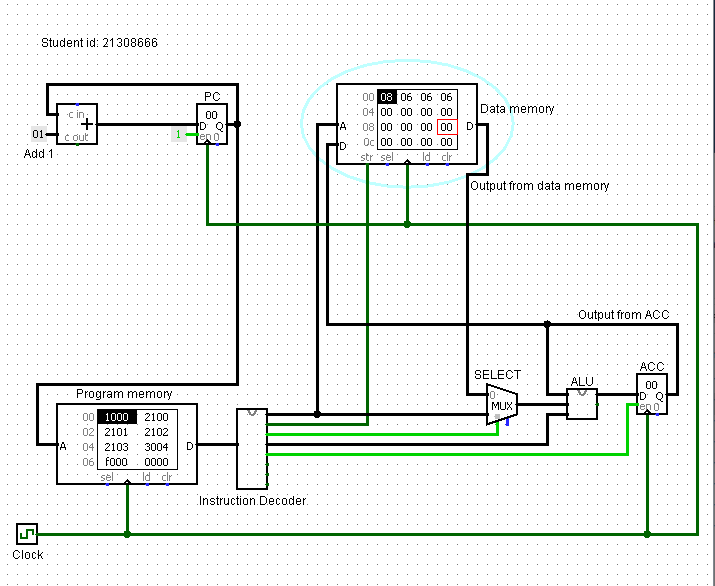
# Introduction

Logisim is a computer software used to design and simulate logic circuits used to teach the basics of computational logic. In this report I will be developing a simulation of an extremely basic CPU using Logisim. The simulation requires a reset button that sets the program back to its initial state, a functional halt instruction to stop the program, a functional ALU to simulate all the calculations and arithmetic logic, functional branch instructions to repeat instructions so that they do not have to be re-entered more than once as well as a function to stop repeating the instructions under certain conditions and working input/output devices that can be used and/or seen by the user. MARS (MIPS Assembler and Runtime Simulator) is a software used to run basic assembly language on the MIPS architecture. In this report I will be developing a program that requires me to improve the messages the program outputs to the user so that it is more clear and easily understandable, use a subroutine to repeat code, enhance the functionality of the program to multiply user inputted numbers as well as add them, incorporate a loop to repeat code and use an array to store the digits that have been entered by the user and output them back to the user for them to see.

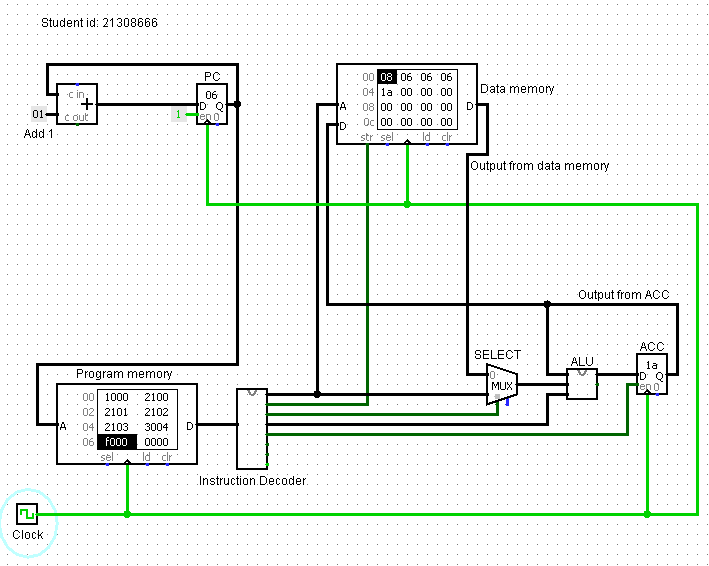
# Logisim CPU Simulation

## Testing the CPU

### Data loaded



### End of Program



Total of digits in decimal is … 8 + 6 + 6 + 6 = 26

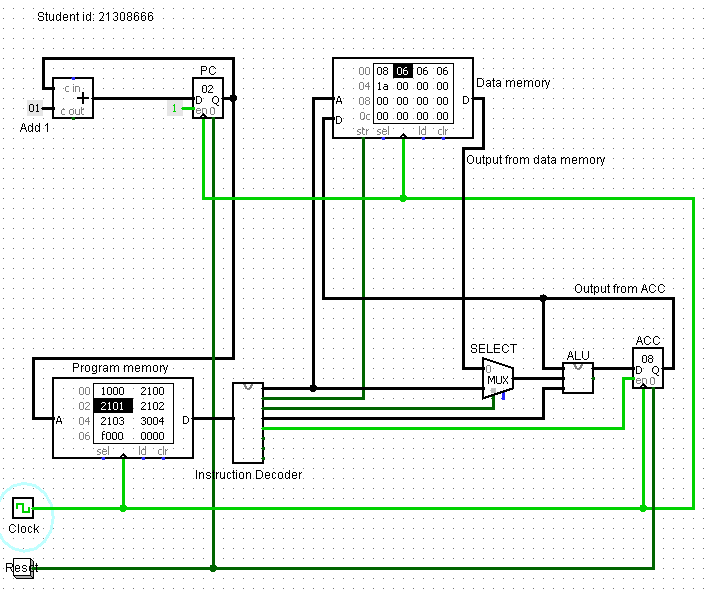
In hexadecimal is … 1a

This task is complete.

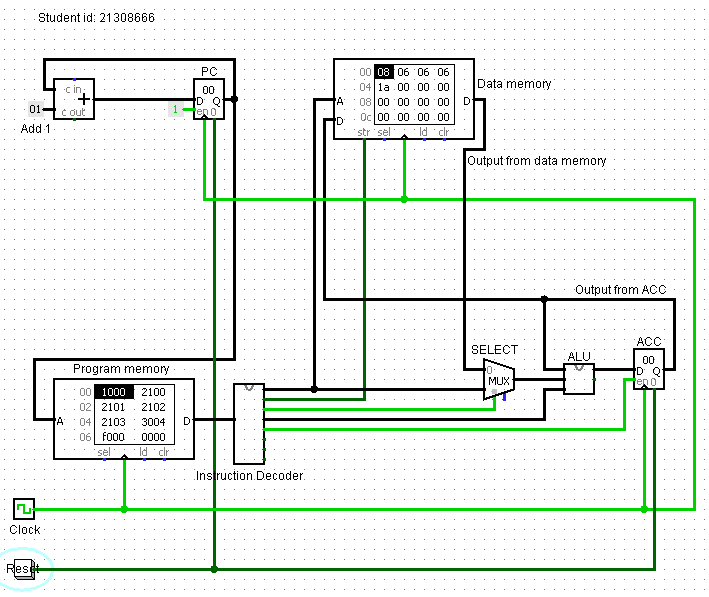
## Amendments to the CPU

### Implementing a reset button

#### Data loaded

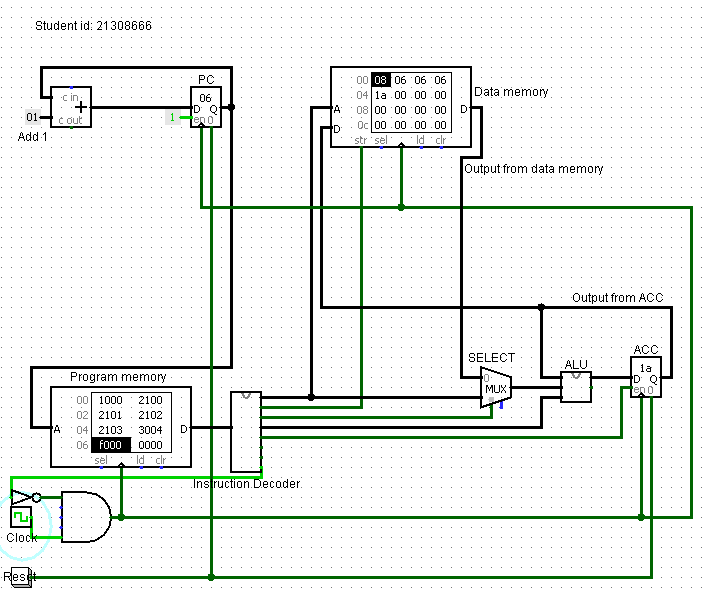


#### After reset



The reset task is complete.

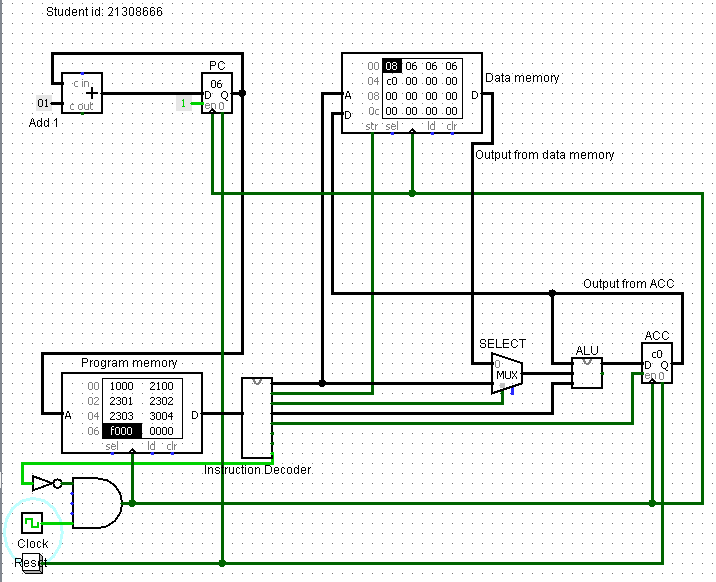
### Adding the halt instruction



The halt task is complete.

### Adding functionality to the ALU

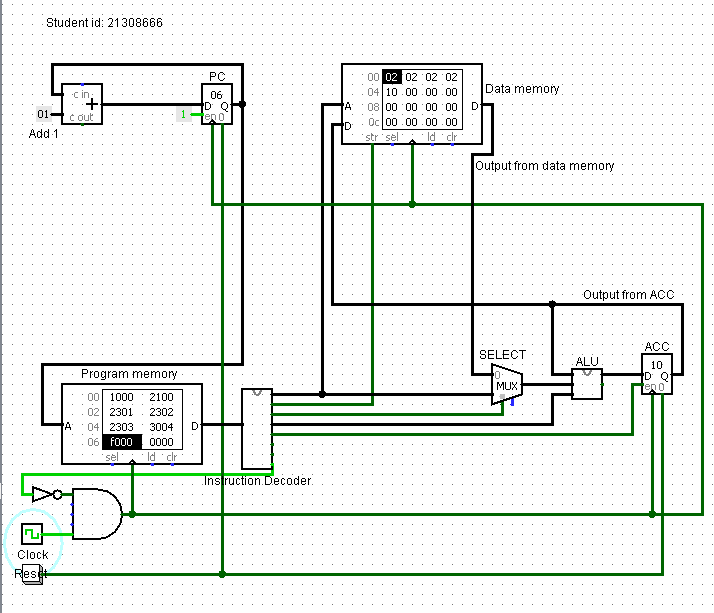
#### Multiplication



Multiplying the digits in decimal gives … 1728

In hexadecimal this is … 6c0

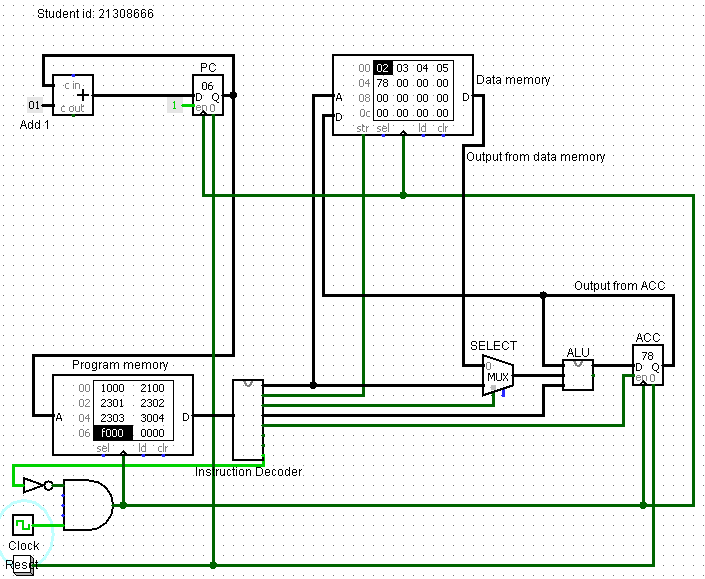
The result in the data memory shows that ... the number that is above 255 cannot be shown by an 8-bit CPU so the final 2 numbers of the hexadecimal number 6c0 are shown on the accumulator as just c0.



Multiplying the digits in decimal gives … 16

In hexadecimal this is … 10

The result in the data memory shows that ... my calculation was correct, and the accumulator holds the hexadecimal value of 10.



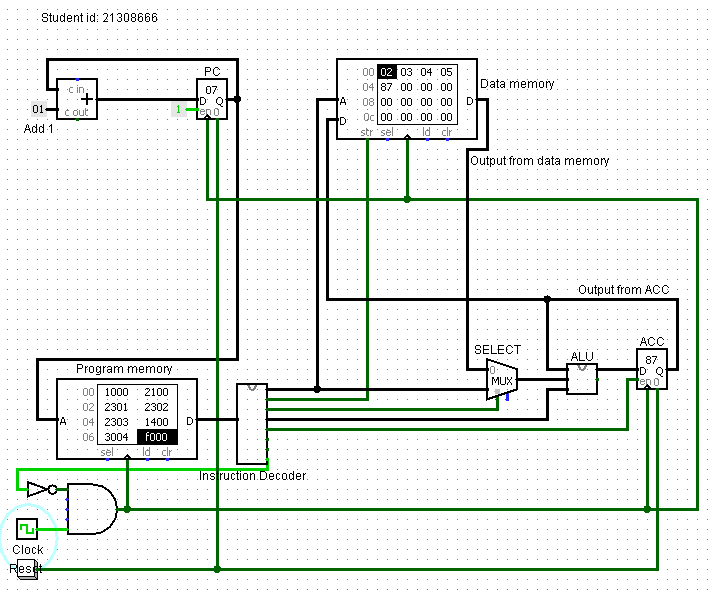
Multiplying the digits in decimal gives … 120

In hexadecimal this is … 78

The result in the data memory shows that ... my calculation was correct, and the accumulator holds the hexadecimal value of 78.

The multiplication task is complete.

#### Inversion



The hex value of the result before inversion is … 78

The binary equivalent of this is … 01111000

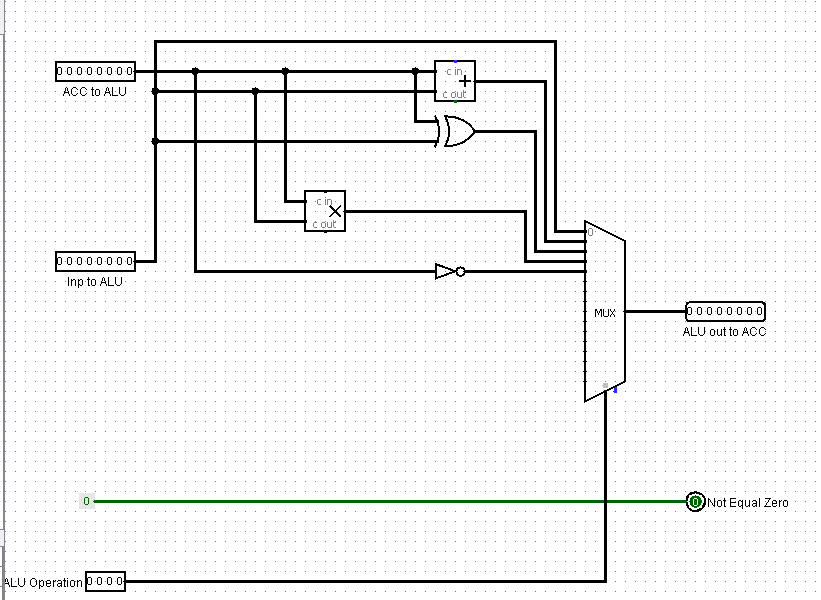
After inversion this is … 10000111

When converted back to hex, this is … 87

The result in the data memory shows that … the hexadecimal number 78 has been inverted into the hexadecimal number 87

The inversion task is complete.

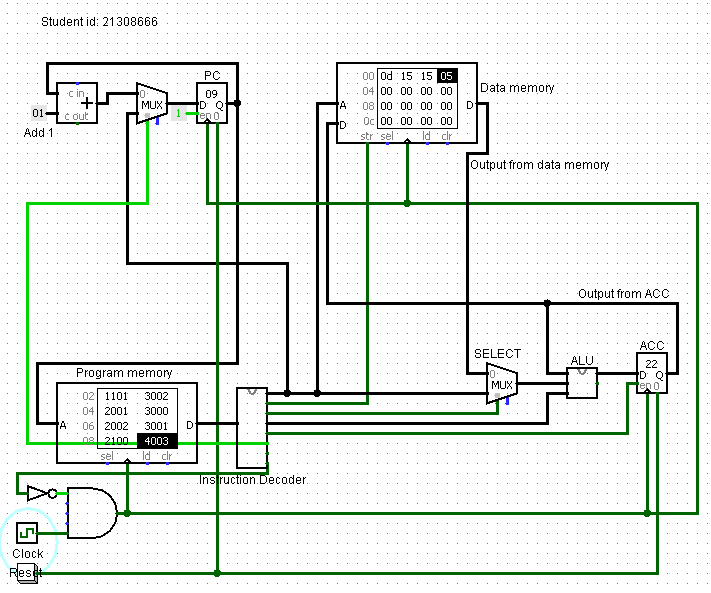
#### Final ALU



The ALU amendment tasks are complete.

### Adding functionality for branch instructions

#### Branch Always

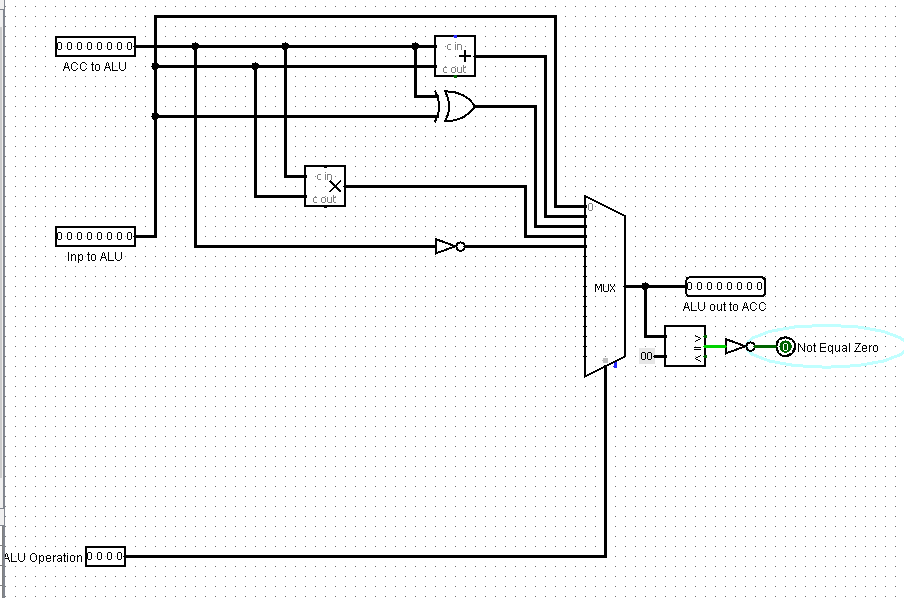


Each number is the sum of the previous 2 numbers in the data memory. The hexadecimal number 22 is equal to the denary number 34 which is the 8th number in the Fibonacci sequence.

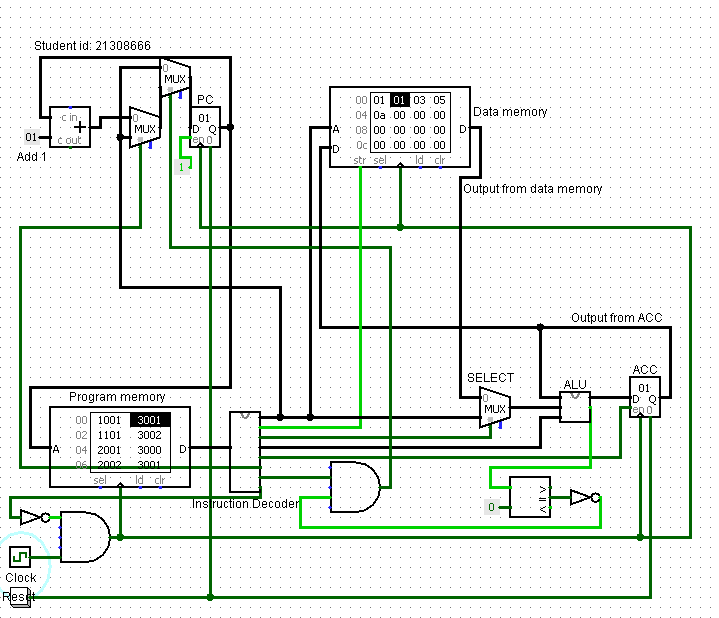
For some reason it is the 8th number instead of the 6th in the sequence which would be 0d (hex), I believe this is due to the values in my data memory not starting from 1 as a normal Fibonacci sequence does.

The branch always task is complete.

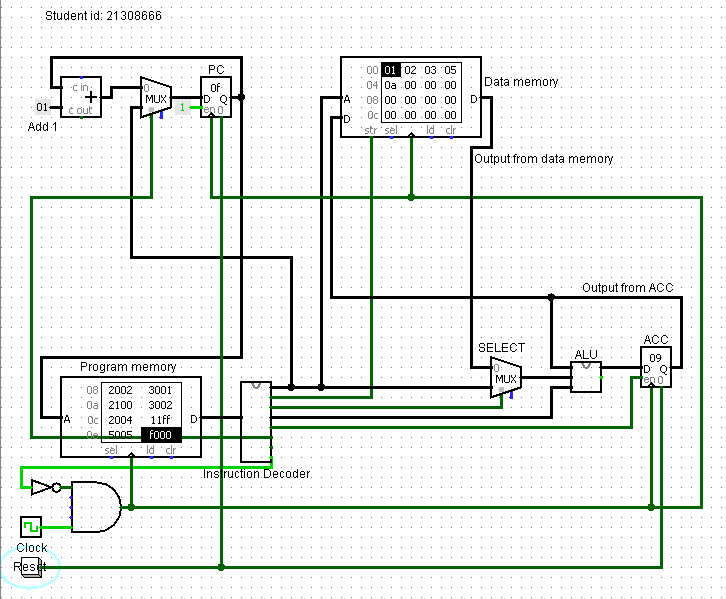
#### Branch Conditionally



The ALU amendment to output 1 when the ACC is not zero is partially complete.



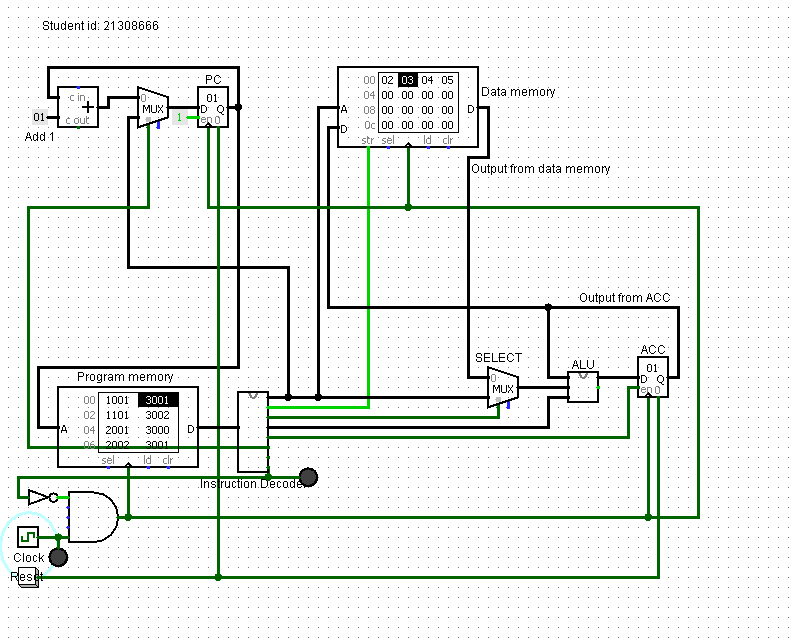
This is my attempt at the instruction to set the program counter with the value of the address when branch always is set or (branch NE zero AND Not Equal Zero) are set. It does not work so I will be going back to a previous circuit that I have saved.

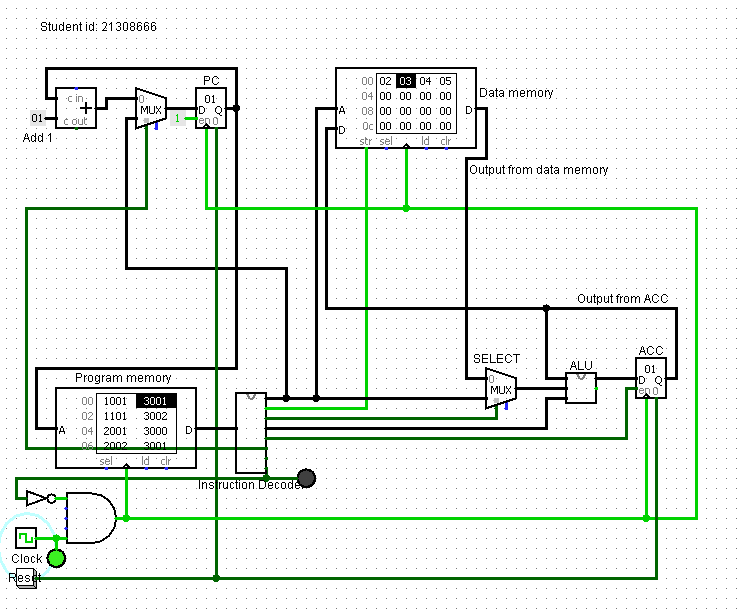


The result in the data memory is the hexadecimal number 09 which is equal to the denary number 9, this is not a Fibonacci number. The program should have stopped when the number in the counter was 0 but it messed up for some reason, maybe because the previous task was incomplete.

The branch conditionally task is partially complete.

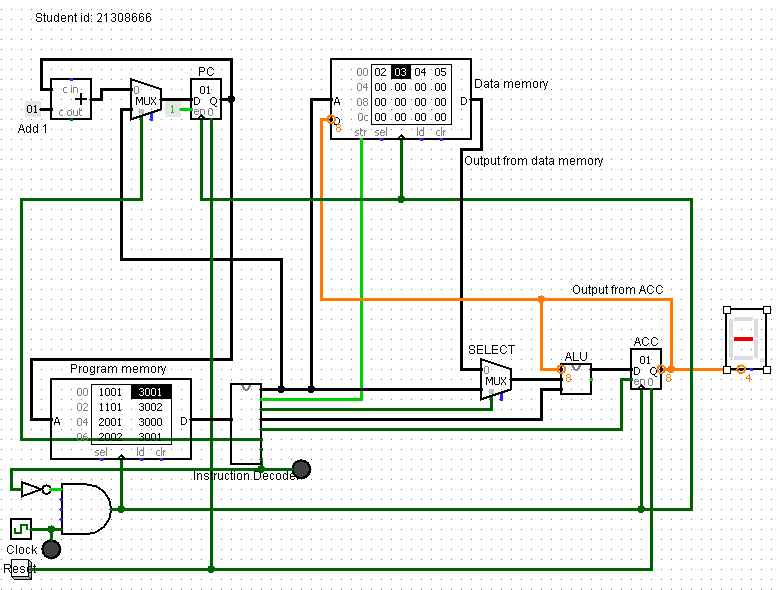
### Incorporating Input and Output devices



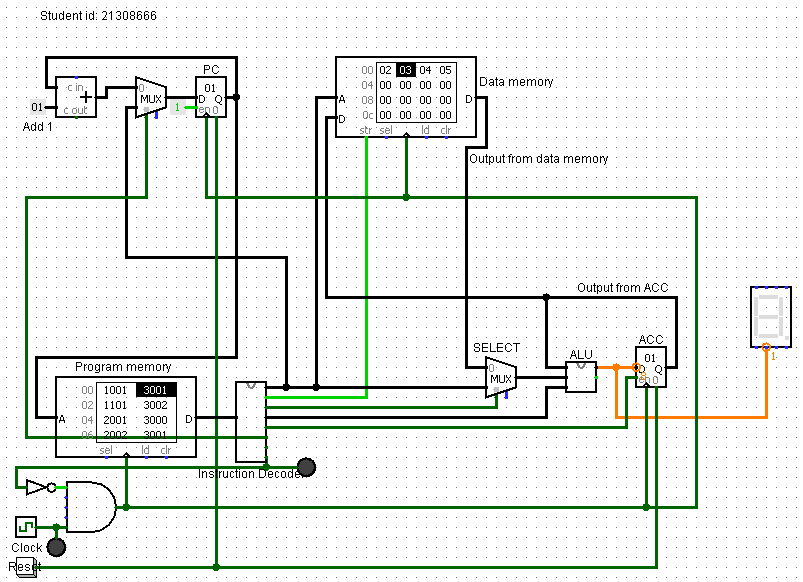


I have added 2 LED lights as outputs for this task, one is a green LED that turns on whenever the clock is on as it is connected to the same wire leading from the clock to an and gate. The other is a red LED that is connected to the 8th pin of the instruction decoder that sends halt commands, seeing the red light would signal that the program has halted.

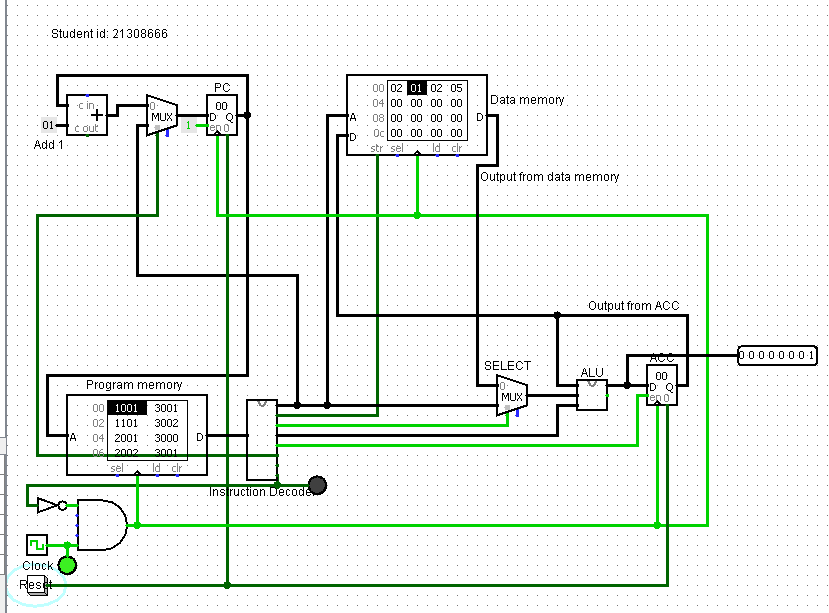
Next, I will attempt to show an output of numbers from either the ALU or the Accumulator.

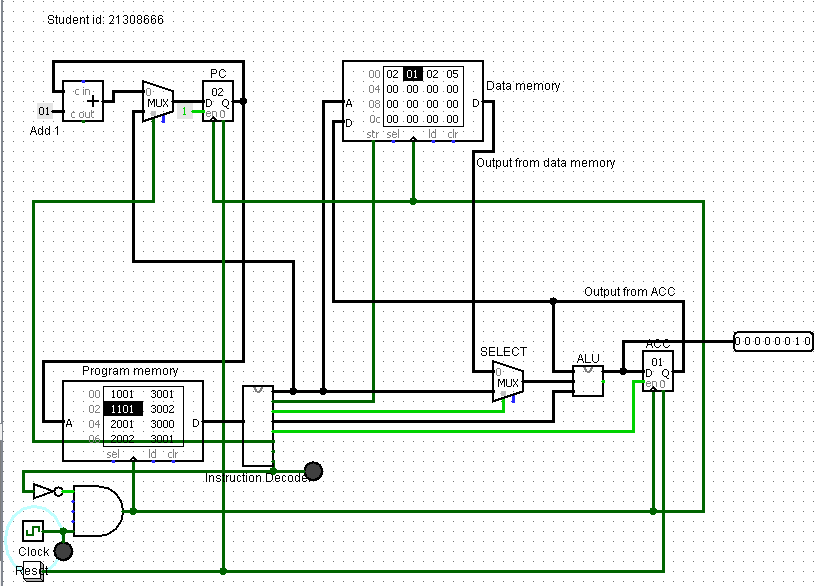


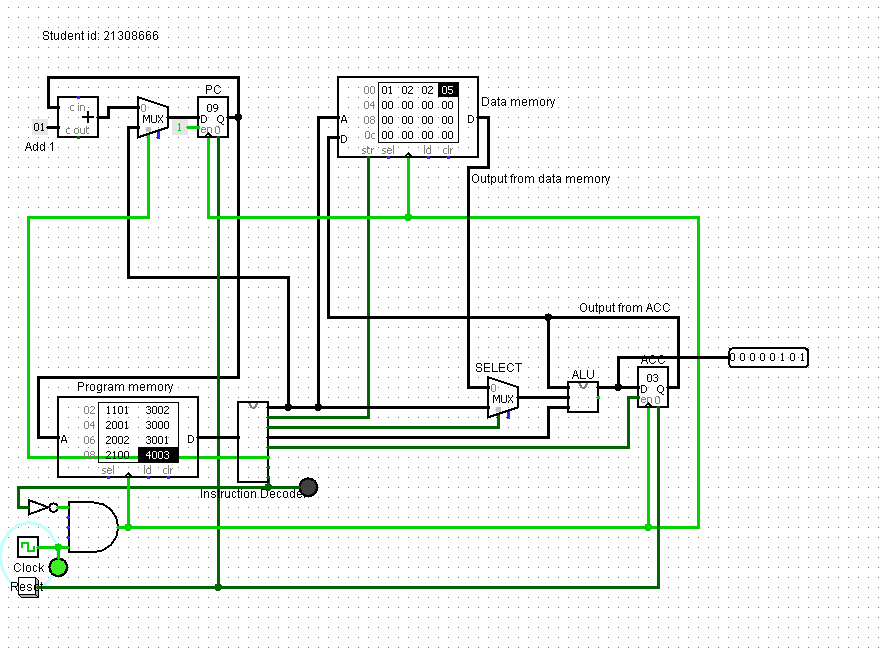
I first decided to test a hex display with the accumulator as it shows a hexadecimal value, but it does not work as the hex display is a 4-bit device, next I will try to connect a 7-segment display to the ALU.



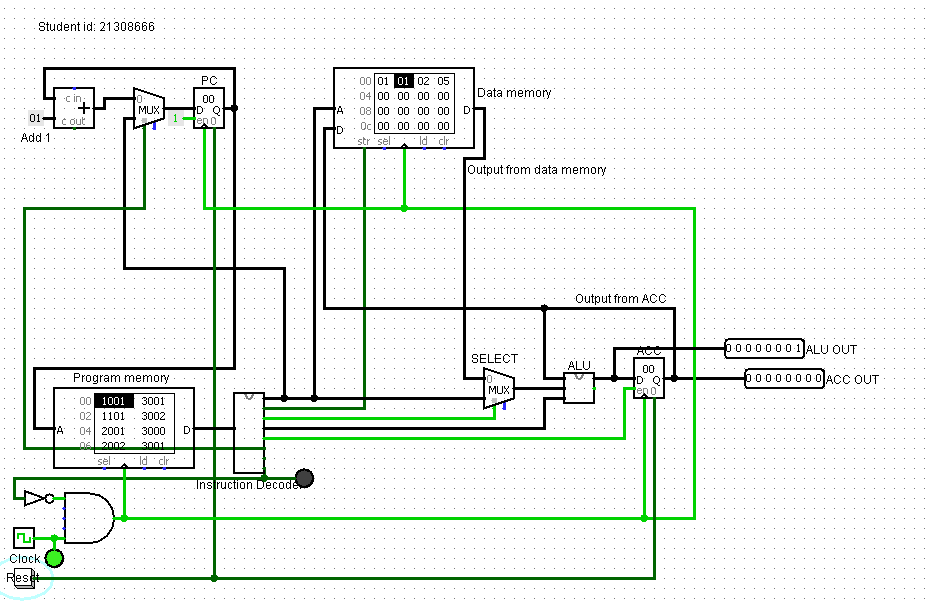
The same error occurs, I will instead try to connect a regular output to the ALU.

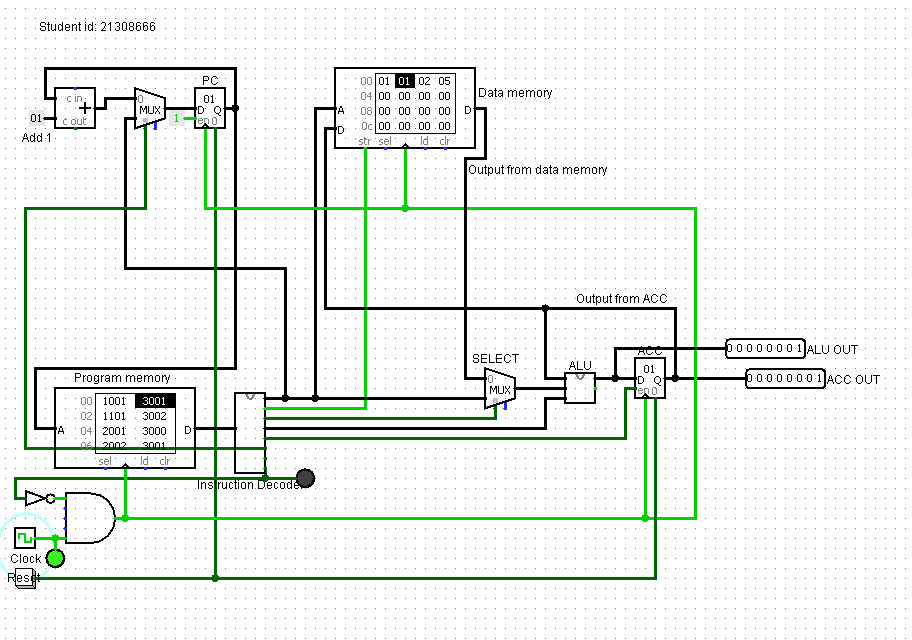


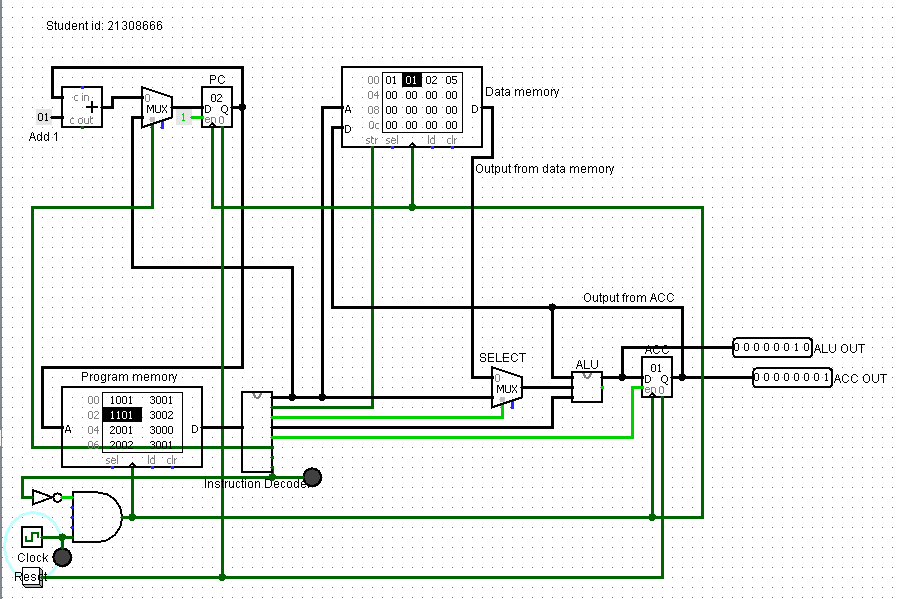


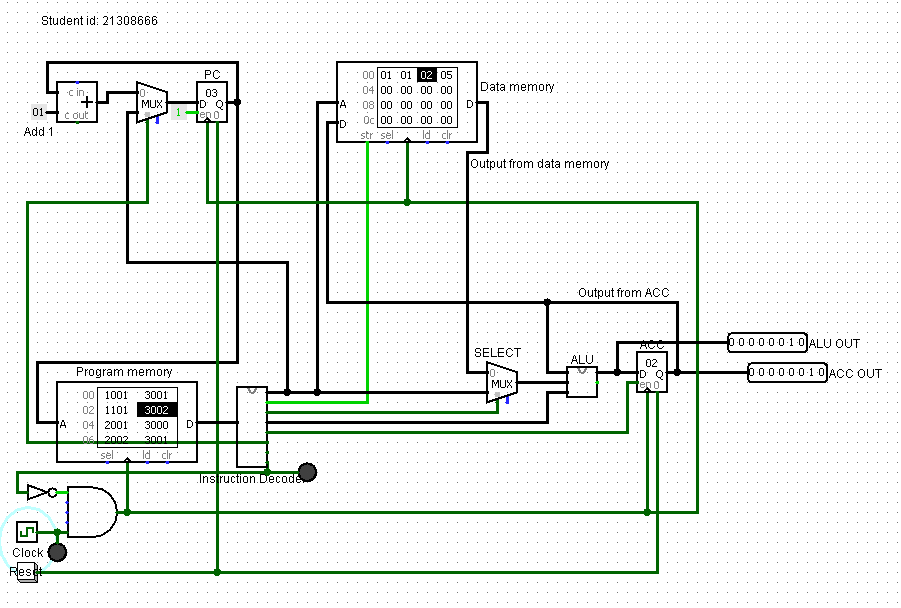


This output works and changes with each tick of the program. I will next try the same thing with the ACC.

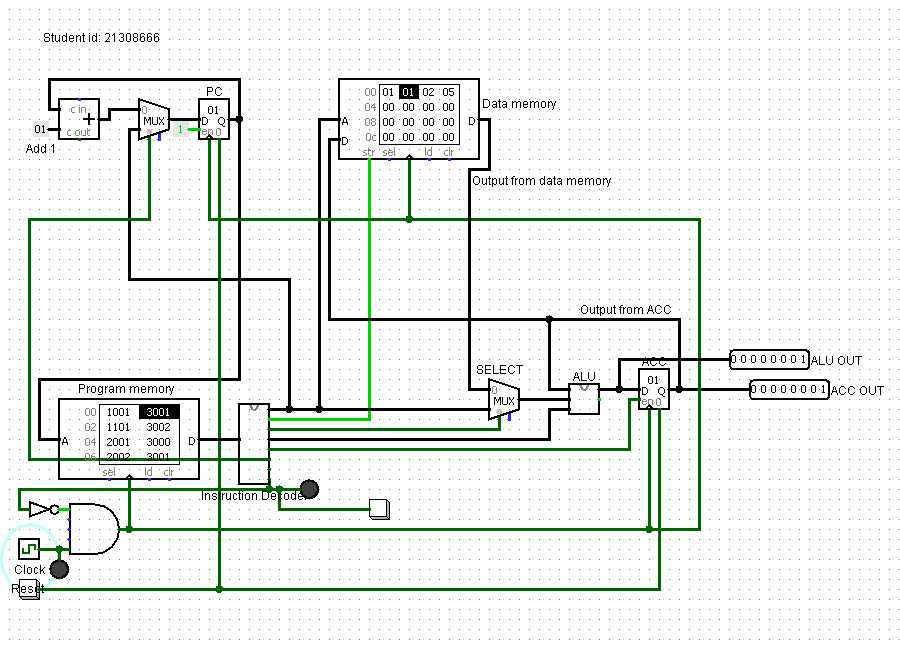




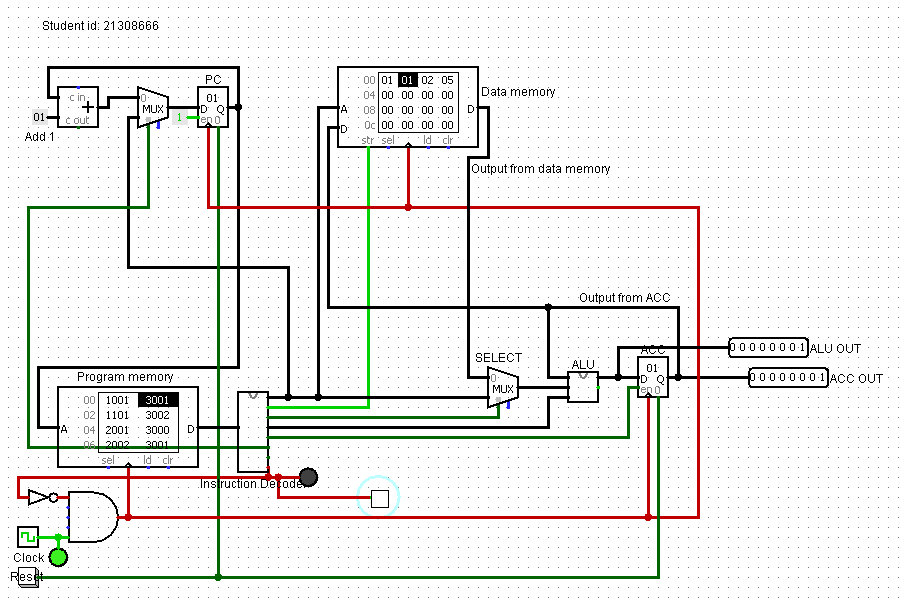




The above images show the visible output pins from the ALU and ACC and show their values.



I have added a button connected to the halt wire as an input, in theory this should cause the program to halt.

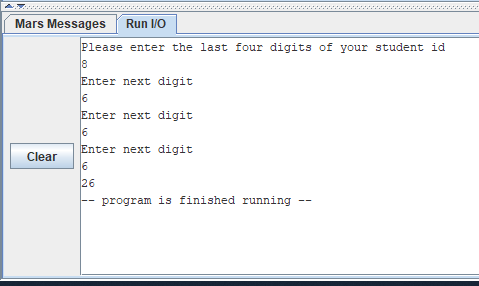


Instead of causing the program to halt it crashes it so I shall remove this input.

The task relating to input and output devices is complete.

# MIPS Assembly Language Programming

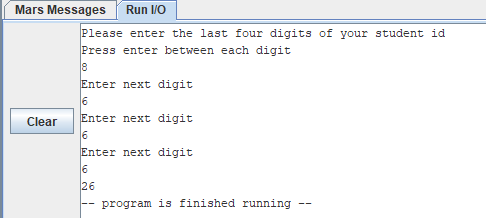
## Testing the existing program

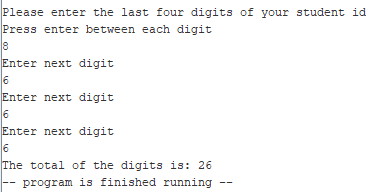


This task is complete.

## Amendments to the program

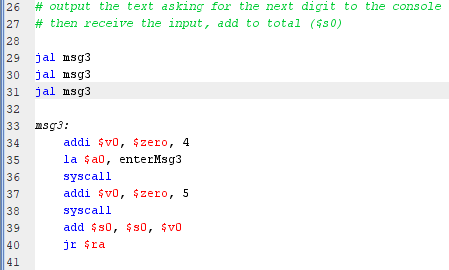
### Improvements to messages output

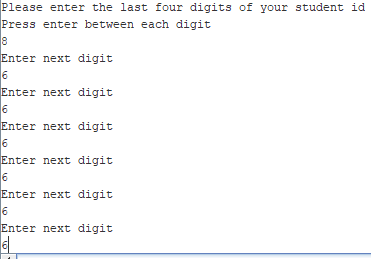




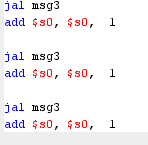
The improved messages task is complete.

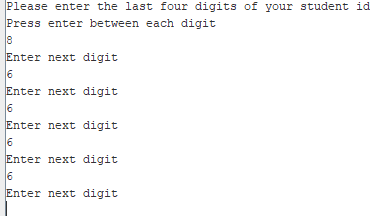
### Using a subroutine



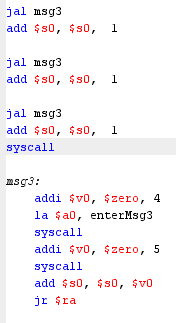


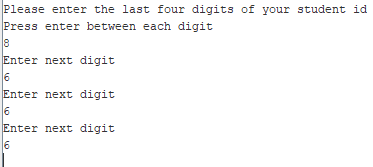
This does not work, the code that was repeated was placed into a sub routine and called back 3 times as It was previously repeated 3 times in the code, to fix this I will clear the registers after each jal call.



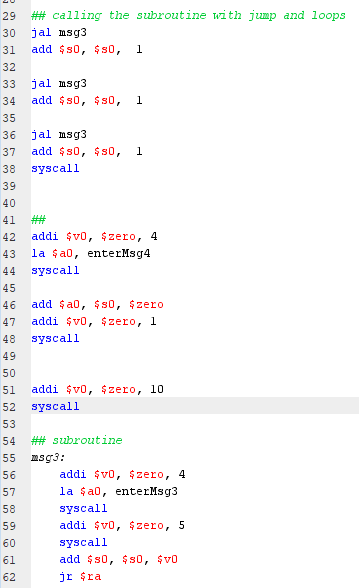


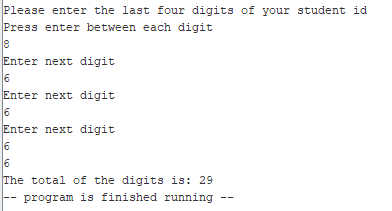
The same error occurs so clearing the registers was not the right option. I will add a syscall command after the last jal.



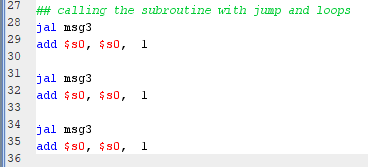


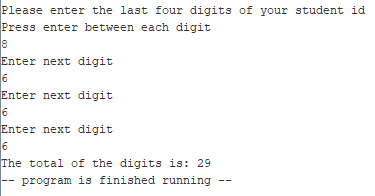
Now it does not keep asking for more digits anymore, I will now move my subroutine to the bottom of the document to see if this is a formatting error.





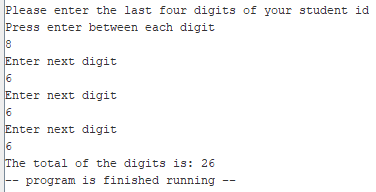
The subroutine now works but it needs an extra value after the 4th input, I believe this is because of my syscall so I will remove it.





The error now is that the total score is wrong I will remove the code after each jal that clears registers to fix this.

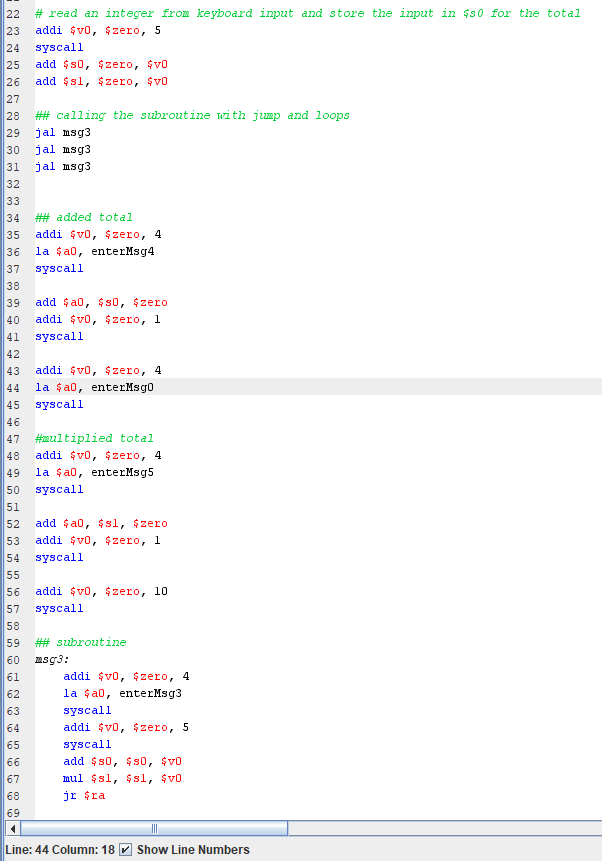


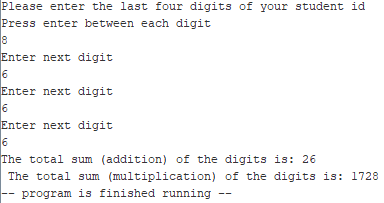


The program now works.

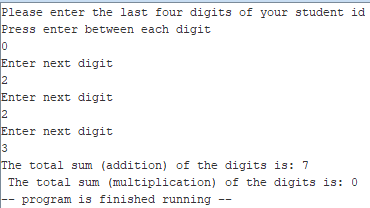
The subroutine task is complete.

### Adding functionality





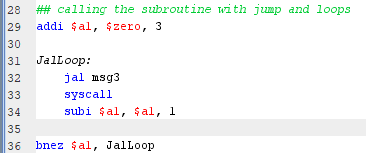
This program works fine now.



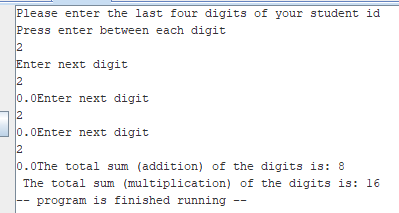
This is a test showing the input value being 0 so that the end result is also 0

The multiplication functionality task is complete.

### Adding a loop

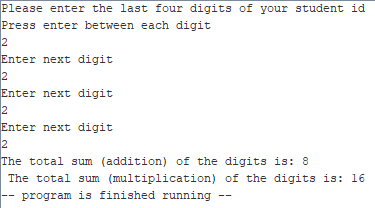


I am using a1 to store the counter for the number of times I want the loop to be iterated.



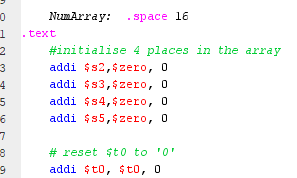
The value 0.0 is outputted before the messages, I think this is because of the syscall in the loop, I will fix it.



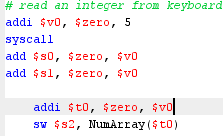


The loop task is complete.

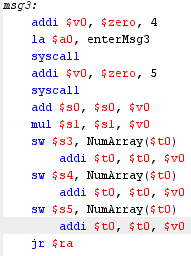
### Using an array



I have initialized an array with 4 slots and set the value of $t0 to 0. (Peralta, 2021)



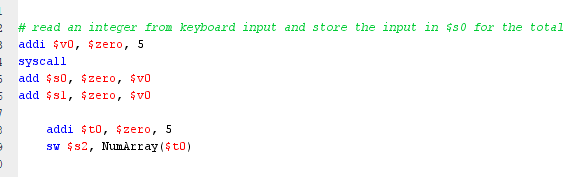
I am adding the entered value to the temporary register $t0 then storing it into the register $s2.

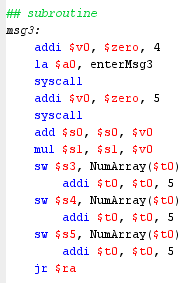


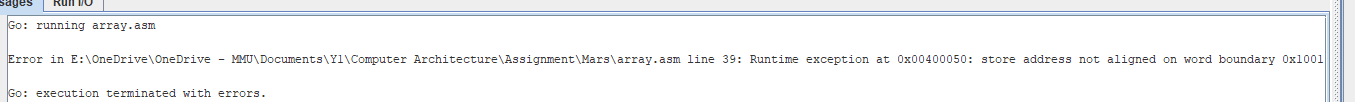
I am storing the values of each input into the registers.



I will change $v0 to 5

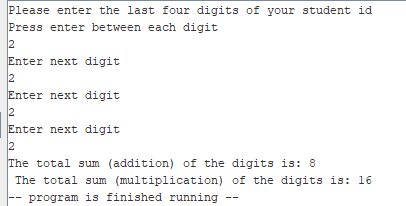




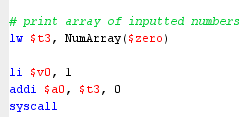


The store address is not aligned on a word boundary, according to stack overflow this can be fixed by changing the store word command sw to the store byte command sb.

(error: "store address not aligned on word boundary", 2021)



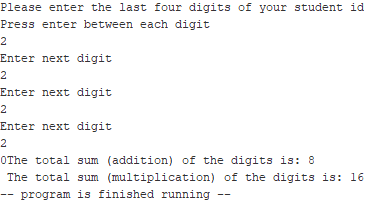
The program appears to run so I will now output the array.



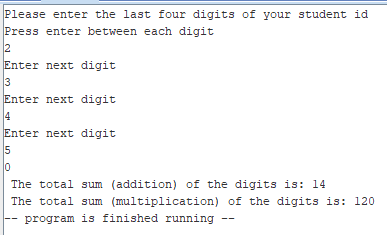


The error says that the fetch address is not stored on the word boundary, I will change the load word lw command to a load byte lb one.

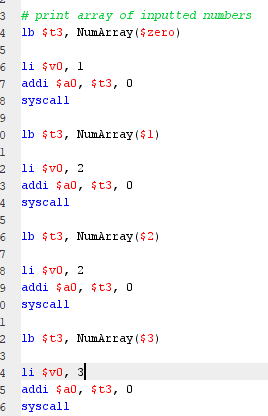


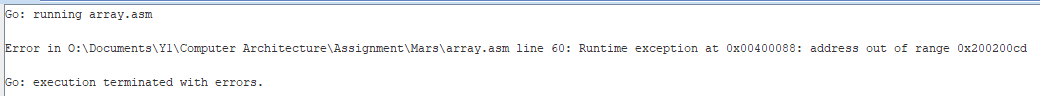


This needs a gap, also the wrong value was outputted so I need to fix this.

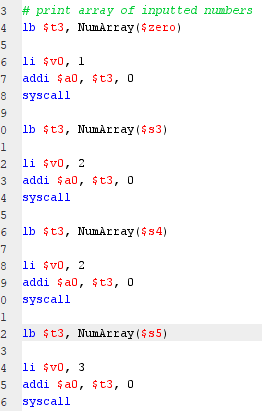


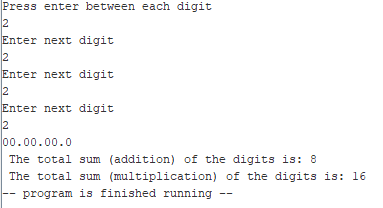
I now need to make it output all the values of the array.





The address that I passed to the array is out of range.





The value is now being outputted to the user, but the array is not holding the correct values. It should hold the value that were previously inputted to the user, I will need to fix this, so those inputs are appended to the array.

The array task is not complete.

# Discussion

## The Logisim CPU

The Logisim CPU runs a program using the data file and the program file that are stored in the program memory and data memory ROM’s. The data.dat file contains 4 numbers that are used as starting values for the different programs to run through, these numbers undergo many calculations such as being added together, multiplied with each other, and inverted.

The circuit has a clock signal which when turned on runs 1 tick of the program.

The Program Counter contains the address of the next instruction that the CPU will fetch, decode and execute.

The program.dat file was the first program that I ran, I had to change the values it contained within itself accordingly depending on which task I was doing, the instruction 2100 2101 2102 2103 added 4 numbers together as pin 1 on the ALU is used for calculations pertaining to addition. Pin 1 is selected by the multiplexer and chooses which calculation to run. The added value is shown on the accumulator.

The next change I had to make to this was to allow the CPU to multiply the 4 values together instead of adding them for this the numbers 2100 2101 2102 2103 had to be changed to 2100 2301 2302 2303, this is because pin 3 of the ALU is responsible for all calculations pertaining to multiplication. The first value 2100 just selects the first number so that remains unchanged, the next three numbers have been adjusted from 2101 2102 2103 to 2301 2302 2303 as the number 3 is for multiplication, the instruction has been changed from adding the numbers together to multiplying them instead.

The next task was to invert the result of the multiplication, this was done by adding a not gate from the input of the ALU into pin 4 of the ALU, because of this pin 4 is responsible for all inversion commands. To invert the value, I had to add the number 1400 onto the sequence of 2100 2301 2302 2303 so it changed to 2100 2301 2302 2303 1400. The number 1 tells the CPU to use the given number as the immediate value and pass it on while the number 4 tells the program to select pin 4 of the ALU which is responsible for inversion, so the program takes the already multiplied value and inverts it.

It is then followed by the values 3004 and f000. The value 3004 copies the current value that is stored inside of the accumulator to the area in the data memory given by the address which in this case is 04. Whereas f000 is the command to call the end of the program, so if this runs the program will not continue anymore.

The final values of the program.dat file was 1000 2100 2301 2302 2303 1400 3004 f000.

Another program that the CPU runs is a Fibonacci loop, this program calculates values of a sequence by adding the 2 previous values together. For example, 1,2,3,5,8,13,21,34.

## MIPS Assembly Language compared to the Logisim CPU

Some of the similarities that I have noticed between writing and testing a program on the Logisim CPU and in MIPS Assembly language using the MARS simulator are that they both have some semblance of a Graphical User interface (GUI), though Logisim’s GUI is much more fleshed out and easy to use.

Another similarity between Logisim and MARS that I have found is that they both do some level of logic, Logisim does this much more as it is geared around simulations and logic-based calculations such as the ALU (Arithmetic Logic Unit) in a CPU (Central Processing Unit) that does mathematical logic and calculations. Logisim also does logic based on whatever data is in the instruction set in the instruction decoder. MARS does this through mathematical equations also such as the addi and subbi commands, that add and subtract from a stored value, and storing / loading values from different registers.

Both of the programs (Logisim and MARS) are oriented towards the design, development, and creation of computer programs. Logisim’s programs are more oriented towards computer / electronical circuits and. According to google MARS programs are more oriented towards hardware device drivers, embedded systems and computer games requiring direct hardware access.

Some of the differences that I have noticed between writing and testing a program on the Logisim CPU and in MIPS Assembly language using the MARS simulator are that MIPS does not have a very extensive GUI (Graphical User Interface) and is mainly CLI (command line interface) based, this means that there is not many icons and visuals and the program itself looks very plain and barebones, this leads to there being less distractions on the program.

Another difference that I have noticed is that Logisim is more oriented towards circuits and logic whilst MARS is more oriented towards programs due to the fact that it requires you to actually code in assembly language, Logisim is also a lot more user friendly due to its GUI and the drag and drop interface.

The errors in MARS are easily recognizable and fixable as the program gives you error messages and error codes that you can google and find fixes for on forums but errors on Logisim do not have these and must be fixed by the user alone, this is good as they can be a valuable learning tool, but it is bad as you can get permanently stuck.

## Links to programming in a high-level language

Low level Programming languages are things like binary, machine code and assembly language that are almost unintelligible to humans.

High level programming languages are much closer to normal human languages and as such are easier to learn, they also come with a lot of features that allow for very specific common tasks such as creating loops and arrays to be done very easily.

Developing a program in a high-level programming language, such as Java for example, is much easier than doing so in a low-level programming language such as MIPS assembly language. The rules are set out much more concretely and there are strict guidelines, so it is easier to learn, there is also a larger instruction set in high-level languages like java so there are a lot more options for things such as loops and arrays, due to this we do not need to reuse the limited code we have available to us to create and initialise every single part of an array including the number of spaces inside of it. Low level programming languages have a smaller instruction set on the other hand so there is a lot less options for the developer to use but this means that it can be a lot simpler in most cases. MIPS also does not require a lot of coding conventions that are quite common in higher level programming languages such as java and c# like semi colons (;) after each line of code or each command.

Low level programming languages are also a lot more flexible than high level ones as they allow for many things that are impossible in many high-level ones such as changing the screens refresh rate, they are also fast and memory efficient. However, they also require the user to memorise and remember various kinds of obscure information such as the name of certain registers like $a0 which is extremely far removed from normal human language and could prove hard to remember.

There is also a lot more learning resources for high level programming languages compared to low level programming languages. This is because they are more in demand and as such there is a larger market for people who need to learn them.

The difference between a high-level language such as java and a program like Logisim is even more apparent as Logisim does not use any actual code, it instead uses icons and wiring to create circuits, the closes thing that Logisim has to a programming language is an instruction set that the user can create.

# Conclusion

In conclusion, both Logisim and MARS are exceptionally good educational programs to teach the basics of computational logic, electronic circuits, and assembly language. These skills can form the basis of your knowledge and help you to understand the underlying principles behind how code works, so you will understand both how to code in high level languages as well as how the computer processes your code and turns it into actual commands and instructions that it can follow for you. I now understand how values and registers work thanks to the work I did using MARS and understand a lot of the underlying principles behind how high-level languages store values into variables and that they retrieve them from the address (where the variable is initialised) whenever you call the variables. Thanks to the work in Logisim I now understand how an extremely basic CPU works and how the flow of data can affect many different calculations and the logic behind how these calculations work. I can use this to help me understand how computers process the code and how software can interact with hardware components to acquire different desired outcomes. I believe that the two programs, MARS and Logisim, are almost perfectly suited to the task of teaching people who are new to logic and circuits, how to understand and use them.

# References

Peralta, A., 2021. *MIPS Tutorial 27 Arrays*. [online] Youtube.com. Available at: <https://www.youtube.com/watch?v=BfHcogmKM20> [Accessed 14 December 2021].

Stack Overflow. 2021. *error: "store address not aligned on word boundary"*. [online] Available at: <https://stackoverflow.com/questions/9830892/error-store-address-not-aligned-on-word-boundary> [Accessed 14 December 2021].

# Appendix A: The Logisim Code

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<project source="2.7.1" version="1.0">

This file is intended to be loaded by Logisim (<http://www.cburch.com/logisim/>).

<lib desc="#Wiring" name="0">

<tool name="Splitter">

<a name="facing" val="west"/>

<a name="appear" val="center"/>

</tool>

<tool name="Pin">

<a name="facing" val="north"/>

<a name="width" val="24"/>

<a name="tristate" val="false"/>

</tool>

<tool name="Probe">

<a name="radix" val="10signed"/>

</tool>

<tool name="Tunnel">

<a name="facing" val="east"/>

<a name="width" val="24"/>

<a name="label" val="RES"/>

</tool>

<tool name="Pull Resistor">

<a name="facing" val="east"/>

</tool>

<tool name="Clock">

<a name="facing" val="north"/>

</tool>

<tool name="Constant">

<a name="width" val="24"/>

<a name="value" val="0x0"/>

</tool>

</lib>

<lib desc="#Gates" name="1"/>

<lib desc="#Plexers" name="2"/>

<lib desc="#Arithmetic" name="3"/>

<lib desc="#Memory" name="4">

<tool name="ROM">

<a name="contents">addr/data: 8 8

0

</a>

</tool>

</lib>

<lib desc="#I/O" name="5"/>

<lib desc="#Base" name="6">

<tool name="Text Tool">

<a name="text" val=""/>

<a name="font" val="SansSerif plain 12"/>

<a name="halign" val="center"/>

<a name="valign" val="base"/>

</tool>

<tool name="Text">

<a name="text" val="Select"/>

</tool>

</lib>

<main name="Instruction Decoding Unit"/>

<options>

<a name="gateUndefined" val="ignore"/>

<a name="simlimit" val="1000"/>

<a name="simrand" val="0"/>

</options>

<mappings>

<tool lib="6" map="Button2" name="Menu Tool"/>

<tool lib="6" map="Button3" name="Menu Tool"/>

<tool lib="6" map="Ctrl Button1" name="Menu Tool"/>

</mappings>

<toolbar>

<tool lib="6" name="Poke Tool"/>

<tool lib="6" name="Edit Tool"/>

<tool lib="6" name="Text Tool">

<a name="text" val=""/>

<a name="font" val="SansSerif plain 12"/>

<a name="halign" val="center"/>

<a name="valign" val="base"/>

</tool>

<sep/>

<tool lib="0" name="Pin">

<a name="tristate" val="false"/>

</tool>

<tool lib="0" name="Pin">

<a name="facing" val="west"/>

<a name="output" val="true"/>

<a name="labelloc" val="east"/>

</tool>

<tool lib="1" name="NOT Gate"/>

<tool lib="1" name="AND Gate"/>

<tool lib="1" name="OR Gate"/>

</toolbar>

<circuit name="CPU main">

<a name="circuit" val="CPU main"/>

<a name="clabel" val=""/>

<a name="clabelup" val="east"/>

<a name="clabelfont" val="SansSerif plain 12"/>

<wire from="(650,570)" to="(700,570)"/>

<wire from="(520,440)" to="(570,440)"/>

<wire from="(20,530)" to="(270,530)"/>

<wire from="(240,160)" to="(240,420)"/>

<wire from="(220,630)" to="(660,630)"/>

<wire from="(110,150)" to="(110,160)"/>

<wire from="(40,630)" to="(220,630)"/>

<wire from="(500,460)" to="(500,470)"/>

<wire from="(550,360)" to="(550,430)"/>

<wire from="(150,180)" to="(150,260)"/>

<wire from="(210,180)" to="(210,260)"/>

<wire from="(130,170)" to="(130,320)"/>

<wire from="(550,450)" to="(550,480)"/>

<wire from="(620,400)" to="(620,430)"/>

<wire from="(270,490)" to="(630,490)"/>

<wire from="(130,320)" to="(290,320)"/>

<wire from="(110,150)" to="(140,150)"/>

<wire from="(600,430)" to="(620,430)"/>

<wire from="(620,430)" to="(640,430)"/>

<wire from="(320,160)" to="(320,450)"/>

<wire from="(290,450)" to="(320,450)"/>

<wire from="(270,530)" to="(300,530)"/>

<wire from="(240,120)" to="(240,160)"/>

<wire from="(270,480)" to="(550,480)"/>

<wire from="(40,590)" to="(60,590)"/>

<wire from="(30,260)" to="(30,500)"/>

<wire from="(680,430)" to="(750,430)"/>

<wire from="(480,160)" to="(490,160)"/>

<wire from="(190,170)" to="(200,170)"/>

<wire from="(130,170)" to="(140,170)"/>

<wire from="(50,170)" to="(60,170)"/>

<wire from="(50,150)" to="(60,150)"/>

<wire from="(100,160)" to="(110,160)"/>

<wire from="(40,420)" to="(240,420)"/>

<wire from="(700,260)" to="(700,570)"/>

<wire from="(670,430)" to="(680,430)"/>

<wire from="(40,420)" to="(40,480)"/>

<wire from="(370,200)" to="(370,460)"/>

<wire from="(220,180)" to="(220,630)"/>

<wire from="(50,120)" to="(240,120)"/>

<wire from="(290,320)" to="(290,450)"/>

<wire from="(30,260)" to="(150,260)"/>

<wire from="(60,590)" to="(60,600)"/>

<wire from="(680,360)" to="(680,430)"/>

<wire from="(270,520)" to="(270,530)"/>

<wire from="(30,500)" to="(270,500)"/>

<wire from="(320,450)" to="(490,450)"/>

<wire from="(20,530)" to="(20,550)"/>

<wire from="(410,260)" to="(700,260)"/>

<wire from="(270,460)" to="(370,460)"/>

<wire from="(470,210)" to="(470,430)"/>

<wire from="(200,480)" to="(240,480)"/>

<wire from="(620,400)" to="(730,400)"/>

<wire from="(270,470)" to="(500,470)"/>

<wire from="(50,120)" to="(50,150)"/>

<wire from="(170,160)" to="(200,160)"/>

<wire from="(330,360)" to="(550,360)"/>

<wire from="(550,430)" to="(570,430)"/>

<wire from="(550,450)" to="(570,450)"/>

<wire from="(320,160)" to="(340,160)"/>

<wire from="(270,450)" to="(290,450)"/>

<wire from="(470,430)" to="(490,430)"/>

<wire from="(470,210)" to="(490,210)"/>

<wire from="(40,480)" to="(60,480)"/>

<wire from="(130,520)" to="(130,570)"/>

<wire from="(330,180)" to="(330,360)"/>

<wire from="(650,450)" to="(650,570)"/>

<wire from="(330,180)" to="(340,180)"/>

<wire from="(120,570)" to="(130,570)"/>

<wire from="(230,160)" to="(240,160)"/>

<wire from="(490,160)" to="(490,210)"/>

<wire from="(550,360)" to="(680,360)"/>

<wire from="(20,550)" to="(30,550)"/>

<wire from="(60,550)" to="(70,550)"/>

<wire from="(60,590)" to="(70,590)"/>

<wire from="(130,570)" to="(650,570)"/>

<wire from="(410,200)" to="(410,260)"/>

<wire from="(210,260)" to="(410,260)"/>

<wire from="(630,440)" to="(630,490)"/>

<wire from="(660,450)" to="(660,630)"/>

<wire from="(630,440)" to="(640,440)"/>

<comp lib="6" loc="(511,511)" name="Text"/>

<comp lib="0" loc="(750,430)" name="Pin">

<a name="facing" val="west"/>

<a name="output" val="true"/>

<a name="width" val="8"/>

<a name="label" val="ACC OUT"/>

<a name="labelloc" val="east"/>

</comp>

<comp loc="(270,450)" name="Instruction Decoding Unit">

<a name="label" val="Instruction Decoder"/>

<a name="labelloc" val="south"/>

</comp>

<comp lib="6" loc="(127,434)" name="Text">

<a name="text" val="Program memory"/>

</comp>

<comp lib="6" loc="(102,83)" name="Text">

<a name="text" val="Student id: 21308666"/>

</comp>

<comp lib="0" loc="(190,170)" name="Constant"/>

<comp lib="2" loc="(170,160)" name="Multiplexer">

<a name="width" val="8"/>

</comp>

<comp lib="6" loc="(500,415)" name="Text">

<a name="text" val="SELECT"/>

</comp>

<comp lib="4" loc="(200,480)" name="RAM">

<a name="dataWidth" val="16"/>

</comp>

<comp lib="4" loc="(230,160)" name="Register">

<a name="label" val="PC"/>

</comp>

<comp lib="0" loc="(40,590)" name="Clock">

<a name="label" val="Clock"/>

<a name="labelloc" val="south"/>

</comp>

<comp lib="5" loc="(40,630)" name="Button">

<a name="label" val="Reset"/>

</comp>

<comp lib="0" loc="(730,400)" name="Pin">

<a name="facing" val="west"/>

<a name="output" val="true"/>

<a name="width" val="8"/>

<a name="label" val="ALU OUT"/>

<a name="labelloc" val="east"/>

</comp>

<comp lib="1" loc="(120,570)" name="AND Gate"/>

<comp lib="4" loc="(480,160)" name="RAM">

<a name="bus" val="separate"/>

</comp>

<comp lib="3" loc="(100,160)" name="Adder"/>

<comp lib="6" loc="(542,226)" name="Text">

<a name="text" val="Output from data memory"/>

</comp>

<comp lib="6" loc="(42,194)" name="Text">

<a name="text" val="Add 1"/>

</comp>

<comp lib="1" loc="(60,550)" name="NOT Gate"/>

<comp lib="6" loc="(627,355)" name="Text">

<a name="text" val="Output from ACC"/>

</comp>

<comp lib="2" loc="(520,440)" name="Multiplexer">

<a name="width" val="8"/>

</comp>

<comp lib="4" loc="(670,430)" name="Register">

<a name="label" val="ACC"/>

</comp>

<comp lib="5" loc="(60,600)" name="LED">

<a name="facing" val="north"/>

<a name="color" val="#3cf022"/>

</comp>

<comp lib="5" loc="(300,530)" name="LED"/>

<comp lib="6" loc="(520,149)" name="Text">

<a name="text" val="Data memory"/>

</comp>

<comp lib="6" loc="(525,444)" name="Text"/>

<comp lib="0" loc="(50,170)" name="Constant">

<a name="width" val="8"/>

</comp>

<comp loc="(600,430)" name="ALU">

<a name="label" val="ALU"/>

</comp>

</circuit>

<circuit name="ALU">

<a name="circuit" val="ALU"/>

<a name="clabel" val=""/>

<a name="clabelup" val="east"/>

<a name="clabelfont" val="SansSerif plain 12"/>

<wire from="(250,100)" to="(250,230)"/>

<wire from="(470,140)" to="(530,140)"/>

<wire from="(530,260)" to="(580,260)"/>

<wire from="(280,80)" to="(280,210)"/>

<wire from="(130,80)" to="(190,80)"/>

<wire from="(340,220)" to="(520,220)"/>

<wire from="(190,80)" to="(190,280)"/>

<wire from="(520,270)" to="(580,270)"/>

<wire from="(250,230)" to="(300,230)"/>

<wire from="(250,100)" to="(430,100)"/>

<wire from="(460,280)" to="(580,280)"/>

<wire from="(540,250)" to="(580,250)"/>

<wire from="(640,320)" to="(680,320)"/>

<wire from="(190,280)" to="(430,280)"/>

<wire from="(150,100)" to="(250,100)"/>

<wire from="(120,590)" to="(600,590)"/>

<wire from="(190,80)" to="(280,80)"/>

<wire from="(620,320)" to="(640,320)"/>

<wire from="(640,370)" to="(660,370)"/>

<wire from="(700,380)" to="(720,380)"/>

<wire from="(750,380)" to="(770,380)"/>

<wire from="(540,90)" to="(540,250)"/>

<wire from="(280,210)" to="(300,210)"/>

<wire from="(550,240)" to="(580,240)"/>

<wire from="(410,80)" to="(430,80)"/>

<wire from="(410,130)" to="(430,130)"/>

<wire from="(150,150)" to="(430,150)"/>

<wire from="(130,270)" to="(150,270)"/>

<wire from="(150,50)" to="(150,100)"/>

<wire from="(530,140)" to="(530,260)"/>

<wire from="(150,100)" to="(150,150)"/>

<wire from="(150,50)" to="(550,50)"/>

<wire from="(410,80)" to="(410,130)"/>

<wire from="(520,220)" to="(520,270)"/>

<wire from="(550,50)" to="(550,240)"/>

<wire from="(600,400)" to="(600,590)"/>

<wire from="(150,150)" to="(150,270)"/>

<wire from="(280,80)" to="(410,80)"/>

<wire from="(640,320)" to="(640,370)"/>

<wire from="(650,390)" to="(660,390)"/>

<wire from="(470,90)" to="(540,90)"/>

<comp lib="0" loc="(680,320)" name="Pin">

<a name="facing" val="west"/>

<a name="output" val="true"/>

<a name="width" val="8"/>

<a name="label" val="ALU out to ACC"/>

<a name="labelloc" val="south"/>

</comp>

<comp lib="0" loc="(130,270)" name="Pin">

<a name="width" val="8"/>

<a name="tristate" val="false"/>

<a name="label" val="Inp to ALU"/>

<a name="labelloc" val="south"/>

</comp>

<comp lib="0" loc="(650,390)" name="Constant">

<a name="width" val="8"/>

<a name="value" val="0x0"/>

</comp>

<comp lib="0" loc="(770,380)" name="Pin">

<a name="facing" val="west"/>

<a name="output" val="true"/>

<a name="label" val="Not Equal Zero"/>

<a name="labelloc" val="east"/>

</comp>

<comp lib="0" loc="(120,590)" name="Pin">

<a name="width" val="4"/>

<a name="tristate" val="false"/>

<a name="label" val="ALU Operation"/>

</comp>

<comp lib="1" loc="(460,280)" name="NOT Gate">

<a name="width" val="8"/>

</comp>

<comp lib="1" loc="(750,380)" name="NOT Gate"/>

<comp lib="3" loc="(700,380)" name="Comparator"/>

<comp lib="2" loc="(620,320)" name="Multiplexer">

<a name="select" val="4"/>

<a name="width" val="8"/>

</comp>

<comp lib="0" loc="(130,80)" name="Pin">

<a name="width" val="8"/>

<a name="tristate" val="false"/>

<a name="label" val="ACC to ALU"/>

<a name="labelloc" val="south"/>

</comp>

<comp lib="3" loc="(470,90)" name="Adder"/>

<comp lib="1" loc="(470,140)" name="XOR Gate">

<a name="width" val="8"/>

<a name="size" val="30"/>

<a name="inputs" val="2"/>

</comp>

<comp lib="3" loc="(340,220)" name="Multiplier"/>

</circuit>

<circuit name="Instruction Decoding Unit">

<a name="circuit" val="Instruction Decoding Unit"/>

<a name="clabel" val=""/>

<a name="clabelup" val="east"/>

<a name="clabelfont" val="SansSerif plain 12"/>

<wire from="(330,110)" to="(580,110)"/>

<wire from="(530,250)" to="(580,250)"/>

<wire from="(250,250)" to="(310,250)"/>

<wire from="(330,130)" to="(520,130)"/>

<wire from="(250,60)" to="(560,60)"/>

<wire from="(530,120)" to="(530,250)"/>

<wire from="(330,90)" to="(380,90)"/>

<wire from="(520,290)" to="(580,290)"/>

<wire from="(310,240)" to="(310,250)"/>

<wire from="(380,80)" to="(380,90)"/>

<wire from="(540,210)" to="(580,210)"/>

<wire from="(250,170)" to="(250,250)"/>

<wire from="(350,540)" to="(590,540)"/>

<wire from="(380,80)" to="(490,80)"/>

<wire from="(200,150)" to="(230,150)"/>

<wire from="(560,170)" to="(580,170)"/>

<wire from="(330,100)" to="(490,100)"/>

<wire from="(560,60)" to="(560,170)"/>

<wire from="(250,60)" to="(250,160)"/>

<wire from="(520,130)" to="(520,290)"/>

<wire from="(330,230)" to="(350,230)"/>

<wire from="(520,90)" to="(540,90)"/>

<wire from="(230,40)" to="(230,150)"/>

<wire from="(140,160)" to="(160,160)"/>

<wire from="(230,40)" to="(250,40)"/>

<wire from="(540,90)" to="(540,210)"/>

<wire from="(350,230)" to="(350,540)"/>

<wire from="(380,90)" to="(380,140)"/>

<wire from="(180,160)" to="(250,160)"/>

<wire from="(180,170)" to="(250,170)"/>

<wire from="(330,120)" to="(530,120)"/>

<wire from="(380,140)" to="(580,140)"/>

<comp lib="1" loc="(520,90)" name="OR Gate">

<a name="size" val="30"/>

<a name="inputs" val="2"/>

</comp>

<comp lib="2" loc="(310,240)" name="Decoder">

<a name="select" val="4"/>

</comp>

<comp lib="0" loc="(590,540)" name="Pin">

<a name="facing" val="west"/>

<a name="output" val="true"/>

<a name="label" val="Halt"/>

<a name="labelloc" val="east"/>

</comp>

<comp lib="0" loc="(580,140)" name="Pin">

<a name="facing" val="west"/>

<a name="output" val="true"/>

<a name="label" val="Use Immediate"/>

<a name="labelloc" val="east"/>

</comp>

<comp lib="0" loc="(580,110)" name="Pin">

<a name="facing" val="west"/>

<a name="output" val="true"/>

<a name="label" val="Write Memory"/>

<a name="labelloc" val="east"/>

</comp>

<comp lib="0" loc="(160,160)" name="Splitter">

<a name="fanout" val="4"/>

<a name="incoming" val="16"/>

<a name="appear" val="center"/>

<a name="bit1" val="0"/>

<a name="bit2" val="0"/>

<a name="bit3" val="0"/>

<a name="bit4" val="1"/>

<a name="bit5" val="1"/>

<a name="bit6" val="1"/>

<a name="bit7" val="1"/>

<a name="bit8" val="2"/>

<a name="bit9" val="2"/>

<a name="bit10" val="2"/>

<a name="bit11" val="2"/>

<a name="bit12" val="3"/>

<a name="bit13" val="3"/>

<a name="bit14" val="3"/>

<a name="bit15" val="3"/>

</comp>

<comp lib="0" loc="(580,290)" name="Pin">

<a name="facing" val="west"/>

<a name="output" val="true"/>

<a name="label" val="Branch NE zero"/>

<a name="labelloc" val="east"/>

</comp>

<comp lib="0" loc="(580,170)" name="Pin">

<a name="facing" val="west"/>

<a name="output" val="true"/>

<a name="width" val="4"/>

<a name="label" val="ALU Operation"/>

<a name="labelloc" val="east"/>

</comp>

<comp lib="0" loc="(140,160)" name="Pin">

<a name="width" val="16"/>

<a name="tristate" val="false"/>

<a name="label" val="Instruction"/>

</comp>

<comp lib="0" loc="(200,150)" name="Splitter">

<a name="facing" val="west"/>

<a name="incoming" val="8"/>

<a name="appear" val="center"/>

<a name="bit1" val="0"/>

<a name="bit2" val="0"/>

<a name="bit3" val="0"/>

<a name="bit4" val="1"/>

<a name="bit5" val="1"/>

<a name="bit6" val="1"/>

<a name="bit7" val="1"/>

</comp>

<comp lib="0" loc="(250,40)" name="Pin">

<a name="facing" val="west"/>

<a name="output" val="true"/>

<a name="width" val="8"/>

<a name="label" val="Address or Immediate"/>

<a name="labelloc" val="east"/>

</comp>

<comp lib="0" loc="(580,210)" name="Pin">

<a name="facing" val="west"/>

<a name="output" val="true"/>

<a name="label" val="Write ACC"/>

<a name="labelloc" val="east"/>

</comp>

<comp lib="0" loc="(580,250)" name="Pin">

<a name="facing" val="west"/>

<a name="output" val="true"/>

<a name="label" val="Branch always"/>

<a name="labelloc" val="east"/>

</comp>

</circuit>

</project>

# Appendix B: The MIPS Code

# MIPS code for computer architecture assignment - By Husnain Ahmed

.data

enterMsg0: .asciiz " \n "

enterMsg1: .asciiz "Please enter the last four digits of your student id \n"

enterMsg2: .asciiz "Press enter between each digit \n"

enterMsg3: .asciiz "Enter next digit \n"

enterMsg4: .asciiz "The total sum (addition) of the digits is: "

enterMsg5: .asciiz "The total sum (multiplication) of the digits is: "

NumArray: .space 16

.text

#initialise 4 places in the array

addi $s2,$zero, 0

addi $s3,$zero, 0

addi $s4,$zero, 0

addi $s5,$zero, 0

# reset $t0 to '0'

addi $t0, $t0, 0

# output msg1

addi $v0, $zero, 4

la $a0, enterMsg1

syscall

## output msg2

addi $v0, $zero, 4

la $a0, enterMsg2

syscall

# read an integer from keyboard input and store the input in $s0 for the total

addi $v0, $zero, 5

syscall

add $s0, $zero, $v0

add $s1, $zero, $v0

addi $t0, $zero, 5

sb $s2, NumArray($t0)

## calling the subroutine with jump and loops

addi $a1, $zero, 3

JalLoop:

jal msg3

subi $a1, $a1, 1

bnez $a1, JalLoop

# print array of inputted numbers

lb $t3, NumArray($zero)

li $v0, 1

addi $a0, $t3, 0

syscall

lb $t3, NumArray($s3)

li $v0, 2

addi $a0, $t3, 0

syscall

lb $t3, NumArray($s4)

li $v0, 2

addi $a0, $t3, 0

syscall

lb $t3, NumArray($s5)

li $v0, 3

addi $a0, $t3, 0

syscall

#gap

addi $v0, $zero, 4

la $a0, enterMsg0

syscall

## added total

addi $v0, $zero, 4

la $a0, enterMsg4

syscall

add $a0, $s0, $zero

addi $v0, $zero, 1

syscall

addi $v0, $zero, 4

la $a0, enterMsg0

syscall

#multiplied total

addi $v0, $zero, 4

la $a0, enterMsg5

syscall

add $a0, $s1, $zero

addi $v0, $zero, 1

syscall

addi $v0, $zero, 10

syscall

## subroutine

msg3:

addi $v0, $zero, 4

la $a0, enterMsg3

syscall

addi $v0, $zero, 5

syscall

add $s0, $s0, $v0

mul $s1, $s1, $v0

sb $s3, NumArray($t0)

addi $t0, $t0, 5

sb $s4, NumArray($t0)

addi $t0, $t0, 5

sb $s5, NumArray($t0)

addi $t0, $t0, 5

jr $ra

# Appendix C: Completed Marking Grid

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Criteria | 0-29% | 30-39% | 40-49% | 50-59% | 60-69% | 70-85% | 86-100% |
| Logisim CPU simulation amendment and testing | There is little or no evidence of using or developing the Logisim CPU simulation.  There is little or no evidence of a professional approach to the tasks. | There is some evidence of trying at least one of the Logisim CPU tasks.  There is little evidence of a professional approach to the tasks. | There is evidence that at least two of the Logisim CPU tasks are complete and working appropriately.  There is some evidence of considering a professional approach to the completed tasks. | There is evidence that at least two of the Logisim CPU tasks are complete and working appropriately with attempts at further tasks.  There is some evidence of applying a professional approach to the completed tasks. | There is evidence that most of the Logisim CPU tasks are complete and working appropriately.  Most of the tasks have been done with a suitably professional approach. | There is evidence that most of the Logisim CPU tasks are complete and working appropriately with an attempt at all tasks.  Most of the tasks have been done with a highly professional approach. | There is evidence that all of the Logisim CPU tasks are complete and working, although there may be issues with some aspects on one or two tasks.  Most of the tasks have been done with a highly professional approach. |
| MIPS Assembly language amendment and testing | There is little or no evidence of using or developing the MIPS Assembly language program.  There is little or no evidence of a professional approach to the tasks. | There is some evidence of trying at least one of the tasks to amend the supplied MIPS Assembly language program.  There is little evidence of a professional approach to the tasks. | There is evidence that at least two of the MIPS Assembly language tasks are complete and working appropriately.  There is some evidence of considering a professional approach to the completed tasks | There is evidence that at least two of the MIPS Assembly language tasks are complete and working appropriately with attempts at further tasks.  There is some evidence of applying a professional approach to the completed tasks. | There is evidence that most of the MIPS Assembly language tasks are complete and working appropriately.  Most of the tasks have been done with a suitably professional approach. | There is evidence that most of the MIPS Assembly language tasks are complete and working appropriately with an attempt at all tasks.  Most of the tasks have been done with a highly professional approach. | There is evidence that all of the MIPS Assembly language tasks are complete and working, although there may be issues with some aspects on one or two tasks.  Most of the tasks have been done with a highly professional approach. |
| Report | There is little or no attempt at the written discussion part of the report.  Little or no part of the report structure has been completed. | There is some attempt at the written discussion part of the report, but this part is very limited or lacks clarity and/or accuracy.  Several parts of the report structure are complete. There may be some errors in the content. | There is a reasonable attempt at the written discussion part of the report, but it is mainly descriptive and may have errors or a lack of clarity.  Most parts of the report structure are complete. There may be some errors in the content. | The discussion part of the report contains mainly description very little analysis. There may be some errors.  The report is mainly completed to an acceptable standard. | The discussion part of the report includes mainly accurate description and some analysis, making suitable connections.  The report is completed to a good standard, with use of technical language and some references, mainly presented appropriately. | The discussion part of the report includes accurate description and critical analysis.  The report is completed to a high standard, with appropriate use of technical language and of references. | The discussion part of the report includes accurate description and critical analysis showing insight in the points made.  The report is completed to a very high standard, with appropriate use of technical language and of references. |