[Mohammed mahin ibnay mamun]

[23643001]

[Year]

[1. Introduction 1](#_Toc144972793)

[2. CPU Simulation 2](#_Toc144972794)

[2.1 Calculating the sum of data stored in data memory 2](#_Toc144972795)

[2.2 Calculating the sum of Immediate values 2](#_Toc144972796)

[2.3 Bitwise OR 2](#_Toc144972797)

[2.4 Completing the stop instruction 3](#_Toc144972798)

[2.5 Branch always 3](#_Toc144972799)

[2.6 Amending the ALU 3](#_Toc144972800)

[2.7 Conditional branch instruction 4](#_Toc144972801)

[2.8 Incorporating Input and Output devices 4](#_Toc144972802)

[3. RISC-V Assembly Language Programming 5](#_Toc144972803)

[3.1 Test the existing program with minor adjustments 5](#_Toc144972804)

[3.2 Improve the messages that the program outputs 5](#_Toc144972805)

[3.3 Adding in a subroutine 5](#_Toc144972806)

[3.4 Enhancing the functionality of the program 5](#_Toc144972807)

[3.5 Splitting Numbers 6](#_Toc144972808)

[3.6 Splitting numbers with a loop 6](#_Toc144972809)

[3.7 Making use of arrays 6](#_Toc144972810)

[3.8 Designing a game 7](#_Toc144972811)

[4. Conclusion 8](#_Toc144972812)

[5. References 9](#_Toc144972813)

[6. Appendix A: The CPU Simulation Code 10](#_Toc144972814)

[7. Appendix B: The RISC-V Assembly Language Code 11](#_Toc144972815)

# Introduction

The current report is split into 2. the first part will be making circuits with logism evolution and the second part using rars to make CPU operations. I will include screenshots of my test examples including some before and afters. I will also write an explanation of what am doing on each task and how this affects the output.

# CPU Simulation

## Calculating the sum of data stored in data memory

### Test data

v2.0 raw

3 0 0 1

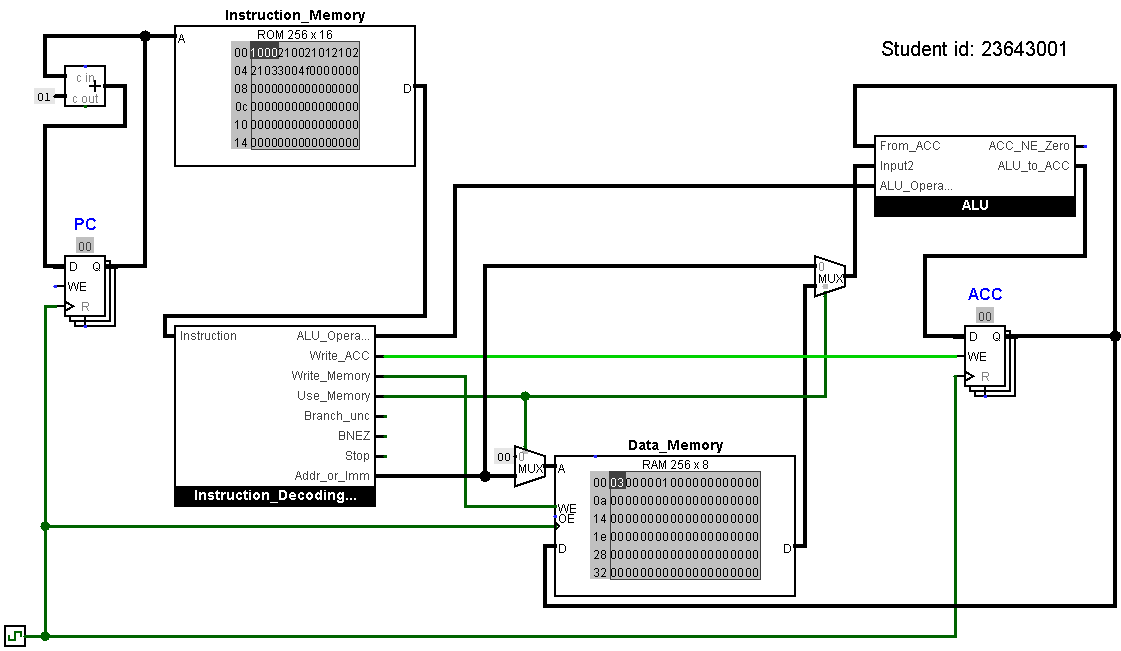
### Expected result

When we get the 4 digits in this scenario and add them up, this is what we get from the equation.

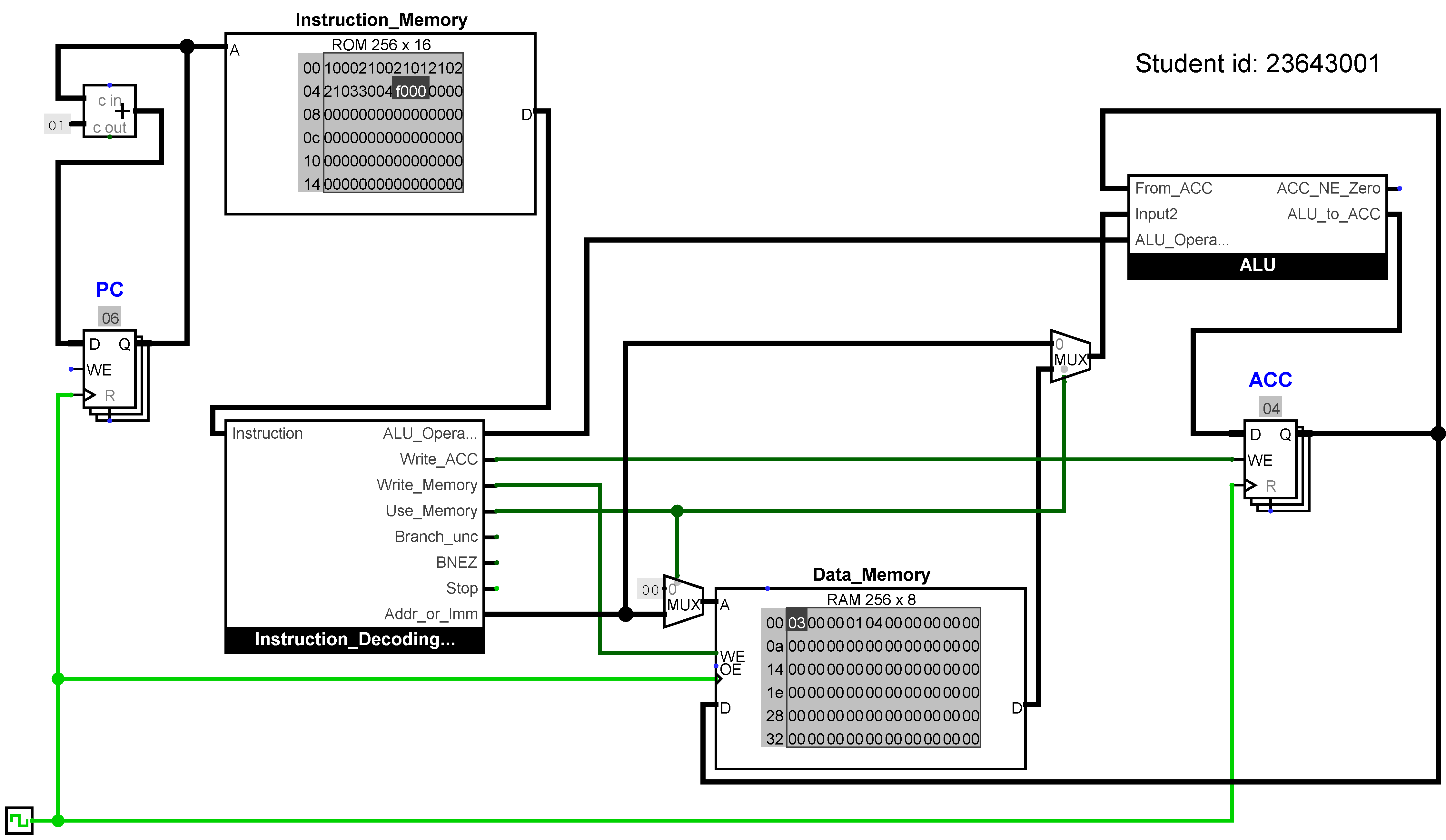
3 + 0 + 0 + 1 = 4

In hexadecimal this is equal to = 4 which can also be displayed as 0x4.

### Circuit with data loaded



### Circuit at end of program



### Result

The hex value 0x4 is at address 04 in both the accumulator which can be seen in the screenshot above labelled in blue writing = ACC. the data memory can be seen with our values 03, 0 0, 0 0, 01 and 04. The 5th value in the data memory is our result and the first 4 are out input values which got added up.

### Explanation

The pc in our circuit which is the program counter is used to return the address of the instruction we are running. The counter gets incremented by 1 when our circuit is running. The accumulator which is labelled ACC in our circuit keeps the running total and goes back into the arithmetic logic unit which is labelled as ALU to add the next value. The values make a change when the clock is clicked. The program counter increases by 1 which allows our instruction memory to highlight the next bit of data. This triggers our ALU which adds a value of 01 to our accumulator.

## Calculating the sum of Immediate values

### Test program

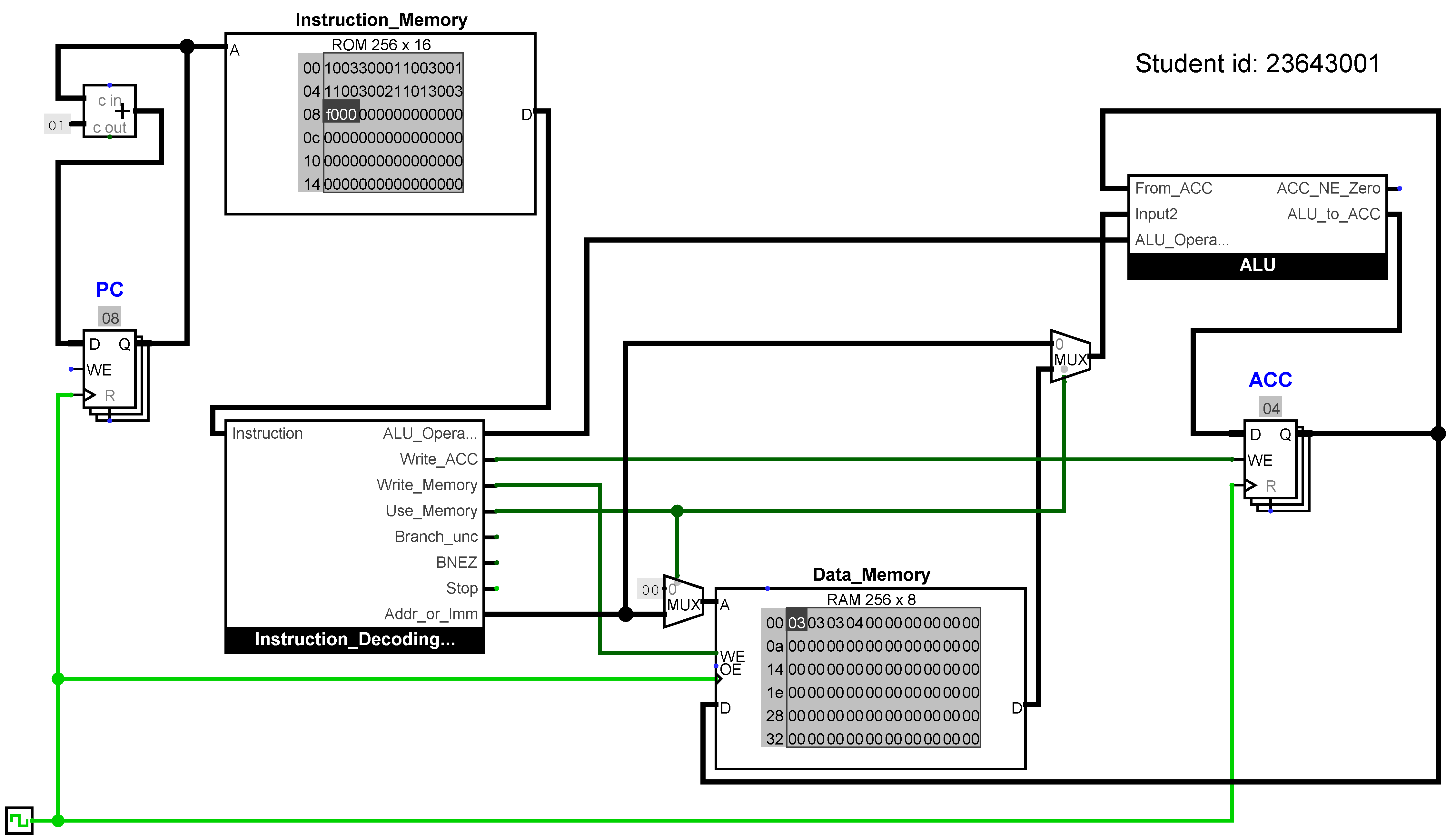
v2.0 raw

1003 3000 1100 3001 1100 3002 1101 3003 f000

### Expected result

|  |  |  |
| --- | --- | --- |
| Input Data | Running Total | |
| Decimal | Hexadecimal |
| 3 | 3 | 3 |
| 0 | 3 | 3 |
| 0 | 3 | 3 |
| 1 | 4 | 4 |

### Circuit at end of program



### Result

My expected results are equal to my data memory once I have run the circuit. an addition has been done therefore the outputs are 3 3 3 then 4. 3+0+0+1. The final value is 4 which is correct.

### Explanation

The input2 to the arithmetic logic unit (ALU) travels in via the wires from either the Addr\_or\_imm which we can see is at the bottom of our instruction Decoding area or from data memory. The use memory signal goes inside the multiplexer to choose which one to use.

## Bitwise OR

### Test program

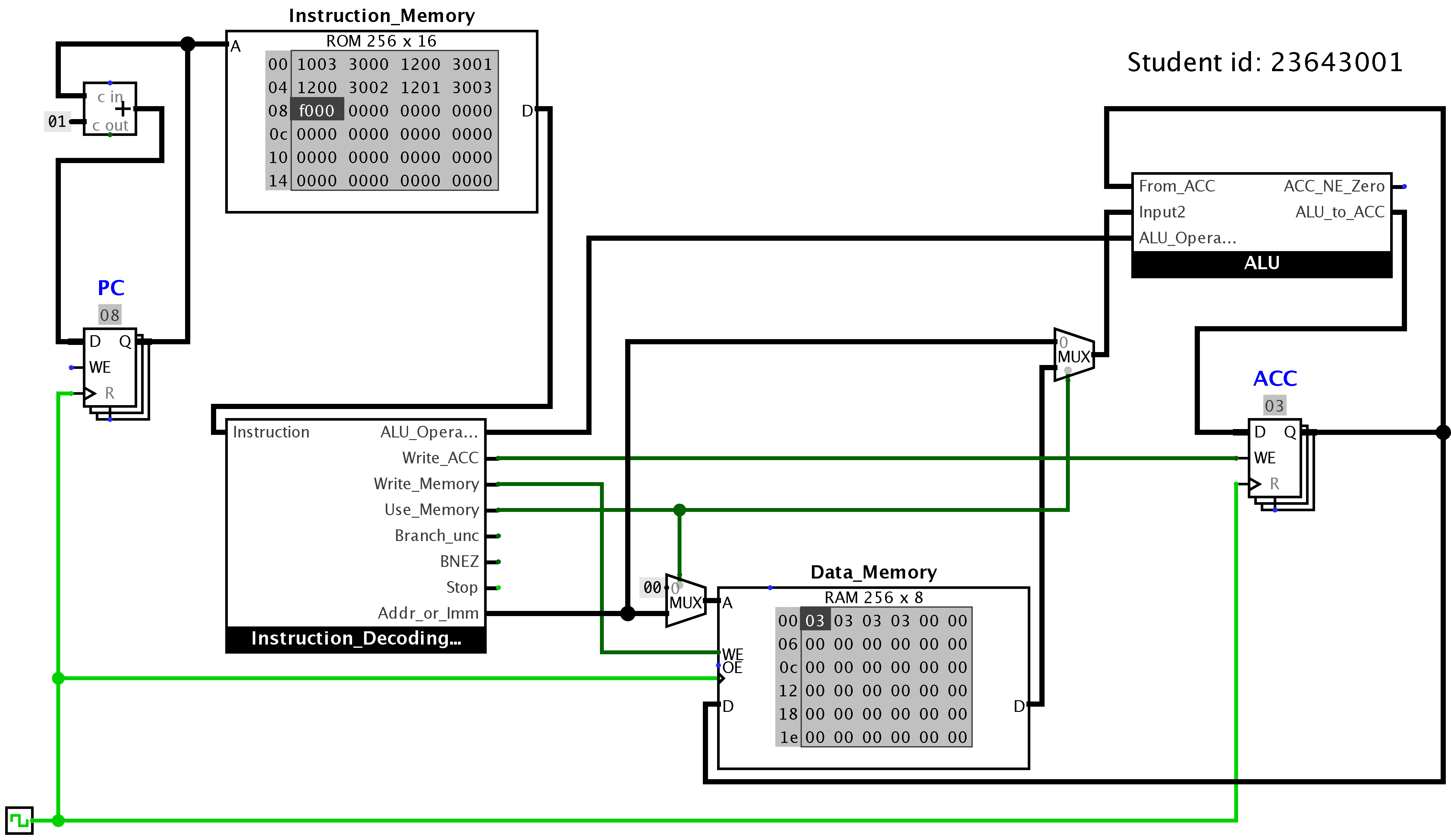
v2.0 raw

1003 3000 1200 3001 1200 3002 1201 3003 f000

### Expected result

|  |  |  |  |
| --- | --- | --- | --- |
| Input data | | Expected Results | |
| Decimal | Binary | Binary | Hexadecimal |
| 3 | 00000011 | 00000011 | 3 |
| 0 | 00000000 | 00000011 | 3 |
| 0 | 00000000 | 00000011 | 3 |
| 1 | 00000001 | 00000011 | 3 |

### Circuit at end of program



### Result

As shown in my screenshot above, by loading the new set of data into the instruction memory and running the clock, the data memory value match up with my expected results which is stated above.

### Explanation

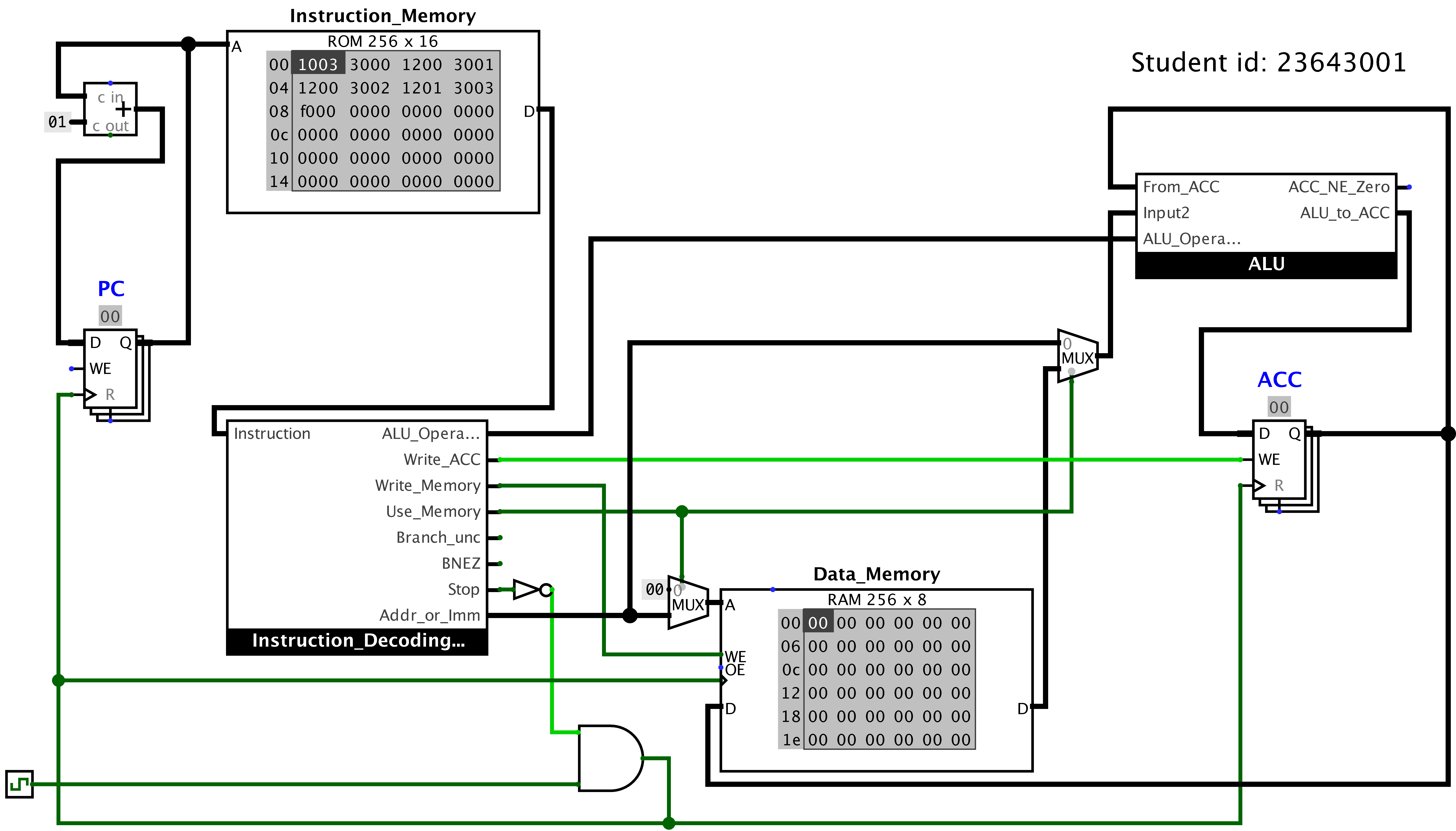
The ALU operation goes to a multiplexer inside the ALU operation 1 adds the two inputs and operation 2 does a bitwise OR

## Completing the stop instruction

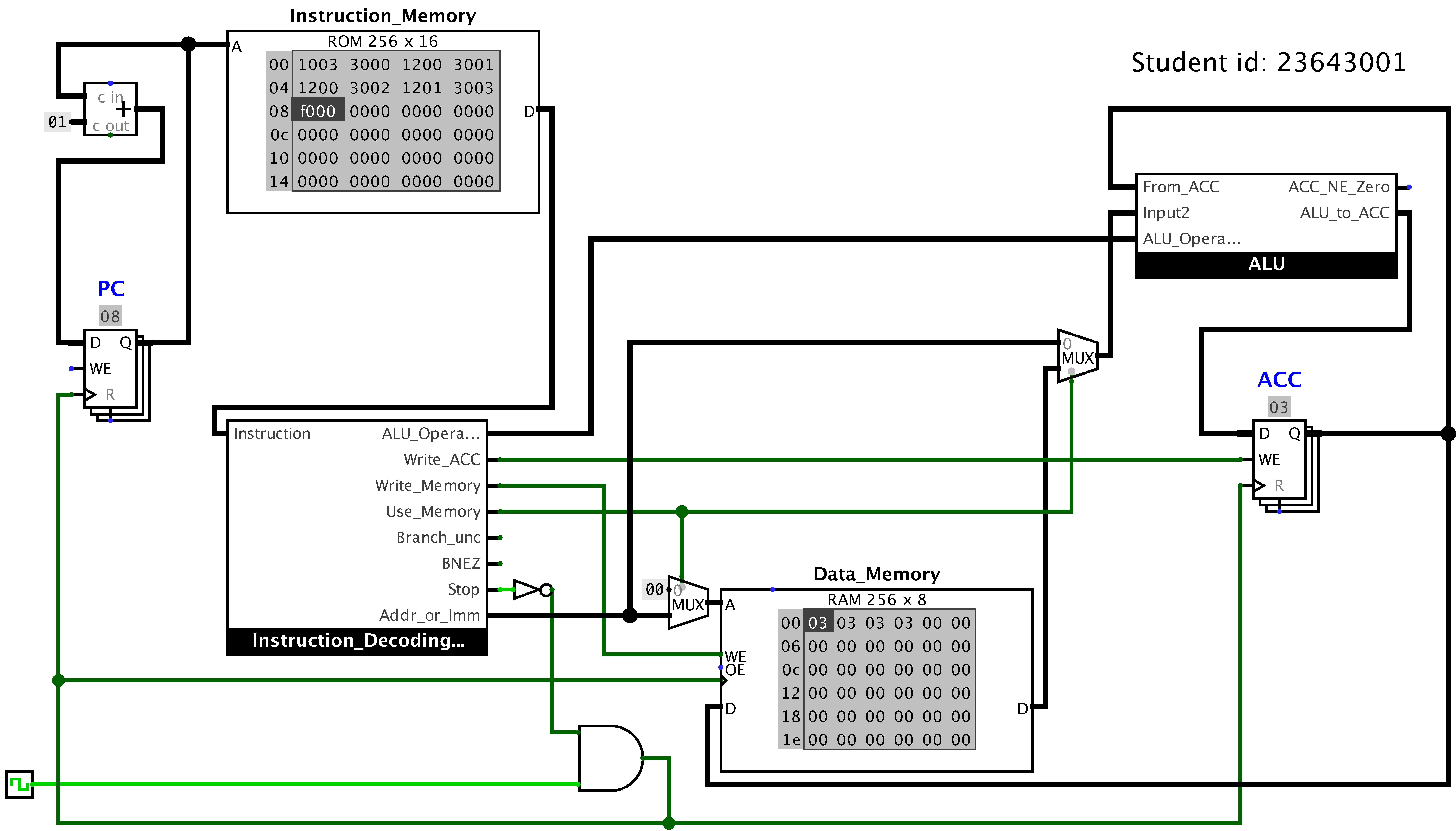
### Expected result

We are expecting the same results as the last test 2.3 because we have not removed any or loaded any new data to our program. We are going to add 2 gates to our program and expect a same outcome if done correctly. I am also expecting the circuit to look a little different, reason for this is because I will be adding in 2 gates and wiring them to the circuit.

### Circuit at start of Test



### Circuit at end of Test



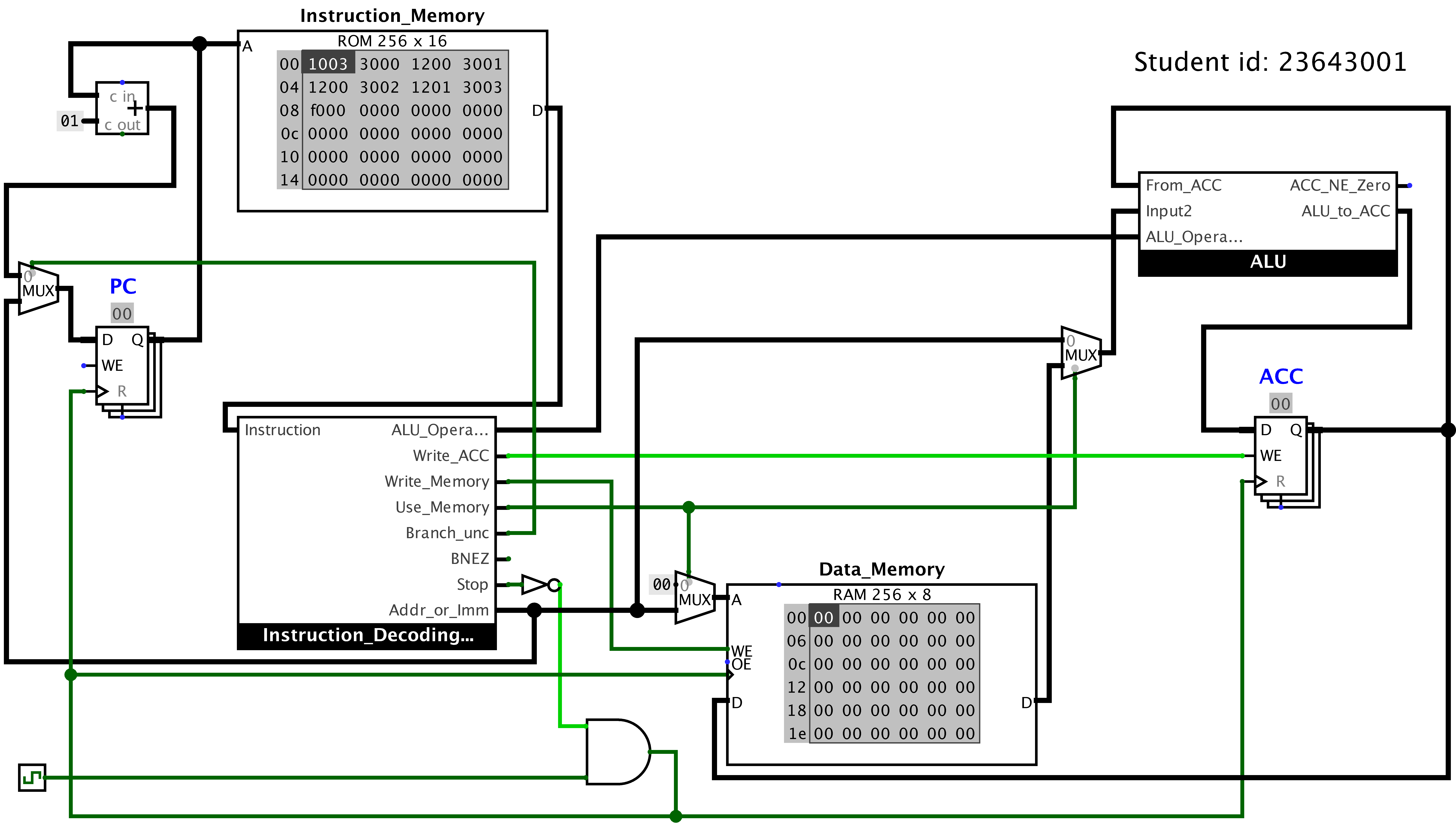
### Explanation

As show in the above screenshots, 2.4.2 shows the circuit at the start of the test. 2.4.2 shows the same circuit but when the test has been completed. our data memory remains the same Aswell as our program counter and accumulator. All outputs remain the same, the reason for this is simply because no data was changed but new components and wiring was amended.

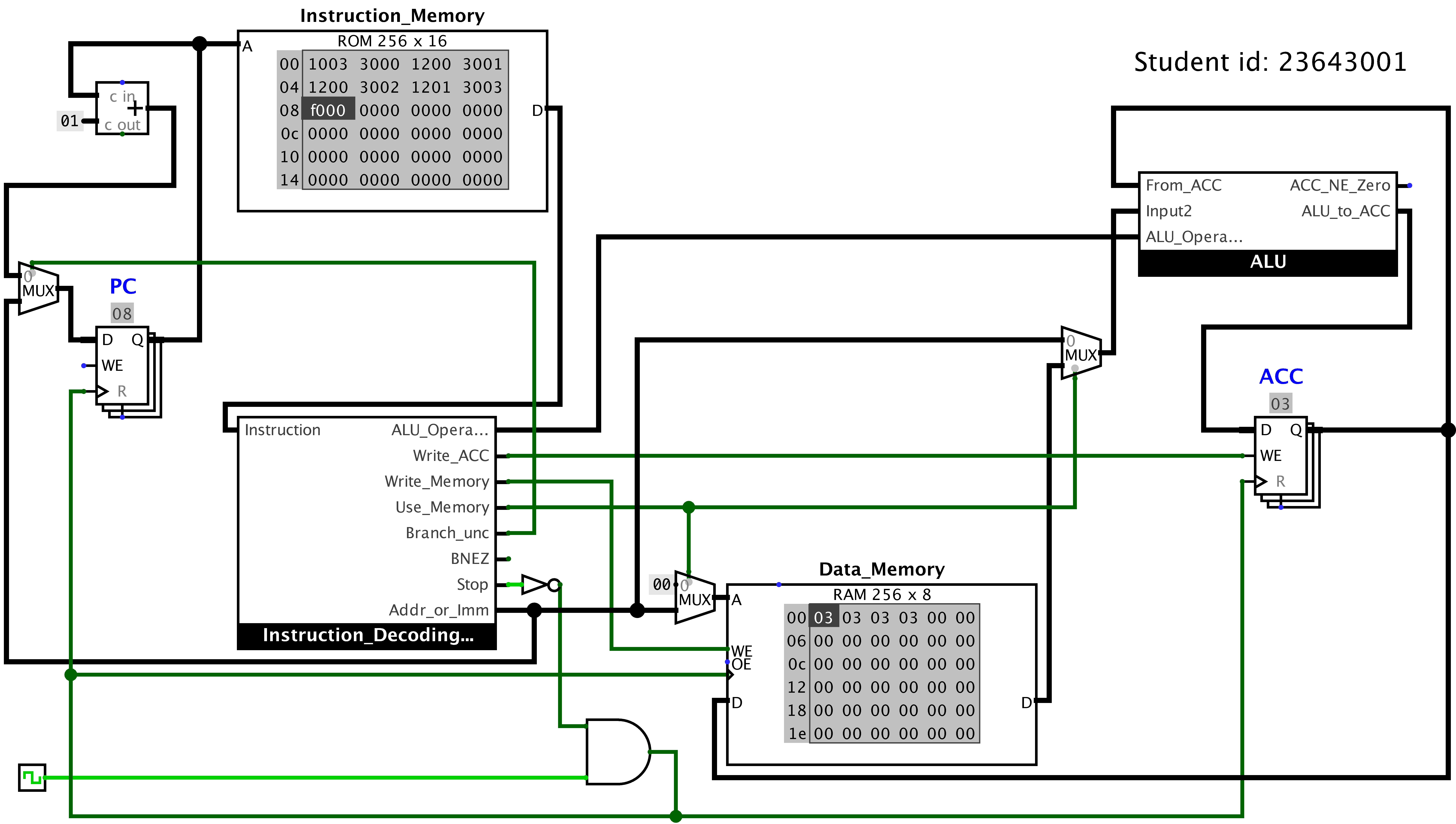
The first change I made to the circuit for this task was, I removed the clock output which was connected to the wires which were running our circuit. Then I made use of the stop function in the instruction decoder to do this, I dragged out a wire from this output and joined it to the input of a new NOT gate. Then I got a new AND gate and joint the output of the not gate into one of the inputs for the AND gate. The next input for this AND gate came straight from the clock. Finally, the output for this AND gate connected back to the wires which were previously connected to the clock.

## Branch always

### Test Program



The previous circuit which I made in the task above has now been modified in multiple ways by adding a multiplexer tool, from the ‘Plexers’ tab. The benefit of this tool is that is allows me to choose where to get data from in my circuit. I placed this in a suitable area of on my circuit then connected the 2 input pins. The first one is connected from my adder and the second is connect from the instruction decoder’s Addr\_or\_imm. The output of this multiplier in my circuit was then connected to my program counter’s input “D.” Then I corrected the properties of the multiplier (bits). Finally, I wired up the Branch\_unc from my instruction decoder to the select on my multiplexer.



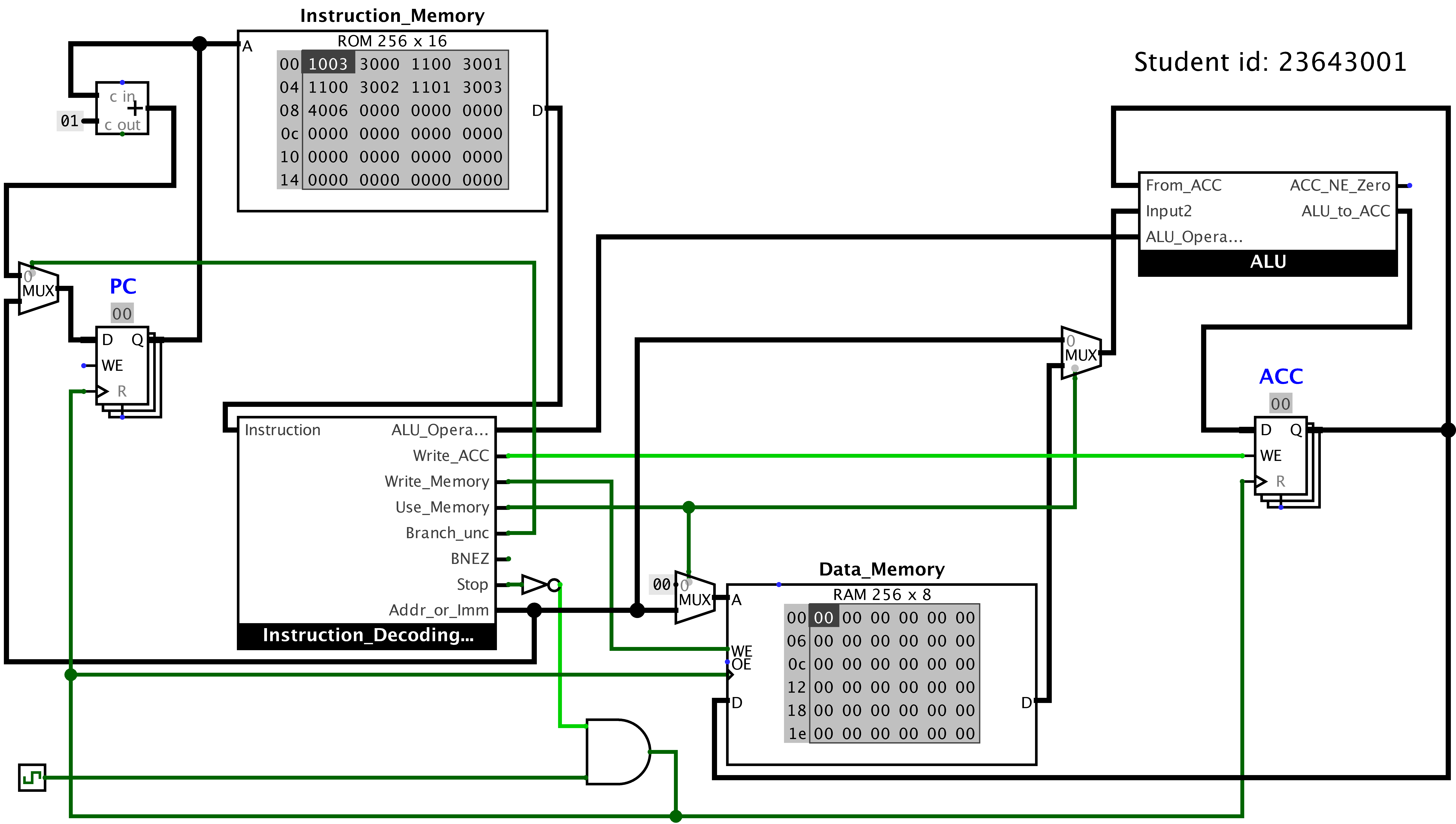
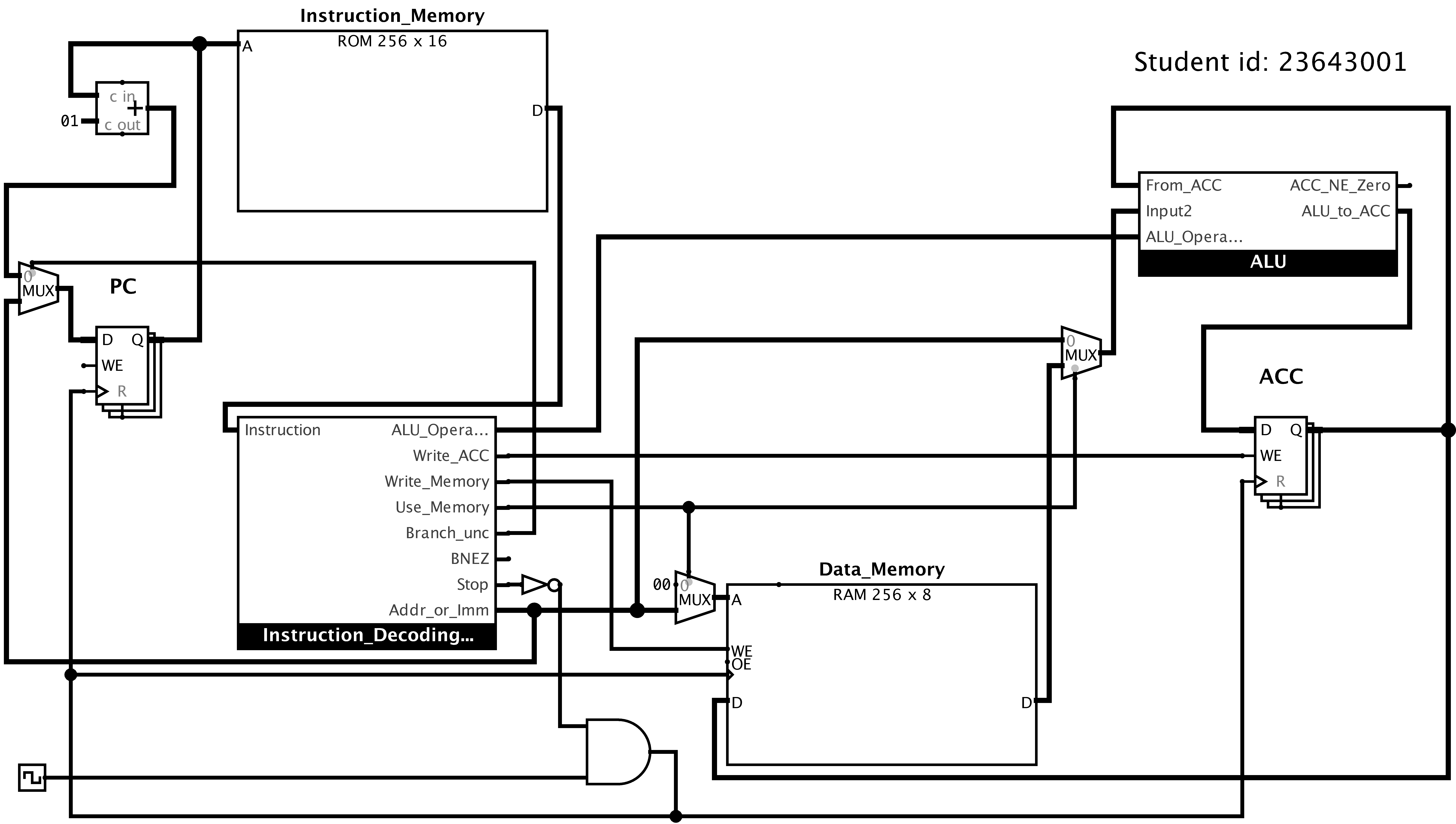
### Expected Result

The circuit should take the values of the last 4 digits of the student id and add them up. For example, in my scenario, my student ID ends in 3001. Therefore, the circuit will perform an addition on these values 3 + 0 + 0 + 1 = 4. this has been completed before in a previous task. Now i will loop the circuit so it will run the arithmetic operation 5 times, so I am expecting the result of 20 as 4 x 5 is 20.

### Circuit after adding 5 times

So now my circuit runs in an infinite loop. I had to edit the instruction from my instruction memory the data was originally 1003 3000 1100 3001 1100 3002 1101 3003 f000. The end of this data, which is f000, was used originally to end the data. I made changes to this data file and replaced f000 with 4006. The value 4 as the first digit is needed so the instruction memory knows we are using an unconditional branch instruction. Then the second digit of this 4digit sequence is set to 0. the reason for this because the alu is not used. Finally, 06 is used in the 4 digits at the end of the .data, this tells the loop to go back to address 06 and loop from there.

### Result

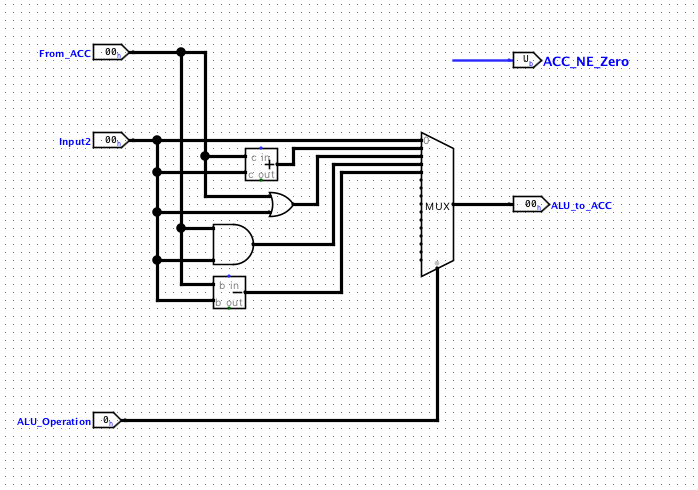


### Explanation

So now my circuit runs in an infinite loop. I had to edit the instruction from my instruction memory the data was originally 1003 3000 1100 3001 1100 3002 1101 3003 f000. The end of this data, which is f000, was used originally to end the data. I made changes to this data file and replaced f000 with 4006. The value 4 as the first digit is needed so the instruction memory knows we are using an unconditional branch instruction. Then the second digit of this 4digit sequence is set to 0. the reason for this because the alu is not used. Finally, 06 is used in the 4 digits at the end of the .data, this tells the loop to go back to address 06 and loop from there. The outcome now displays the data memory adding on each time.

## Amending the ALU

### ALU circuit



### Test Program

|  |  |  |  |
| --- | --- | --- | --- |
|  | Decimal | Binary | Hexadecimal |
| Initial Value | 3 | 11 | 0x3 |
| After subtracting 1 | 2 | 10 | 0x2 |
| After Bitwise AND |  |  |  |

### Expected result

### Circuit at end of program

### Result

### Explanation

## Conditional branch instruction

### ALU circuit

### Test Program

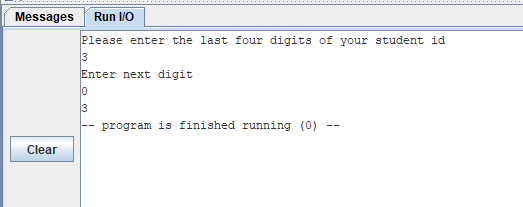
### Test evidence

### Explanation

# RISC-V Assembly Language Programming

## Test the existing program with minor adjustments

### Starter Program



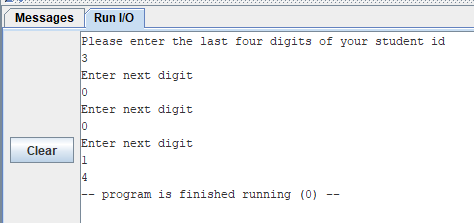
This image is a screenshot of the program result before any modification

### Expected Result

The expected results should be the sum of the last 4 digits of the student id:

3 + 0 + 0 + 1 = 4

### Result



The result outputs the expected result therefore it is working.

### Explanation

A7 performs instruction.

When the register A7 is used in RARS, and is equal to 1, an integer is printed out.

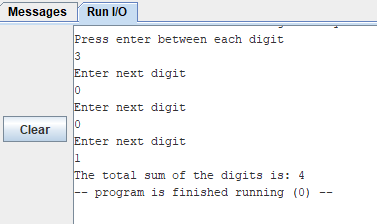
When the register A7 is used in RARS and is equal to 4 a string is printed out.

When the register A7 is used in RARS and is equal to 5 the command reads out integers.

When the register A7 is used in RARS and is equal to 7 the program exits.

## Improve the messages that the program outputs

### Results



An already defined string was used to instruct the user at the beginning to press the enter between each digit. A further message was defined and output at the end before the result saying, “The total sum of the digits is:”

### Explanation

Clear messages are very import in programming to ensure the user of the program clearly understands the aims and objectives:

User knows what to do, clarity, properly instruct.

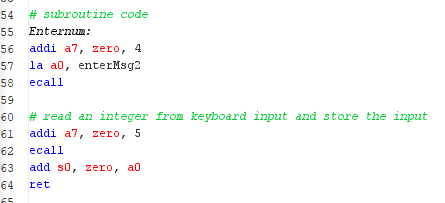
Directives. Data and .text

.data is used to define use of memory.

.text is used to write the program instructions.

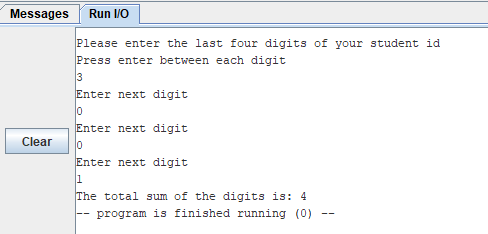
## Adding in a subroutine

### Subroutine code



A section of repeated code has been placed into a subroutine which can be called with Jal function.

### Result



The subroutine works correctly as the result is identical to the program from question 2.

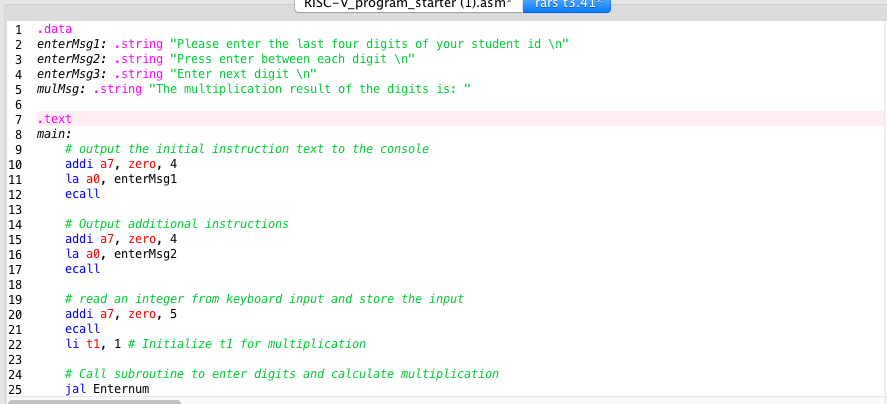
### Explanation

Subroutines used to reduce code and make code more efficient.

Jal used, jump and link. The use of this function (Jal) in the code allows us to run chunks of code over and over without too much repetition. This makes the code less stacked and easier to read.

## Enhancing the functionality of the program

### Multiplication Code

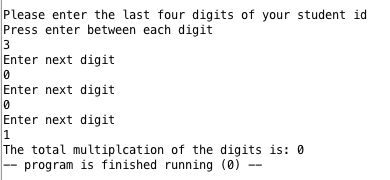


### Expected Result

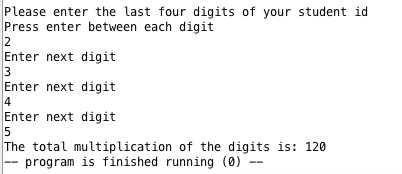
With this code, i will run 2 sets of code.one of the test results will use my last 4 digits of user id which includes the value 0. by having 0 in my input the output will be 0 due to multiplication ruling. Therefore, a second till will take place using digits other than 0.

### Result

My user id including 0.



Test number 2 without digit 0.



### Explanation

Above is screenshot of my code and testing of my multiplication code. The code being by using messages stored as strings in the .data and .text. Then my codes run an operation which allows it to read the input from the user's keyboard and store the value into the register t0. Then my program calls the Enternum subroutine three times, this allows user to enter values and store them into register. 4 digits are printed, my code displays the message saying what the total multiplication of the code is.

## Splitting Numbers

### Pseudocode

Set register s0 to have the value from a0 (the user input)

Set register s1 to 10 (this will be used for division later)

Set register t0 to the sum of 1000 plus zero

After these: s0=3001, s1=10, t0=1000

Divide s0 by t0 and store the integer result in t1

After this: s0=3001, s1=10, t0=1000, t1=3

Update s0 to be the remainder of s0 divided by t0

After this: s0=001, s1=10, t0=1000, t1=3

Divide t0 by s1 storing the integer result back in t0

After this: s0=001, s1=10, t0=100, t1=3

Divide new s0 by new t0 and store the integer result in t2

After this: s0=321, s1=10, t0=100, t1=3, t2=3

Update s0 to be the remainder of s0 divided by t0

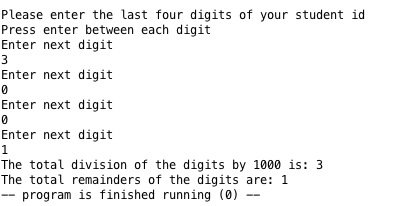
After this: s0=01, s1=10, t0=100, t1=3

Above is an example of the start of my pseudocode for the division program.

### Expected Result

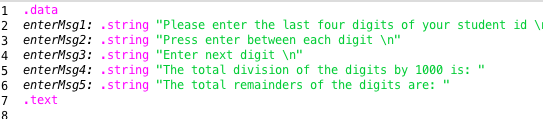
I can use the operation in rars (rem) to find reminders of values after division (div) has taken place. I found these operators in the riscV basic instructions tab which is located besides the syscals tab. So, the expected result of my code should be division = 3 and remainders also known as. Quotient =1. Because leading 0 are not needed.

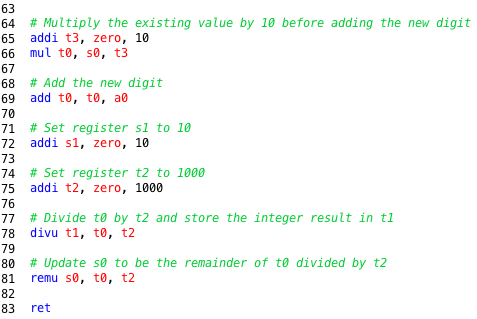
### Result



### Explanation

After amending my code with the pseudocode written above, my program now displays division and remainders of code. To this, i had to read the pseudocode which is half English half code and then i interpreted this into small chunks of code following the rars coding language. Using syscalls and basic instructions helped me complete this task.

I made a new copy of my previous task code and then i needed to add this new code to Aswell as new enter messages.



## Splitting numbers with a loop

### Pseudocode

Set register t3 to value 4

Set register t4 to value 0

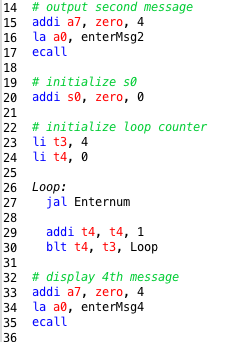
Create a loop then

Call upon our previously used subroutine saved as enterNum

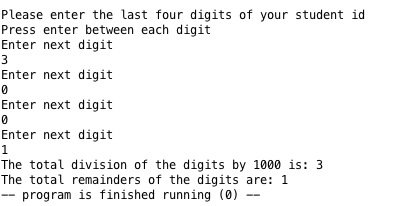
Add 1 to our register t4

If the register t4 is less than register t3, then we want to repeat if not the end loop

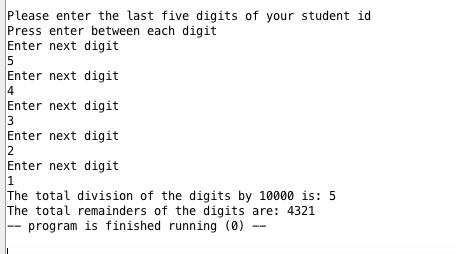
### Loop Code



### 4-Digit Result



### 5-Digit Result



### Explanation

Above is a screenshot of my run io. I used for this examples the values 54321. I have tested my code with different numbers and they are all working as expected. I made 2 main amendments to my code from the previous task. To start with I added an extra 0 to the end of my constat for deviding. Id i didnt do this then my program would still divide by 1000 and i would have 54 as div and 321 as rem. I have included a screenshot from my code of what this looks like.



The second change i made to my code from the previous code was changing the value of my loop. By doing this, I updated the value from 4 to 5 so now the loop would run 5 times which means enternum subroutine will also be called 5 times letter user enter 5 digits instead of 4 and assigning them into the suitable register.



## Making use of arrays

### Pseudocode

Create empty loop and use 5 for memory to later

Int [] digits = new [5]

### Expected Results

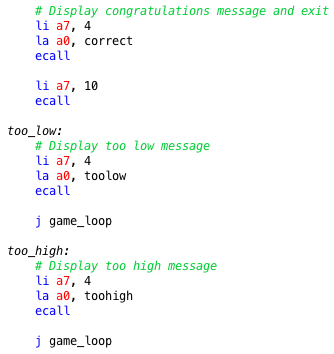
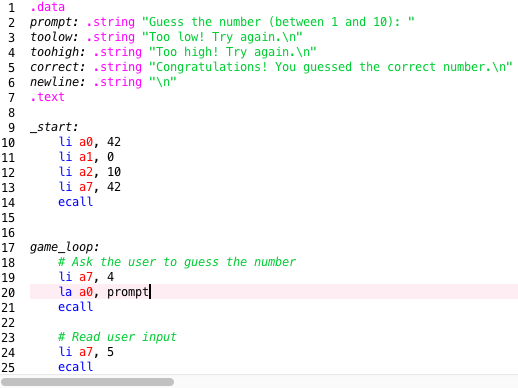
### Result

### Conclusion

### Explanation

## Designing a game

Attempted task but not completed



# Conclusion

At the beginning of this document, i installed logism evolution which is a very useful tool for creating ciruits. I began by working my way through each task and experience more difficult challenge along the way. I added screenshots and explained the importance of each task and what made them different. It went from calculating sums to making a change and using new tools to create a bitwise or. I also made circuit for stops Aswell branching. Unfortunately, i was unable to complete the last task of the first section due to difficult and time.

The second part of this report which I have made covers a range of topics which take place over tasks, including basic arithmetic operations, conditional branches, RISC-V assembly language programming, Aswell as usage of subroutines and many functions. I made programs for adding, multiplying, dividing and loops. Unfortunately, I was unable to complete the game task and arrays task due to time.

# References

I used this link which led to a playlist of several videos to help me understand more about this coding language and get the tasks above completed.

<https://www.youtube.com/playlist?list=PLCNL4SrVOxUUANtygTb0fpRUbD_N-K538>

I went through lecture videos and slides from Moodle Aswell as lab sheet tasks, this also helped me with this written report and programming.