2024

Computer Architecture Report

DEPARTMENT OF COMPUTING AND MATHEMATICS

RODRIGO PICAZO COBALEDA

1. CPU SIMULATION 1
1.1 Calculating the sum of values 1
1.2 Completing the stop instruction 2
1.3 Bitwise Operations 4
1.4 Amending the ALU 5
1.5 Output Device 8
1.6 Branch instruction 10
1.7 Loop 16
1.8 Game 17
2. RISC-V ASSEMBLY LANGUAGE PROGRAMMING 18
2.1 Test the existing program with minor adjustments 18
2.2 Improve the messages that the program outputs 20
2.3 Enhancing the functionality of the program 22
2.4 Adding a subroutine 24
2.5 Swapping Numbers 26
2.6 Counting up with a loop 35
2.7 Making use of arrays 38
2.8 Designing a game 42
3. REFERENCES 43
4. APPENDIX A: THE CPU SIMULATION CODE 44
5. APPENDIX B: THE RISC-V ASSEMBLY LANGUAGE CODE 56
5.1 Tasks 2.1 to 2.4 amending the starter program provided 56
5.2 Tasks 2.5 to 2.7 ordering numbers and using a loop and array. 59
5.3 Task 2.8 Creating a console based game 62
6. APPENDIX C: USE OF GENERATIVE AI 63

1. CPU Simulation

1.1 Calculating the sum of values

1.1.1 Test data

For my test data which will be used to calculate various mathematical operations I'll be using the last four numbers of my MMU student ID,4702.

The last four digits of my MMU student ID will be split into two different two-digit numbers.

Decimal:

1.) **47**
2.) **02**

Hexadecimal:

1.) **2F**
2.) **02**

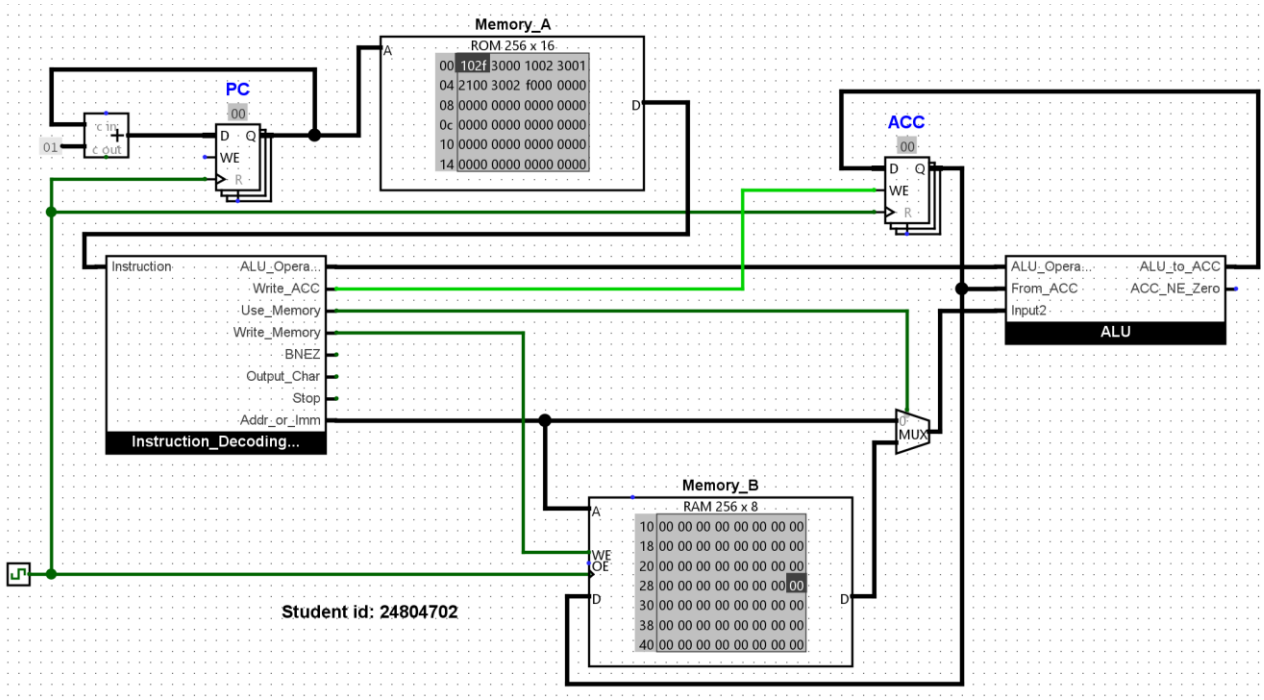
1.1.2 Test program

```
v2.0 raw
102F 3000   # 1. put first value to ACC and address 00
1002 3001   # 2. put second value to ACC and address 01
2100        # 3. add ACC value to value from address 00
3002        # 4. store ACC value at address 02
f000        # 5. stop
```

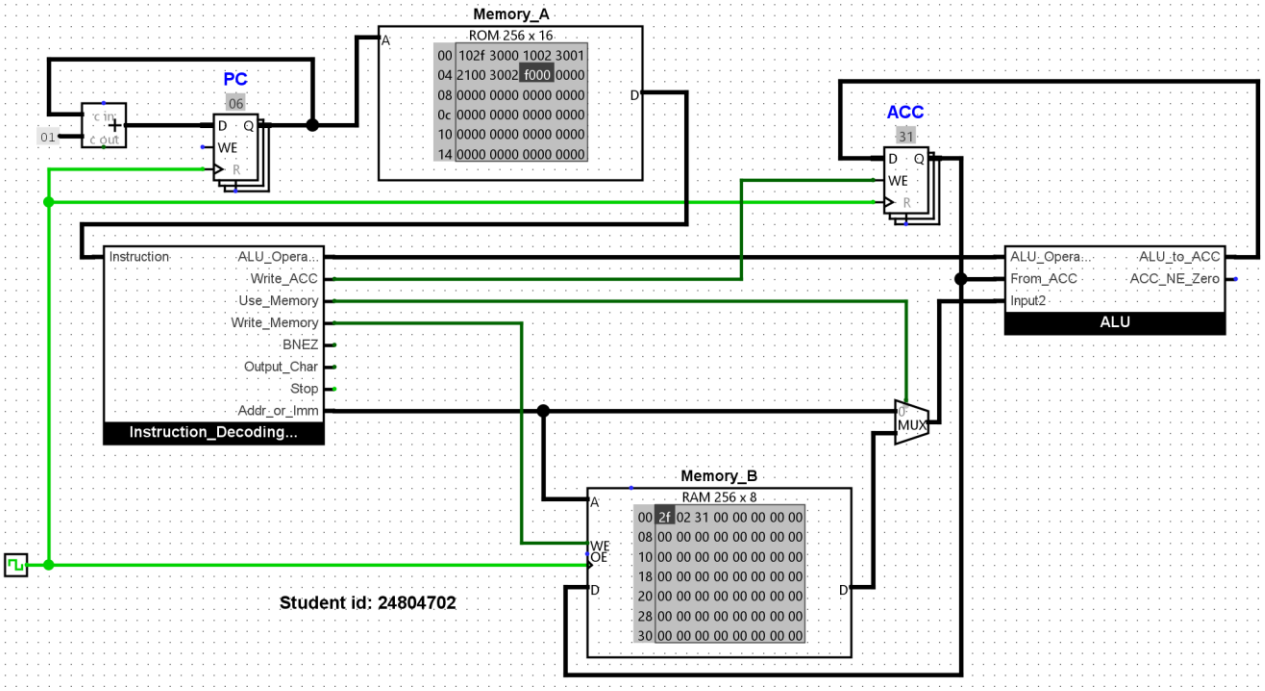
1.1.3 Expected result

The result of adding the 2 hex numbers would be:
31

1.1.4 Circuit with program loaded



1.1.5 Circuit at end of program



1.1.6 Result

As you can see the result of my circuit and my expected result for adding 2F and 02 together is identical, this means that the program is correctly adding the 2 numbers and storing it in Memory_B

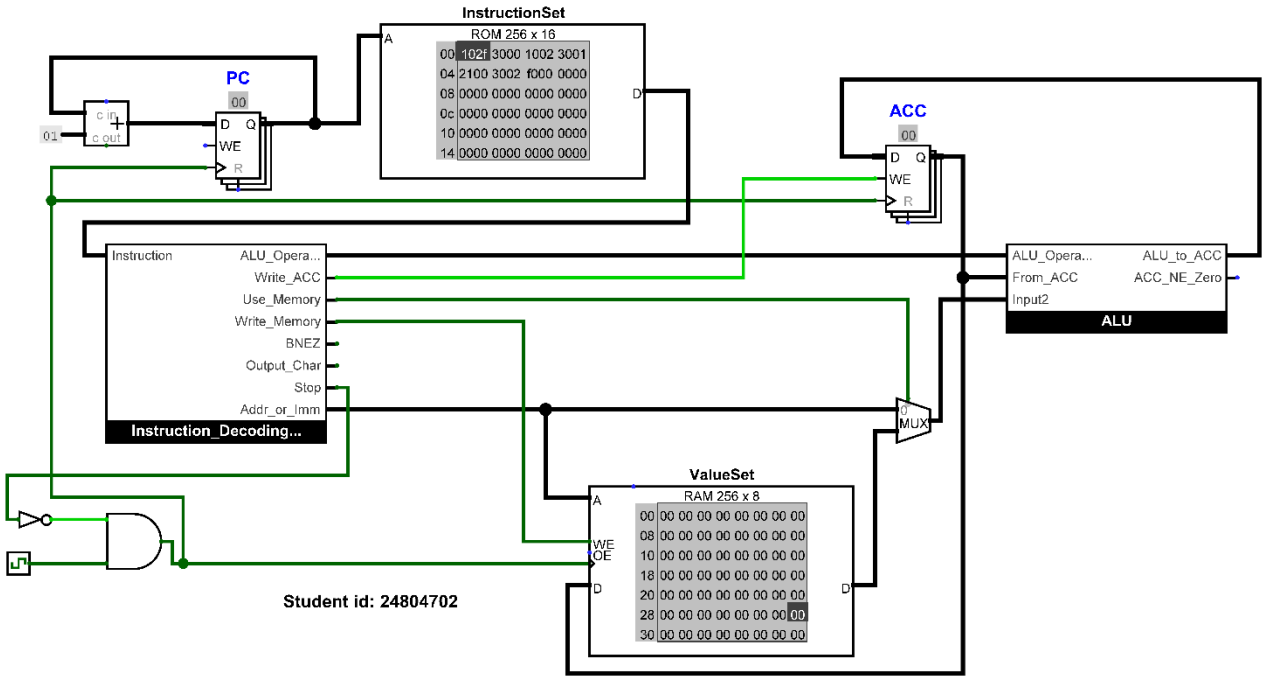
1.2 Completing the stop instruction

1.2.1 Memory Components

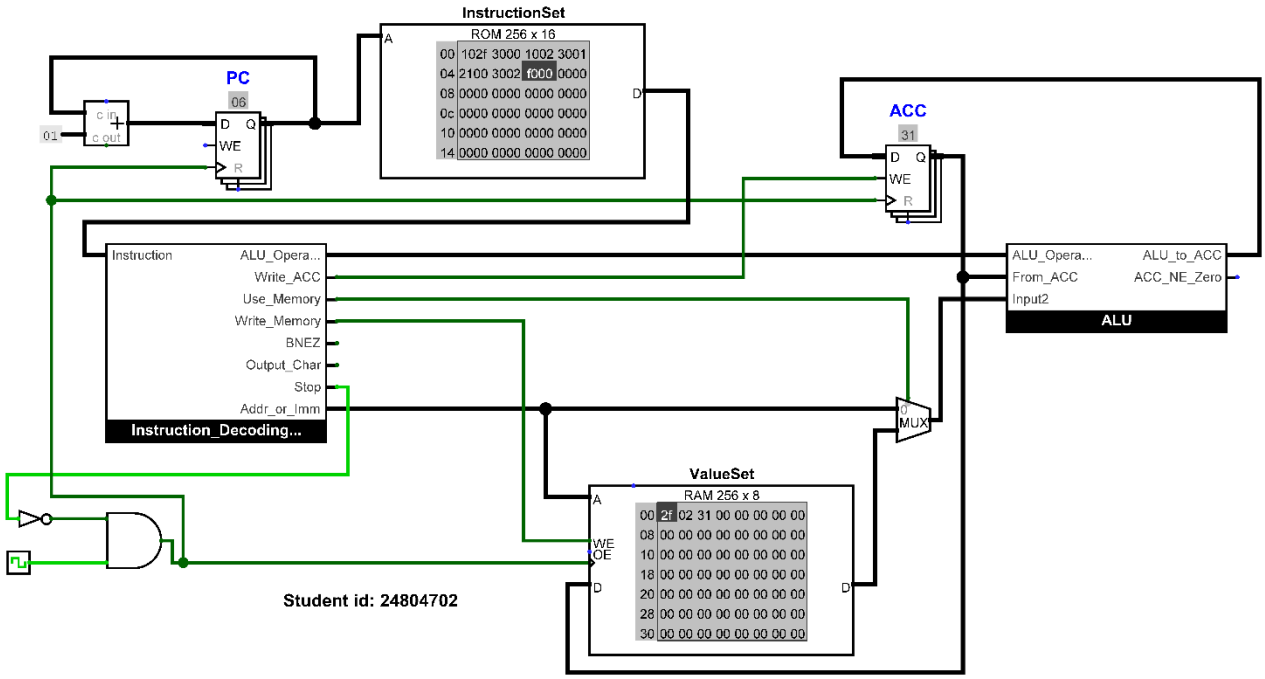
Memory component A stores the instructions for the CPU, the CPU executes the instruction and depending on what the instruction is it'll move onto the next instruction, or it'll stop.

Memory component B stores the values that is achieved through the CPU executing the instructions and stores it in an address.

1.2.2 Circuit at start of Test



1.2.3 Circuit at end of Test



1.2.4 Explanation

The reason why InstructionSet doesn't need a clock signal is because it depends on the Program Counter register to increment by one to know which address it needs to go to next however, the ValueSet does need a direct input from the clock because it depends on it to tell it when to store down memory.

1.3 Bitwise Operations

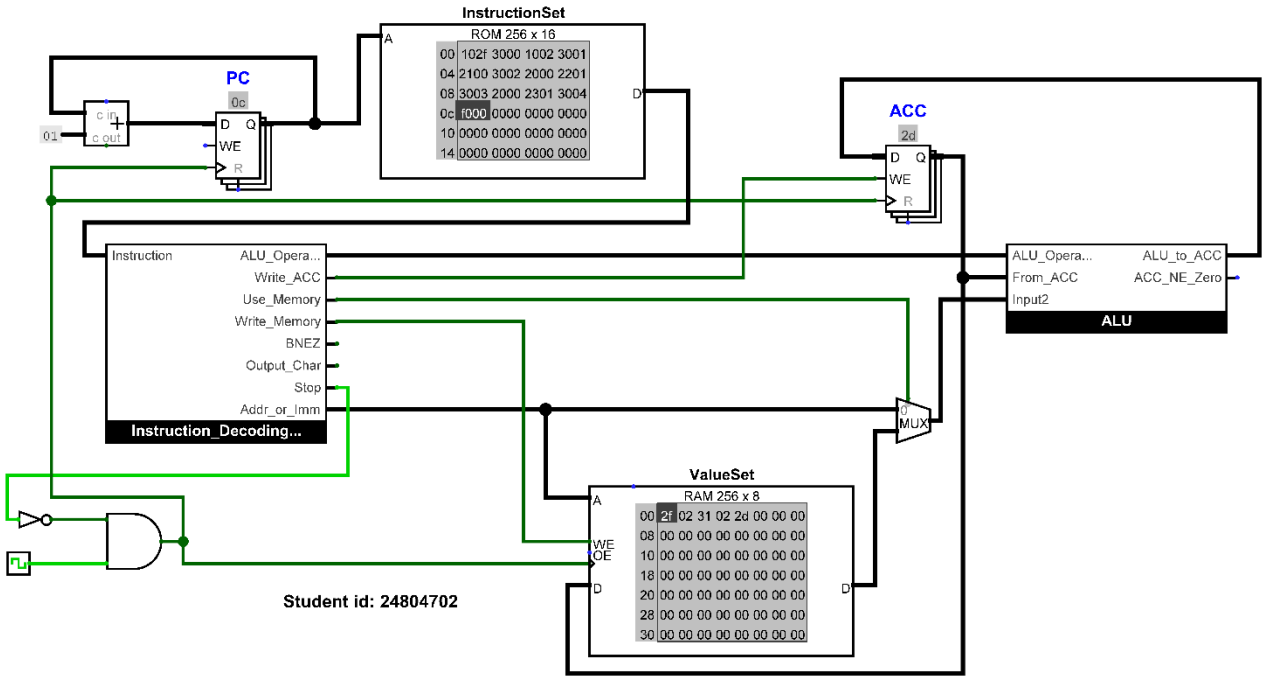
1.3.1 Test program

```
v2.0 raw
102F 3000   # 1. put first value to ACC and store to 00
1002 3001   # 2. put second value to ACC and store to 01
2100        # 3. add ACC value to value from address 00
3002        # 4. store ACC value to address 02
2000 2201   # 5. load from 00 to ACC, bitwise AND with value from 01
3003        # 6. store ACC value to address 03
2000 2301   # 7. load from 00 to ACC, bitwise XOR with value from 01
3004        # 8. store ACC value to address 04
f000
```

1.3.2 Expected result

	Hex	Binary
First number	2F	00101111
Second number	02	00000010
After bitwise AND	02	00000010
After bitwise XOR	2D	00101101

1.3.3 Result



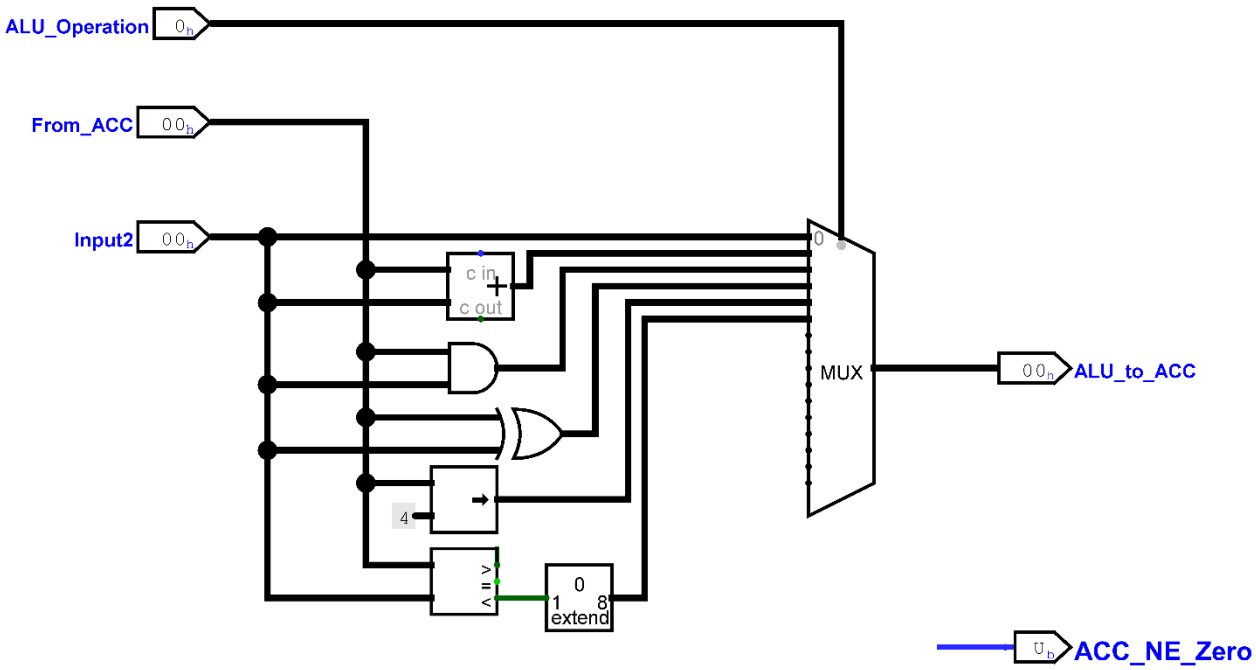
As we can see from the ValueSet the values stored in it perfectly match up with our hex values we got from the expected results table which indicate it's working perfectly.

1.3.4 Explanation

The main reason why I keep reusing the instruction of 2000 is because we need to keep loading the value of 2f in address 00 of the ram to be used in both the bitwise AND operation and bitewise XOR operation.

1.4 Amending the ALU

1.4.1 ALU circuit



1.4.2 Test Program

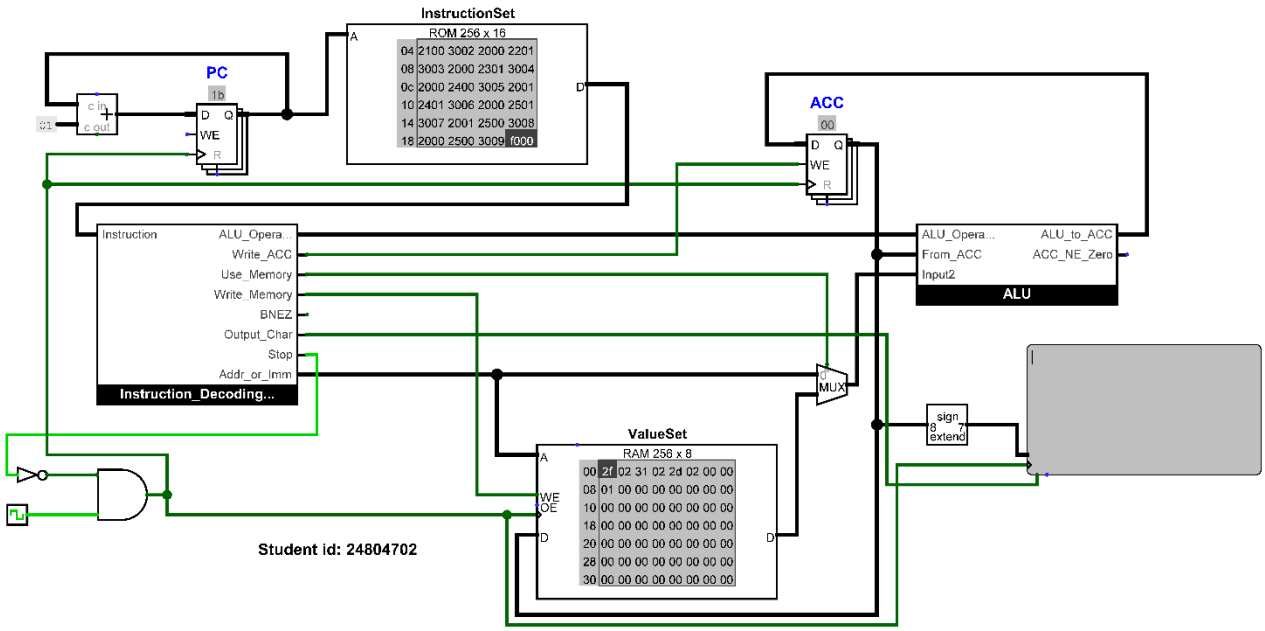
```
v2.0 raw
#--- ADDING OPERATION ---#
102F 3000   # 1. put first value to ACC and store to 00
1002 3001   # 2. put second value to ACC and store to 01
2100        # 3. add ACC value to value from address 00
3002        # 4. store ACC value to address 02
#--- AND OPERATION ---#
2000 2201   # 5. load from 00 to ACC, bitwise AND with value from 01
3003        # 6. store ACC value to address 03
#--- XOR OPERATION ---#
2000 2301   # 7. load from 00 to ACC, bitwise XOR with value from 01
3004        # 8. store ACC value to address 04
#--- LOGICAL RIGHT SHIFT OPERATION ---#
2000 2400   # 9. Load value from 00 to ACC, Logical right shift of value from
00
3005        # 10. store ACC value to address 05

2001 2401   # 11. Load value from 01 to ACC, Logical right shift of value from
01
3006        # 12. Store in address 06
#--- COMPARATOR OPERATION ---#
2000 2501   #13.  Load value from 00 to ACC, Comparator of value from 01
3007        #14.  Store ACC value in address 07

2001 2500   #15.  Load value from 01 to ACC, comparator of value from 01
3008        #16.  Store ACC value in address 08

2000 2500   #17.  Load value from 00 & 00
3009        #18.  Store value in address 09
#--- STOP OPERATION ---#
```

```
f000        # Stop command
```

1.4.3 Expected result

Test description	Value at 00	Value at 01	Right shift of first value	Set less than value
First number smaller	02	2f	00	01
First number bigger	2f	02	02	00
Both numbers the same	2f	2f	02	00

1.4.4 Circuit at end of program



1.4.5 Result

As we can see from the end of circuit images all the values, all the values match up with the expected values.

The right shift results are stored in address 05 and 06 where we can see as 02 and 00, this matches up to our expected result from our chart since 2f is being right shifted first and stored in address 05, next the 02 is right shifted which returns 00 in address 06.

The comparator results are all also correct and matching up with our expected results.

In the first command the first 2 values to be compared and stored in address 07 is 2f and 02, this returns a value of 00 since the second number is not bigger than the first number.

In the second command the values 02 and 2f is compared and returned with a value of 01 since 2f is bigger than 02.

In the third command the values 2f and 2f is compared and returned with 00 since 2f and 2f is the same value.

1.4.6 Explanation

To process the output of the comparator to be 8 bits I first started by trying to do it with splitters, whilst this method worked it would take up unnecessary space on the diagram so I tried to find a component which would do it compactly.

I had a look on the official Logisim evolution documentation to find if there was any component.

Once I found the comparator component on the official documentation which “transforms a value into a value of another bit width”(Bit Extender ,n.d), I added it to my circuit and had to do some small tweaks to get it to work.

When developing my program to be able to carry out the logical right shift operation and set less than operation I first tested the ALU to check what value the ALU_operation had to be to carry out the logical right shift and comparator.

Once I found out the value that it needed (4 for a logical right shift and 5 for the comparator) I followed the pattern the other commands followed and edited it to use 4 for my right shift and 5 for my comparator.

1.5 Output Device

1.5.1 Test Data

```
v2.0 raw
48 #H
45 #E
4C #L
4C #L
4F #O

20 #Space

52 #R
4F #O
```

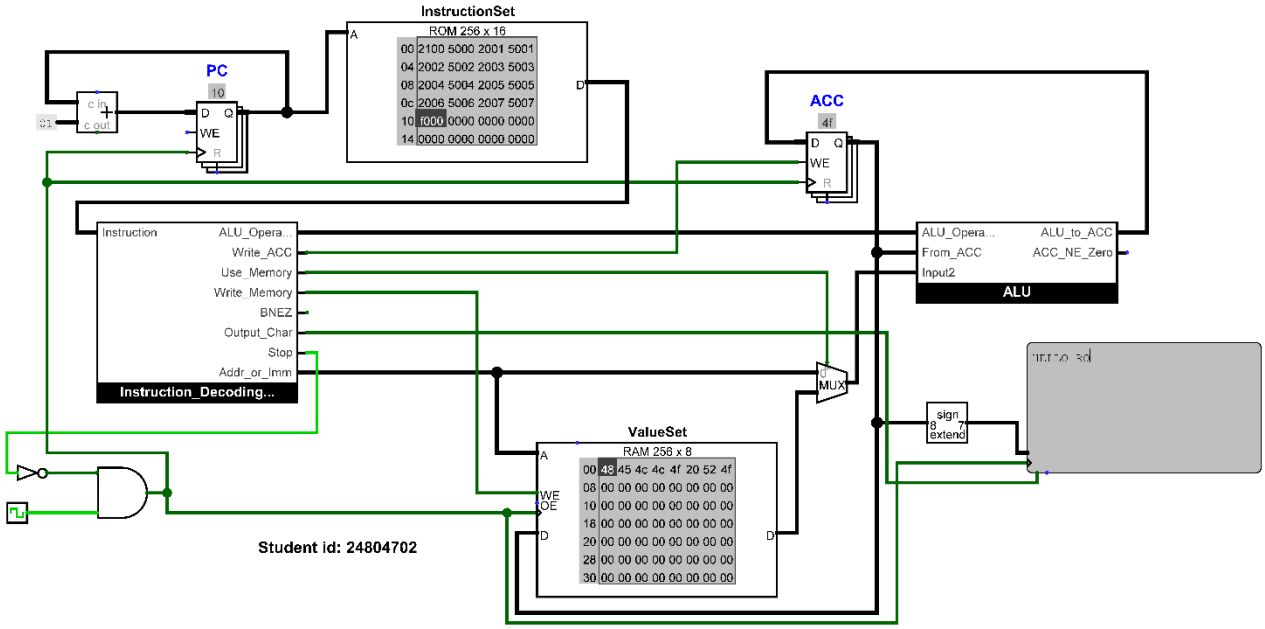
With this “HELLO RO” should be outputted to the terminal

1.5.2 Test Program

```
v2.0 raw
#H
2100
5000
#E
2001
5001
#L
2002
5002
```

```
#L
2003
5003
#O
2004
5004
#Space
2005
5005
#R
2006
5006
#O
2007
5007

f000        # Stop command
```

1.5.3 Circuit at end of program



1.5.4 Result

As we see here the result of the terminal is identical to the expected result

1.5.5 Explanation

The way the TTY is referenced in the official Logisim evolution documentation “This component implements a very simple dumb terminal.” (TTY, no date).

TTY first was “An electromechanical device that can be used to send and receive typed messages through various communications channel, in both point-to-point and point-to-multipoint configurations” (Wikipedia, 2024).

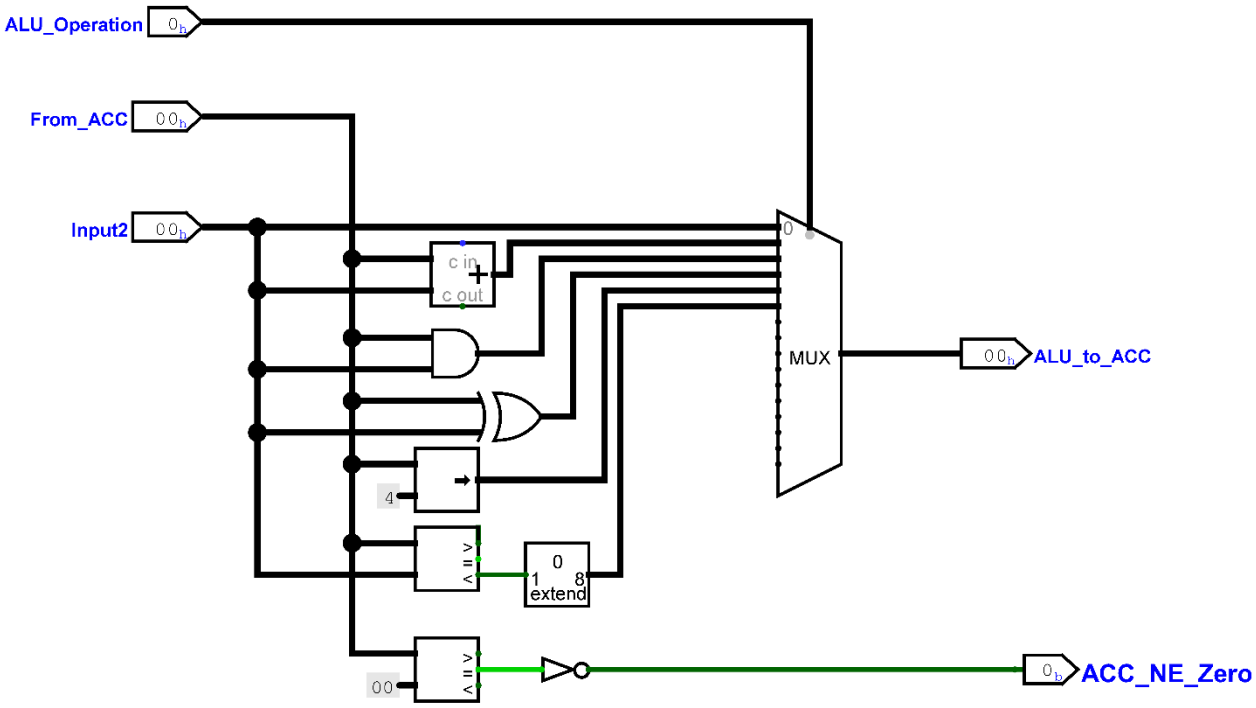
However, “With the development of early computers in the 1950s, teleprinters were adapted to allow typed data to be sent to a computer, and responses printed.” (Wikipedia,2024).

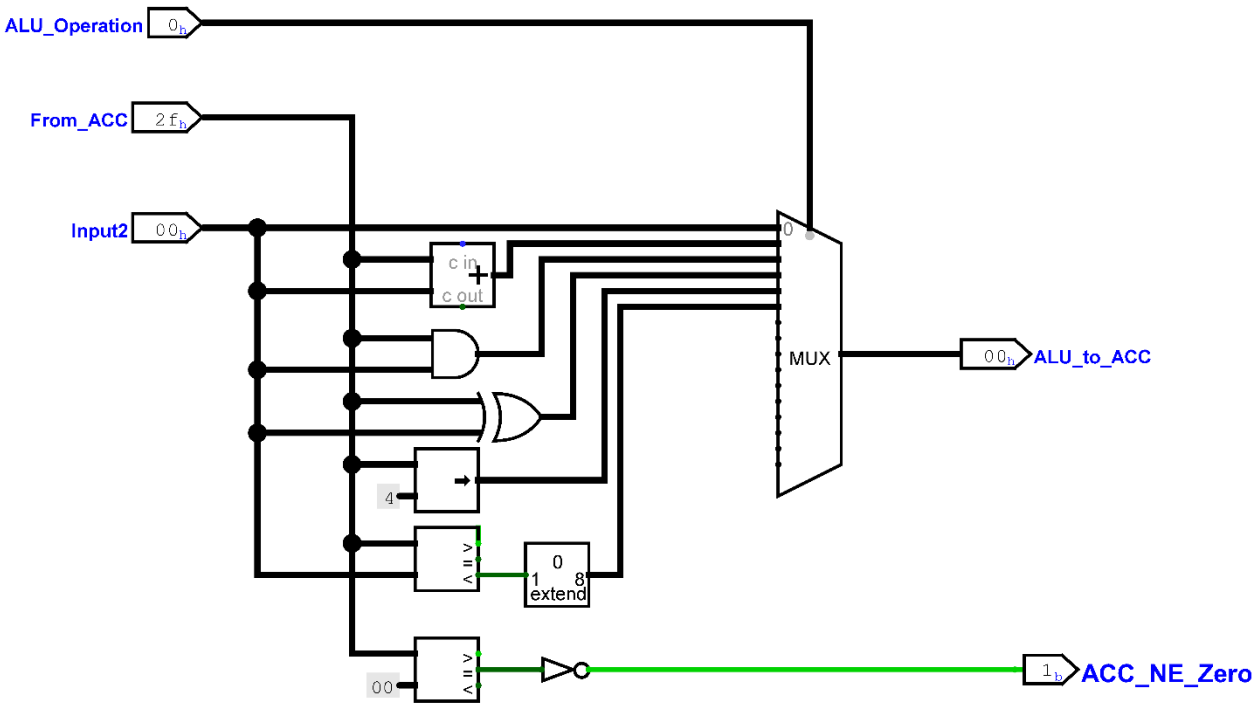
A method of clearing the TTY screen without using the clear input is to go back into the RAM data and rewrite the addresses with the hex code for space (20) and write it to the terminal.

1.6 Branch instruction

1.6.1 ALU circuit and testing

In order to test if my ACC_NE_Zero works I used one of my 2 digit student numbers as my test data and as we can see the ACC_NE_Zero displays 0 when it the value in From_ACC is equal to zero and displays one when it is not equal to zero.





1.6.2 Test program

```
v2.0 raw
#--- Values to be sorted ---#
1002 3000   # Stores the value of 02 in address 00
102f 3001   # Stores the value of 2f in address 01
#--- Branch segment ---#
2000 2501   #1.  Load value from 00 to ACC, Comparator of value from 01
400F        #2.  If accumulator is 01 then branch to exit code
#--- Sorting ---#
2000 3002   #1.  Load value from address 00 to ACC and store at address 02 in
memory B
2001 3000   #2.  Load value from address 01 to ACC and store at address 00 in
memory B
2002 3001   #3.  Load value from address 02 to ACC and store at address 01 in
memory B
2010 3002   #4.  Set the value at address 02 in memory B to be 00
#--- Stop ---#
f000        #Stop
```

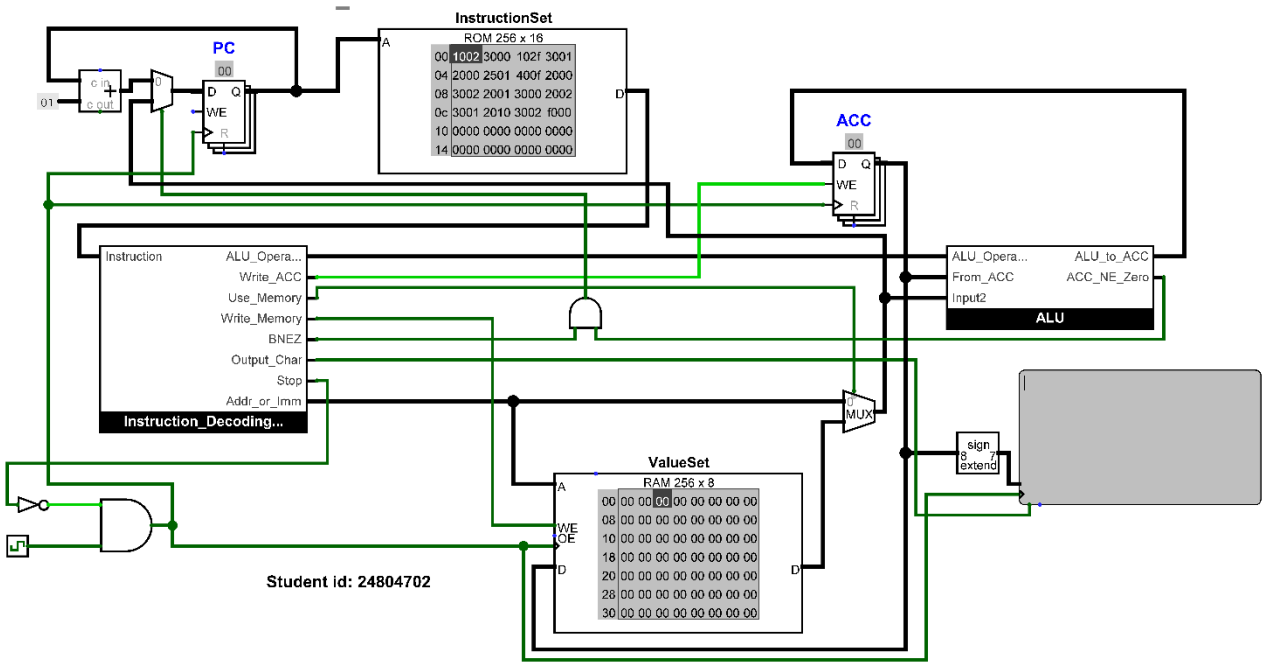
1.6.3 Expected Results

Test description	Value at 00	Value at 01	Set less than value
First number smaller	02	2f	02 2f
First number bigger	2f	02	02 2f
Both numbers the same	2f	2f	2f 2f

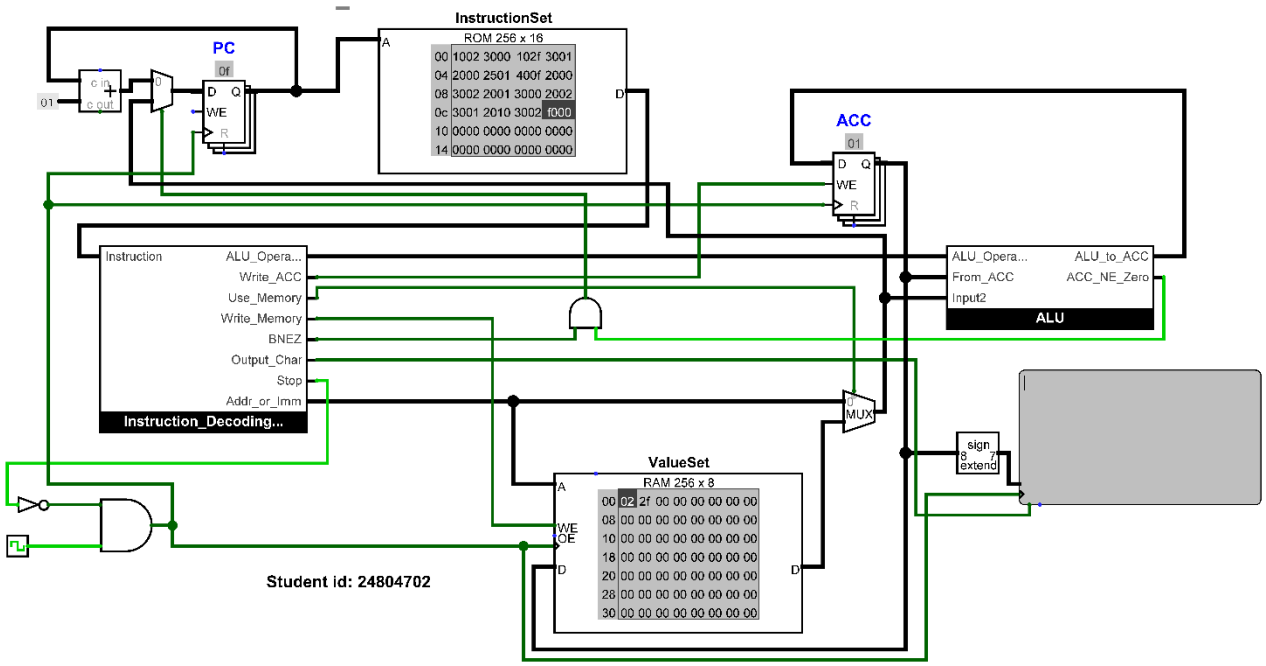
1.6.4 Results

First number smaller:

Start:



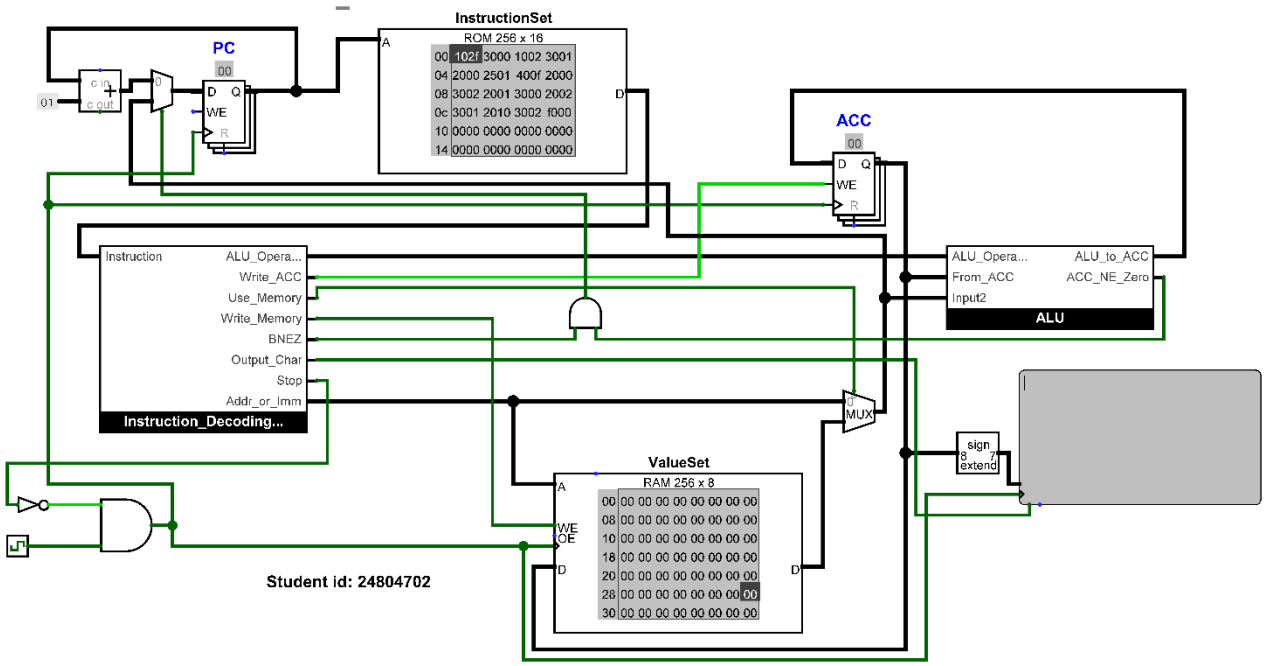
End:



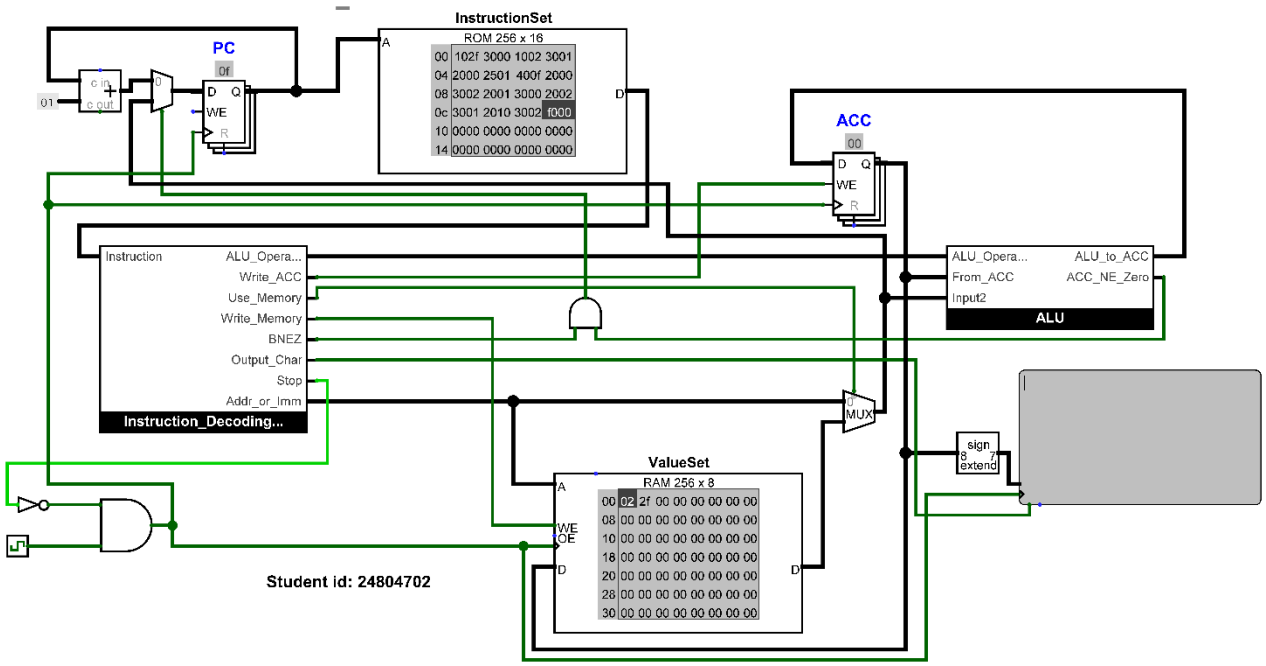
The ACC is set to 01 and the program branches to address 0f/15 which turns off the program so the list doesn't become unsorted

First number bigger:

Start:



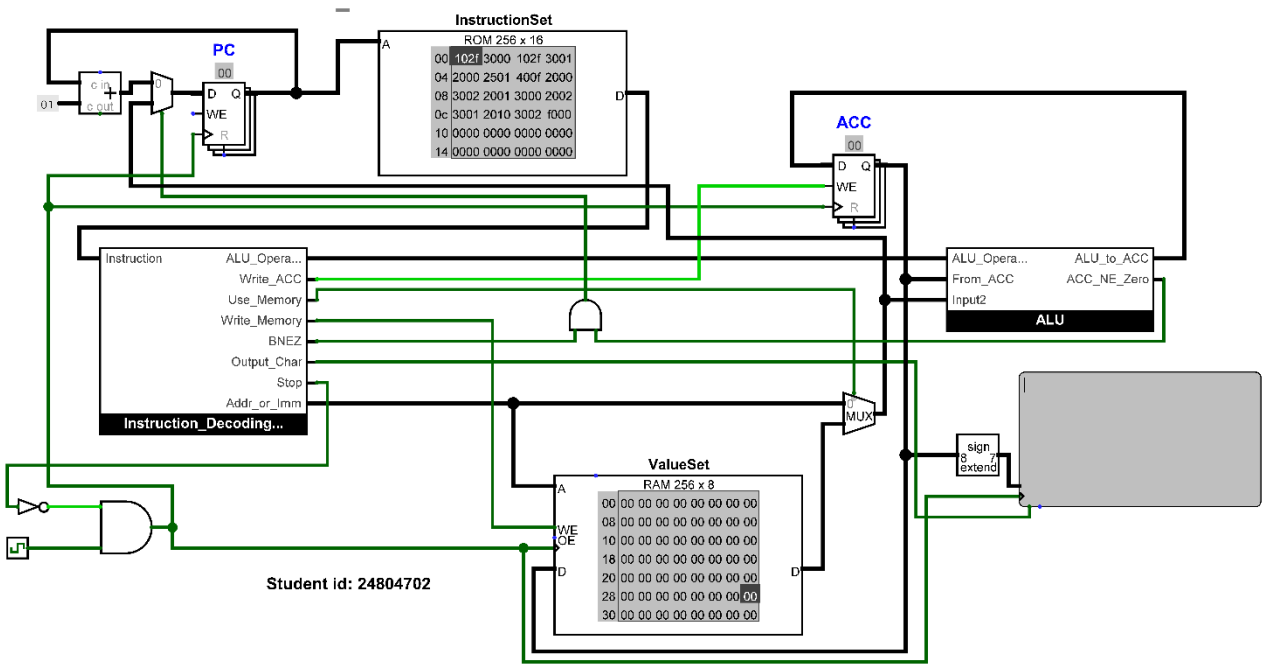
End:



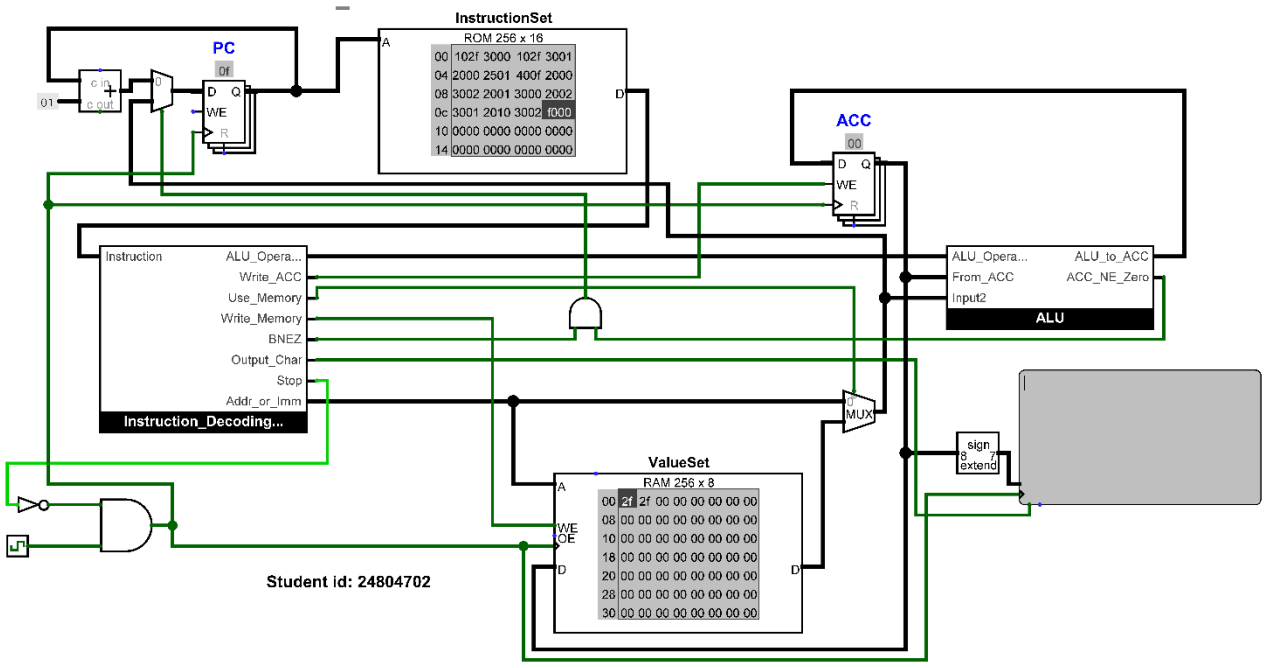
The ACC isn't set to 01 because From_ACC is smaller than input2 which prevents the branch command to occur and the program just continues and sorts the values

Both numbers are the same:

Start:



End:



Even though both numbers are the same and don't need sorting the program still sorts it because it's not equal to zero

1.6.5 Explanation

`Set ACC to 1 if value at address 00 is less than at 01 (slt)`

Use 2000 to load value of address 00 into accumulator and compare it with the value of address 01 with the command 2501.

Branch to instruction f000 at end of program if ACC is not zero

If the accumulator is not equal to zero and the branch is not equal to zero the command 4000 will branch to an address in ROM, we count all the commands in ROM to figure out where the stop command is stored and then replace the last 2 digits of the 4000 with, for example my stop command is at 15 so my branch is 400F.

Load value from address 00 to ACC and store at address 02 in memory B

Since we're loading the value of address 00 into ACC we'll be using 2000 and then storing it in the address 02 with the command 3002.

Load value from address 01 to ACC and store at address 00 in memory B

same as before but since we're working with a different address we change the last two digits so 2000 and 3002 is 2001 and 3000.

Load value from address 02 to ACC and store at address 01 in memory B

Same as before, 2002 3001.

Set the value at address 02 in memory B to be 00

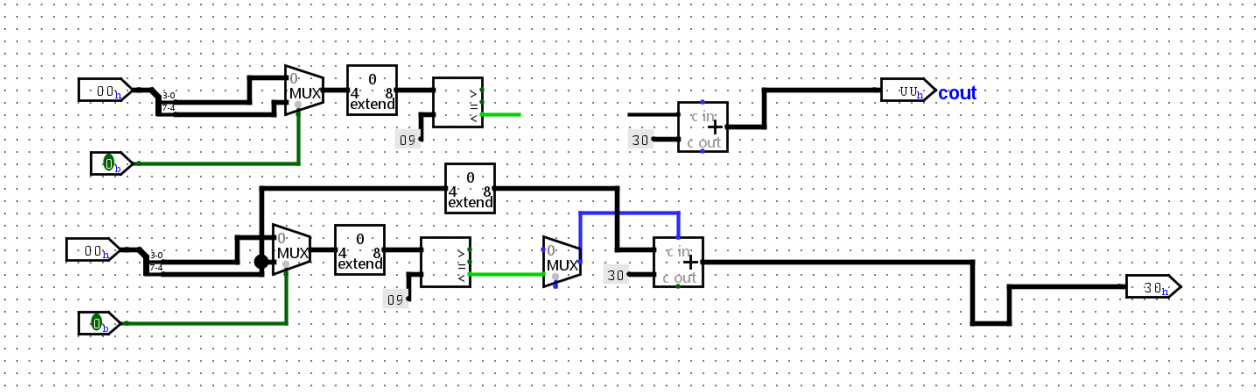
Almost the same as before however since we're changing the address to hold the value of 00 I grab it from an unused address and rewrite it using 3000, 2010 3002

For branching instead of using not equal to zero (bnez in RISC-V) we could the other 2 other features of the comparator where we can loop if its bigger than zero/input2 or less then input2, these commands also exist in RISC-V in the form as blt (branch if less than) and bgt (branch if greater than). To alter the circuit so it could use the blt and bgt then we'd need to swap the constant of 0 to the input2.

1.7 Loop

1.7.1 Displaying Hex Output

Beginner programs:



This is some of the converting ACC to ASCII tables I used at the start of development

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

ASCII table I used to check if my output was correct (Zack West, no date,

1.7.2 Running Loop

1.7.3 Explanation

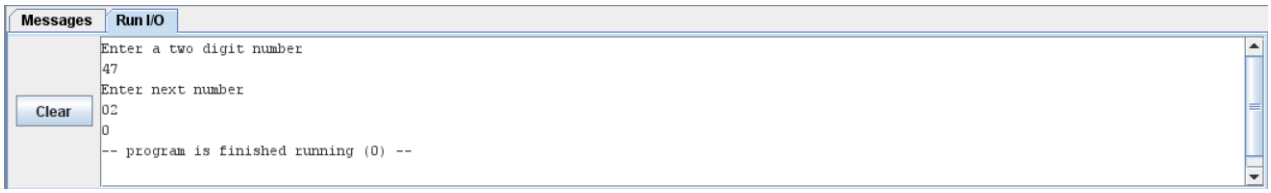
1.8 Game

2. RISC-V Assembly Language Programming

2.1 Test the existing program with minor adjustments

2.1.1 Starter Program

```
.data

enterMsg1: .string "Please use the last four digits of your student id as two
2-digit numbers \n"
enterMsg2: .string "Enter a two digit number\n"
enterMsg3: .string "Enter next number \n"

.text

###

# output the instruction text to the console
addi a7, zero, 4
la a0, enterMsg2
ecall

# read an integer from keyboard input and store in s0
addi a7, zero, 5
ecall
add s0, zero, a0


# output the text asking for the next number to the console
# then receive the input and store in s1

addi a7, zero, 4
la a0, enterMsg3
ecall

addi a7, zero, 5
ecall
add s1, zero, a0

## add the two values together and store in s2
##



# output the value from s2
add a0, s2, zero
addi a7, zero, 1
ecall


addi a7, zero, 10
ecall
```



Currently when I run the program and run the RISC-V Assembly code it compile perfectly however when I try to add my test data it only outputs the value 0.

2.1.2 Expected Result

For the expected result it should be 49 since 47 + 02 is 49 so the Run I/O should contain:
Enter a two digit number
47
Enter next number
02
49

2.1.3 Program code

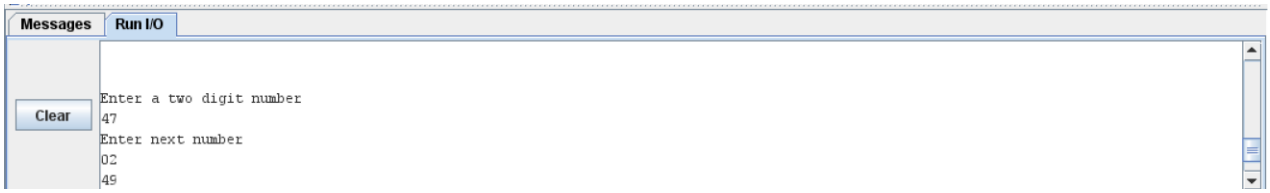
```
35  ## add the two values together and store in s2
36  ##
37
38  add s2,s0,s1 # s2 = input1 (a0) + input2 (a1)
```

```
43  add a0, s2, zero #Adds s2 to register a0 and out puts it in a0: a0 = s2 + 0
44  addi a7, zero, 1
45  ecall
```

```
.data

enterMsg1: .string "Please use the last four digits of your student id as two
2-digit numbers \n"
enterMsg2: .string "Enter a two digit number\n"
enterMsg3: .string "Enter next number \n"

.text

###

# output the instruction text to the console

addi a7, zero, 4
la a0, enterMsg2
ecall

# read an integer from keyboard input and store in s0
addi a7, zero, 5
ecall
add s0, zero, a0


# output the text asking for the next number to the console
```

```
# then receive the input and store in s1

addi a7, zero, 4
la a0, enterMsg3
ecall

addi a7, zero, 5
ecall
add s1, zero, a0

## add the two values together and store in s2
##

add s2,s0,s1 # s2 = input1 (a0) + input2 (a1)


# output the value from s2

add a0, s2, zero #Adds s2 to register a0 and out puts it in a0: a0 = s2 + 0
addi a7, zero, 1
ecall


addi a7, zero, 10
ecall
```

2.1.4 Result



As you can see here the program now correctly adds the values in a0 (47) and a1 (02)

2.2 Improve the messages that the program outputs

2.2.1 Program Code

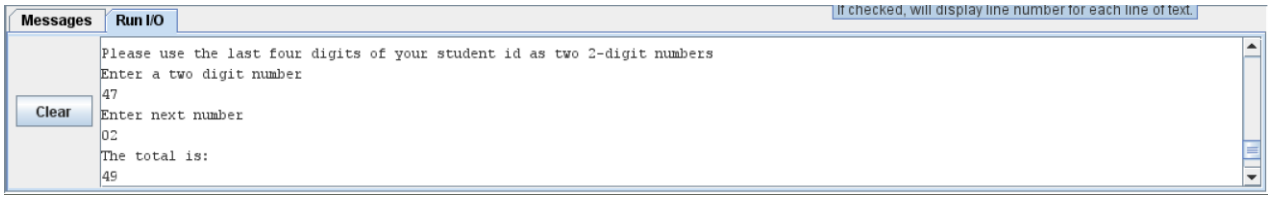
```
1  .data
2
3  enterMsg1: .string "Please use the last four digits of your student id as two 2-digit numbers \n"
4  enterMsg2: .string "Enter a two digit number\n"
5  enterMsg3: .string "Enter next number \n"
6  outputMsg: .string "The total is:\n"
12 # output the instruction text to the console
13
14 addi a7, zero, 4 # outputs "Please use the last four digits of your student id as two 2-digit numbers" and makes a new line
15 la a0, enterMsg1
16 ecall
```

```
45  # output the value from s2
46  addi a7,zero,4  #Outputs the string msg
47  la a0, outputMsg
48  ecall
```

```
.data

enterMsg1: .string "Please use the last four digits of your student id as two
2-digit numbers \n"
enterMsg2: .string "Enter a two digit number\n"
enterMsg3: .string "Enter next number \n"
outputMsg: .string "The total is:\n"

.text

###

# output the instruction text to the console

addi a7, zero, 4 # outputs "Please use the last four digits of your student id
as two 2-digit numbers" and makes a new line
la a0, enterMsg1
ecall

addi a7, zero, 4
la a0, enterMsg2
ecall

# read an integer from keyboard input and store in s0
addi a7, zero, 5
ecall
add s0, zero, a0


# output the text asking for the next number to the console
# then receive the input and store in s1

addi a7, zero, 4
la a0, enterMsg3
ecall

addi a7, zero, 5
ecall
add s1, zero, a0

## add the two values together and store in s2
##

add s2,s0,s1 # s2 = input1 (a0) + input2 (a1)
```

```
# output the value from s2
addi a7,zero,4  #Outputs the string msg
la a0, outputMsg
ecall

add a0, s2, zero #Adds s2 to register a0 and out puts it in a0: a0 = s2 + 0
addi a7, zero, 1
ecall

addi a7, zero, 10
ecall
```

2.2.2 Results



2.2.3 Explanation

When developing a program that a user will use it is important to create clear instructions and comments that the user can read and understand otherwise the user may use the program incorrectly and not get the desired result.

2.3 Enhancing the functionality of the program

2.3.1 Program Code

```
.data
enterMsg1: .string "Please use the last four digits of your student id as two 2-digit numbers \n"
enterMsg2: .string "Enter a two digit number\n"
enterMsg3: .string "Enter next number \n"
outputMsg: .string "The total is:\n"
andMsg: .string "\nThe total of the values in a bitwise and operation is:\n"
xorMsg: .string "\nThe total of the values in a bitwise xor operation is:\n"
# ------------------------------------------------- mathematical equations ------------------------------------------------- #
## add the two values together and store in s2
add s2,s0,s1 # s2 = input1 (a0) + input2 (a1)

## bitwise and operation on s0 and s1
and s3,s0,s1

## bitwise xor operation on s0 and s1
xor s4,s0,s1
```

```
#------ Outputs the add result ------#
addi a7,zero,4  #Outputs the string msg
la a0, outputMsg
ecall

add a0, s2, zero #Adds s2 to register a0 and out puts it in a0: a0 = s2 + 0
addi a7, zero, 1
ecall
```

```
#------ Outputs the and result ------#

addi a7, zero,4 #Outputs variable andMsg
la a0, andMsg
ecall

add a0,s3,zero #Adds s3 to register 01 and outputs
addi a7,zero,1
ecall
```

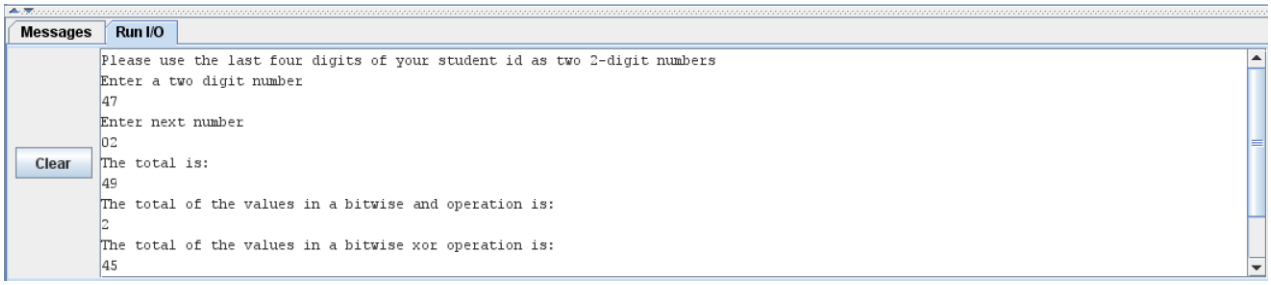
```
#------ Outputs the xor result ------#

addi a7,zero,4 #Outputs xorMsg
la a0, xorMsg
ecall

add a0,s4,zero #Adds s4 to register 01 and outputs
addi a7,zero,1
ecall
```

2.3.2 Expected Result

Expected Result:

The total is:
49
The total of the values in a bitwise and operation is:
2
The total of the values in a bitwise xor operation is:
45

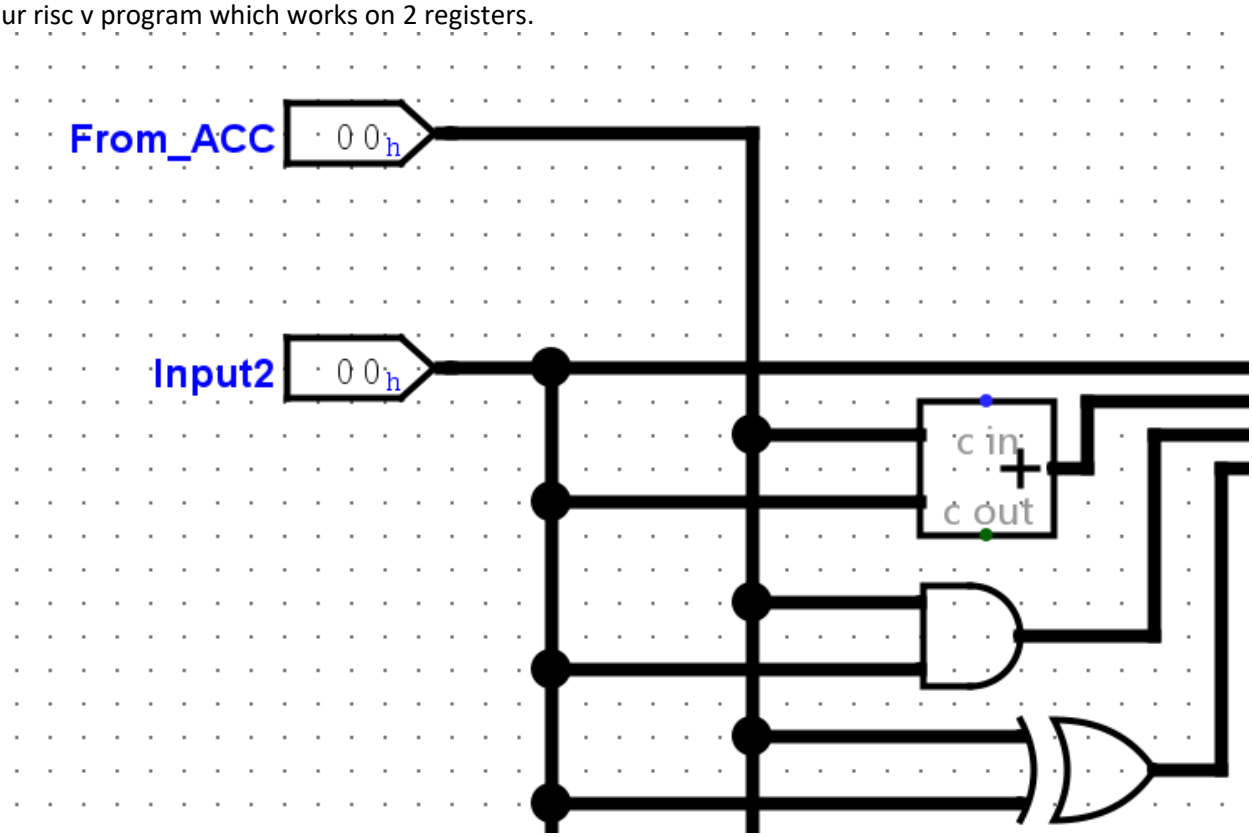
2.3.3 Result



As you can see from the results, they're the same as the expected results which means that the program is working perfectly.

2.3.4 Explanation

'add', 'and' and 'xor' are all logical operations which all perform the same mathematical calculations. They're based on the logic gates we have also developed in our circuit which work on 2 inputs similar to our risc v program which works on 2 registers.



To be able to use these operations we use 2 to pass through to the alu, which ever number the operation is on (e.g. add is on one so it would be 21 so far) and then the address of the value we want to perform the calculation on (for example address 00, a completed working code for adding a value from address 00 is 2100).

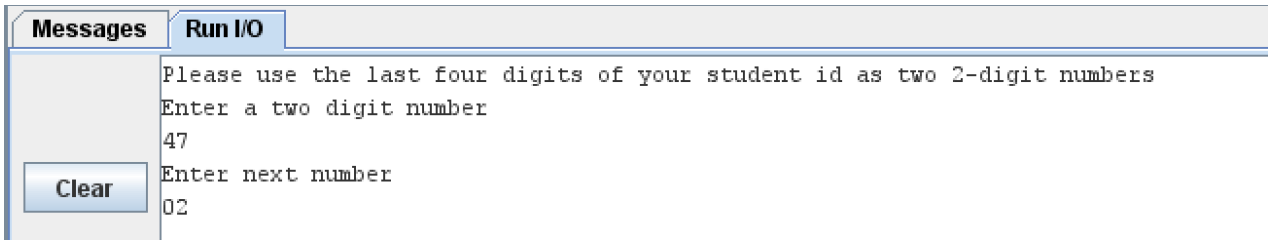
2.4 Adding a subroutine

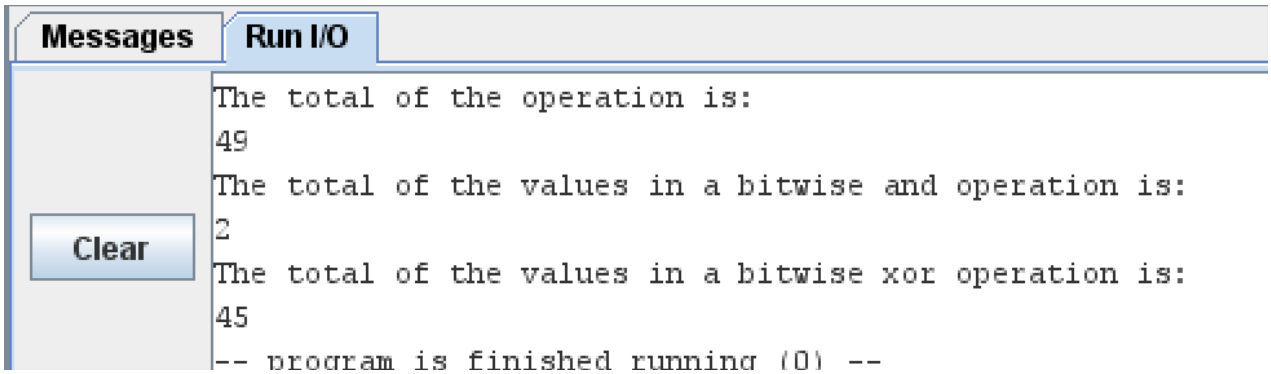
2.4.1 Subroutine code

```
51  # --- Add --- #
52  addi a1, s2, 0
53  la a0, outputMsg
54  jal output
55  # --- And --- #
56  addi a1, s3, 0
57  la a0, andMsg
58  jal output
59  # --- Xor --- #
60  addi a1, s4, 0
61  la a0, xorMsg
62  jal output
```

```
69  output:
70          addi a7,zero, 4
71          ecall
72
73          addi a7,zero, 1
74          add a0, a1,zero
75          ecall
76          ret
```

2.4.2 Result





2.4.3 Explanation

When using subroutines in RISC-V assembly it's extremely important to use `jal` (jump and link) and `ret` (return).

`jal` is used where you want the subroutine to be run and add the label of the subroutine after the `jal` but on the same line, to use it correctly you'd need to first declare the actual subroutine at the bottom of the RISC-V file after the exit ecall operation (otherwise the procedure could be called indefinitely).

`ret` is used in the subroutine to state when the subroutine is ready to exit, after the subroutine is finished the program will jump back to where the `jal` line was executed and continue from there.

An example of how to actually properly use `jal` and `ret` in RISC-V assembly would be:

```
.data
outputMsg: .string "Hello world!"
.text
la a0, outputMsg
jal print

exit:
      addi a7,zero 10
      ecall

print:
      addi a7,zero,4
      ecall
      ret
```

If we didn't use jal and ret the RISC-V program wouldn't be able to run and the program would either crash or run into a logic error.

2.5 Swapping Numbers

2.5.1 Program Code

```
# --- Sorting --- #
slt t1,s0,s1 #if (s0 < s1) {t1 = 1} else {t1 = 0}
bnez t1,sorted #if (t1 != 0) { sorted(); } else
add s2,s0,zero
add s0,s1,zero
add s1,s2,zero
```

```
j sorted
```

2.5.2 Expected Result

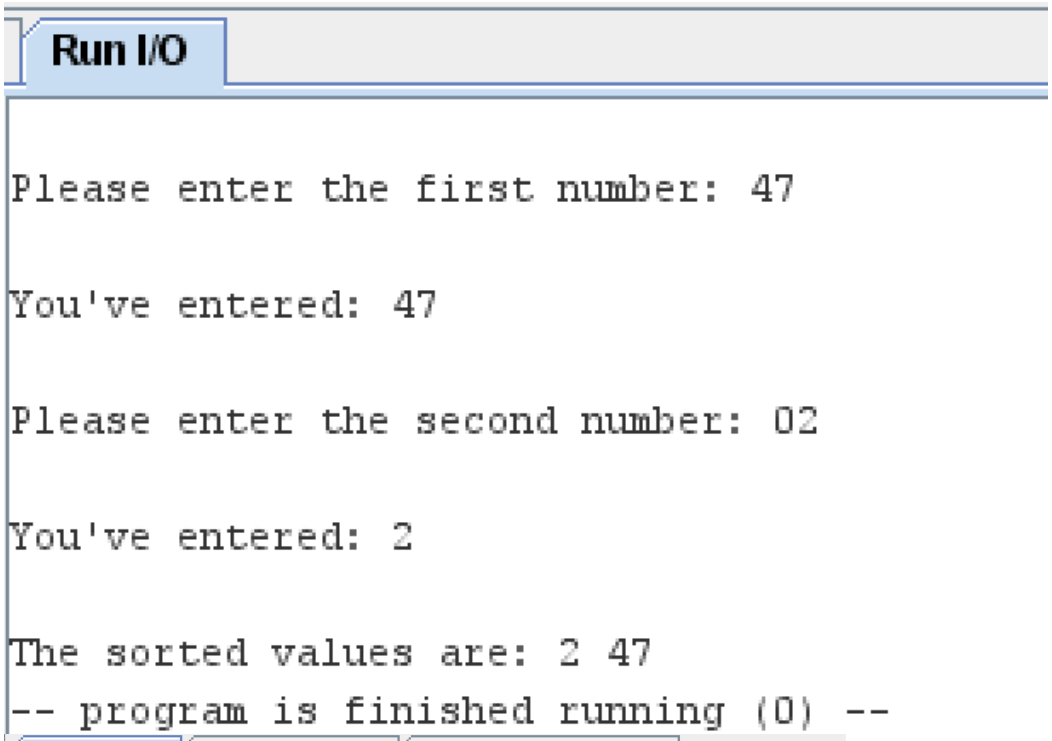
Test data:
47
02
The values are not in the right order:
47,02 -> 02 47

The values are already in the right order:
02 , 47 -> 02 47

The two values are the same:
47, 47 -> 47 47

2.5.3 Result

The values are not in the right order:



Registers | Floating Point | Control and Status

Name	Number	Value
zero	0	0
ra	1	4194384
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	0
s0	8	2
s1	9	47
a0	10	47
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	10
s2	18	47

Here since the values aren't in the correct order instead of immediately branching to the sorted subroutine because t1 isn't 0 so the value of s0 is s2

The values are already in the right order:

Run I/O

```
Please enter the first number: 02

You've entered: 2

Please enter the second number: 47

You've entered: 47

The sorted values are: 2 47
-- program is finished running (0) --
```

Registers | Floating Point | Control and Status

Name	Number	Value
zero	0	0
ra	1	4194384
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	1
t2	7	0
s0	8	2
s1	9	47
a0	10	47
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	10

Since the values are already in the right order the t1 isn't set to 0 which immediately branches to the sorted code block without copying the values to any registers

The two values are the same:

Run I/O

```
Please enter the first number: 47

You've entered: 47

Please enter the second number: 47

You've entered: 47

The sorted values are: 47 47
-- program is finished running (0) --
```

Registers | Floating Point | Control and Status

Name	Number	Value
zero	0	0
ra	1	4194384
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	0
s0	8	47
s1	9	47
a0	10	47
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	10
s2	18	47

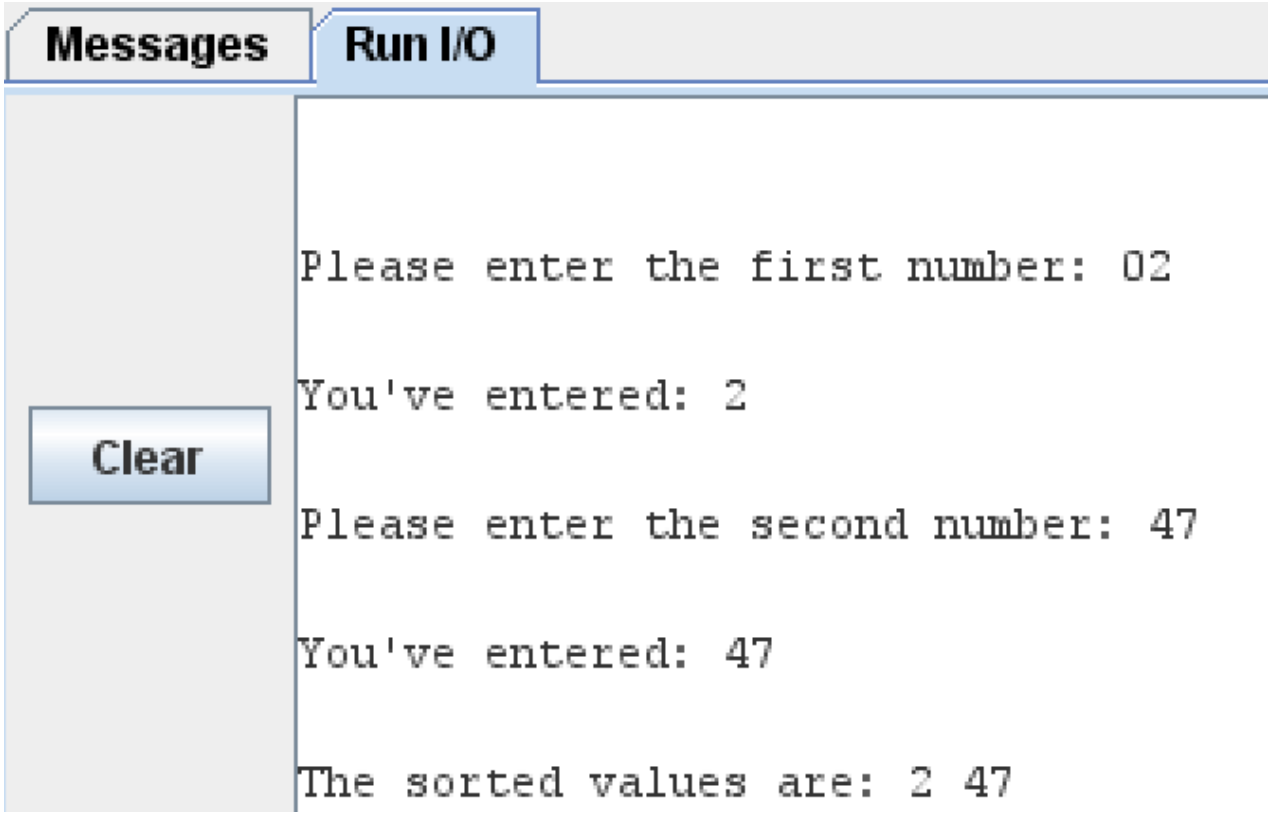
2.5.4 Alternative approach

```
35  # --- Sorting --- #
36  slt t1,s0,s1 #if (s0 < s1) {t1 = 1} else {t1 = 0}
37  bnez t1,sorted #if (t1 != 0) { sorted(); } else
38  mv s2,s0
39  mv s0,s1
40  mv s1,s2
41  j sorted
```

For my alternative method I decided to look into a RISC-V ebook since it could offer more of an insight into how to copy values from one register into another, according to Stephen Smith "("mv x5, x6" actually translates to "addi x5,x6,0"…)" (2024,pp. 36).

From this I can infer that another way to copy values from one register to another is to do "mv {register1}, {register2}", this would have the same output as doing "addi {register1}, {register2}, 0".

The values are in the correct order:



Registers | Floating Point | Control and Status

Name	Number	Value
zero	0	0
ra	1	4194384
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	1
t2	7	0
s0	8	2
s1	9	47
a0	10	47
a1	11	0

The values are not in the correct order:

Messages | Run I/O

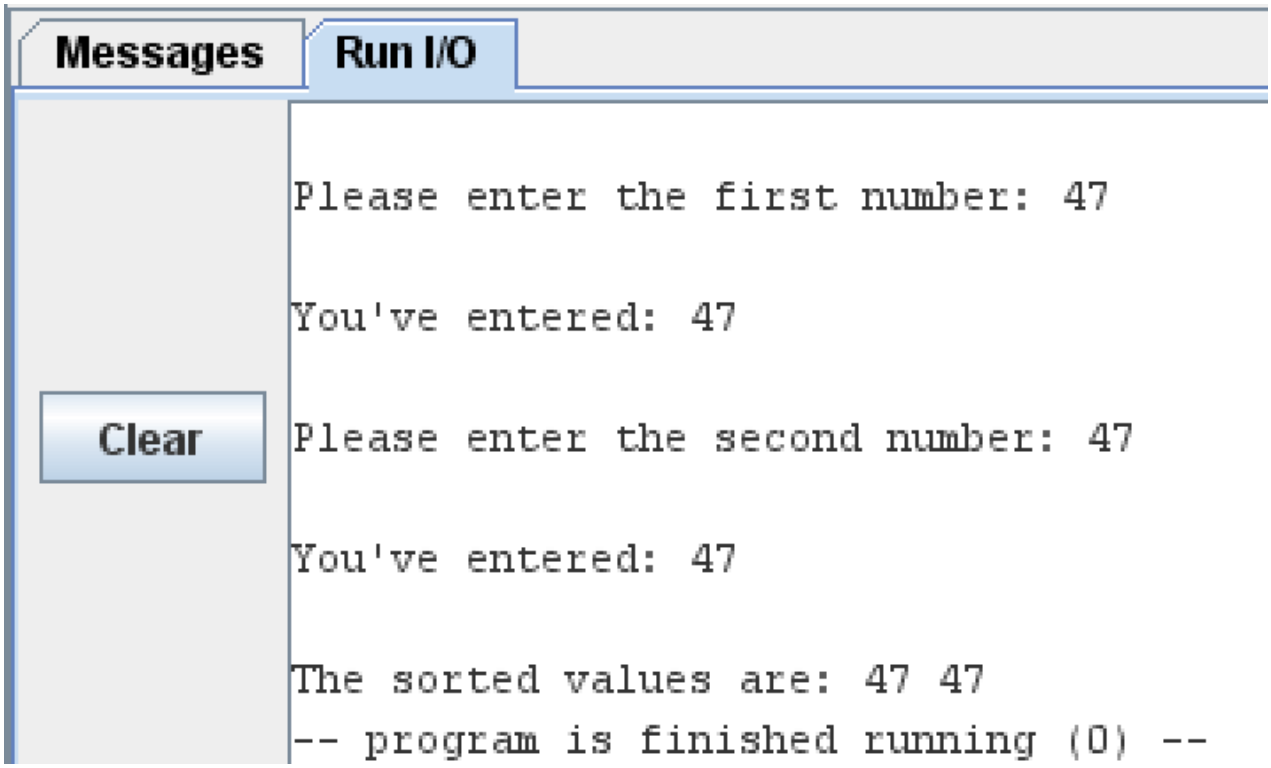
Clear

```
Please enter the first number: 47

You've entered: 47

Please enter the second number: 02

You've entered: 2

The sorted values are: 2 47
-- program is finished running (0) --
```

Registers | Floating Point | Control and Status

Name	Number	Value
zero	0	0
ra	1	4194384
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	0
s0	8	2
s1	9	47
a0	10	47
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	10
s2	18	47

Two values are the same:



Registers | Floating Point | Control and Status

Name	Number	Value
zero	0	0
ra	1	4194384
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	0
t2	7	0
s0	8	47
s1	9	47
a0	10	47
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	10
s2	18	47

2.5.5 Explanation

The mv command is a pseudoinstruction used to make the code more readable for the developers on RISC-V however both “add {register1}, {register2},zero” and “mv {register1}, {register2}” get translated into the same piece of code once it’s assembled, in code you can use both with no difference however some could argue that “mv” would be more readable than “add”.

2.6 Counting up with a loop

2.6.1 Loop Code

```
60  loop:
61          jal calling
62          addi s0,s0,1
63          slt t1,s0,s1
64          bnez t1,loop
65          j loopdone
66  loopdone:
67          jal calling
68          j exit
69
70  calling:
71          addi a7,zero,1
72          add a0,zero,s0
73          ecall
74
75          addi a7,zero,11
76          addi a0,zero, ' '
77          ecall
78          ret
```

2.6.2 Results

```
Please enter the first number: 02

You've entered: 2

Please enter the second number: 47

You've entered: 47
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
-- program is finished running (0) --
```


```
Please enter the first number: 47

You've entered: 47

Please enter the second number: 47

You've entered: 47
47 48
-- program is finished running (0) --
```

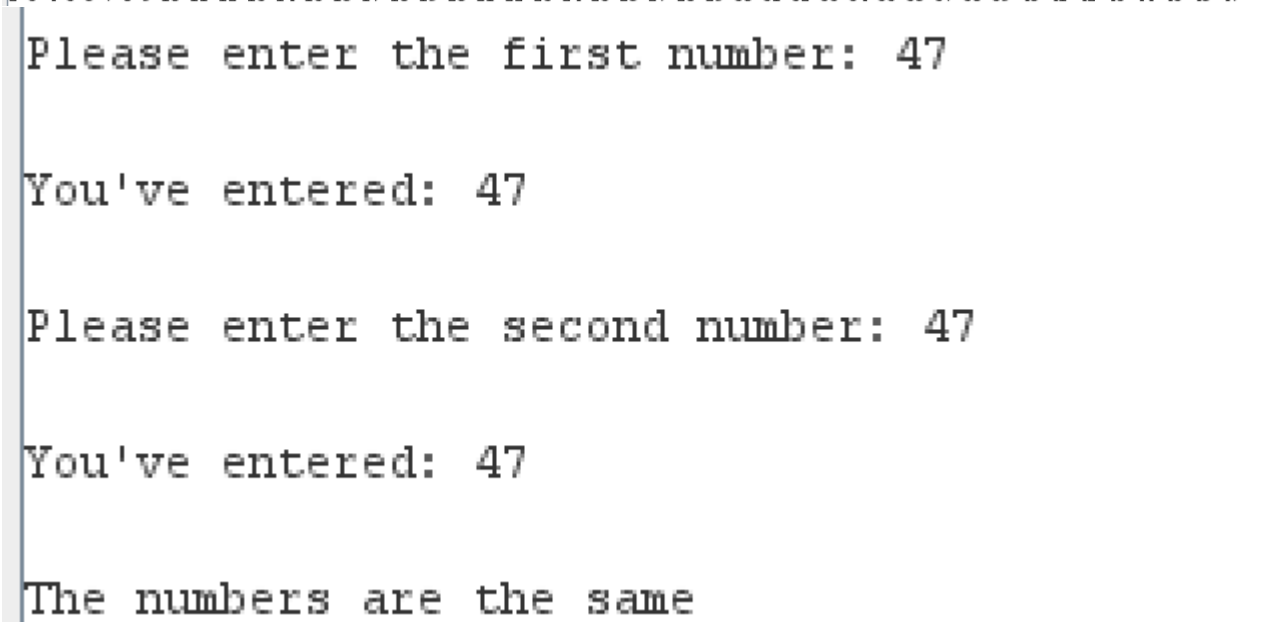
2.6.3 Alternative approach

```
38  beq s0,s1,equal #if (s0 == s1) {equal();}
39  slt t1,s0,s1 #if (s0 < s1) {t1 = 1} else {t1 = 0}
40  bnez t1,loop #if (t1 != 0) { sorted(); } else
41  mv s2,s0
42  mv s0,s1
43  mv s1,s2
44  j loop
```

```
88  equal:
89          la a0, equalMsg
90          jal strPrint #cout << equalMsg;
91          j exit #exit();
```

```
Please enter the first number: 02

You've entered: 2

Please enter the second number: 47

You've entered: 47
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
```

```
Please enter the first number: 47

You've entered: 47

Please enter the second number: 02

You've entered: 2
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
```



2.6.4 Explanation

Before the program would only check which of the 2 numbers entered was the smallest and then call a loop to count up from it until it reaches the highest number, however now by introducing a beq (branch if equal) the program will jump to a subroutine called equal and output the message “The numbers are the same”

2.7 Making use of arrays

2.7.1 Pseudocode

```
.data
Create array and give it the space of 100 bytes
…
.text
Load address into a register

Make a copy of the original register address
…
add 1 to the 2nd input in order to be able to store the last number in the
array before branching to loop
…
after sorting take away one from smaller register and add one to bigger
register

Loop:
add 1 to a register to keep as a counter

Store smaller register register to array

add 1 to the smaller register

Add 4 to register with array to move onto the next index in array

Set t1 to 0 if first input is smaller than s1

If t1 is not equal to zero then jump to label loop

Jump to loopdone

Loopdone:
If register with 0 is greater than register with length of array then branch
to label exit

Load copy array address to a register

Output the element

Add 1 to the register with 0

Increment the copy array by 4

Jump to exit
```

2.7.2 Expected Results

For my test data I'll be using 02 and 47 from my student id number, I will test for when the input is: smaller to bigger, bigger to smaller, and equal.

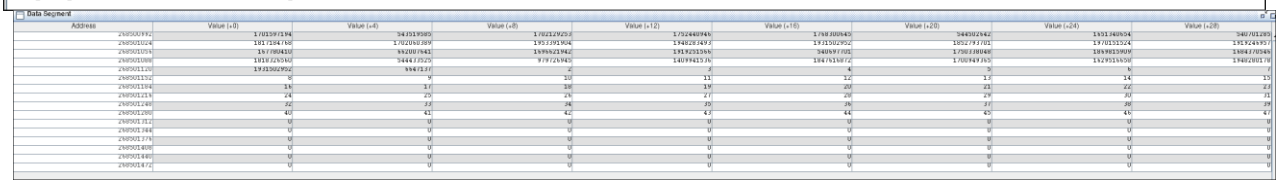
For smaller to bigger and bigger to smaller I expect to see:
2, 3, 4, 5, …, 47
In my I/O and in my data segment.

For when they're both the same I expect to see:
"The numbers are the same" in the I/O and no numbers in an array in my data segment

2.7.3 Result

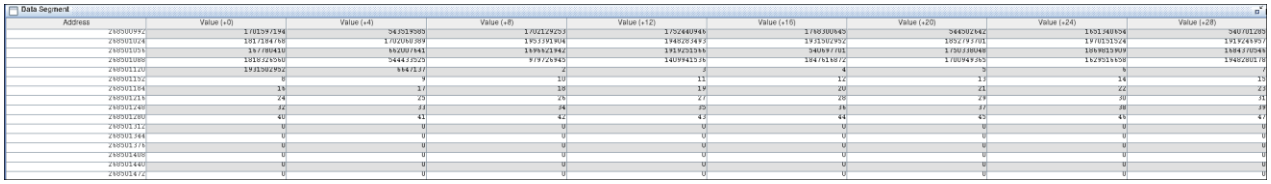
First number is smaller:

```
Please enter the first number: 02

You've entered: 2

Please enter the second number: 47

You've entered: 47
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
-- program is finished running (0) --
```



First number is bigger:

```
Please enter the first number: 47

You've entered: 47

Please enter the second number: 02

You've entered: 2
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
-- program is finished running (0) --
```



Both are the same:

```
Please enter the first number: 47

You've entered: 47

Please enter the second number: 47

You've entered: 47

The numbers are the same
-- program is finished running (0) --
```

As we can see from our actual results they match what I put in my expected results in both the I/O segment and the data segment

2.7.4 Explanation

General explanation:
An array is a linear data structure which stores data at continuous memory locations (GeeksForGeeks, 2024), e.g. 2 at index 0, 3 at index 1, 0 at index 2 ; Whilst the values entered doesn't have to be continuous the index value is.

Start of the program:
In order to use arrays you first need to allocate dedicated memory to them so that they do not get overwritten with other data, in my RISC-V code I decided to do this in my .data segment by creating my array name, choosing what type of data I want it to hold (words or bits) and then the amount of space allocated in the program however, you could also do that in the .text segment by loading upper immediate into a variable which holds the variable address.

Loop:
In my loop I set up a counter register (a3) to keep count of how many times the loop repeats in order to work as an end counter in loop done so the subroutine doesn't keep repeating until the end of the actual array and keep outputting 0 after all the numbers have been outputted.

To store the values in the array I used store word with my register of s0 and would add 1 to s0 and 4 to a1 to move the index of a1 to the next available one until the loop finished

Loopdone:
At the start of my loopdone subroutine I set up bge a2,a3,exit which would work in a similar fashion to a for loop in high level languages, a2 would be a variable that initially have 0 stored in it and a3 was the counter I used in loop which kept count of how many times the loop subroutine repeated itself, everytime loopdone loops I added 1 onto a2 until it reached the same value or greater of a3 which would then jump to the exit subroutine I made before.

In my code I made a copy of the blank array in my register a5 in order to be able to loop through it again when outputting each element in the array, I had to do this because when I was trying to output each element in the original array it would continue from the last address where I stored the last number and output 0 because the indexes didn't have any values.

I would load a value from a5 into the register a0 and then ecall it to be printed into the console and then add 4 to a5 to move to the next index.

Circuit:
In order to be able to use arrays in my circuit I'd first need to create a command to store a certain amount of bits into my ram and actually implement the option to use it in my Instruction_Decoding_Unit and use variations of the command 2000 in order to be able to grab and store values in the array.

To help me develop this assembly code I used a similar method to Sarah Harris to be able to loop thought my array and output it (DDCA Ch6 – Part 10:Arrays, 2022,3:48).

And How to use an array in RISC-V Assembly by Beep (2020) to be able to understand how to create arrays in the .data and in load it into a register.

2.8 Designing a game

3. References

TTY (n.d.) Cburch.com Available at: http://www.cburch.com/logisim/docs/2.7/en/html/libs/io/tty.html. (Accessed: 4 December 2024)

Teleprinter (2024). Wikipedia, The Free Encyclopedia. Available at: https://en.wikipedia.org/w/index.php?title=Teleprinter&oldid=1259820203. (Accessed: 06/12/2024).

Bit Extender(n.d.) Cburch.com Available at: http://www.cburch.com/logisim/docs/2.7/en/html/libs/wiring/extender.html. (Accessed 11/12/2024)

Smith, Stephen. (2024) *RISC-V Assembly language programming: unlock the power of the risc-v instruction set*. Available at: https://doi.org/10.1007/979-8-8688-0137-2 (Accessed: January 07,2025).

Sarah Harris (2022) *DDCA Ch6 – Part 10:Arrays*. 12th September. Available at: https://www.youtube.com/watch?v=XQDKFlPE_mo (Accessed: January 07,2025).

GeeksForGeeks (2024) *What is Array?* Available at: https://www.geeksforgeeks.org/what-is-array/ (Accessed: January 09, 2025).

'File:ASCII-Table-wide.svg' (2010) *Wikimedia commons*. Available at: https://commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg (Accessed: January 09,2025).

Beep. (2020) 'How to use an array in RISC-V Assembly', *stackoverflow*, 19/01. Available at: https://stackoverflow.com/questions/59813759/how-to-use-an-array-in-risc-v-assembly (Accessed: 08/01/2025).

4. Appendix A: The CPU Simulation Code

The code within this section needs to be as plain text which could be copied and pasted into another file to test your circuit.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project source="3.9.0" version="1.0">
  This file is intended to be loaded by Logisim-evolution
v3.9.0(https://github.com/logisim-evolution/).

  <lib desc="#Wiring" name="0">
    <tool name="Pin">
      <a name="appearance" val="classic"/>
    </tool>
  </lib>
  <lib desc="#Gates" name="1"/>
  <lib desc="#Plexers" name="2"/>
  <lib desc="#Arithmetic" name="3"/>
  <lib desc="#Memory" name="4"/>
  <lib desc="#I/O" name="5"/>
  <lib desc="#TTL" name="6"/>
  <lib desc="#TCL" name="7"/>
  <lib desc="#Base" name="8"/>
  <lib desc="#BFH-Praktika" name="9"/>
  <lib desc="#Input/Output-Extra" name="10"/>
  <lib desc="#Soc" name="11"/>
  <main name="CPU_main"/>
  <options>
    <a name="gateUndefined" val="ignore"/>
    <a name="simlimit" val="1000"/>
    <a name="simrand" val="0"/>
  </options>
  <mappings>
    <tool lib="8" map="Button2" name="Menu Tool"/>
    <tool lib="8" map="Button3" name="Menu Tool"/>
    <tool lib="8" map="Ctrl Button1" name="Menu Tool"/>
  </mappings>
  <toolbar>
    <tool lib="8" name="Poke Tool"/>
    <tool lib="8" name="Edit Tool"/>
    <tool lib="8" name="Wiring Tool"/>
    <tool lib="8" name="Text Tool"/>
    <sep/>
    <tool lib="0" name="Pin"/>
    <tool lib="0" name="Pin">
      <a name="facing" val="west"/>
      <a name="output" val="true"/>
    </tool>
    <sep/>
    <tool lib="1" name="NOT Gate"/>
    <tool lib="1" name="AND Gate"/>
    <tool lib="1" name="OR Gate"/>
    <tool lib="1" name="XOR Gate"/>
    <tool lib="1" name="NAND Gate"/>
```

```
    <tool lib="1" name="NOR Gate"/>
    <sep/>
    <tool lib="4" name="D Flip-Flop"/>
    <tool lib="4" name="Register"/>
  </toolbar>
  <circuit name="CPU_main">
    <a name="appearance" val="logisim_evolution"/>
    <a name="circuit" val="CPU_main"/>
    <a name="circuitnamedboxfixedsize" val="true"/>
    <a name="simulationFrequency" val="2.0"/>
    <comp lib="0" loc="(40,520)" name="Clock"/>
    <comp lib="0" loc="(70,90)" name="Constant">
      <a name="width" val="8"/>
    </comp>
    <comp lib="0" loc="(980,430)" name="Bit Extender">
      <a name="out_width" val="7"/>
    </comp>
    <comp lib="1" loc="(160,500)" name="AND Gate"/>
    <comp lib="1" loc="(580,280)" name="AND Gate">
      <a name="facing" val="north"/>
      <a name="size" val="30"/>
    </comp>
    <comp lib="1" loc="(60,480)" name="NOT Gate"/>
    <comp lib="2" loc="(180,80)" name="Multiplexer">
      <a name="size" val="20"/>
      <a name="width" val="8"/>
    </comp>
    <comp lib="2" loc="(860,390)" name="Multiplexer">
      <a name="selloc" val="tr"/>
      <a name="width" val="8"/>
    </comp>
    <comp lib="3" loc="(130,80)" name="Adder"/>
    <comp lib="4" loc="(200,50)" name="Register">
      <a name="appearance" val="logisim_evolution"/>
      <a name="label" val="PC"/>
    </comp>
    <comp lib="4" loc="(380,20)" name="ROM">
      <a name="appearance" val="classic"/>
      <a name="contents">addr/data: 8 16
102f 3000 1002 3001 2000 2501 400f 2000
3002 2001 3000 2002 3001 2010 3002 2600
3003 2602 3004 5000 5001 f000
</a>
      <a name="dataWidth" val="16"/>
      <a name="label" val="InstructionSet"/>
      <a name="labelfont" val="SansSerif bold 14"/>
      <a name="labelvisible" val="true"/>
    </comp>
    <comp lib="4" loc="(550,450)" name="RAM">
      <a name="appearance" val="classic"/>
      <a name="asyncread" val="true"/>
      <a name="clearpin" val="true"/>
      <a name="label" val="ValueSet"/>
      <a name="labelfont" val="SansSerif bold 14"/>
      <a name="labelvisible" val="true"/>
    </comp>
```

```
<comp lib="4" loc="(810,120)" name="Register">
  <a name="appearance" val="logisim_evolution"/>
  <a name="label" val="ACC"/>
</comp>
<comp lib="5" loc="(1000,470)" name="TTY"/>
<comp lib="8" loc="(350,560)" name="Text">
  <a name="text" val="Student id: 24804702"/>
</comp>
<comp loc="(1140,240)" name="ALU"/>
<comp loc="(320,240)" name="Instruction_Decoding_Unit"/>
<wire from="(1010,480)" to="(1010,490)"/>
<wire from="(1140,240)" to="(1160,240)"/>
<wire from="(1140,260)" to="(1140,320)"/>
<wire from="(1160,80)" to="(1160,240)"/>
<wire from="(130,80)" to="(140,80)"/>
<wire from="(140,170)" to="(660,170)"/>
<wire from="(140,70)" to="(140,80)"/>
<wire from="(140,70)" to="(160,70)"/>
<wire from="(140,90)" to="(140,170)"/>
<wire from="(140,90)" to="(160,90)"/>
<wire from="(160,500)" to="(180,500)"/>
<wire from="(170,100)" to="(170,180)"/>
<wire from="(170,180)" to="(580,180)"/>
<wire from="(180,460)" to="(180,500)"/>
<wire from="(180,500)" to="(180,520)"/>
<wire from="(180,520)" to="(520,520)"/>
<wire from="(180,80)" to="(200,80)"/>
<wire from="(20,440)" to="(20,480)"/>
<wire from="(20,440)" to="(330,440)"/>
<wire from="(20,480)" to="(30,480)"/>
<wire from="(200,120)" to="(200,160)"/>
<wire from="(260,80)" to="(300,80)"/>
<wire from="(300,20)" to="(300,80)"/>
<wire from="(300,80)" to="(340,80)"/>
<wire from="(320,240)" to="(920,240)"/>
<wire from="(320,260)" to="(690,260)"/>
<wire from="(320,270)" to="(320,280)"/>
<wire from="(320,270)" to="(840,270)"/>
<wire from="(320,300)" to="(490,300)"/>
<wire from="(320,320)" to="(570,320)"/>
<wire from="(320,340)" to="(900,340)"/>
<wire from="(320,360)" to="(330,360)"/>
<wire from="(320,380)" to="(510,380)"/>
<wire from="(330,360)" to="(330,440)"/>
<wire from="(340,0)" to="(350,0)"/>
<wire from="(340,30)" to="(340,80)"/>
<wire from="(340,30)" to="(380,30)"/>
<wire from="(40,520)" to="(110,520)"/>
<wire from="(490,300)" to="(490,500)"/>
<wire from="(490,500)" to="(550,500)"/>
<wire from="(510,380)" to="(510,460)"/>
<wire from="(510,380)" to="(830,380)"/>
<wire from="(510,460)" to="(550,460)"/>
<wire from="(520,520)" to="(520,630)"/>
<wire from="(520,520)" to="(550,520)"/>
<wire from="(520,630)" to="(910,630)"/>
```

```
  <wire from="(530,540)" to="(530,620)"/>
  <wire from="(530,540)" to="(550,540)"/>
  <wire from="(530,620)" to="(890,620)"/>
  <wire from="(570,310)" to="(570,320)"/>
  <wire from="(580,180)" to="(580,280)"/>
  <wire from="(590,310)" to="(590,320)"/>
  <wire from="(590,320)" to="(1140,320)"/>
  <wire from="(60,160)" to="(200,160)"/>
  <wire from="(60,160)" to="(60,190)"/>
  <wire from="(60,190)" to="(60,460)"/>
  <wire from="(60,190)" to="(810,190)"/>
  <wire from="(60,20)" to="(300,20)"/>
  <wire from="(60,20)" to="(60,70)"/>
  <wire from="(60,460)" to="(180,460)"/>
  <wire from="(60,480)" to="(110,480)"/>
  <wire from="(60,70)" to="(90,70)"/>
  <wire from="(620,80)" to="(640,80)"/>
  <wire from="(640,80)" to="(640,210)"/>
  <wire from="(660,170)" to="(660,220)"/>
  <wire from="(660,220)" to="(870,220)"/>
  <wire from="(690,170)" to="(690,260)"/>
  <wire from="(690,170)" to="(810,170)"/>
  <wire from="(70,90)" to="(90,90)"/>
  <wire from="(780,150)" to="(810,150)"/>
  <wire from="(780,80)" to="(1160,80)"/>
  <wire from="(780,80)" to="(780,150)"/>
  <wire from="(790,540)" to="(810,540)"/>
  <wire from="(810,400)" to="(810,540)"/>
  <wire from="(810,400)" to="(830,400)"/>
  <wire from="(840,270)" to="(840,370)"/>
  <wire from="(860,390)" to="(870,390)"/>
  <wire from="(870,150)" to="(890,150)"/>
  <wire from="(870,220)" to="(870,280)"/>
  <wire from="(870,280)" to="(870,390)"/>
  <wire from="(870,280)" to="(920,280)"/>
  <wire from="(890,150)" to="(890,260)"/>
  <wire from="(890,260)" to="(890,430)"/>
  <wire from="(890,260)" to="(920,260)"/>
  <wire from="(890,430)" to="(890,620)"/>
  <wire from="(890,430)" to="(940,430)"/>
  <wire from="(90,210)" to="(640,210)"/>
  <wire from="(90,210)" to="(90,240)"/>
  <wire from="(90,240)" to="(100,240)"/>
  <wire from="(900,340)" to="(900,490)"/>
  <wire from="(900,490)" to="(1010,490)"/>
  <wire from="(910,470)" to="(1000,470)"/>
  <wire from="(910,470)" to="(910,630)"/>
  <wire from="(980,430)" to="(990,430)"/>
  <wire from="(990,430)" to="(990,460)"/>
  <wire from="(990,460)" to="(1000,460)"/>
</circuit>
<circuit name="Instruction_Decoding_Unit">
  <a name="appearance" val="logisim_evolution"/>
  <a name="circuit" val="Instruction_Decoding_Unit"/>
  <a name="circuitnamedboxfixedsize" val="true"/>
  <a name="simulationFrequency" val="2.0"/>
```

```
<comp lib="0" loc="(160,170)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="label" val="Instruction"/>
  <a name="labelfont" val="SansSerif bold 12"/>
  <a name="radix" val="16"/>
  <a name="width" val="16"/>
</comp>
<comp lib="0" loc="(170,170)" name="Splitter">
  <a name="appear" val="right"/>
  <a name="bit0" val="2"/>
  <a name="bit1" val="2"/>
  <a name="bit10" val="0"/>
  <a name="bit11" val="0"/>
  <a name="bit12" val="1"/>
  <a name="bit13" val="1"/>
  <a name="bit14" val="1"/>
  <a name="bit15" val="1"/>
  <a name="bit3" val="2"/>
  <a name="bit4" val="2"/>
  <a name="bit5" val="2"/>
  <a name="bit6" val="2"/>
  <a name="bit7" val="2"/>
  <a name="bit8" val="0"/>
  <a name="bit9" val="0"/>
  <a name="fanout" val="3"/>
  <a name="incoming" val="16"/>
  <a name="spacing" val="3"/>
</comp>
<comp lib="0" loc="(600,140)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="label" val="Write_ACC"/>
  <a name="labelfont" val="SansSerif bold 12"/>
  <a name="output" val="true"/>
</comp>
<comp lib="0" loc="(600,190)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="label" val="Use_Memory"/>
  <a name="labelfont" val="SansSerif bold 12"/>
  <a name="output" val="true"/>
</comp>
<comp lib="0" loc="(600,250)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="label" val="Write_Memory"/>
  <a name="labelfont" val="SansSerif bold 12"/>
  <a name="output" val="true"/>
</comp>
<comp lib="0" loc="(600,280)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="label" val="BNEZ"/>
  <a name="labelfont" val="SansSerif bold 12"/>
  <a name="output" val="true"/>
</comp>
```

```
<comp lib="0" loc="(600,320)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="label" val="Output_Char"/>
  <a name="labelfont" val="SansSerif bold 12"/>
  <a name="output" val="true"/>
</comp>
<comp lib="0" loc="(600,640)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="label" val="Stop"/>
  <a name="labelfont" val="SansSerif bold 12"/>
  <a name="output" val="true"/>
</comp>
<comp lib="0" loc="(620,60)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="label" val="ALU_Operation"/>
  <a name="labelfont" val="SansSerif bold 12"/>
  <a name="output" val="true"/>
  <a name="radix" val="16"/>
  <a name="width" val="4"/>
</comp>
<comp lib="0" loc="(630,690)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="label" val="Addr_or_Imm"/>
  <a name="labelfont" val="SansSerif bold 12"/>
  <a name="output" val="true"/>
  <a name="radix" val="16"/>
  <a name="width" val="8"/>
</comp>
<comp lib="1" loc="(570,190)" name="OR Gate">
  <a name="size" val="30"/>
</comp>
<comp lib="1" loc="(580,140)" name="OR Gate">
  <a name="size" val="30"/>
</comp>
<comp lib="2" loc="(330,220)" name="Decoder">
  <a name="select" val="4"/>
  <a name="selloc" val="tr"/>
</comp>
<wire from="(160,170)" to="(170,170)"/>
<wire from="(190,180)" to="(250,180)"/>
<wire from="(190,210)" to="(330,210)"/>
<wire from="(190,240)" to="(190,690)"/>
<wire from="(190,690)" to="(630,690)"/>
<wire from="(250,60)" to="(250,180)"/>
<wire from="(250,60)" to="(620,60)"/>
<wire from="(330,210)" to="(330,220)"/>
<wire from="(350,230)" to="(400,230)"/>
<wire from="(350,240)" to="(420,240)"/>
<wire from="(350,250)" to="(530,250)"/>
<wire from="(350,260)" to="(530,260)"/>
<wire from="(350,270)" to="(510,270)"/>
<wire from="(350,370)" to="(370,370)"/>
```

```
    <wire from="(370,370)" to="(370,640)"/>
    <wire from="(370,640)" to="(600,640)"/>
    <wire from="(400,130)" to="(400,230)"/>
    <wire from="(400,130)" to="(550,130)"/>
    <wire from="(420,150)" to="(420,180)"/>
    <wire from="(420,150)" to="(550,150)"/>
    <wire from="(420,180)" to="(420,240)"/>
    <wire from="(420,180)" to="(540,180)"/>
    <wire from="(510,270)" to="(510,320)"/>
    <wire from="(510,320)" to="(600,320)"/>
    <wire from="(530,200)" to="(530,250)"/>
    <wire from="(530,200)" to="(540,200)"/>
    <wire from="(530,250)" to="(600,250)"/>
    <wire from="(530,260)" to="(530,280)"/>
    <wire from="(530,280)" to="(600,280)"/>
    <wire from="(570,190)" to="(600,190)"/>
    <wire from="(580,140)" to="(600,140)"/>
  </circuit>
  <circuit name="ALU">
    <a name="appearance" val="logisim_evolution"/>
    <a name="circuit" val="ALU"/>
    <a name="circuitnamedboxfixedsize" val="true"/>
    <a name="simulationFrequency" val="2.0"/>
    <comp lib="0" loc="(210,120)" name="Pin">
      <a name="appearance" val="NewPins"/>
      <a name="label" val="From_ACC"/>
      <a name="labelfont" val="SansSerif bold 12"/>
      <a name="radix" val="16"/>
      <a name="width" val="8"/>
    </comp>
    <comp lib="0" loc="(210,190)" name="Pin">
      <a name="appearance" val="NewPins"/>
      <a name="label" val="Input2"/>
      <a name="labelfont" val="SansSerif bold 12"/>
      <a name="radix" val="16"/>
      <a name="width" val="8"/>
    </comp>
    <comp lib="0" loc="(210,60)" name="Pin">
      <a name="appearance" val="NewPins"/>
      <a name="label" val="ALU_Operation"/>
      <a name="labelfont" val="SansSerif bold 12"/>
      <a name="radix" val="16"/>
      <a name="width" val="4"/>
    </comp>
    <comp lib="0" loc="(330,360)" name="Constant">
      <a name="value" val="0x4"/>
      <a name="width" val="3"/>
    </comp>
    <comp lib="0" loc="(330,480)" name="Constant">
      <a name="value" val="0x0"/>
      <a name="width" val="8"/>
    </comp>
    <comp lib="0" loc="(450,410)" name="Bit Extender">
      <a name="in_width" val="1"/>
      <a name="out_width" val="8"/>
      <a name="type" val="zero"/>
```

```
</comp>
<comp lib="0" loc="(680,270)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="label" val="ALU_to_ACC"/>
  <a name="labelfont" val="SansSerif bold 12"/>
  <a name="output" val="true"/>
  <a name="radix" val="16"/>
  <a name="width" val="8"/>
</comp>
<comp lib="0" loc="(720,470)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="label" val="ACC_NE_Zero"/>
  <a name="output" val="true"/>
</comp>
<comp lib="1" loc="(380,270)" name="AND Gate">
  <a name="size" val="30"/>
  <a name="width" val="8"/>
</comp>
<comp lib="1" loc="(420,310)" name="XOR Gate">
  <a name="size" val="30"/>
  <a name="width" val="8"/>
</comp>
<comp lib="1" loc="(450,470)" name="NOT Gate"/>
<comp lib="2" loc="(610,270)" name="Multiplexer">
  <a name="select" val="4"/>
  <a name="selloc" val="tr"/>
  <a name="width" val="8"/>
</comp>
<comp lib="3" loc="(380,350)" name="Shifter">
  <a name="shift" val="lr"/>
</comp>
<comp lib="3" loc="(380,400)" name="Comparator">
  <a name="mode" val="unsigned"/>
</comp>
<comp lib="3" loc="(380,470)" name="Comparator"/>
<comp lib="3" loc="(390,220)" name="Adder"/>
<comp loc="(520,520)" name="ASCII_To_Hex"/>
<wire from="(210,120)" to="(300,120)"/>
<wire from="(210,190)" to="(240,190)"/>
<wire from="(210,60)" to="(590,60)"/>
<wire from="(240,190)" to="(240,230)"/>
<wire from="(240,190)" to="(570,190)"/>
<wire from="(240,230)" to="(240,280)"/>
<wire from="(240,230)" to="(350,230)"/>
<wire from="(240,280)" to="(240,320)"/>
<wire from="(240,280)" to="(350,280)"/>
<wire from="(240,320)" to="(240,410)"/>
<wire from="(240,320)" to="(380,320)"/>
<wire from="(240,410)" to="(340,410)"/>
<wire from="(300,120)" to="(300,210)"/>
<wire from="(300,210)" to="(300,260)"/>
<wire from="(300,210)" to="(350,210)"/>
<wire from="(300,260)" to="(300,300)"/>
<wire from="(300,260)" to="(350,260)"/>
```

```
    <wire from="(300,300)" to="(300,340)"/>
    <wire from="(300,300)" to="(380,300)"/>
    <wire from="(300,340)" to="(300,390)"/>
    <wire from="(300,340)" to="(340,340)"/>
    <wire from="(300,390)" to="(300,460)"/>
    <wire from="(300,390)" to="(340,390)"/>
    <wire from="(300,460)" to="(300,520)"/>
    <wire from="(300,460)" to="(340,460)"/>
    <wire from="(330,360)" to="(340,360)"/>
    <wire from="(330,480)" to="(340,480)"/>
    <wire from="(380,270)" to="(420,270)"/>
    <wire from="(380,350)" to="(460,350)"/>
    <wire from="(380,380)" to="(380,390)"/>
    <wire from="(380,410)" to="(410,410)"/>
    <wire from="(380,470)" to="(420,470)"/>
    <wire from="(390,220)" to="(400,220)"/>
    <wire from="(400,200)" to="(400,220)"/>
    <wire from="(400,200)" to="(570,200)"/>
    <wire from="(420,210)" to="(420,270)"/>
    <wire from="(420,210)" to="(570,210)"/>
    <wire from="(420,310)" to="(440,310)"/>
    <wire from="(440,220)" to="(440,310)"/>
    <wire from="(440,220)" to="(570,220)"/>
    <wire from="(450,410)" to="(470,410)"/>
    <wire from="(450,470)" to="(720,470)"/>
    <wire from="(460,230)" to="(460,350)"/>
    <wire from="(460,230)" to="(570,230)"/>
    <wire from="(470,240)" to="(470,410)"/>
    <wire from="(470,240)" to="(570,240)"/>
    <wire from="(490,260)" to="(490,500)"/>
    <wire from="(490,260)" to="(570,260)"/>
    <wire from="(490,500)" to="(520,500)"/>
    <wire from="(520,500)" to="(520,520)"/>
    <wire from="(520,540)" to="(550,540)"/>
    <wire from="(550,250)" to="(550,540)"/>
    <wire from="(550,250)" to="(570,250)"/>
    <wire from="(590,60)" to="(590,190)"/>
    <wire from="(610,270)" to="(680,270)"/>
  </circuit>
  <circuit name="ASCII_To_Hex">
    <a name="appearance" val="logisim_evolution"/>
    <a name="circuit" val="ASCII_To_Hex"/>
    <a name="circuitnamedboxfixedsize" val="true"/>
    <a name="simulationFrequency" val="2.0"/>
    <comp lib="0" loc="(1030,1220)" name="Pin">
      <a name="appearance" val="NewPins"/>
      <a name="facing" val="west"/>
      <a name="label" val="cout1"/>
      <a name="output" val="true"/>
      <a name="radix" val="16"/>
      <a name="width" val="8"/>
    </comp>
    <comp lib="0" loc="(1030,1360)" name="Pin">
      <a name="appearance" val="NewPins"/>
      <a name="facing" val="west"/>
      <a name="label" val="cout2"/>
```

```
  <a name="output" val="true"/>
  <a name="radix" val="16"/>
  <a name="width" val="8"/>
</comp>
<comp lib="0" loc="(210,1220)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="label" val="From_ACC"/>
  <a name="radix" val="16"/>
  <a name="width" val="8"/>
</comp>
<comp lib="0" loc="(220,1220)" name="Splitter">
  <a name="appear" val="right"/>
  <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="incoming" val="8"/>
</comp>
<comp lib="0" loc="(280,1190)" name="Probe">
  <a name="appearance" val="NewPins"/>
  <a name="radix" val="16"/>
</comp>
<comp lib="0" loc="(280,1270)" name="Probe">
  <a name="appearance" val="NewPins"/>
  <a name="radix" val="16"/>
</comp>
<comp lib="0" loc="(400,1190)" name="Bit Extender">
  <a name="in_width" val="4"/>
  <a name="out_width" val="8"/>
  <a name="type" val="zero"/>
</comp>
<comp lib="0" loc="(400,1270)" name="Bit Extender">
  <a name="in_width" val="4"/>
  <a name="out_width" val="8"/>
  <a name="type" val="zero"/>
</comp>
<comp lib="0" loc="(470,1200)" name="Constant">
  <a name="value" val="0x9"/>
  <a name="width" val="8"/>
</comp>
<comp lib="0" loc="(470,1280)" name="Constant">
  <a name="value" val="0x9"/>
  <a name="width" val="8"/>
</comp>
<comp lib="0" loc="(600,1320)" name="Constant">
  <a name="value" val="0x57"/>
  <a name="width" val="8"/>
</comp>
<comp lib="0" loc="(600,1380)" name="Constant">
  <a name="value" val="0x30"/>
  <a name="width" val="8"/>
</comp>
<comp lib="0" loc="(750,1230)" name="Constant">
```

```
  <a name="value" val="0x30"/>
  <a name="width" val="8"/>
</comp>
<comp lib="0" loc="(760,1150)" name="Constant">
  <a name="value" val="0x57"/>
  <a name="width" val="8"/>
</comp>
<comp lib="1" loc="(570,1200)" name="OR Gate">
  <a name="size" val="30"/>
</comp>
<comp lib="1" loc="(570,1280)" name="OR Gate">
  <a name="size" val="30"/>
</comp>
<comp lib="2" loc="(910,1360)" name="Multiplexer">
  <a name="width" val="8"/>
</comp>
<comp lib="2" loc="(920,1200)" name="Multiplexer">
  <a name="width" val="8"/>
</comp>
<comp lib="3" loc="(510,1190)" name="Comparator"/>
<comp lib="3" loc="(510,1270)" name="Comparator"/>
<comp lib="3" loc="(810,1140)" name="Adder"/>
<comp lib="3" loc="(810,1220)" name="Adder"/>
<comp lib="3" loc="(820,1330)" name="Adder"/>
<comp lib="3" loc="(820,1390)" name="Adder"/>
<wire from="(210,1220)" to="(220,1220)"/>
<wire from="(240,1230)" to="(300,1230)"/>
<wire from="(240,1240)" to="(300,1240)"/>
<wire from="(280,1190)" to="(300,1190)"/>
<wire from="(280,1270)" to="(300,1270)"/>
<wire from="(300,1190)" to="(300,1230)"/>
<wire from="(300,1190)" to="(360,1190)"/>
<wire from="(300,1240)" to="(300,1270)"/>
<wire from="(300,1270)" to="(360,1270)"/>
<wire from="(400,1190)" to="(450,1190)"/>
<wire from="(400,1270)" to="(420,1270)"/>
<wire from="(420,1270)" to="(420,1340)"/>
<wire from="(420,1270)" to="(450,1270)"/>
<wire from="(420,1340)" to="(420,1400)"/>
<wire from="(420,1340)" to="(780,1340)"/>
<wire from="(420,1400)" to="(780,1400)"/>
<wire from="(450,1180)" to="(450,1190)"/>
<wire from="(450,1180)" to="(460,1180)"/>
<wire from="(450,1260)" to="(450,1270)"/>
<wire from="(450,1260)" to="(470,1260)"/>
<wire from="(460,1160)" to="(460,1180)"/>
<wire from="(460,1160)" to="(660,1160)"/>
<wire from="(460,1180)" to="(470,1180)"/>
<wire from="(510,1190)" to="(540,1190)"/>
<wire from="(510,1200)" to="(540,1200)"/>
<wire from="(510,1270)" to="(540,1270)"/>
<wire from="(510,1280)" to="(540,1280)"/>
<wire from="(540,1200)" to="(540,1210)"/>
<wire from="(540,1280)" to="(540,1290)"/>
<wire from="(570,1200)" to="(570,1230)"/>
<wire from="(570,1230)" to="(670,1230)"/>
```

```
    <wire from="(570,1280)" to="(930,1280)"/>
    <wire from="(600,1320)" to="(780,1320)"/>
    <wire from="(600,1380)" to="(780,1380)"/>
    <wire from="(660,1130)" to="(660,1160)"/>
    <wire from="(660,1130)" to="(770,1130)"/>
    <wire from="(660,1160)" to="(660,1210)"/>
    <wire from="(660,1210)" to="(770,1210)"/>
    <wire from="(670,1230)" to="(670,1250)"/>
    <wire from="(670,1250)" to="(900,1250)"/>
    <wire from="(750,1230)" to="(770,1230)"/>
    <wire from="(760,1150)" to="(770,1150)"/>
    <wire from="(810,1140)" to="(880,1140)"/>
    <wire from="(810,1220)" to="(830,1220)"/>
    <wire from="(820,1330)" to="(860,1330)"/>
    <wire from="(820,1390)" to="(860,1390)"/>
    <wire from="(830,1210)" to="(830,1220)"/>
    <wire from="(830,1210)" to="(890,1210)"/>
    <wire from="(860,1330)" to="(860,1350)"/>
    <wire from="(860,1350)" to="(880,1350)"/>
    <wire from="(860,1370)" to="(860,1390)"/>
    <wire from="(860,1370)" to="(880,1370)"/>
    <wire from="(880,1140)" to="(880,1190)"/>
    <wire from="(880,1190)" to="(890,1190)"/>
    <wire from="(890,1380)" to="(890,1400)"/>
    <wire from="(890,1400)" to="(930,1400)"/>
    <wire from="(900,1220)" to="(900,1250)"/>
    <wire from="(910,1360)" to="(1030,1360)"/>
    <wire from="(920,1200)" to="(940,1200)"/>
    <wire from="(930,1280)" to="(930,1400)"/>
    <wire from="(940,1200)" to="(940,1220)"/>
    <wire from="(940,1220)" to="(1030,1220)"/>
  </circuit>
</project>
```

5. Appendix B: The RISC-V Assembly Language Code

The code within this section needs to be as plain text which could be copied and pasted into RARS to test your work. You should include additional subheadings to show the different stages.

5.1 Tasks 2.1 to 2.4 amending the starter program provided

2.3

```
.data
enterMsg1: .string "Please use the last four digits of your student id as two 2-digit
numbers \n"
enterMsg2: .string "Enter a two digit number\n"
enterMsg3: .string "Enter next number \n"
outputMsg: .string "The total is:\n"
andMsg: .string "\nThe total of the values in a bitwise and operation is:\n"
xorMsg: .string "\nThe total of the values in a bitwise xor operation is:\n"

.text

###

# ------------------------------------------------------ Input segment  -------------
---------------------------------------------------- #
# output the instruction text to the console

addi a7, zero, 4 # outputs "Please use the last four digits of your student id as two
2-digit numbers" and makes a new line
la a0, enterMsg1
ecall

addi a7, zero, 4
la a0, enterMsg2
ecall

# read an integer from keyboard input and store in s0
addi a7, zero, 5
ecall
add s0, zero, a0


# output the text asking for the next number to the console
# then receive the input and store in s1

addi a7, zero, 4
la a0, enterMsg3
ecall

addi a7, zero, 5
ecall

add s1, zero, a0

# ------------------------------------------------------ mathematical equations -----
---------------------------------------------------- #
## add the two values together and store in s2
```

```
add s2,s0,s1 # s2 = input1 (a0) + input2 (a1)

## bitwise and operation on s0 and s1
and s3,s0,s1

## bitwise xor operation on s0 and s1
xor s4,s0,s1

# ------------------------------------------------------ Run I/O segment -----------
---------------------------------------------------- #

# output the value from s2
addi a7,zero,4  #Outputs the string msg
la a0, outputMsg
ecall

add a0, s2, zero #Adds s2 to register a0 and out puts it in a0: a0 = s2 + 0
addi a7, zero, 1
ecall

addi a7, zero,4 #Outputs variable andMsg
la a0, andMsg
ecall

add a0,s3,zero #Adds s3 to register 01 and outputs
addi a7,zero,1
ecall

addi a7,zero,4 #Outputs xorMsg
la a0, xorMsg
ecall

add a0,s4,zero #Adds s4 to register 01 and outputs
addi a7,zero,1
ecall

addi a7, zero, 10
ecall
```

2.4

```
.data
enterMsg1: .string "Please use the last four digits of your student id as two 2-digit
numbers \n"
enterMsg2: .string "Enter a two digit number\n"
enterMsg3: .string "Enter next number \n"
outputMsg: .string "\nThe total of the operation is:\n"
andMsg: .string "\nThe total of the values in a bitwise and operation is:\n"
xorMsg: .string "\nThe total of the values in a bitwise xor operation is:\n"
.text
###
# ------------------------------------------------------ Input segment  -------------
---------------------------------------------------- #
# output the instruction text to the console

addi a7, zero, 4 # outputs "Please use the last four digits of your student id as two
2-digit numbers" and makes a new line
```

```
la a0, enterMsg1
ecall

addi a7, zero, 4
la a0, enterMsg2
ecall

# read an integer from keyboard input and store in s0
addi a7, zero, 5
ecall
add s0, zero, a0


# output the text asking for the next number to the console
# then receive the input and store in s1

addi a7, zero, 4
la a0, enterMsg3
ecall

addi a7, zero, 5
ecall

add s1, zero, a0

# ------------------------------------------------------ mathematical equations -----
------------------------------------------------------ #
## add the two values together and store in s2
add s2,s0,s1 # s2 = input1 (a0) + input2 (a1)

## bitwise and operation on s0 and s1
and s3,s0,s1

## bitwise xor operation on s0 and s1
xor s4,s0,s1


# ------------------------------------------------------- Run I/O segment -----------
------------------------------------------------------ #

# --- Add --- #
addi a1, s2, 0
la a0, outputMsg
jal output
# --- And --- #
addi a1, s3, 0
la a0, andMsg
jal output
# --- Xor --- #
addi a1, s4, 0
la a0, xorMsg
jal output

#------ Ends the program ------#

addi a7, zero, 10 #Exit code
ecall #Syscall
```

```
output:
      addi a7,zero, 4
      ecall

      addi a7,zero, 1
      add a0, a1,zero
      ecall
      ret
```

5.2 Tasks 2.5 to 2.7 ordering numbers and using a loop and array.

2.6

```
.data
# --- Declaring constants in the program --- #s
inputPrompt1: .string "\nPlease enter the first number: "
inputPrompt2: .string "\nPlease enter the second number: "
enterValidation: .string "\nYou've entered: "
solutionMsg: .string "\nThe sorted values are: "
.text
# --- Starting the program --- #
# --- First int input --- #

la a0, inputPrompt1 #Loads string inputPrompt1 into register a0
jal strPrint #Jumps and link subroutine strPrint which displays what's stored at a0
(Which is
jal intInput #Jumps and link subroutine intInput which stores the integer entered into
a0

add t3, a0,zero #Adds the integer entered in a0 and stores it in t3 to output

add s0,t3,zero  #Copies the value from t3 into s0

la a0, enterValidation
jal strPrint
jal intPrint

# --- Second int input --- #
la a0, inputPrompt2
jal strPrint
jal intInput
add t3, a0, zero

add s1, t3, zero

la a0, enterValidation
jal strPrint
jal intPrint

# --- Loop --- #
slt t1,s0,s1 #if (s0 < s1) {t1 = 1} else {t1 = 0}
bnez t1,loop #if (t1 != 0) { sorted(); } else
mv s2,s0
```

```
mv s0,s1
mv s1,s2
j loop

# --- Subroutines --- #
exit: #Exit program
      addi a7,zero,10
      ecall
strPrint: #Prints a string
      addi a7,zero,4
      ecall
      ret
intPrint: #Prints an integer
      addi a7,zero,1
      add a0,t3,zero
      ecall

      addi a7,zero,11
      addi a0, zero, '\n'
      ecall
      ret
intInput: #Takes in an input and stores it in a0
      addi a7,zero,5
      ecall
      ret
loop:
      jal calling #calling();
      addi s0,s0,1 #s0 = s0 + 1;
      slt t1,s0,s1 #if (s0 < s1) { t1 = 1;} else {t1 = 0;}
      bnez t1,loop #if (t1 != 0) {
      j loopdone #      loopdone(); }

loopdone:
      jal calling #calling();
      j exit #exit();

calling:
      addi a7,zero,1
      add a0,zero,s0
      ecall

      addi a7,zero,11
      addi a0,zero, ' '
      ecall
      ret
```

2.7

```
.data
# --- Declaring constants in the program --- #s
inputPrompt1: .string "\nPlease enter the first number: "
inputPrompt2: .string "\nPlease enter the second number: "
enterValidation: .string "\nYou've entered: "
solutionMsg: .string "\nThe sorted values are: "
equalMsg: .string "\nThe numbers are the same"
numbers : .word 100
```

```
.text
# --- Starting the program --- #
# --- Array --- #
la a1,numbers #  Stores the address of the array to a1
add a5,zero,a1 # Makes a copy of the initial array in register a5
# --- First int input --- #

la a0, inputPrompt1 #Loads string inputPrompt1 into register a0
jal strPrint #Jumps and link subroutine strPrint which displays what's stored at a0
jal intInput #Jumps and link subroutine intInput which stores the integer entered into
a0

add t3, a0,zero #Adds the integer entered in a0 and stores it in t3 to output

add s0,t3,zero  #Copies the value from t3 into s0

la a0, enterValidation
jal strPrint
jal intPrint

# --- Second int input --- #
la a0, inputPrompt2
jal strPrint
jal intInput
add t3, a0, zero

add s1, t3, zero

la a0, enterValidation
jal strPrint
jal intPrint

# --- Loop --- #


beq s0,s1,equal #if (s0 == s1) {equal();}
addi s1,s1,1 # So it doesnt stop one number before the number entered
slt t1,s0,s1 #if (s0 < s1) {t1 = 1} else {t1 = 0}
bnez t1,loop #if (t1 != 0) { sorted(); } else
mv s2,s0
mv s0,s1
mv s1,s2

addi s0,s0,-1
addi s1,s1,1
j loop

# --- Subroutines --- #
exit: #Exit program
      addi a7,zero,10
      ecall
strPrint: #Prints a string
      addi a7,zero,4
      ecall
      ret
intPrint: #Prints an integer
```

```
	addi a7,zero,1
	add a0,t3,zero
	ecall

	addi a7,zero,11
	addi a0, zero, '\n'
	ecall
	ret
intInput: #Takes in an input and stores it in a0
	addi a7,zero,5
	ecall
	ret
loop:
	addi a3,a3,1 # Counter for how many times numbers were incremented
	sw s0, 0(a1) # Stores each increment of s0 in the array
	addi s0,s0,1 #s0 = s0 + 1;
	addi a1, a1, 4 #Increments the array by 1 index
	slt t1,s0,s1 #if (s0 < s1) { t1 = 1;} else {t1 = 0;}
	bnez t1,loop #if (t1 != 0) {
	j loopdone #	loopdone(); }

loopdone:

	bge a2,a3, exit # if (a2 > a3) {exit();}
	lw a0,0(a5) # Loads the copy of the initial address into the variable a0

	addi a7,zero,1 # cout << a0;
	ecall

	addi a7,zero,11 # cout <<  ' ';
	addi a0,zero, ' '
	ecall

	addi a2,a2,1 # adds 1 to a2 (counter)
	addi a5,a5,4 # Increments the index by one
	j loopdone


calling:
	addi a7,zero,1 #cout << s0;
	add a0,zero,s0
	ecall

	addi a7,zero,11 #cout << ' ';
	addi a0,zero, ' '
	ecall
	ret

equal:
	la a0, equalMsg
	jal strPrint #cout << equalMsg;
	j exit #exit();
```

5.3 Task 2.8 Creating a console based game

6. Appendix C: Use of Generative AI

In this section, name the tool used and paste the full chat prompts and outputs used towards responses for this assessment.