

2025

Computer Architecture Report

YEAR 1 SEMESTER 1

SAM STEWART

1. CPU SIMULATION	3
1.1 Calculating the sum of values	3
1.2 Completing the stop instruction	4
1.3 Bitwise Operations	5
1.4 Amending the ALU.....	6
1.5 Output Device.....	8
1.6 Branch instruction	9
1.7 Loop	16
1.8 Game	22
2. RISC-V ASSEMBLY LANGUAGE PROGRAMMING.....	51
2.1 Test the existing program with minor adjustments	51
2.2 Improve the messages that the program outputs	52
2.3 Enhancing the functionality of the program.....	53
2.4 Adding a subroutine	56
2.5 Swapping Numbers.....	59
2.6 Counting down with a loop.....	61
2.7 Making use of arrays	64
2.8 Designing a game.....	70
3. REFERENCES	140
4. APPENDIX A: THE CPU SIMULATION CODE.....	140
5. APPENDIX B: THE RISC-V ASSEMBLY LANGUAGE CODE	163
5.1 Tasks 2.1 to 2.3	163
5.2 Task 2.4 Including a subroutine	164
5.3 Task 2.5 Swapping numbers	166
5.4 Task 2.6 Loop	167
5.5 Task 2.7 Array	169

5.6 Task 2.8 Creating a console based game 171

6. APPENDIX C: USE OF GENERATIVE AI 229

1. CPU Simulation

1.1 Calculating the sum of values

1.1.1 Test data

Last 4 digits of student number: 0129

Last 4 digits of student number split into two two-digit decimal numbers: 0129

Two-digit decimal numbers converted into hexadecimal: 01=01 29=1D

1.1.2 Test program

v2.0 raw

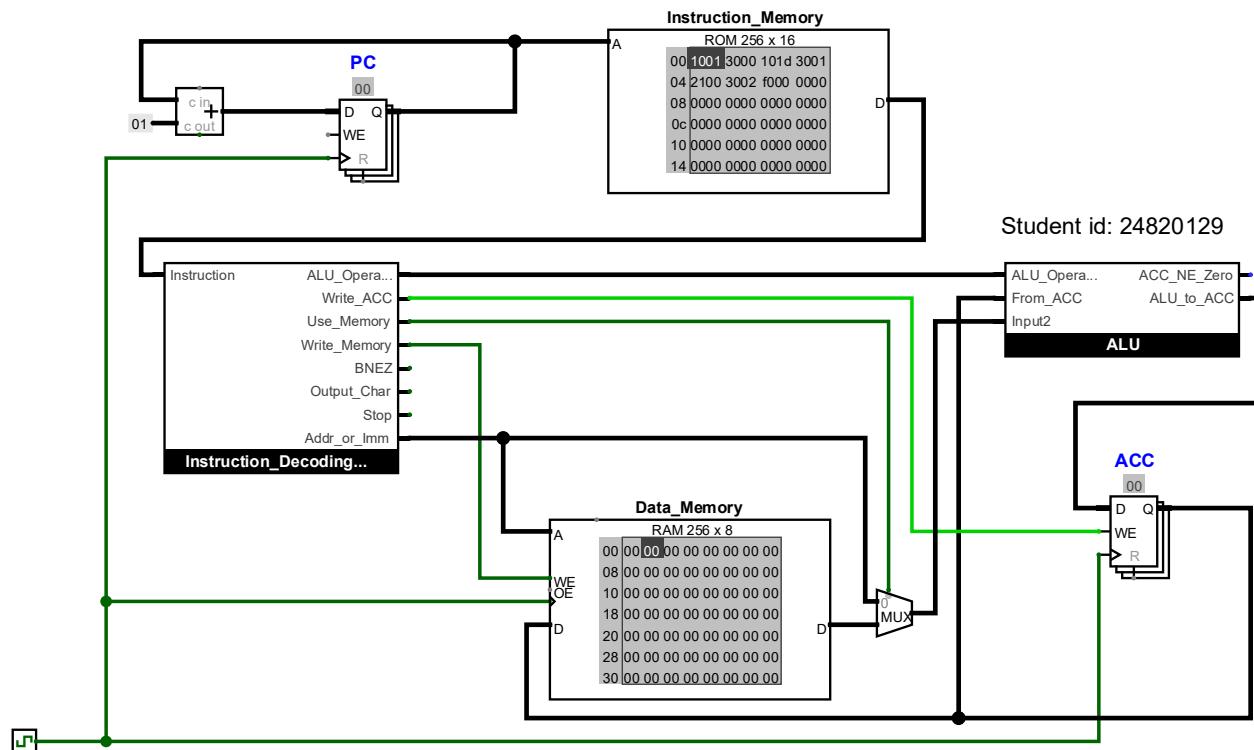
```
1001 3000    # 1. put first value to ACC and address 00
101D 3001    # 2. put second value to ACC and address 01
2100          # 3. add ACC value to value from address 00
3002          # 4. store ACC value at address 02
f000          # 5. stop
```

1.1.3 Expected result

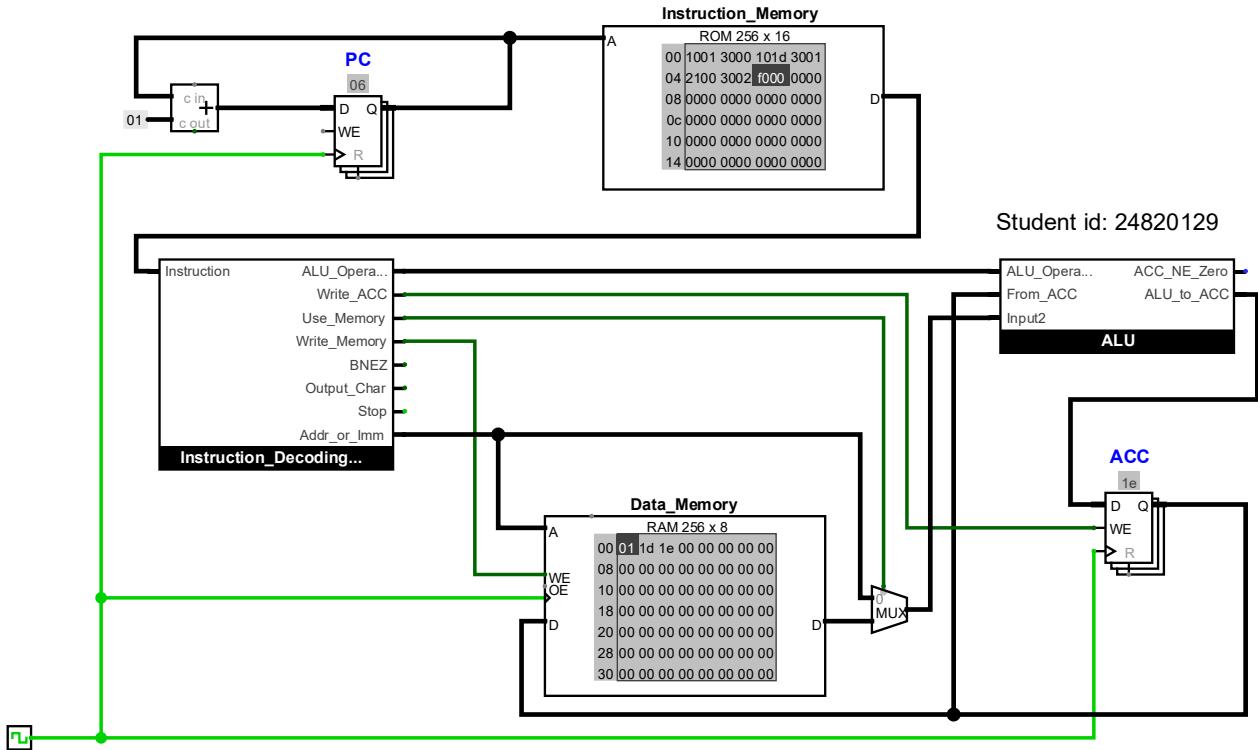
01 + 1D addition expected result in decimal = 30

01 + 1D addition expected result in hexadecimal = 1E

1.1.4 Circuit with program loaded



1.1.5 Circuit at end of program

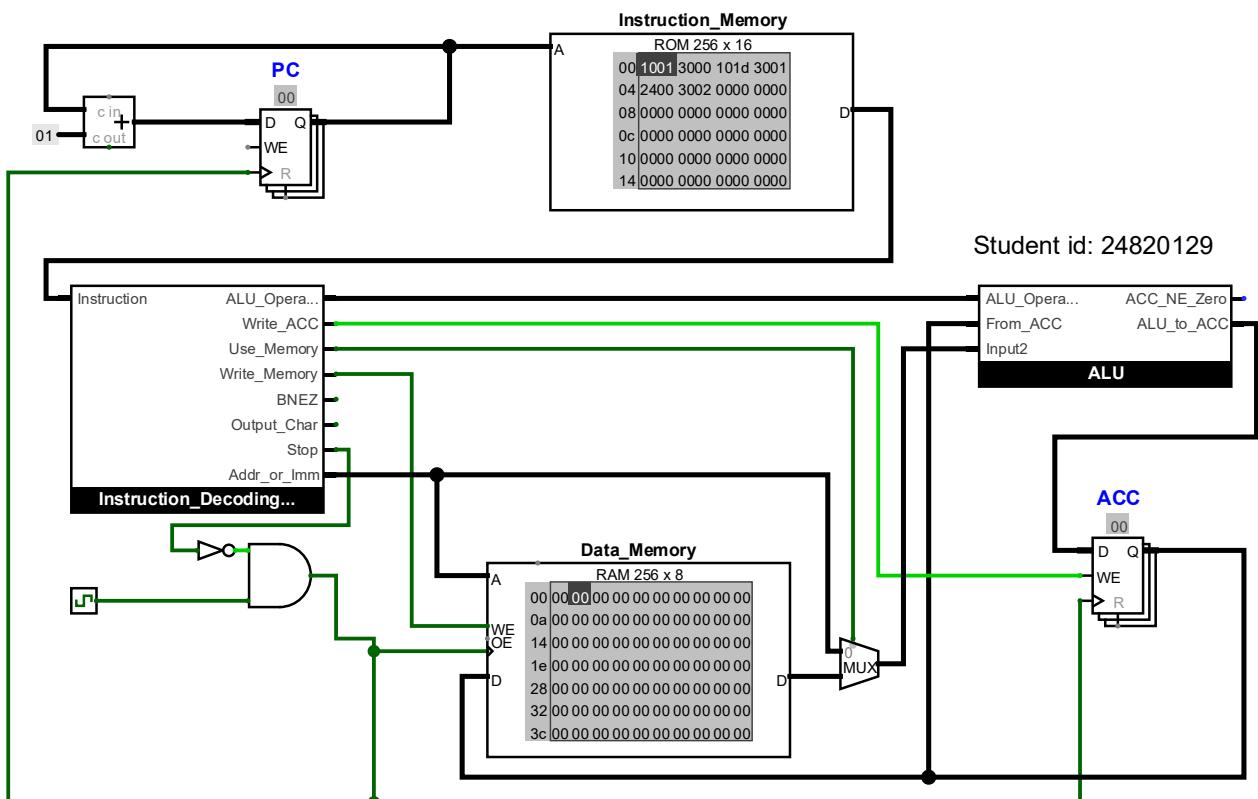


1.1.6 Result

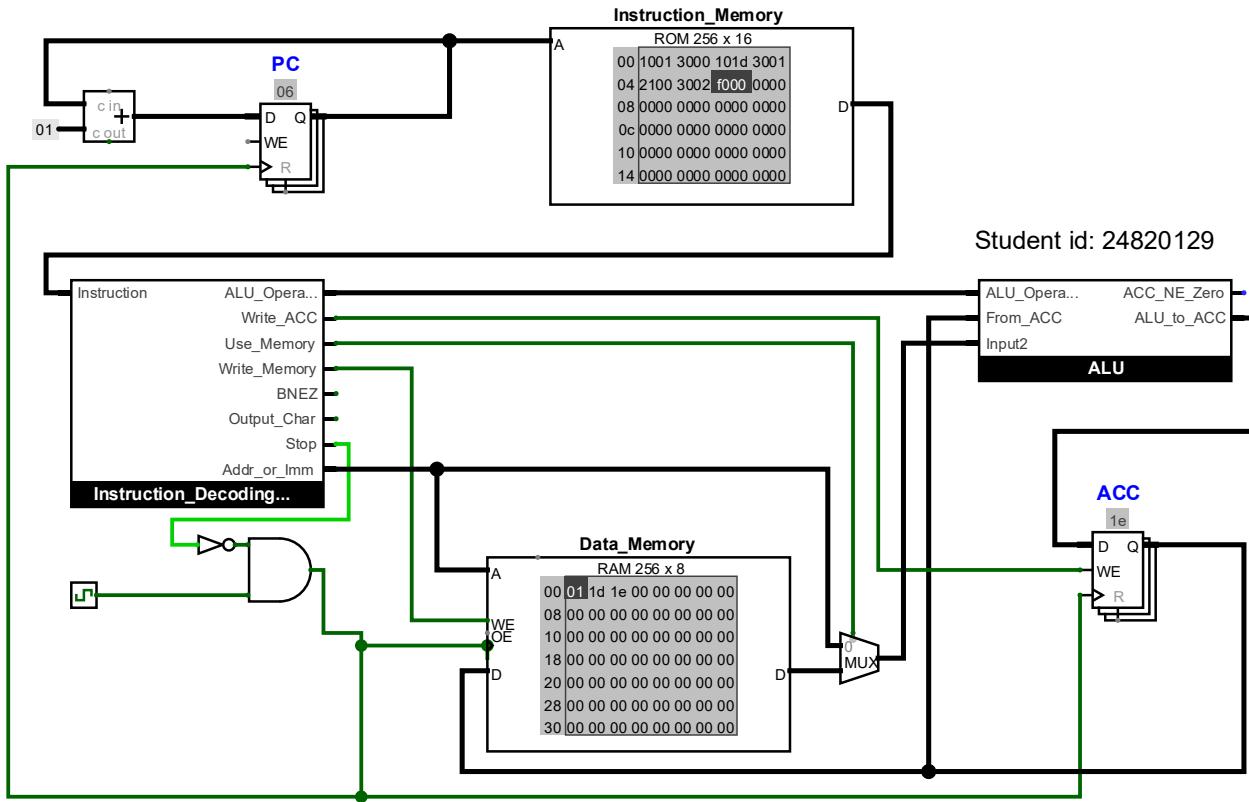
Both operands are now in memory address 00 and 01, with the sum being stored in memory address 02. The sum of 1e is the expected and correct result.

1.2 Completing the stop instruction

1.2.1 Circuit at start of Test



1.2.2 Circuit at end of Test



1.2.3 Explanation

Stop functionality was added to the simulation in this task. When opcode f000 is received by the instruction decoding unit, the stop signal is set to 1. By connecting the inverted stop signal (NOT) to an AND gate with the clock, the clock functions normally until the stop instruction is decoded. Once it is activated the AND gate is locked at value zero, blocking the clock signal from activating the Program Counter and ACC on the rising edge of the clock, halting the program at f000.

1.3 Bitwise Operations

1.3.1 Test program

```
v2.0 raw
1001 3000 # 1. put first value to ACC and store to 00
101D 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 NAND with value from 01
3003      # 6. store ACC value to address 03
2000 2301 # 5. load from 00 to ACC, bitwise OR with value from 01
3004      # 6. store ACC value to address 04
f000
```

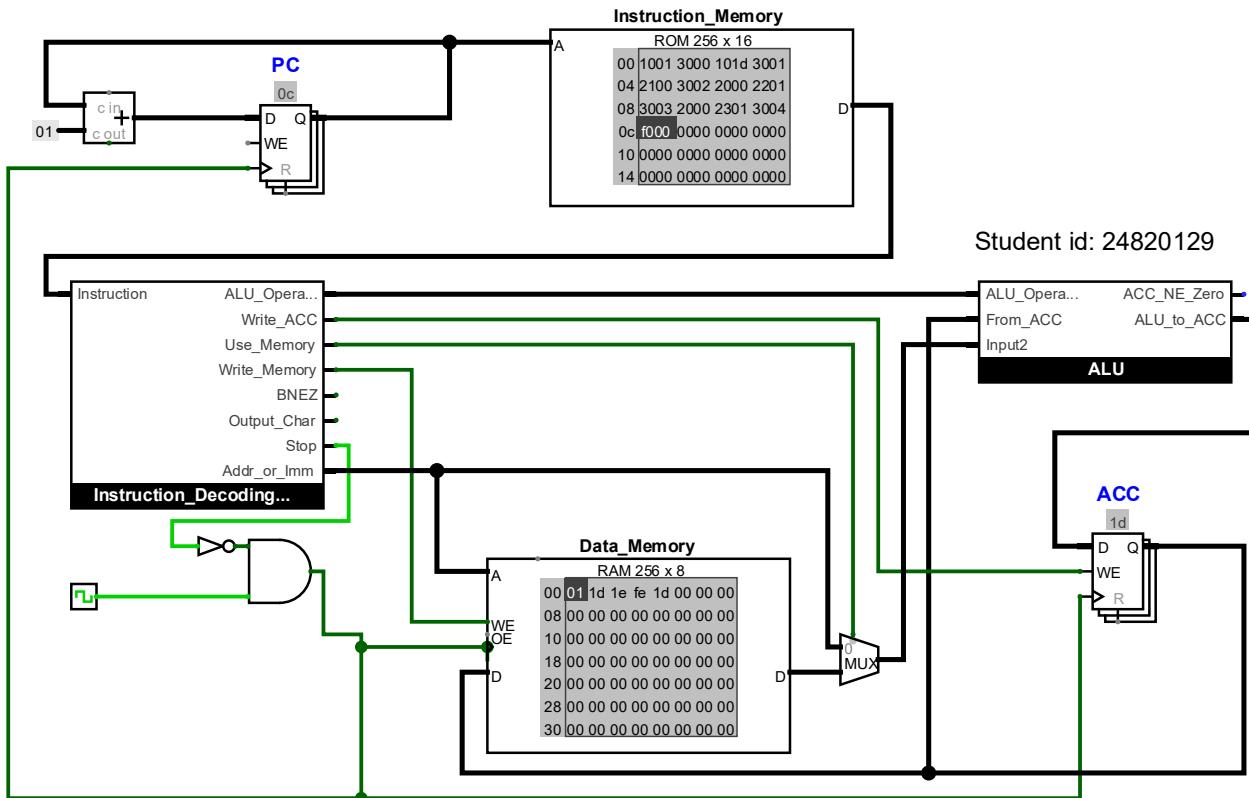
1.3.2 Expected result

Google Gemini 2.5 was used as a guide for the creation of the table below. (appendix No.1).

Binary Operand 1	Binary Operand 2	Bitwise Operation	Binary Working	Binary Result	Hexadecimal Result
00000001	00011101	NAND	$ \begin{array}{l} 00000001 \\ \text{AND } 00011101 \\ = 00000001 \end{array} $	11111110	FE

			NOT 00000001 = 11111110		
00000001	00011101	OR	00000001 OR 00011101 = 00011101	00011101	1D

1.3.3 Circuit at end of program



1.3.4 Result

The program loaded values 00 and 01 and performed the OR and NAND bitwise operations. The program then stored these values in memory locations 03 and 04.

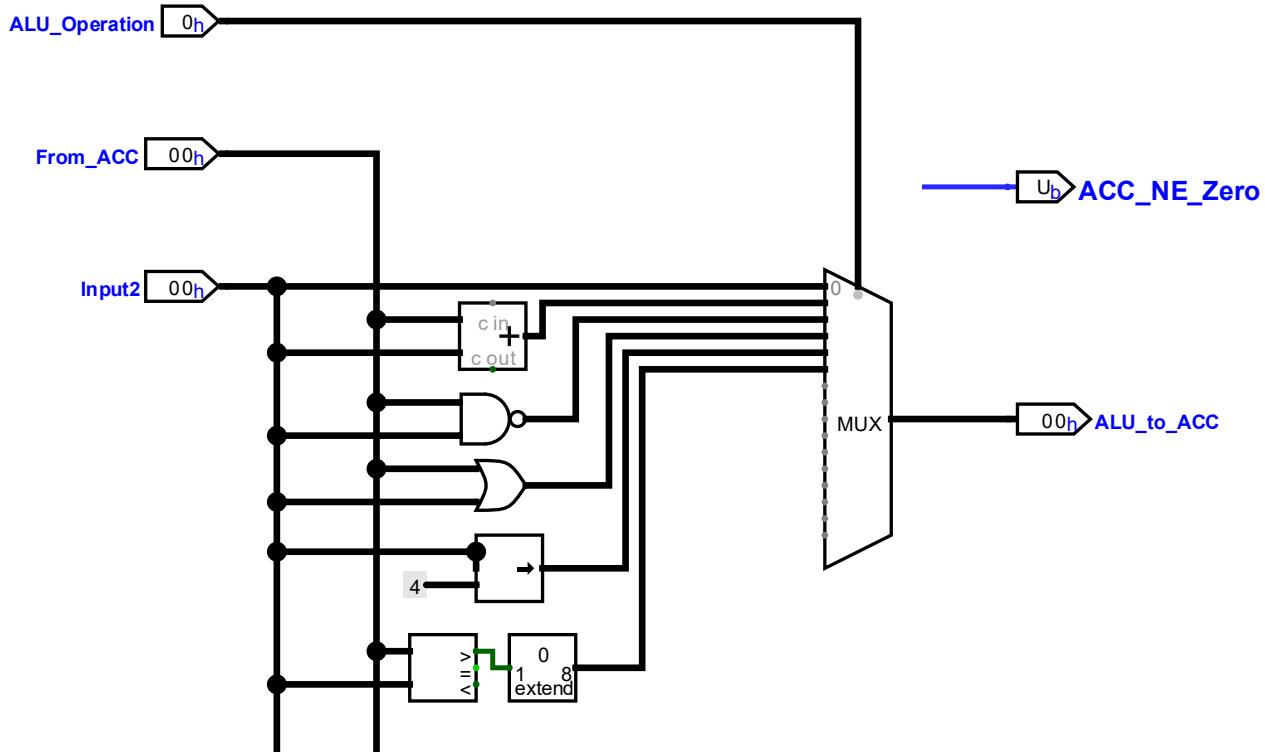
1.3.5 Explanation

The difference between the bitwise NAND and OR instruction is the second hexadecimal digit, the ALU opcode. The decoder sees opcode 2 in the first digit as an ALU operations and sends the second digit '3' to the ALU's MUX to perform the operation.

Observation of the MUX showed that opcode '3' was an OR operation, therefore the second digit of the instruction must be '3' for the ALU to perform a logical OR.

1.4 Amending the ALU

1.4.1 ALU circuit



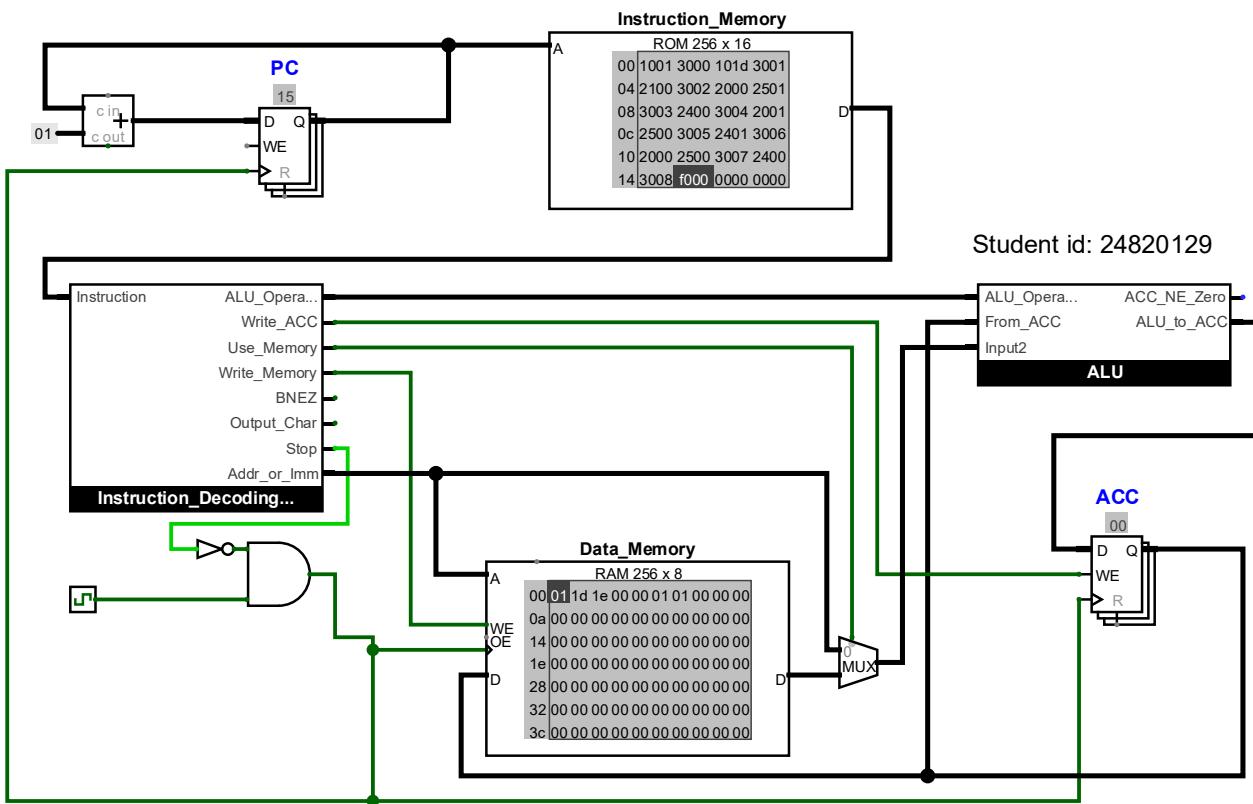
1.4.2 Test Program

```
v2.0 raw
1001 3000    # 1. put first value to ACC and store to 00
101D 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
# first number smaller then right shift of first value
2000 2501    # 5. load address 00 and compare with address 01
3003          # 6. store ACC value in address 03
2400          # 7. bit shift right with address 00
3004          # 8. store ACC value in address 04
# first number bigger then right shift of first value
2001 2500    # 9. load address 01 and compare with address 00
3005          # 10. store ACC value in address 05
2401          # 11. bit shift right with address 01
3006          # 12. store ACC value in address 06
# numbers equal then right shift of first value
2000 2500    # 13. load address 00 and compare with address 00
3007          # 14. store ACC value in address 07
2400          # 15. bit shift right with address 00
3008          # 16. store ACC value in address 08
f000          # 17. end program
```

1.4.3 Expected result

Test	FROM_ACC Value	Input 2 Value	Right shift of first value	Set greater than value
First number smaller	01	1d	00	00
First number bigger	1d	01	01	01
Both numbers equal	01	01	00	00

1.4.4 Circuit at end of program



1.4.5 Result

The program ran as expected, address 05 stored value 01 as FROM_ACC value was greater than input 2 and address 06 stored value 01, the output from a bit shift right on hex value 1d. all other addresses stored 00 matching the output of their respective operations.

1.4.6 Explanation

The 1-bit output of the comparator is converted to an 8-bit output using a bit extender, this is necessary to ensure compatibility with the ACC and CPU which use 8-bit representation.

1.5 Output Device

1.5.1 Test Data and Expected Result

v2.0 raw

48 65 6C 6C 6F 20 53 61 6D 20 30 31 32 39

Expected result:

Store "Hello Sam 0129" at respective memory addresses.

1.5.2 Test Program

v2.0 raw

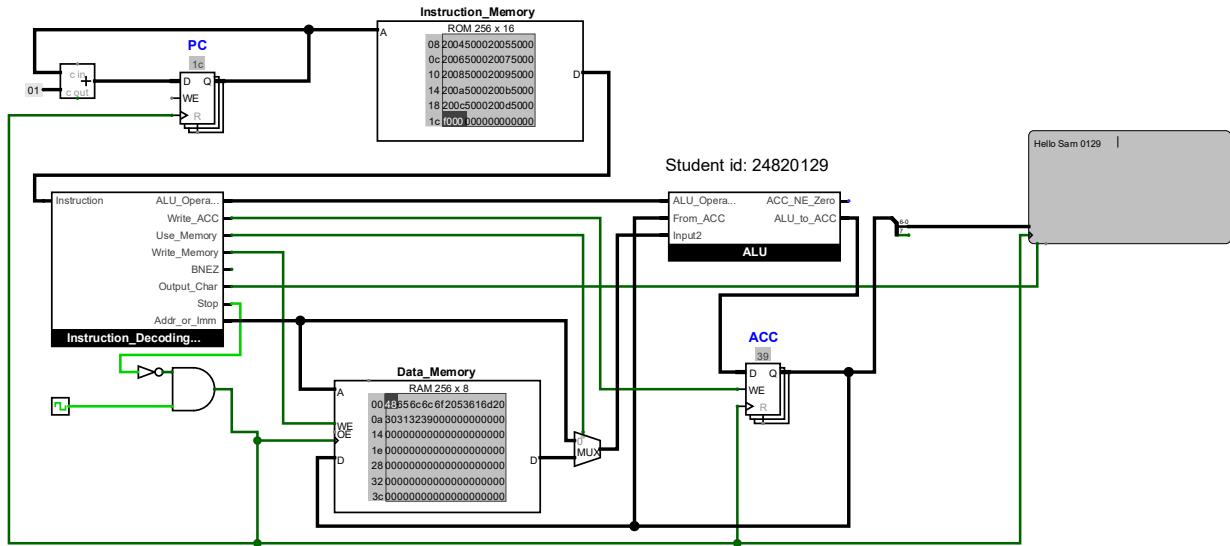
```
2000 5000 # load address 00 (H) and sent to ACC to display on TTY
2001 5000 # load address 01 (e) and sent to ACC to display on TTY
2002 5000 # load address 02 (l) and sent to ACC to display on TTY
```

```

2003 5000 # load address 03 (l)and sent to ACC to display on TTY
2004 5000 # load address 04 (o)and sent to ACC to display on TTY
2005 5000 # load address 05 (space)and sent to ACC to display on TTY
2006 5000 # load address 06 (S)and sent to ACC to display on TTY
2007 5000 # load address 07 (a)and sent to ACC to display on TTY
2008 5000 # load address 08 (m)and sent to ACC to display on TTY
2009 5000 # load address 09 (space)and sent to ACC to display on TTY
200A 5000 # load address 0A (0)and sent to ACC to display on TTY
200B 5000 # load address 0B (1)and sent to ACC to display on TTY
200C 5000 # load address 0C (2)and sent to ACC to display on TTY
200D 5000 # load address 0D (9)and sent to ACC to display on TTY
f000      # stop

```

1.5.3 Circuit at end of program



1.5.4 Result

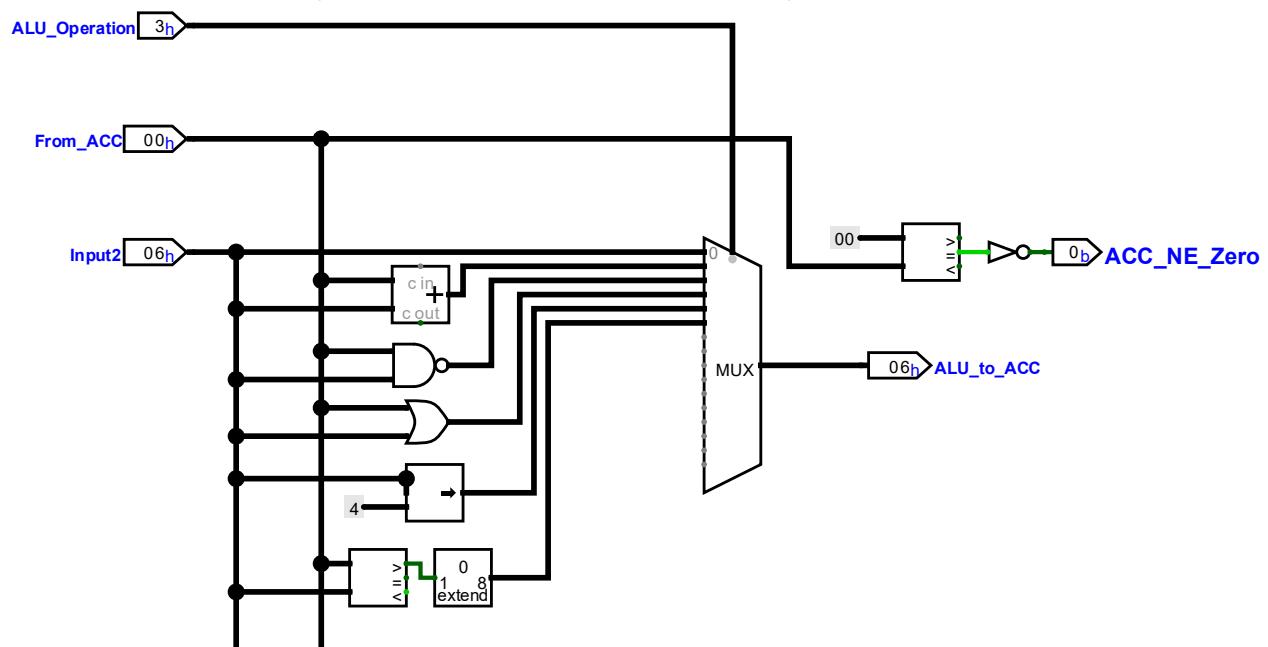
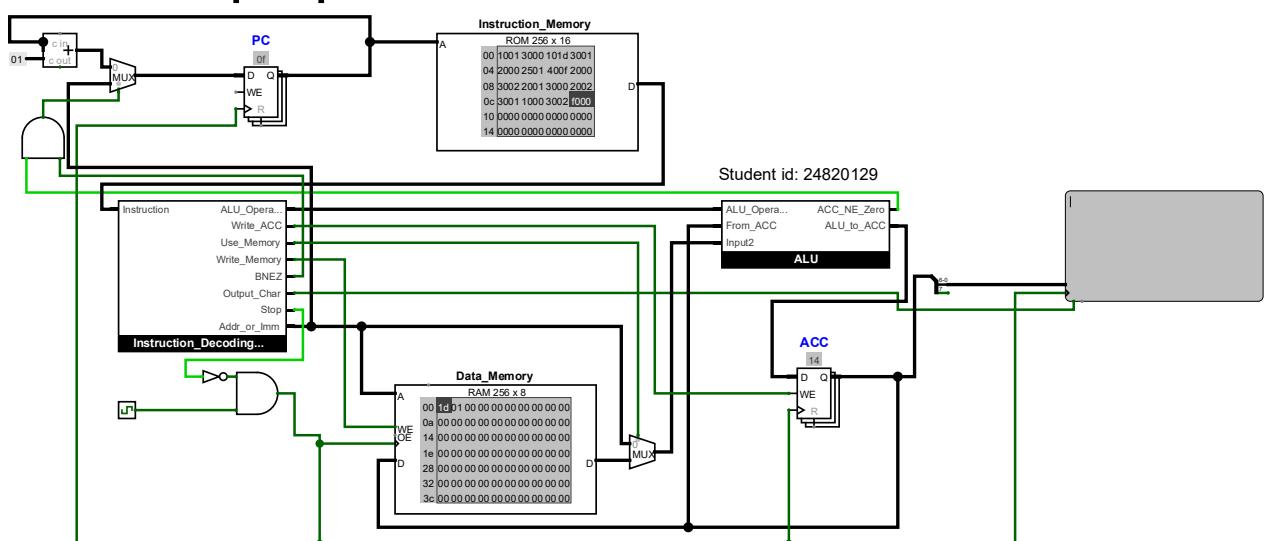
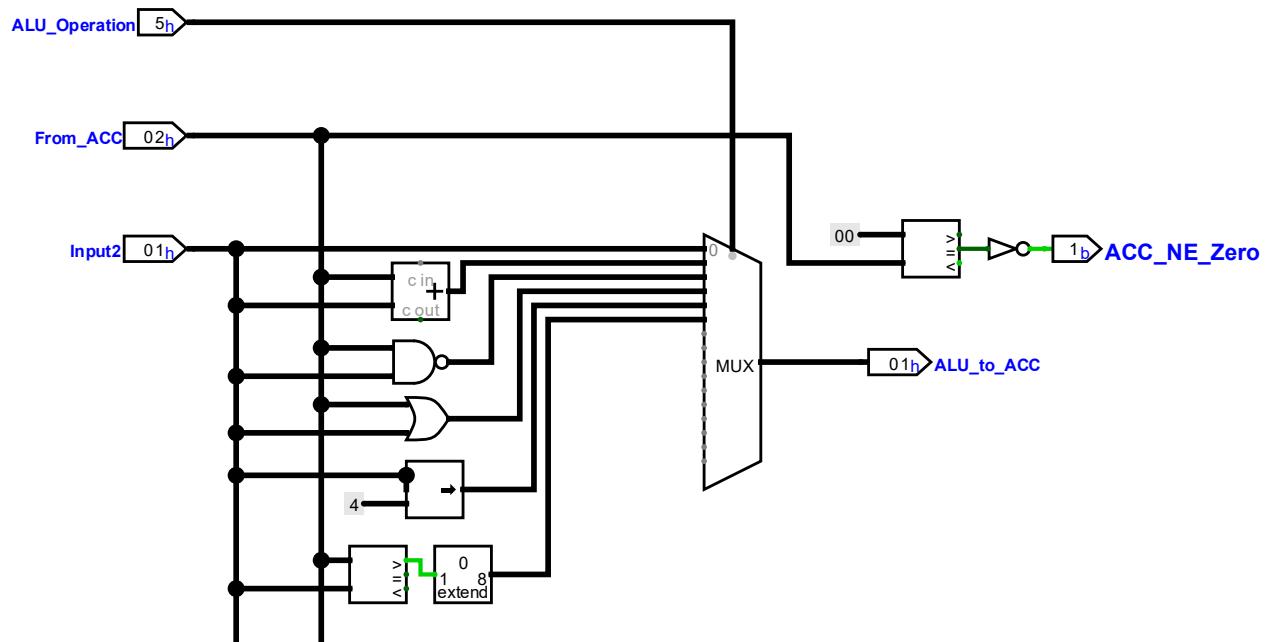
Message “Hello Sam 0129” was printed to the TTY.

1.5.5 Explanation

The small size of ASCII is its limitations. ASCII does not allow for any other representation than American English, this is why UTF-8 was developed as an encoding for Unicode standard cleverly using bit patterns in the first byte to allow for dynamic use if 1-4 bytes without reducing efficiency (Tihane, 2023). This research has helped me further understand that to create clever solutions in computing, thinking outside the box about how we utilise each bit is necessary.

1.6 Branch instruction

1.6.1 ALU circuit and testing



1.6.2 Test program

TEST 1

v2.0 raw

1001 3000 # 1. load 01 to ACC and store at 00
101D 3001 # 2. load 1D to and store at 01

2000 2501 # 3. compare ACC (address 00) value with address 01

400F #4. branch to stop program if 00>01 (value in ACC NE ZERO = 1)

2000 3002 #5. Load value from address 00 to ACC and store at address 02 in data memory
2001 3000 #6. Load value from address 01 to ACC and store at address 00 in data memory
2002 3001 #7. Load value from address 02 to ACC and store at address 01 in data memory
1000 3002 #8. Set the value at address 02 in data memory to be 00

f000 #9. end program

TEST 2 (VALUES SWAPPED)

v2.0 raw

101D 3000 # 1. load 1D to ACC and store at 00
1001 3001 # 2. load 01 to and store at 01

2000 2501 # 3. compare ACC (address 00) value with address 01

400F #4. branch to stop program if 00>01 (value in ACC NE ZERO = 1)

2000 3002 #5. Load value from address 00 to ACC and store at address 02 in data memory
2001 3000 #6. Load value from address 01 to ACC and store at address 00 in data memory
2002 3001 #7. Load value from address 02 to ACC and store at address 01 in data memory
1000 3002 #8. Set the value at address 02 in data memory to be 00

f000 #9. end program

TEST 3 (VALUES =)

1001 3000 # 1. load 1D to ACC and store at 00
1001 3001 # 2. load 01 to and store at 01

2000 2501 # 3. compare ACC (address 00) value with address 01

400F #4. branch to stop program if 00>01 (value in ACC NE ZERO = 1)

2000 3002 #5. Load value from address 00 to ACC and store at address 02 in data memory
2001 3000 #6. Load value from address 01 to ACC and store at address 00 in data memory
2002 3001 #7. Load value from address 02 to ACC and store at address 01 in data memory
1000 3002 #8. Set the value at address 02 in data memory to be 00

f000 #9. end program

1.6.3 Expected Results

TEST 1

Step	Code	Description	ACC Value	Memory Values
1	1001 3000	Load value 01 to ACC and store at address 00	01	00: 00 01: 00 02: 00
2	101D 3001	Load value 1D to ACC and store at address 01	1D	00: 01 01: 00 02: 00
3	2000 2501	Load address 00 to ACC and compare with address 01	00	00: 01 01: 1D 02: 00
4	400F	Branch if the ACC does not equal zero, so no branch	00	00: 01 01: 1D 02: 00
5	2000 3002	Load value address 00 to ACC and store at address 02 temporarily	01	00: 01 01: 1D 02: 01
6	2001 3000	Load address 01 to ACC and store at address 00	1D	00: 1D 01: 1D 02: 01
7	2002 3001	Load address 02 to ACC and store at address 01	01	00: 1D 01: 01 02: 01
8	1000 3002	Store value 00 at address 02, clearing address 02	00	00: 1D 01: 01 02: 00
9	f000	End program	00	00: 1D 01: 01 02: 00

TEST 2 (VALUES SWAPPED)

Step	Code	Description	ACC Value	Memory Values
1	1001 3000	Load value 1D to ACC and store at address 00	1D	00: 1D 01: 00 02: 00
2	101D 3001	Load value 01 to ACC and store at address 01	01	00: 1D 01: 01 02: 00

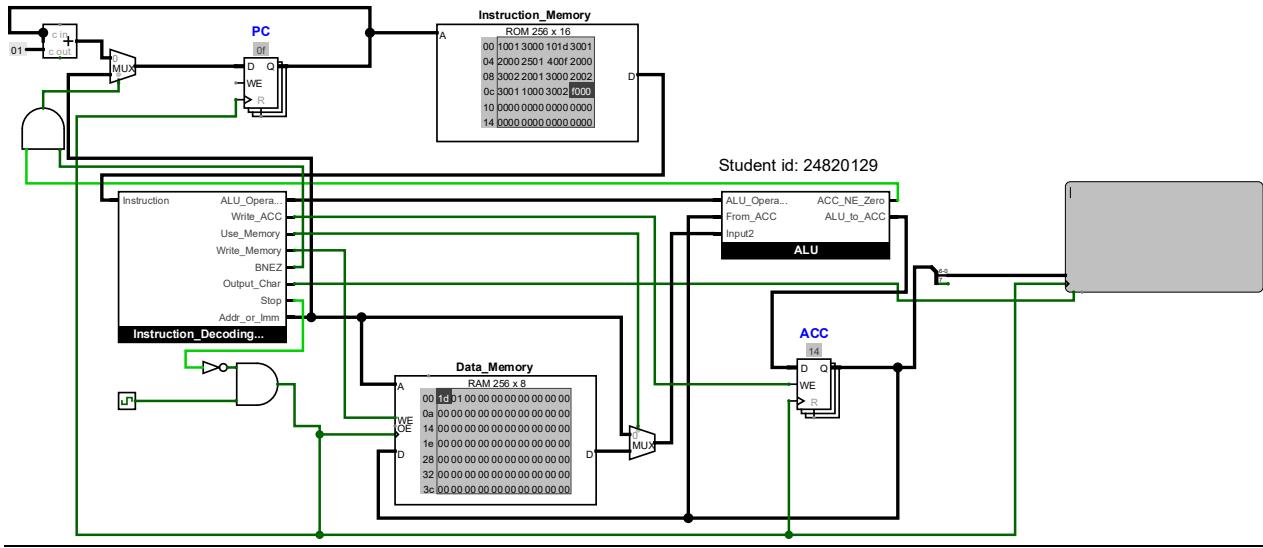
3	2000 2501	Load address 00 to ACC and compare with address 01	01	00: 1D 01: 01 02: 00
4	400F	Branch if the ACC does not equal zero, so branch	01	00: 1D 01: 01 02: 00
5	F000	Stop program	01	00: 1D 01: 01 02: 00

TEST 3 (VALUES =)

Step	Code	Description	ACC Value	Memory Values
1	1001 3000	Load value 01 to ACC and store at address 00	01	00: 00 01: 00 02: 00
2	1001 3001	Load value 01 to ACC and store at address 01	01	00: 01 01: 00 02: 00
3	2000 2501	Load address 00 to ACC and compare with address 01	00	00: 01 01: 01 02: 00
4	400F	Branch if the ACC does not equal zero, so no branch	00	00: 01 01: 01 02: 00
5	2000 3002	Load value address 00 to ACC and store at address 02 temporarily	01	00: 01 01: 01 02: 01
6	2001 3000	Load address 01 to ACC and store at address 00	01	00: 01 01: 01 02: 01
7	2002 3001	Load address 02 to ACC and store at address 01	01	00: 01 01: 01 02: 01
8	1000 3002	Store value 00 at address 02, clearing address 02	00	00: 01 01: 01 02: 00
9	f000	End program	00	00: 01 01: 01 02: 00

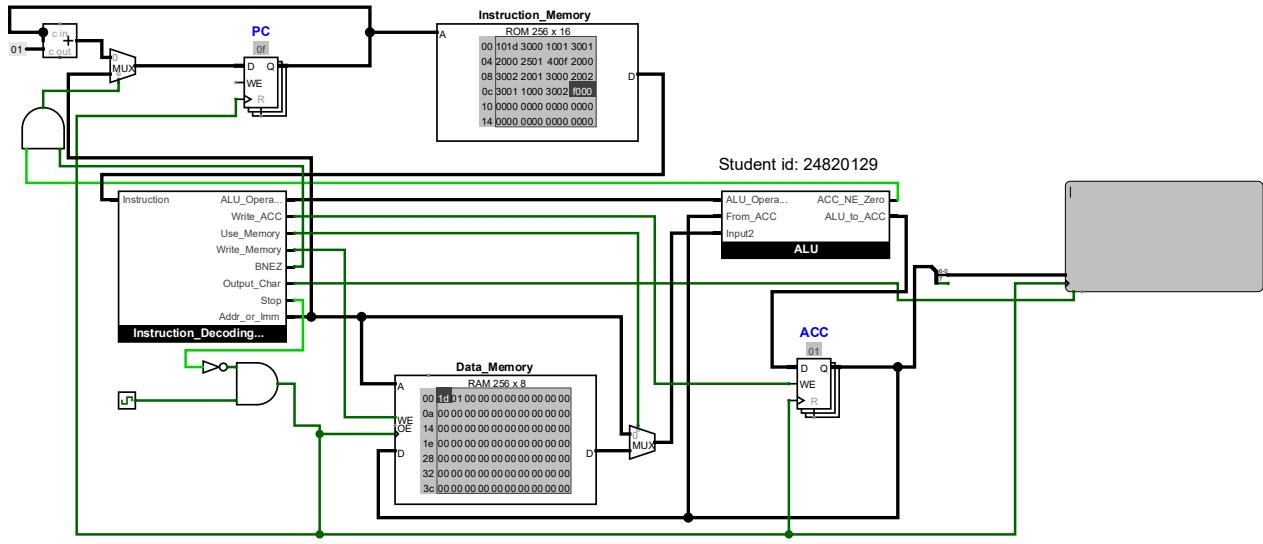
1.6.4 Results

TEST 1



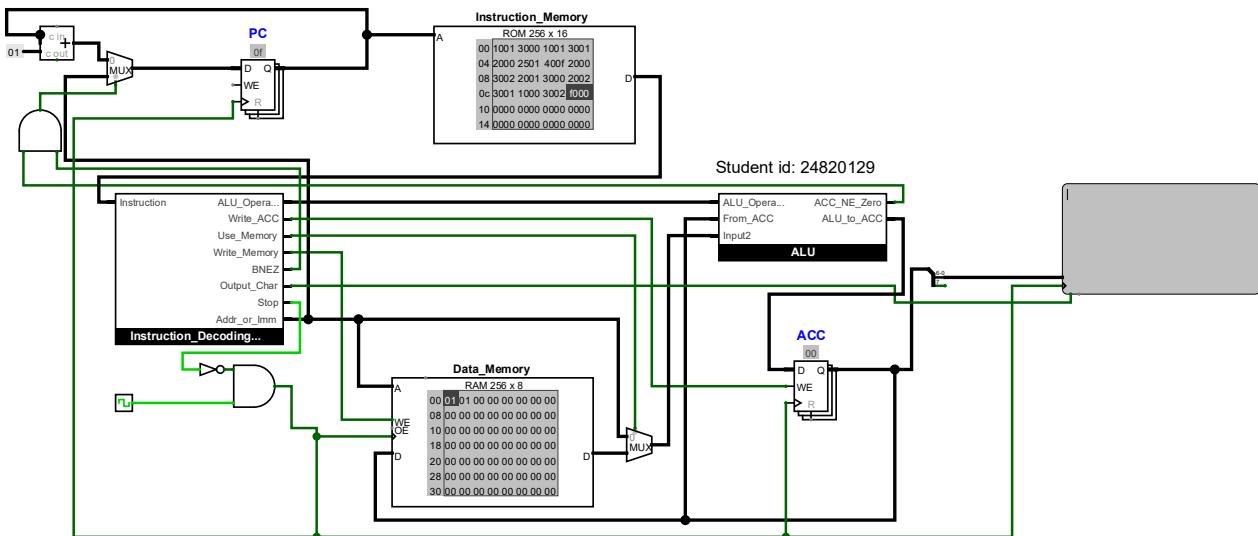
In this test address 00 is greater than address 01, so the branch instruction is not used, and the values are swapped. Then the program stops.

TEST 2 (VALUES SWAPPED)



In this test address 00 is less than address 01, so the branch instruction is executed, and the program stops on the next instruction.

TEST 3(VALUES =)



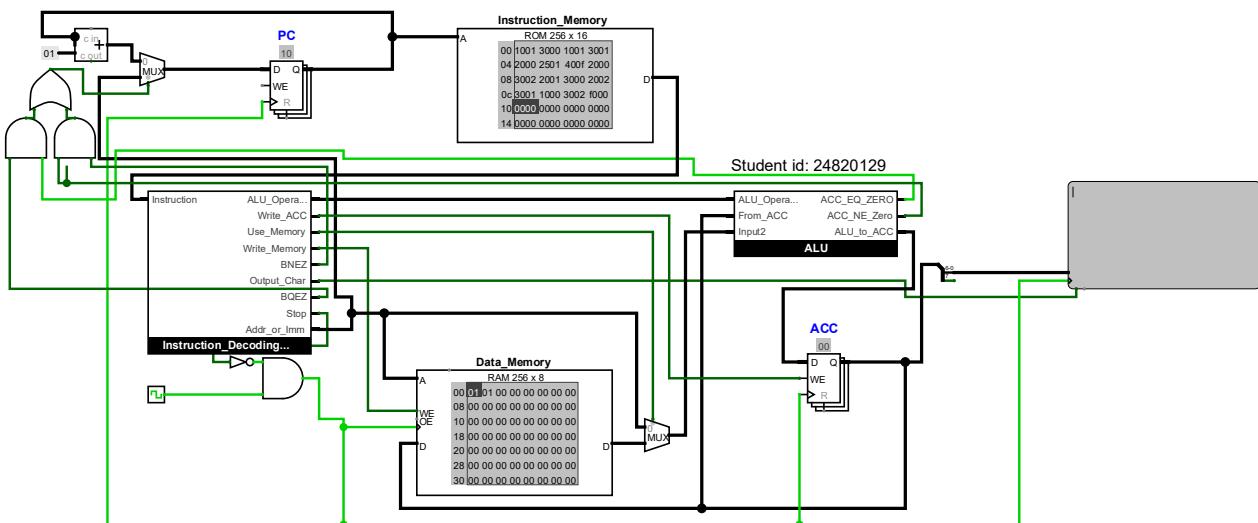
In this test address 00 is equal to 01, so the branch instruction is not executed and the address values are swapped.

1.6.5 Explanation

The opcode for the branch instruction was identified using the “CPU Simulation Background Information document. Considering output 4 of the decoder is the BNEZ signal (Branch Not Equal Zero), CPU opcode 4 is used to initiate this type of branch. When the BNEZ signal is set to 1 and the ACC is not equal to zero, the system sets the Program Counter to the immediate address specified in the instruction (the 2 LSBs) in this case the stop instruction.

Further research into RISC-V architecture uncovered the BEQZ instruction (Branch Equal Zero). This instruction initiates a branch if the value compared (in this CPU the ACC) is equal to zero.(Green 2024)

Implementing a BEQZ instruction involved the creation of a new CPU opcode (6) and the use of a comparator to compare the value inside the ACC to zero. The output of this comparison connected to an AND gate with the BEQZ decoder signal and then to a final OR gate, with the same logic for the BNEZ signal, is then used to control the select bit of the Program Counters MUX. Now if the condition were to be true, the PC is set to the immediate value inside the instruction.



This research provided insight as to the ways in which programs are controlled at a low level. It highlighted the importance of flags to control the flow of programs to allow for more complex programs like sorting algorithms and

games to function. Understanding these concepts has proved useful in the development of simple games in task 1.8 and 2.8.

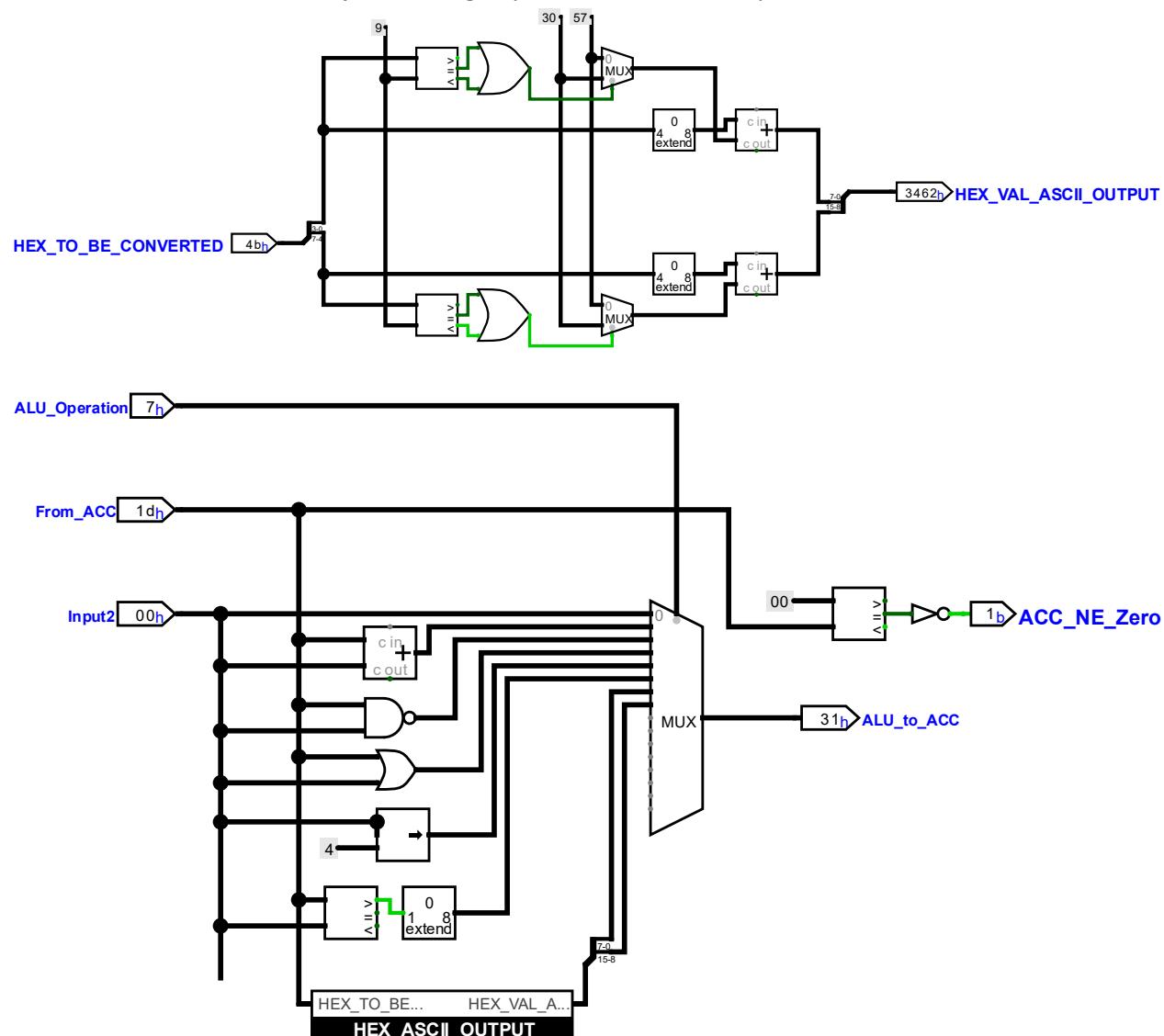
1.7 Loop

1.7.1 Displaying Hex Output

This task requires the conversion of a hexadecimal value to its corresponding ASCII representation. For example, hexadecimal value 0x4b would display the character K, to display “4b” the sub circuit needs to split the hexadecimal value into nibbles and convert the nibbles to their respective ASCII codes.

For hex values 0-9 which appear in succession in the in the ASCII table, the conversion is found by adding an offset of 0x30. For hex letters a-f the offset is 0x57, this value is calculated by subtracting the decimal value of the hex letter ($a = 10$) from the ASCII code converted to decimal ($0x61 = 97$) leaving a difference of decimal 87 (0x57).

The circuit uses a comparator to determine the correct offset. Firstly the 8-bit input is split into two nibbles; if the comparator finds the value of the nibble to be less than or equal to 0x9, the mux selects offset 0x30; otherwise offset 0x57 is used. As there is a bit width mismatch the two nibbles are extended using a zero-bit extender before addition. The ASCII code are rejoined using a splitter for the final output.



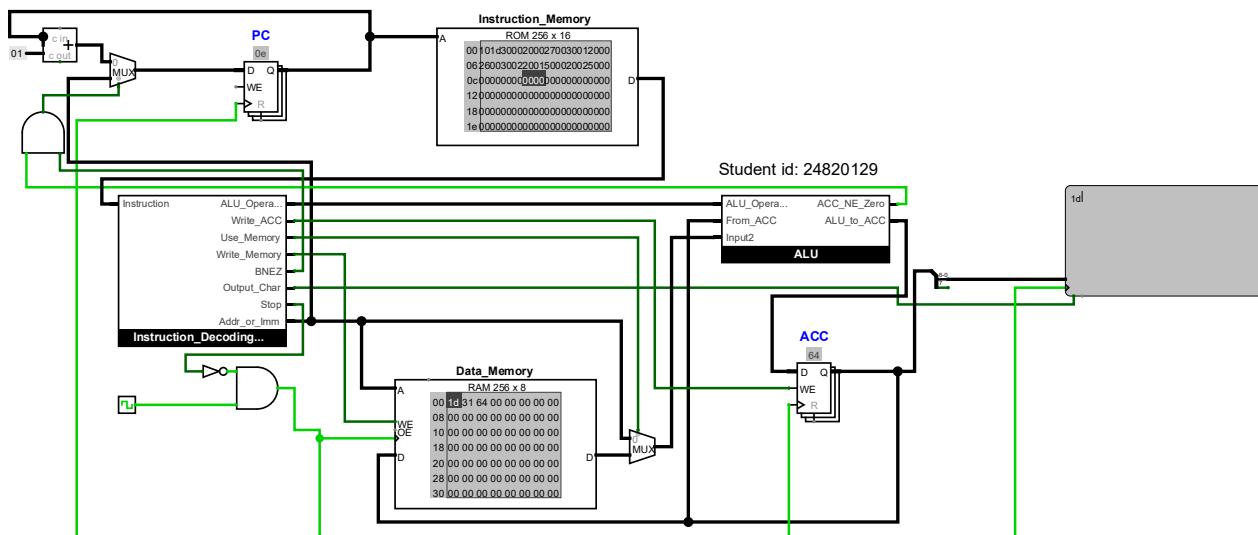
This is the amended ALU with the new conversion subcircuit included, my logic includes 2 new instructions ALU OPCODE 6 will output the LSB from the conversion and OPCODE 7 will output the MSB. This is necessary as the CPU,

ACC, ALU etc only works with 8 bits whereas the output from our conversion is 16bits This value can then be stored and sent to the ALU for output to the TTY using the appropriate CPU code.

Considering ALU and ACC use a bit width of 8 and the conversion output is 16 bits, the output from the conversion is split into two 8-bit outputs; Consequently, opcode 6 deals with the least significant bits from the conversion and opcode 7 deals with the most significant bits.

Test program for amendments to the ALU.

```
v2.0 raw
101D 3000 # 1. send hex value 1D to the ACC and store at address 00
2000 2700 # 2. send address 00 to the ACC and do HEX conversion and output MSB
3001      # 3. store ACC value at address 01
2000 2600 # 4. send address 00 to the ACC and do HEX conversion and output LSB
3002      # 5. store ACC value at address 02
2001 5000 # 6. send address 01 to the ACC and output on TTY
2002 5000 # 7. send address 02 to the ACC and output on TTY
```



Simulation after running the program showing functionality

1.7.2 Running Loop

Test 1

```
v2.0 raw
#1. load values to respective addresses
1001 3000
101D 3001

#start of loop

#2. output value from address 00 and a whitespace char to tty
2000 2700
5000
2000 2600
5000
1020 5000

#3. add -1 to value at 00 and store at 00
```

2000 11FF
3000

2000 2501 #4. compare ACC (address 00) value with address 01 (if 00>01) ACC = 1

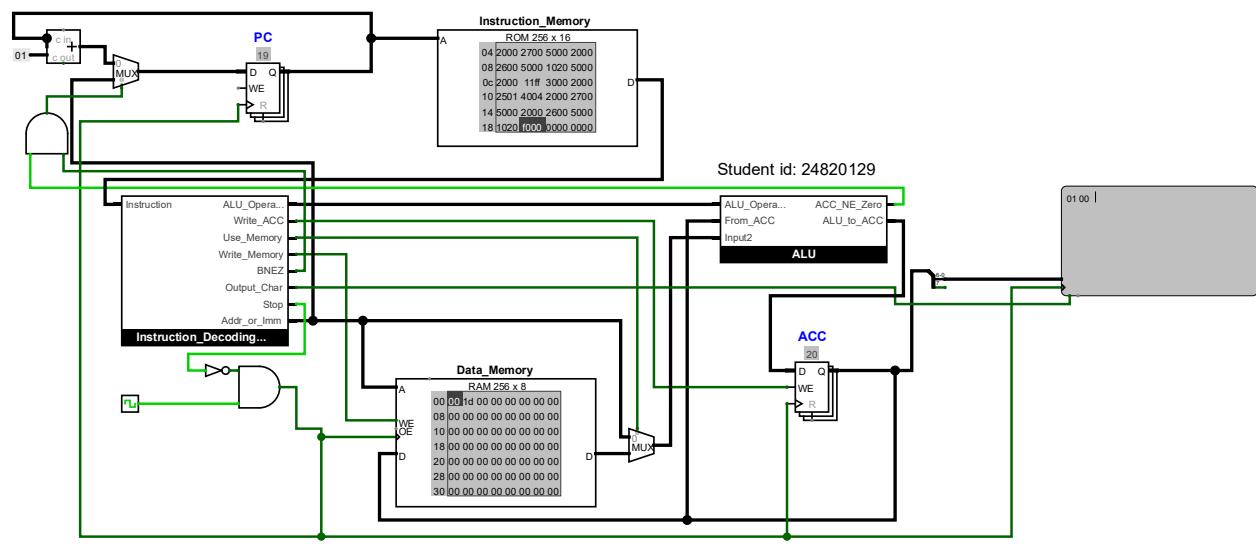
4004 #5. branch to start of loop if ACC NE

#6. output value from address 00 and a white space char

2000 2700
5000
2000 2600
5000
1020

f000 #7. end program

Simulation after program has run:



Test 2 (VALUES SWAPPED)

```
v2.0 raw
#1. load values to respective addresses
101D 3000
1001 3001

#start of loop

#2. output value from address 00 and a whitespace char to tty
2000 2700
5000
2000 2600
5000
1020 5000

#3. add -1 to value at 00 and store at 00
2000 11FF
3000
```

2000 2501 #4. compare ACC (address 00) value with address 01 (if 00>01) ACC = 1

4004 #5. branch to start of loop if ACC NE

#6. output value from address 00 and a whitespace char

2000 2700

5000

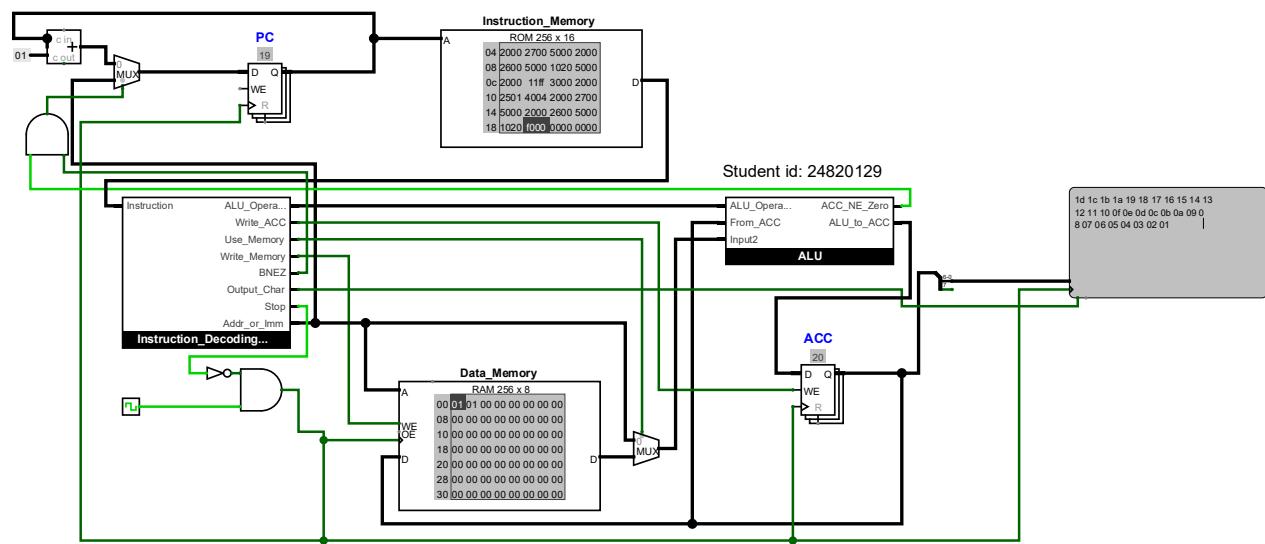
2000 2600

5000

1020

f000 #7. end program

Simulation after program has run:



Test3 (VALUES EQUAL)

v2.0 raw

#1. load values to respective addresses

1001 3000

1001 3001

#start of loop

#2. output value from address 00 and a whitespace char to tty

2000 2700

5000

2000 2600

5000

1020 5000

#3. add -1 to value at 00 and store at 00

2000 11FF

3000

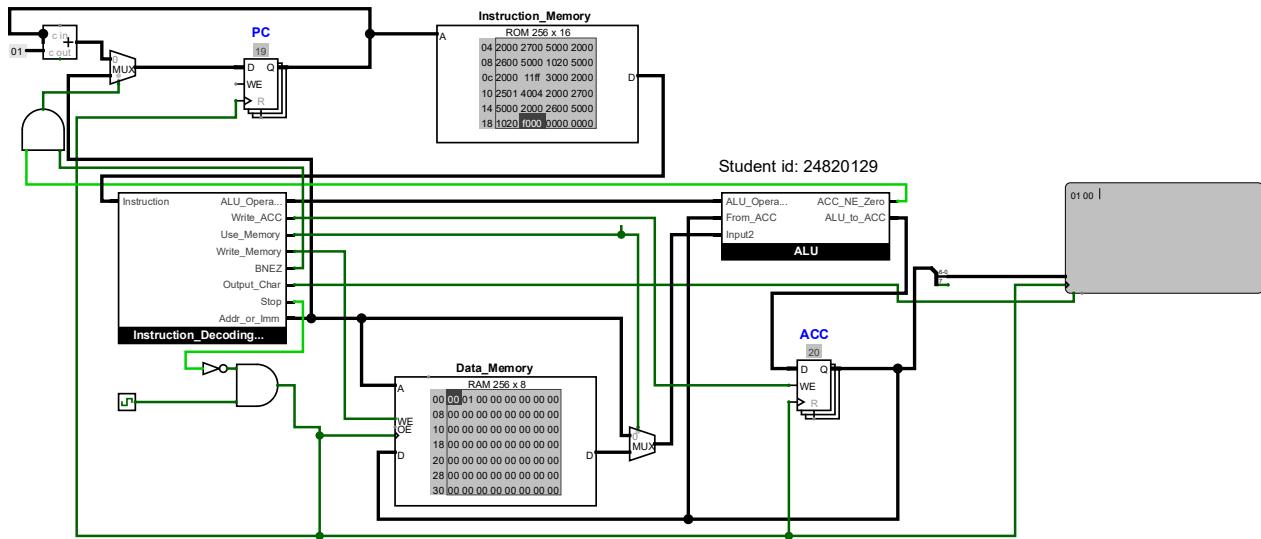
2000 2501 #4. compare ACC (address 00) value with address 01 (if 00>01) ACC = 1

4004 #5. branch to start of loop if ACC NE

#6. output value from address 00 and a white space char
2000 2700
5000
2000 2600
5000
1020

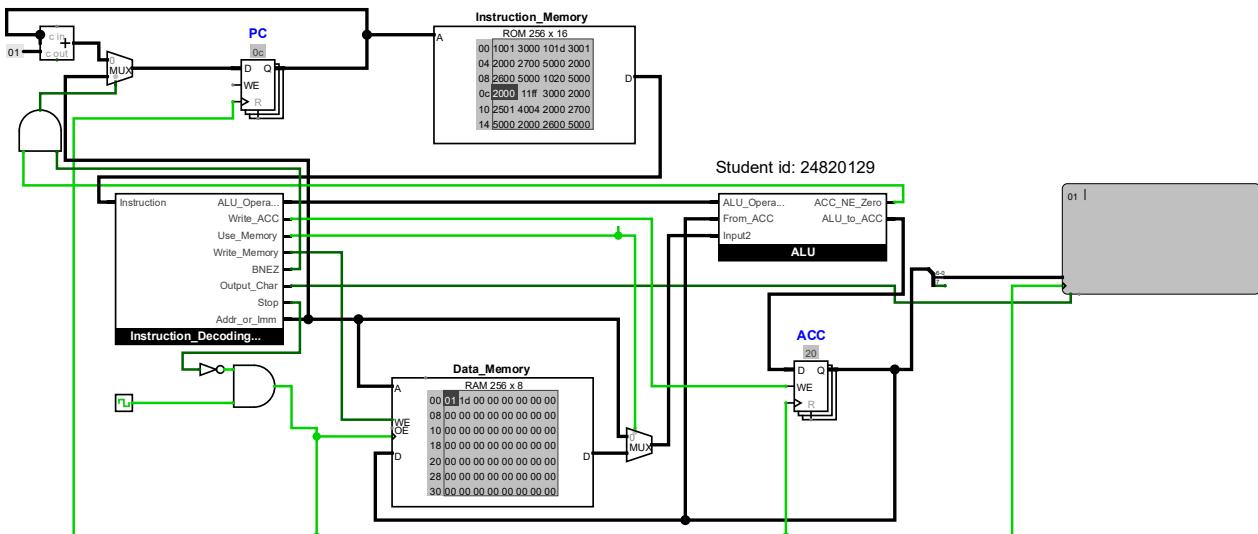
f000 #7. end program

Simulation after program has run:



1.7.3 Explanation

#2. output value from address 00 and a whitespace char to tty
2000 2700
5000
2000 2600
5000
1020 5000

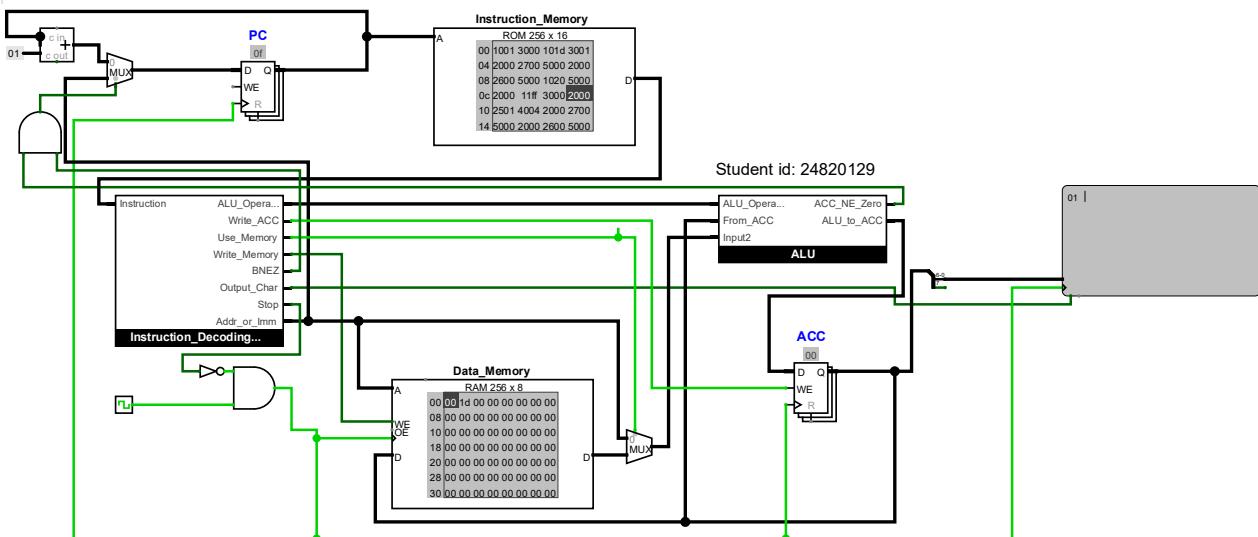


The first part of the loop outputs the HEX value stored at address 00 to the TTY and then a whitespace character using the immediate value of the instruction.

#3. add -1 to value at 00 and store at 00

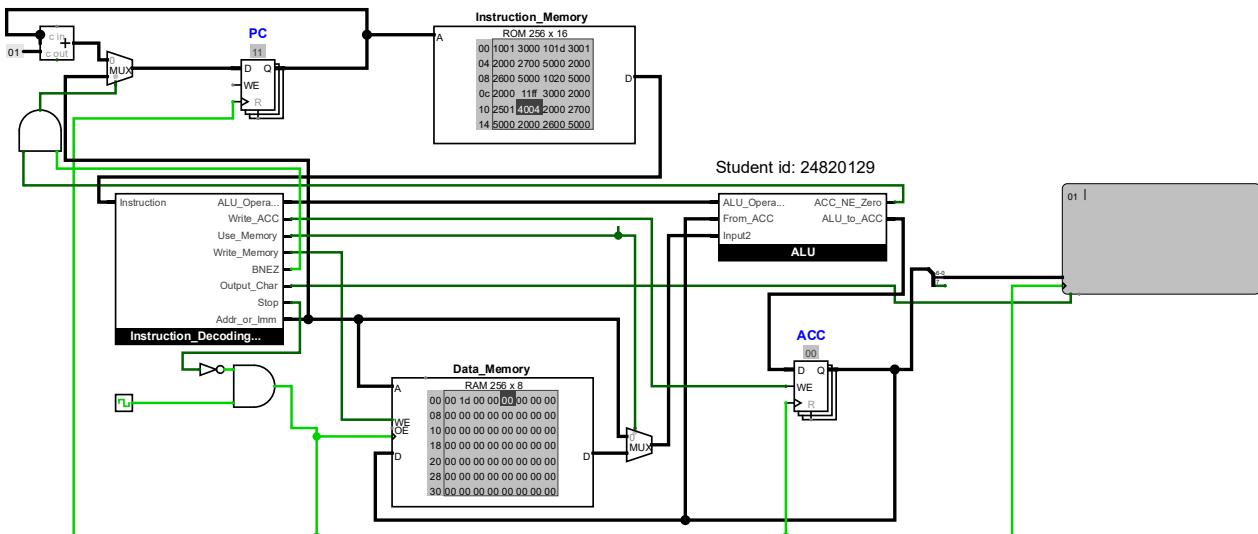
2000 11FF

3000



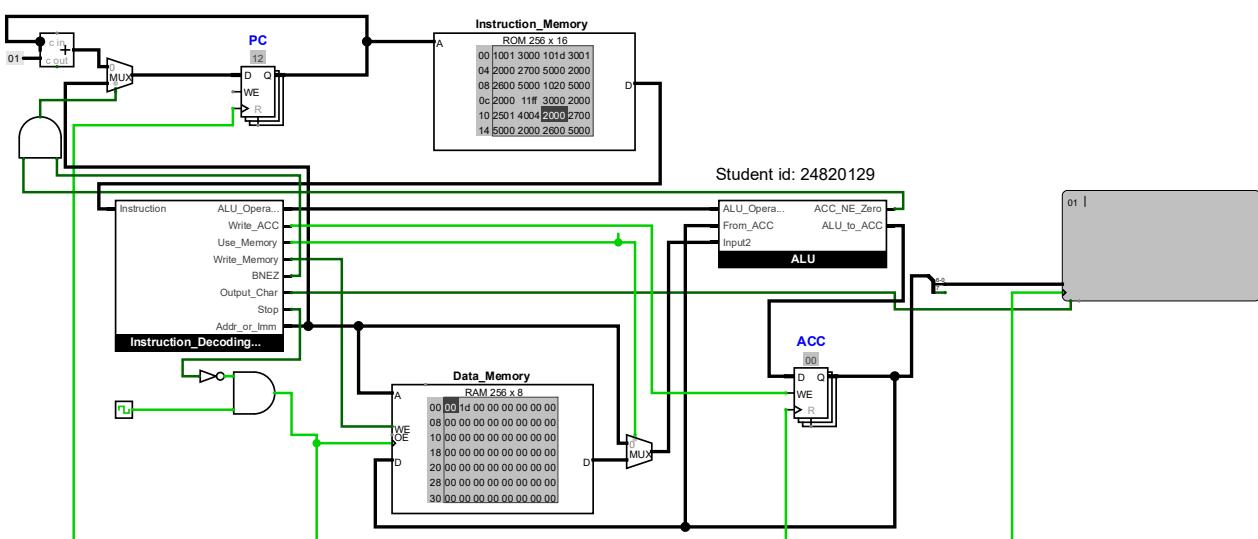
The second part of the loop adds -1 to the value stored at address 00 using the immediate value from the instruction. The value from the ACC is then stored in address 00.

2000 2501 #4. compare ACC (address 00) value with address 01 (if 00>01) ACC = 1



The third part of the loop compares the value in the ACC with the value in address 01. If $00 > 01$ the ACC_NE_ZERO signal is set to 1.

4004 #5. branch to start of loop if ACC NE



The fourth part of the loop will branch to the start of the loop if the ACC_NE_ZERO value is 1 by also setting the BNEZ to 1.

When the two numbers are equal the loop should not run at all, as 01 is not greater than 01; however, the loop runs once. This error occurs because the condition check happens at the end of the loop, forcing the loop to run at least once. To fix the error the comparison needs to be made at the beginning of the loop.

1.8 Game

1.8.1 Game Design and Logic

The chosen game for this task is text-based rock paper scissors as this meets the criteria of taking user input. The players input will be taken from 3 buttons: rock, paper and scissors respectively. The computers turn will be decided using random number generation. To find a starting place for development.

Initial Pseudocode:

```
#GAME MENU
```

```

#START OF MENU LOOP

PRINT TO TTY "WOULD YOU LIKE TO START A GAME OR QUIT?"
PRINT TO TTY "press button 1 to start game"
PRINT TO TTY "press button 2 to exit"

#GAME LOOP

IF PRESS BUTTON == 1
START GAME

#START OF GAME LOOP

PRINT TO TTY "PLEASE SELECT ROCK, PAPER OR SCISSORS"
PRINT TO TTY "1 = Rock 2 = Paper 3 = Scissors"
RANDOM NUMBER GENERATE NUMBER 1-3

#CALCULATE WINNER

# DRAW LOGIC
IF PLAYER INPUT = 1 AND RANDOM = 1
OR
IF PLAYER INPUT = 2 AND RANDOM = 2
OR
IF PLAYER INPUT = 3 AND RANDOM = 3
PRINT TO TTY "ITS A DRAW"
FLAG GAME TO FINISHED AND GO TO START OF MENU LOOP

#PLAYER WINS LOGIC
IF PLAYER INPUT = 1 AND RANDOM = 3
OR
IF PLAYER INPUT = 2 AND RANDOM = 1
OR
IF PLAYER INPUT = 3 AND RANDOM = 2
PRINT TO TTY "THE PLAYER WINS"
FLAG GAME TO FINISHED AND GO TO START OF MENU LOOP

#PLAYER LOSES LOGIC
ELSE
PRINT TO TTY "PLAYER LOSES"
FLAG GAME TO FINISHED AND GO TO START OF MENU LOOP

```

To determine the winner of the round, Google Gemini was used for this formula “Result = (Player - Computer +3) (mod3)” (Appendix C number 2). According to this formula, a result of 0 means a tie, 1 is a player win and 2 is a computer win. With the pseudocode and winner formula created a list of hardware modifications could be created.

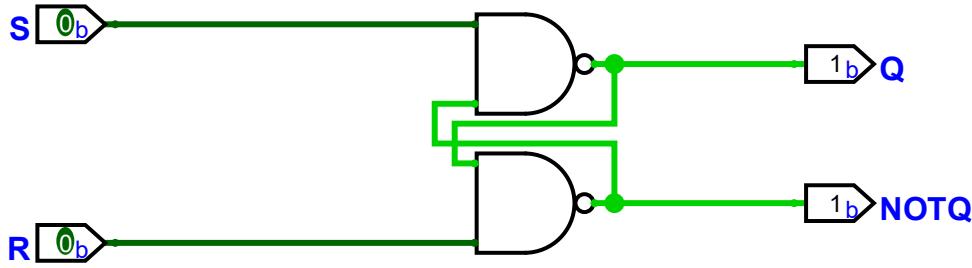
Necessary hardware modifications:

1. User input
2. Random number generation
3. Modulo and Subtraction for winner calculations.

1.8.2 Hardware and Software Modifications

User input

To take user input 3 buttons are used one button for each choice of rock, paper or scissors. Originally registers were used to store the input from the buttons; however, registers only store data on the rise of the clock signal so this would not work for asynchronous user input. To address this SR latches (taught in week 5) were used to hold the user's input, the SR already inside Logisim had the same issues as the original registers so custom SR latches were used (built in week 5).



So user input could be stored, the existing input 2 MUX was expanded. Select bit value 2 (10) now sends user input to the ACC.

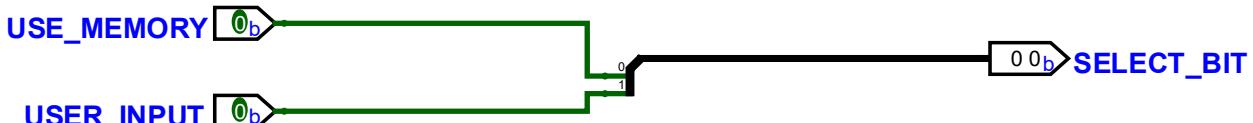
New Opcodes:

CPU opcode 6 User_input: When this signal is active the select_input_2 subcircuit sets the MUX select bit to 2 (10) sending the user input to the ACC.

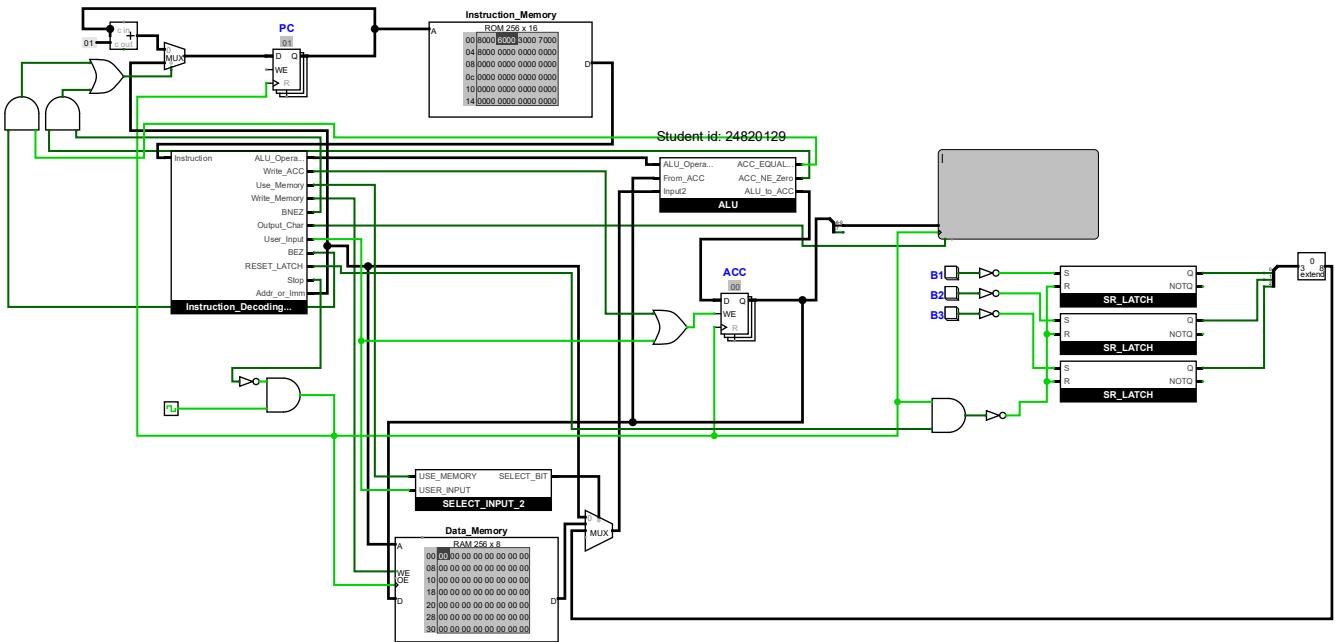
CPU opcode 7 BEZ (branch equal zero): When this signal is active the program will branch to the immediate address specified in the instruction if the ACC value is zero. This allows the program to loop until the ACC is a non-zero value (user input is taken) which eliminates timing issues with taking user input.

CPU opcode 8 RESET_LATCH: When this signal is active, on the rising edge of the clock all latches will be reset. This opcode is used at the start of the program to initialise the latches, removing an issue with the latches being in an illegal state due to unknown previous state. This opcode is also used to wipe previous user input, before taking input again.

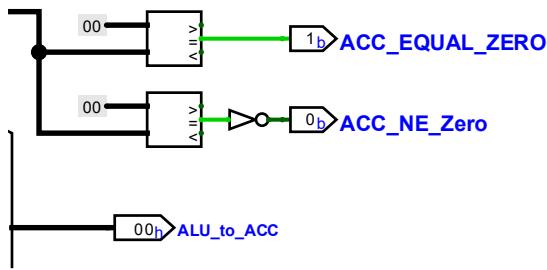
select_input_2 subcircuit:



Computer with user_input, BEZ and RESET_Latch CPU instructions:



ACC_EQUAL_ZERO check for BEZ instruction:



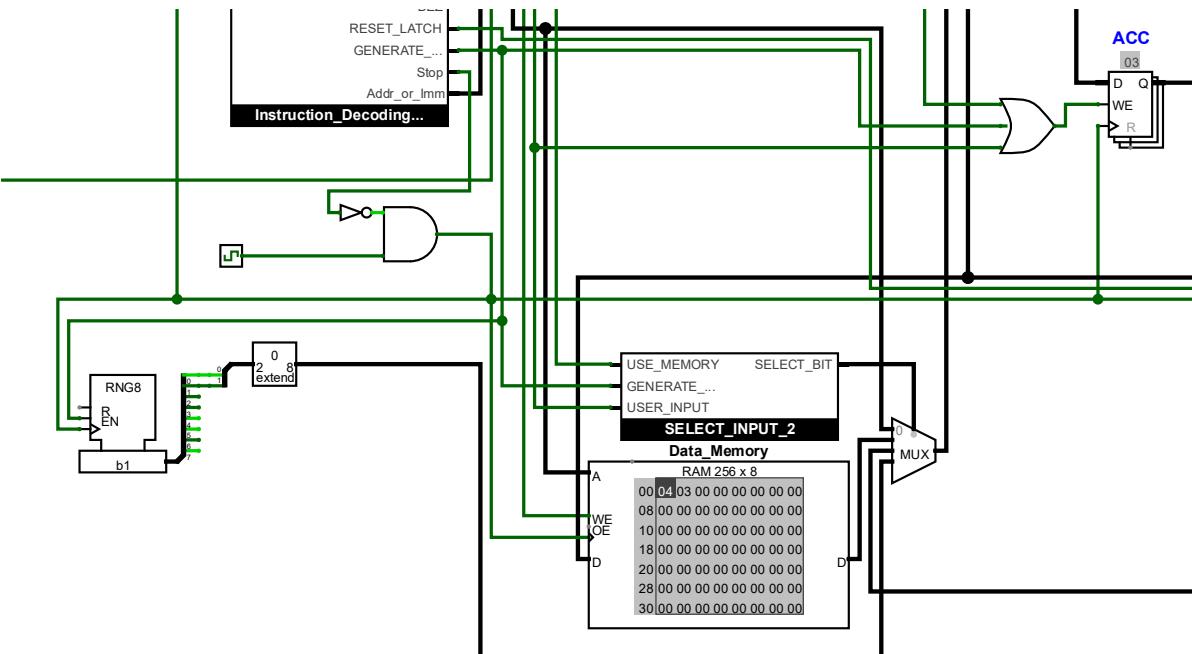
Random number generation

To generate the computer's turn, the built-in Random Generator is used. This component generates a number from 0-255; however, using a splitter to take the first two bits only generates a number from 0-3; to solve bit width issues this value is extended to 8-bits using a zero extension.

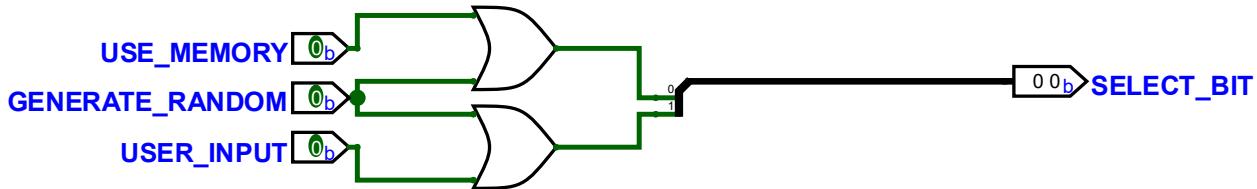
In the same way user input is sent to the ACC, MUX select bit 3 (11) sends the random value to the ACC.

CPU opcode 9 GENERATE_RANDOM: When this signal is active, on the rising edge of the clock a random number is generated and the select_input_2 subcircuit sets the MUX select it to 3(11). This sends the random number to input 2.

Random generation in circuit:



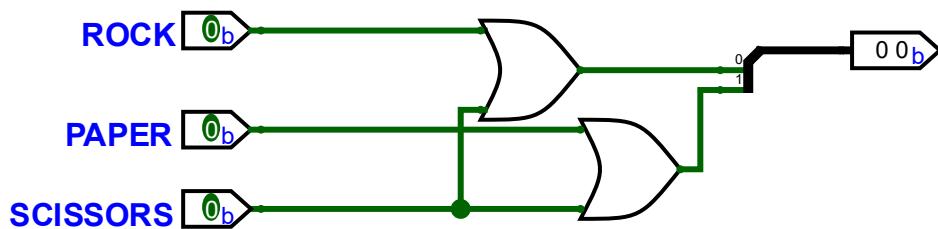
select_input_2 subcircuit with GENERATE_RANDOM added:



Standardising inputs

To perform the winner calculation, it was identified that inputs from the user and computer (random generation) would need to be the standardised. Consequently, the following was chosen: Rock is represented by decimal 1(01), Paper by 2 (10) and Scissors by 3 (11).

Standardising user input hardware implementation:



As the random generation can produce a value of 00, to standardise the computer's input, a loop is used. The loop rejects a value of 00 using the BEZ (CPU opcode 7) instruction; the random value is loaded to the ACC and checked. If it is 00, the loop branches to the beginning of the random generation. To address the bit width mismatch, the output of this subcircuit is extended to 8 bits using a zero extension.

Reject 00 software implementation:

```
9000 #generate random and send to ACC
7000 #generate another number if ACC = 0
3001 #store valid random number
```

Winner calculation

To calculate the winner of a given round this formula is used:

$$\text{Result} = (\text{Player} - \text{Computer} + 3) \text{ (mod3)}.$$

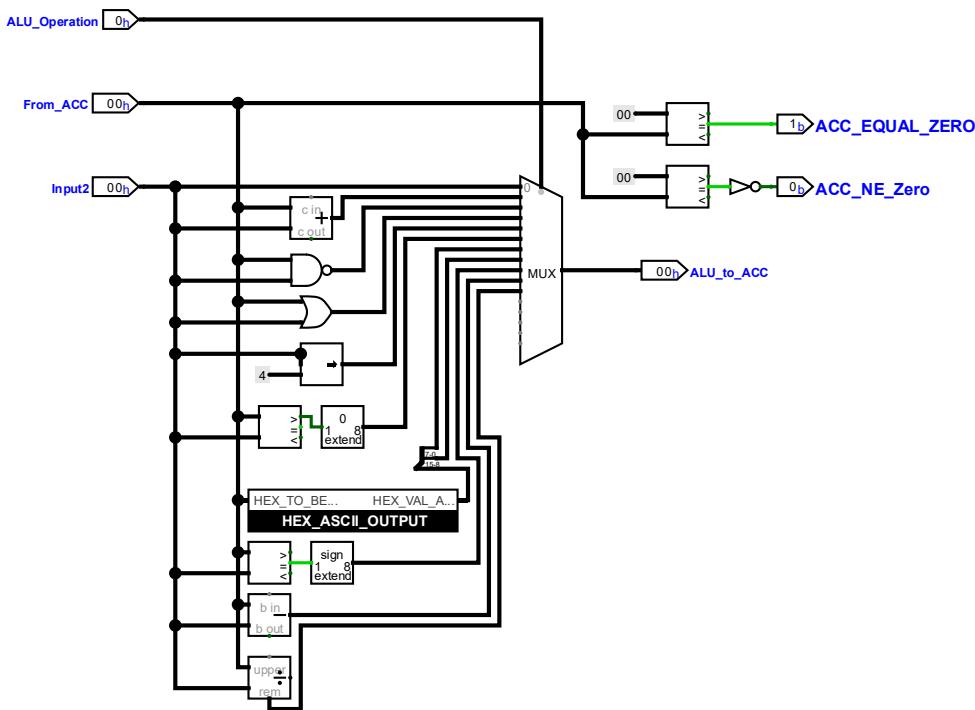
A result of 0 means a tie, 1 is a player win and 2 is a computer win. To utilise this formula the following functionality was added to the ALU:

ALU opcode 7 (equal comparison): Compares input 2 and the value in the ACC and sets the ACC to 1 if the two operands are equal. The output is extended using 8-bit zero extension. This opcode is used to check for a player win, computer win or draw.

ALU opcode 8 (subtraction): Subtracts input 2 and the value in the ACC and stores the result in the ACC. This opcode is used to perform the subtraction part of the formula

ALU opcode 9 (modulus): Divides the value in the ACC by the value in input 2 and stores the remainder from the division in the ACC. This opcode is used to find the final result of the calculation.

ALU with added opcodes:



Rewritten Pseudocode

With the hardware and software modifications in place, the pseudocode was rewritten in more detail.

Rewritten Pseudocode:

```
#MENU PAGE
STORE 0 in GAME COUNT, PLAYER SCORE, COMPUTER SCORE
PRINT TO TTY "ROCK PAPER SCISSORS"
PRINT TO TTY "PRESS ANY BUTTON TO START"
GET USER INPUT
```

```
IF ACC != 0
BRANCH TO SELECT INPUT

IF ACC == 0
CHECK FOR USER INPUT AGAIN

#SELECT INPUT

LOAD GAME COUNT TO ACC AND COMPARE WITH IMMEDIATE VALUE 3
IF ACC != 0
BRANCH TO OVERALL WINNER

LOAD PLAYER SCORE TO ACC AND COMPARE WITH IMMEDIATE VALUE 2
IF ACC != 0
BRANCH TO OVERALL WINNER

LOAD COMPUTER SCORE TO ACC AND COMPARE WITH IMMEDIATE VALUE 2
IF ACC != 0
BRANCH TO OVERALL WINNER

CLEAR TTY
PRINT TO TTY "PLEASE SELECT A CHOICE FROM"
PRINT TO TTY "B1 : Rock"
PRINT TO TTY "B2 : Paper"
PRINT TO TTY "B3 : Scissors"

GET USER INPUT
IF ACC != 0
STORE INPUT IN MEMORY
BRANCH TO RANDOM NUMBER GENERATION

IF ACC == 0
CHECK FOR USER INPUT AGAIN

#RANDOM GENERATION

GET RANDOM FOR RNG
SAVE RNG IN MEMORY

IF ACC == 0
BRANCH TO INPUT COMPARISON

#INPUT COMPARISON

SUBTRACT PLAYER INPUT FROM COMPUTER INPUT
STORE IN MEMORY
ADD 3
```

STORE IN MEMORY
MODULO WITH IMMEDIATE VALUE 3
STORE IN MEMORY

LOAD GAME OUTCOME TO ACC
IF ACC = 0
BRANCH TO DRAW

LOAD GAME OUTCOME TO ACC AND COMPARE WITH IMMEDIATE VALUE 1
IF ACC != ZERO
BRANCH TO PLAYER WINS

LOAD IMMEDIATE VALUE ZERO TO ACC
IF ACC = 0
BRANCH TO COMPUTER WINS

#PLAYER WINS

PRINT TO TTY "YOU WON"

ADD 1 TO GAME COUNT IN MEMORY
ADD 1 TO PLAYER SCORE

LOAD IMMEDIATE VALUE 0 TO ACC
IF ACC == 0
BRANCH TO BEST OF 3

#COMPUTER WINS

PRINT TO TTY "YOU LOST"

ADD 1 TO GAME COUNT IN MEMORY
ADD 1 TO COMPUTER SCORE

LOAD IMMEDIATE VALUE 0 TO ACC
IF ACC == 0
BRANCH TO BEST OF 3

#DRAW

PRINT TO TTY "IT'S A DRAW"

ADD 1 TO GAME COUNT IN MEMORY

LOAD IMMEDIATE VALUE 0 TO ACC
IF ACC == 0
BRANCH TO BEST OF 3

#BEST OF 3

COMPARE BEST OF 3 FLAG WITH IMMEDIATE VALUE 1

IF ACC != 0 BRANCH TO SELECT INPUT

PRINT TO TTY "PRESS B1 FOR BEST OF 3"

PRINT TO TTY "PRESS ANY OTHER BUTTON TO RETURN TO MENU"

SAVE PLAYER INPUT TO MEMORY

COMPARE PLAYER INPUT WITH IMMEDIATE VALUE 1

IF ACC != ZERO

SET BEST OF 3 FLAG IN MEMORY TO 1

BRANCH TO SELECT INPUT

IF ACC == ZERO

BRANCH TO MENU

#OVERALL WINNER

LOAD PLAYER SCORE TO ACC AND COMPARE IF EQUAL TO COMPUTER

IF ACC != 0

BRANCH TO OVERALL DRAW

LOAD PLAYER SCORE TO ACC AND COMPARE IF ACC>COMPUTER SCORE

IF ACC != 0

BRANCH TO PLAYER OVERALL WINS

LOAD IMMEDIATE 0 TO ACC

BRANCH TO COMPUTER OVERALL WINS

#OVERALL DRAW

PRINT TO TTY "IT'S A DRAW"

PRINT TO TTY "PRESS ANY BUTTON TO RETURN TO MENU"

CHECK FOR USER INPUT

IF ACC != 0

BRANCH TO MENU

#PLAYER OVERALL WINS

PRINT TO TTY "YOU WON OVERALL"

PRINT TO TTY "PRESS ANY BUTTON TO RETURN TO MENU"

CHECK FOR USER INPUT

IF ACC != 0

BRANCH TO MENU

#COMPUTER OVERALL WINS

```

PRINT TO TTY "YOU LOST OVERALL"
PRINT TO TTY "PRESS ANY BUTTON TO RETURN TO MENU"

```

```

CHECK FOR USER INPUT
IF ACC != 0
BRANCH TO MENU

```

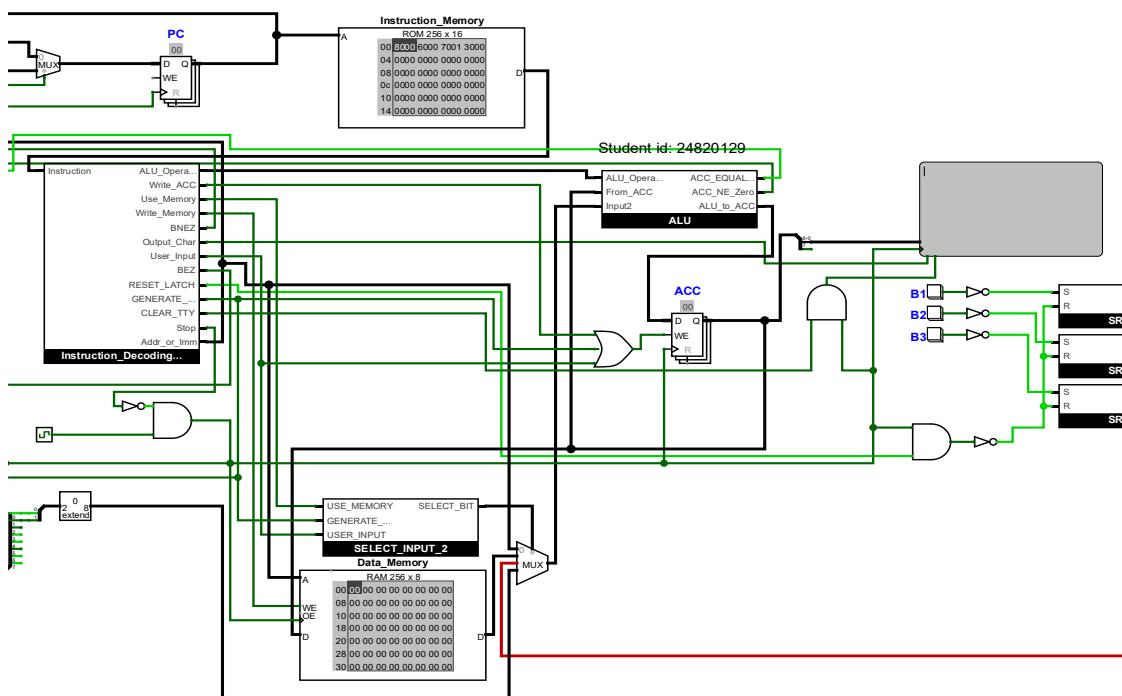
1.8.3 Pointers, Jump, Link and Return

Clearing the TTY

During development of the pseudocode, it was identified that there was no functionality to clear the TTY; this was addressed with the addition of:

CPU opcode A CLEAR_TTY: When this signal is set to active, the TTY is reset on the rising edge of the clock.

Computer with CPU opcode A added:



Pointers

Hardware implementation

Upon starting development, since there was no way to print strings from the data memory, large amounts of the instruction memory were being used to print immediate ASCII values to the TTY; this resulted in the program being too large for the instruction memory. To address this, the computer was changed to add functionality for storing strings in RAM and accessing them using pointers, like in RISC-V RARS.

Further research identified how to address the RAM and terminate string outputs, finding the standard method “stops printing when it reaches a null character” (Pesto, 2010). To allow for this, a pointer opcode was added to the decoder alongside a MUX to decide the data memory’s address input. When the select bit of this MUX is 0, the address or immediate value is used; however, when the pointer signal is active the select bit is set to 1 and the value stored in the ACC is used.

CPU opcode B POINTER: When this signal is active, the MUX that decides the data memory's address pin input is set to 1 using the value in the ACC as the address to be read.

Software implementation

To print the strings a loop was needed with the following:

1. Get character from address stored in pointer variable.
2. Check if the character is null and stop loop if null.
3. Print character to TTY and add 1 to pointer variable.
4. Loop to step 1.

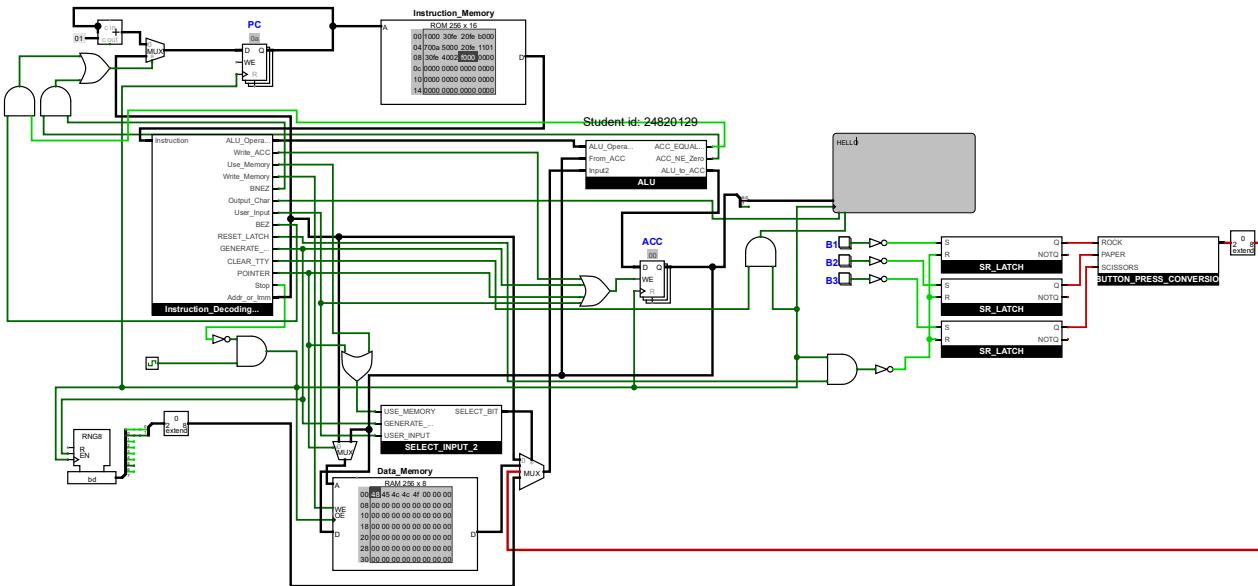
Assembly to test pointer system:

```
v2.0 raw
#1. load 0 into ACC and store at pointer variable FE
1000
30FE
#2. Load pointer variable and get the character from address in the ACC
20FE
B000
#3. Check if null, if null branch to stop
700A
#4. Print character to TTY and add immediate value 1 to pointer variable
5000
20FE
1101
#5. Save the new value of the variable to RAM and loop back to start of loop to print next
character
30FE
4002
F000
```

This program was tested with “Hello” loaded into the data memory starting at address 00.

Test outcome:

The test was successful as seen below.



Jump, Link and Return

Initial Ticket System

A ticket system was used initially to jump to the desired section of the program, due to the absence of jump and link or return functionality. This meant the ACC had to access a return value stored in RAM prior to jumping. This method resulted in more wasted instruction memory and errors with jump or branch addressing.

Assembly for ticket system:

```
#load pointer and jump to print loop
1009
3008
#store ticket and RAM for jump back
1001
3007
1000
70AC

#load ticket to acc and check value for each part of
program
2007
1801
#menu
400C
2007
1802
#select input
4012
2007
1803
#print b1
402A
2007
1804
#print b2
4030
2007
1805
#print b3
```

```

4036
2007
1806
#get input
403C
2007
1807
#update scores
4063
2007
1808
#print bo3 part 2
407B
2007
1809
#bo3 input
4081
2007
180A
#end wait
40D5

```

Link Register

Research into RISCV's implementation of jump and link found that a link register is used to store the current position in the program, which can then be called upon to return to that position ('Link Register' 2025). To implement this a dedicated link register was added to the simulation.

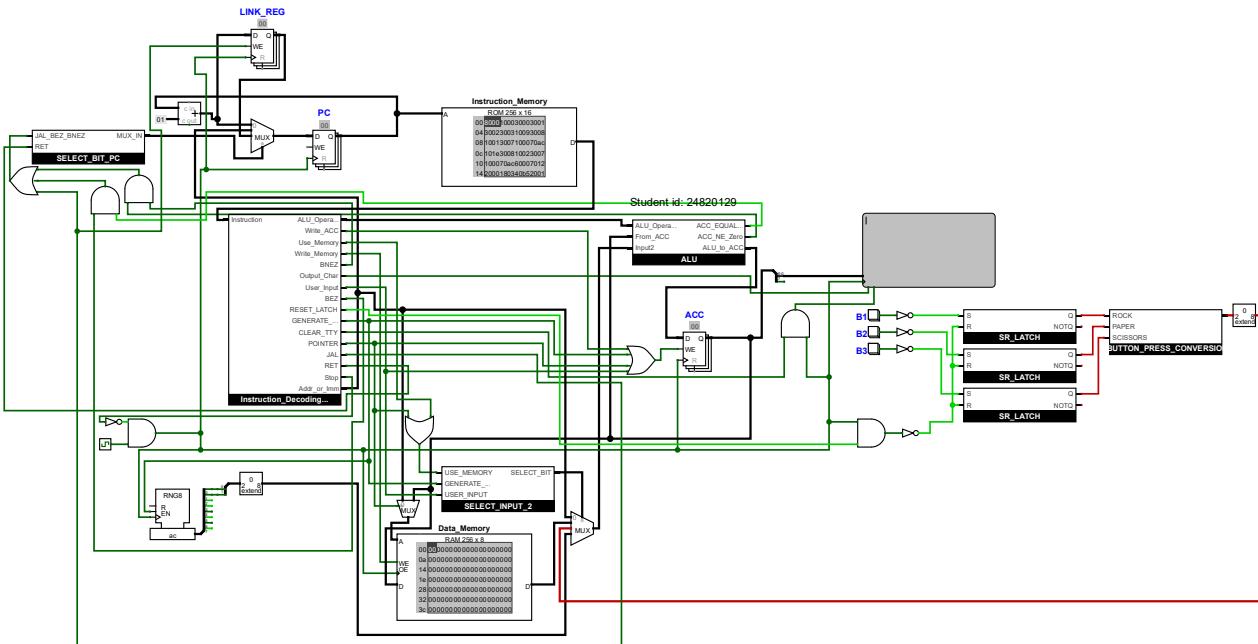
The current MUX which decides the PC's input was expanded upon, now select bit 2 (10) is the input from the link register. When the JAL signal is active (opcode C) the PC is set to the immediate value in the address, causing the program to jump to the address. The current address in the program is also saved to the link register.

SELECT_BIT_PC sub-circuit for PC's MUX:



When the RET opcode is active (opcode D), the PC's MUX is set to 2 (10) sending the data inside the link register to the PC and returning to the position jumped from.

JAL and RET functionality added in the simulation:



Testing Jump Link and Return

To test Jump, Link and Return the following assembly was used:

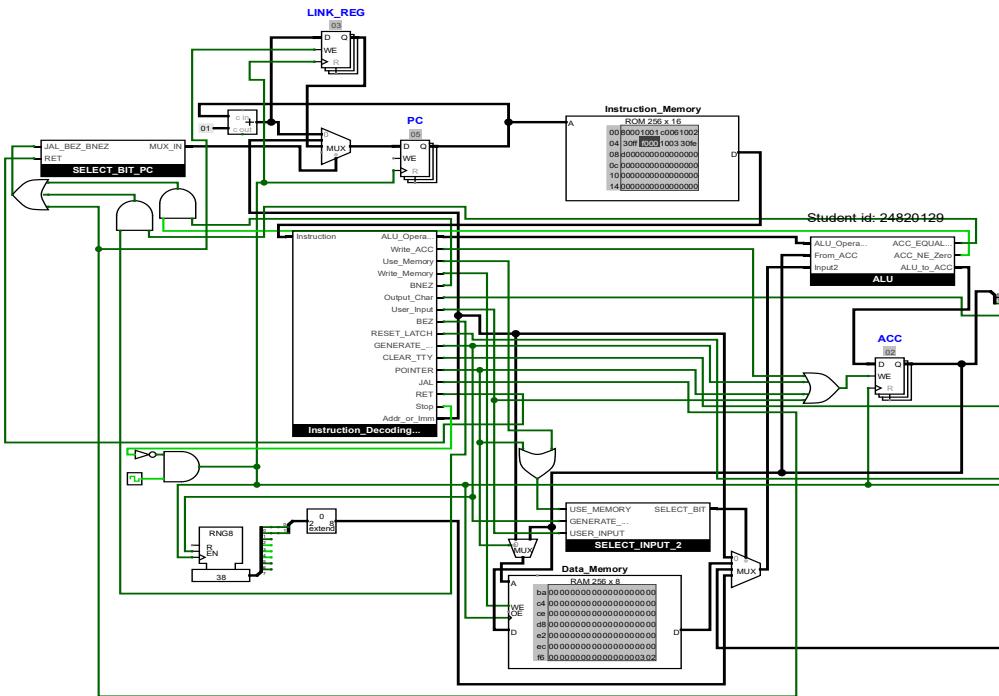
```
#initialise buttons
8000
#jump to instruction 06
C006
#load 2 to acc and store (if this happens it worked)
1002
30FF
F000

#jump here
#load 3 to acc and store (if this happens it worked)
1003
30FE
#return
D000
```

If Jump, Link and Return have been added successfully, 2 will be stored at FF and 3 will be stored at FE.

Test outcome:

Value 02 was stored at address FF and 03 was stored at address FE indicating a successful test.



1.8.4 Final Code

With the functionality of Jump, Link and Returns implementation of subroutines was now possible so the final pseudocode was written:

```
#ROCK PAPER SCISSORS GAME

#initialise buttons and score variables in data memory

#Menu
#print "rock paper scissors"
#print "press any button to start"

#get input using JAL
-----
#GameLoop

#do checks for game over

#load game count
#if game count = 3 branch to calculate overall (if ACC NE ZERO)

#check player score
#if player score = 2 branch to calculate overall

#check computer score
#if computer score = 2 branch to calculate overall

#start round

#clear tty
#print select choice
```

```

#print b1:rock
#print b2:paper
#print b3:scissors
#get input and store at player input

#get cpu input and store at cpu input

#calculate winner
#(player - computer + 3)%3

#subtract player input from cpu input
#store in outcome
#add 3
#store in outcome
#mod 3 with outcome
#store in outcome

#round result

#clear tty

#player won check
#load outcome
#load immediate 1 and compare = with outcome
#if acc NE zero jump to player won

#computer won check
#load outcome
#load immediate 2 and compare = with outcome
#if acc NE zero jump to computer won

#draw (program will continue if all other checks fail)
-----
#round result output

#draw

#prints ITS A DRAW to tty
#increments game score by 1
#jumps to bo3 logic

#player win

#prints YOU WON to tty
#increments player score by 1
#increments game score by 1
#jumps to bo3 logic

#computer win

#prints YOU LOST to tty
#increments computer score by 1
#increments game score by 1
#jumps to bo3 logic

-----
#bo3 logic

#load bo3 flag
#if 1 jump to GameLoop

#this will happen if not in bo3

```

```

#clear tty
#print "PRESS B1 FOR BEST OF 3"
#print "PRESS BUTTON FOR MENU"
#get user input

#if user input = 1
#BO3 flag = 1
#JUMP TO GAMELOOP

>this will happen if other button is pressed
#jump to init

-----
#calculate overall
#load player score to ACC and check if = to computer
#if ACC NE ZERO jump to overall draw
#load player score to acc and compare if greater than computer score
#if acc NE zero jump to player win
#jump to computer win

#overall player win
#clear tty
#print to TTY "PLAYER WINS OVERALL"
#print "PRESS ANY KEY TO CONTINUE"
#jump to user input
#jump to init scores and variables

#overall computer win
#clear tty
#print to TTY "YOU LOST OVERALL"
#print "PRESS ANY KEY TO CONTINUE"
#jump to user input
#jump to init scores and variables

#overall draw
#clear tty
#print to TTY "ITS A DRAW"
#print to TTY "PRESS ANY KEY TO CONTINUE"
#jump to user input
#jump to init scores and variables

-----
#functions

#print function
>this contains the print loop

#get input function
#reset latches
>this contains the input loop

```

To begin development of the final assembly, the data memory was written first so that string addresses would be known. The text strings were converted into their using an online tool, 00 is used as a null terminator to signify the end of the printing loop.

For efficiency and accuracy, Google Gemini 3.5 was used to find the starting memory address for each string. (A1 APPENDIX NO 3).

Data memory:

```
v2.0 raw
# 00-07 variables
# 00: GAME COUNT
# 01: PLAYER SCORE
# 02: COMPUTER SCORE
# 03: BO3 FLAG
# 04: PLAYER INPUT
# 05: COMPUTER INPUT
# 06: OUTCOME
# 07: POINTER
00 00 00 00 00 00 00 00 00

#START: 08 "ROCK PAPER SCISSORS\n"
52 4F 43 4B 20 50 41 50 45 52 20 53 43 49 53 53 4F 52 53 0A 00

#START: 1D "PRESS BUTTON TO START\n"
50 52 45 53 53 20 42 55 54 54 4F 4E 20 54 4F 20 53 54 41 52 54 0A 00

#START: 34 "SELECT A CHOICE\n"
53 45 4C 45 43 54 20 41 20 43 48 4F 49 43 45 0A 00

#START: 45 "B1: Rock\n"
42 31 3A 20 52 6F 63 6B 0A 00

#START: 4F "B2: Paper\n"
42 32 3A 20 50 61 70 65 72 0A 00

#START: 5A "B3: Scissors\n"
42 33 3A 20 53 63 69 73 73 6F 72 73 0A 00

#START: 68 "YOU WON\n"
59 4F 55 20 57 4F 4E 0A 00

#START: 71 "YOU LOST\n"
59 4F 55 20 4C 4F 53 54 0A 00

#START: 7B "IT'S A DRAW\n"
49 54 27 53 20 41 20 44 52 41 57 0A 00

#START: 88 "PRESS B1 FOR BEST OF 3\n"
50 52 45 53 53 20 42 31 20 46 4F 52 20 42 45 53 54 20 4F 46 20 33 0A 00

#START: A0 "PRESS ANY BUTTON FOR MENU\n"
50 52 45 53 53 20 41 4E 59 20 42 55 54 54 4F 4E 20 46 4F 52 20 4D 45 4E 55 0A 00

#START: BB "YOU WON OVERALL\n"
59 4F 55 20 57 4F 4E 20 4F 56 45 52 41 4C 4C 0A 00

#START: CC "YOU LOST OVERALL\n"
59 4F 55 20 4C 4F 53 54 20 4F 56 45 52 41 4C 4C 0A 00

#START: DE "PRESS ANY KEY TO CONTINUE\n"
50 52 45 53 53 20 41 4E 59 20 4B 45 59 20 54 4F 20 43 4F 4E 54 49 4E 55 45 0A 00
```

With the pseudocode and data memory established, development of the final assembly began. Until the end of development, memory addresses for branch and jump instructions could not be determined, therefore; XX placeholders were used for addresses, and square brackets for description of the jumps destination.

After development of the assembly, the final memory addresses were calculated and added manually to ensure accuracy compared to AI tools.

Final Assembly:

```
v2.0 raw
#-----
#ROCK PAPER SCISSORS GAME
#-----
#-----
#initialise buttons and variables in data memory
#-----
#init buttons
8000
#load immediate 00 to ACC
1000
#store 00 at all variable positions
3000
3001
3002
3003
3004
3005
3006
3007
#-----
#Menu
#print "rock paper scissors"
#load address to pointer variable
1008 3007
#jump to print function
C08E#[address of start of print function]

#print "press button to start"
#load address to pointer variable
101D 3007
#jump to print function
C08E#[address of start of print function]

#get input using JAL
#jump to get input function
C098#[address of get input function]
#-----
#GameLoop
#-----
```

```

#do checks for game over

#load game count
2000
#if game count = 3 branch to calculate overall (if ACC NE ZERO)
1803 4068#[branch to calculate overall]

#check player score
2001
#if player score = 2 branch to calculate overall
1802 4068#[branch to calculate overall]

#check computer score
2002
#if computer score = 2 branch to calculate overall
1802 4068#[branch to calculate overall]

#-----
#start round
#-----

#clear tty
A000

#print select choice
#load pointer to acc and store in pointer variable
1034 3007
#JAL
C08E#[jump to print function]

#print b1:rock
#load pointer to acc and store in pointer variable
1045 3007
#JAL
C08E#[jump to print function]

#print b2:paper
#load pointer to acc and store in pointer variable
104F 3007
#JAL
C08E#[jump to print function]

#print b3:scissors
#load pointer to acc and store in pointer variable
105A 3007
#JAL
C08E#[jump to print function]

#-----
#get input and store at player input
C098#[jump to get input]

#get cpu input check if valid and store at cpu input
9000
7028#[jump to start of the cpu input loop (input validation)]
3005

```

```

#-----
#calculate winner
#-----
#(player - computer + 3)%3

#subtract player input from cpu input
2004 2905
#store in outcome
3006
#add 3
2006 1103
#store in outcome
3006
#mod 3 with outcome
2006 1A03
#store in outcome
3006

#-----
#round result

#clear tty
A000

#player won check
#load outcome
2006
#load immediate 1 and compare = with outcome
1801
#if acc NE zero jump to player won
4042#[branch to player won]

#computer_won check
#load outcome
2006
#load immediate 2 and compare = with outcome
1802
#if acc NE zero jump to computer won
404C#[branch to computer won]

#draw (program will continue if all other checks fail)
#-----

#round result output
#-----

#draw

#prints ITS A DRAW to tty
107B 3007
C08E #[jump to print function]
#increments game count by 1
2000 1101
3000

```

```

#jumps to bo3 logic
C056#[jump to bo3 load]

#player win

#prints YOU WON to tty
1068 3007
C08E #[jump to print function]
#increments player score by 1
2001 1101
3001
#increments game count by 1
2000 1101
3000
#jumps to bo3 logic
C056#[jump to bo3 logic]

#computer win

#prints YOU LOST to tty
1071 3007
C08E #[jump to print function]
#increment computer score by 1
2002 1101
3002
#increments game count by 1
2000 1101
3000
#jumps to bo3 logic
C056#[jump to bo3 logic]

#-----
#bo3 logic
#-----

#load bo3 flag
2003
#if 1 jump to GameLoop
1801 4011#[jump to gameloop]
#this will happen if not in bo3
#clear tty
A000
#print "PRESS B1 FOR BEST OF 3"
1088 3007
C08E #[jump to print function]
#print "PRESS ANY BUTTON FOR MENU"
10A0 3007
C08E #[jump to print function]
#get user input
C098#[jump to get user input]

#if user input != 1 jump to overall winner
2004 1801
7068 #[jump to overall winner]

#else this will happen

#B03 flag = 1
2003 1101
3003

```

```

#JUMP TO GAMELOOP
C011#[jump to game loop]

#-----
#calculate overall
#-----

#load player score to ACC and check if = to compute
2001 2802
#if ACC NE ZERO jump to overall draw
4084#[jump to overall draw]
#load player score to acc and compare if greater than computer score
2001 2502
#if acc NE zero jump to player win
4070#[jump to player win]
#jump to computer win
1001 407A#[jump to computer win]

#overall player win
#clear tty
A000
#print to TTY "PLAYER WINS OVERALL"
10BB 3007
C08E#[jump to print]
#print "PRESS ANY KEY TO CONTINUE"
10DE 3007
C08E#[jump to print]
#jump to user input (will return when button is pressed and stored)
C098#[jump to get user input]
#jump to init scores and variables
1001 4000#[jump to init]

#overall computer win
#clear tty
A000
#print to TTY "YOU LOST OVERALL"
10CC 3007
C08E#[jump to print]
#print "PRESS ANY KEY TO CONTINUE"
10DE 3007
C08E#[jump to print]
#jump to user input (will return when button is pressed and stored)
C08E#[jump to get user input]
#jump to init scores and variables
1001 4000#[jump to init]

#overall draw
#clear tty
A000
#print to TTY "ITS A DRAW"
107B 3007
C08E#[jump to print]
#print to TTY "PRESS ANY KEY TO CONTINUE"
10DE 3007
C08E#[jump to print]
#jump to user input(will return when button is pressed and stored)
C098#[jump to get user input]
#jump to init scores and variables
1001 4000#[jump to init]

```

```
#-----
#functions
#-----
#print function

#load pointer to acc
2007
#pointer from acc becomes RAM address input and gets output to the ACC
B000
#if ACC is 00 then return
7097#[address of return]
#output char
5000
#add immediate 1 to the pointer
2007 1101
#store pointer variable at position 07
3007
#loop to start of print loop
1001 408E#[start of print loop]
#RET
D000

#get input function

#get input loop
8000
6000
7099#[address of start of input loop]
3004
#return after input
D000
```

1.8.5 Testing

The game has two states: Best of 3 and Best of 1, proper testing of the program requires verification of win, draw and loss outcomes in both states. The Best of 3 state also requires testing for an early win (2-0) for the player and computer.

To test all functionalities testing of variable storage in memory was also necessary. This will be carried out first and then followed by testing of the game states.

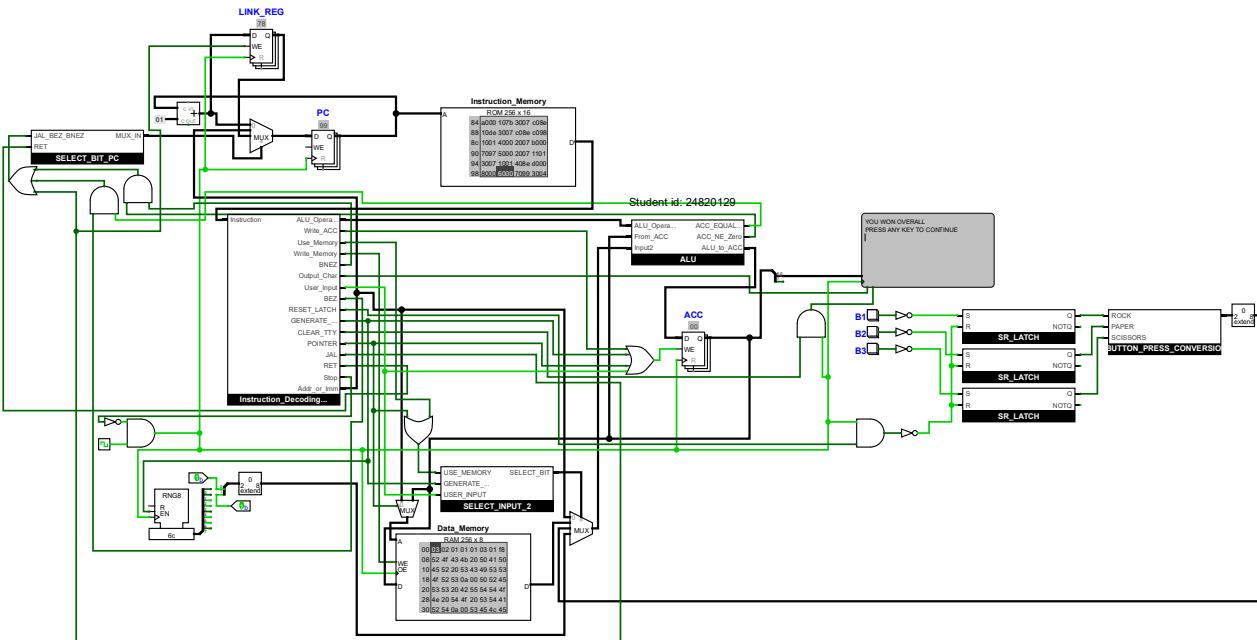
To make the game outcome controllable, the computer input was set to 3 (11) using two input pins.

Testing variable storage in memory:

Check	Expected storage in memory	Actual storage in memory	Pass/Fail
Game Count Variable	03 (3 rounds played) expected to be stored in position 00	03 Stored in position 00	Pass
Player Score	02 (won 2 rounds) expected to be in position 01	02 Stored in position 01	Pass
Computer Score	01 (won 1 round) expected to be in position 02	01 Stored in position 02	Pass
BO3 Flag	01 (flag was set) expected to be in position 03	01 Stored in position 03	Pass
Player Input	01 (rock last input) expected to be in position 04	01 Stored in position 04	Pass
Computer Input	03 (scissors last input) expected to be in position 05	03 Stored in position 05	Pass
Outcome	01 (player won last round) expected to be in position 06	01 Stored in position 06	Pass
Pointer	F8 (address of last string) expected to be in position 07	F8 Stored in position 07	Pass

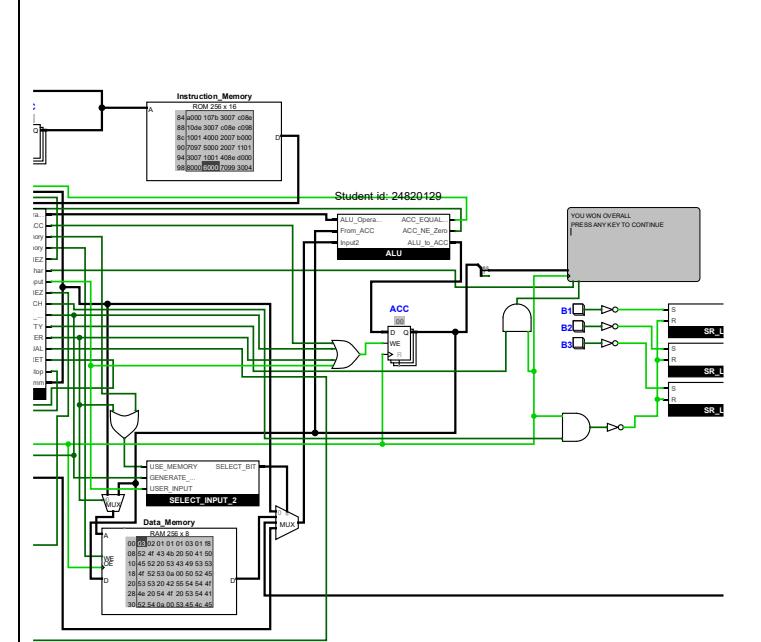
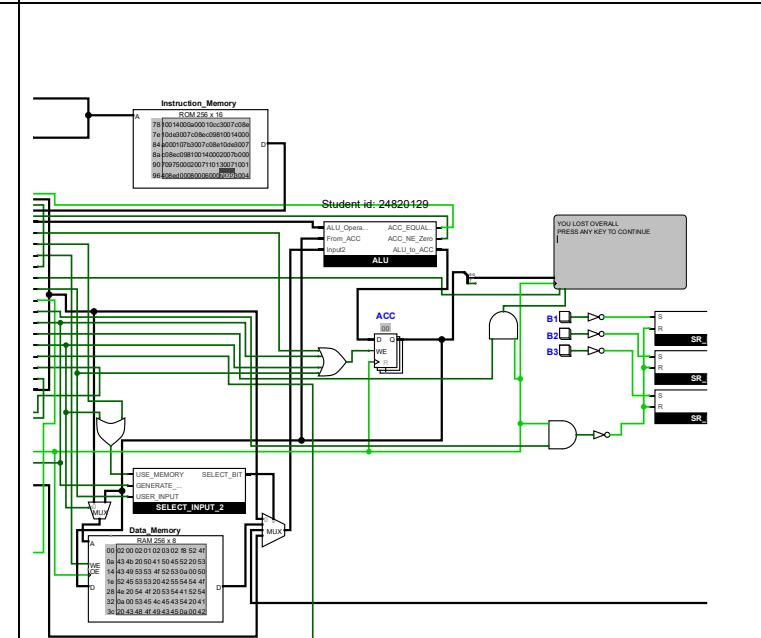
All variable values are correct, and stored in the correct position indicating expected functionality.

Screenshot of variable positions at end of game in memory:

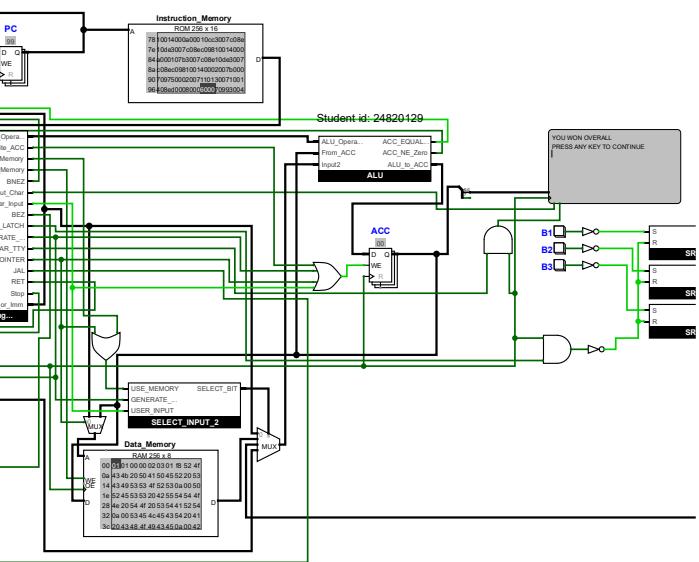
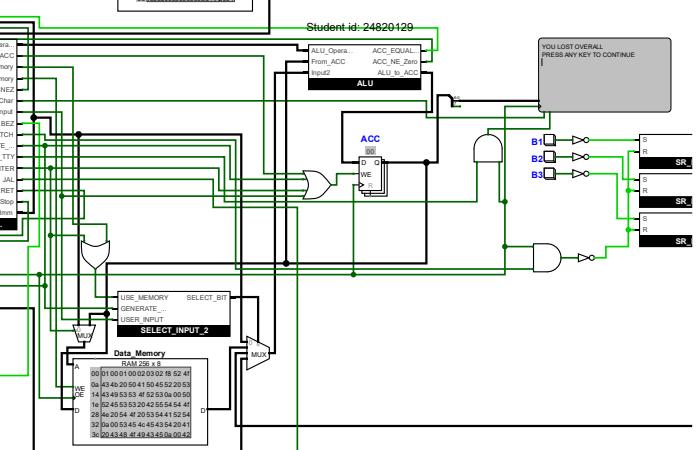


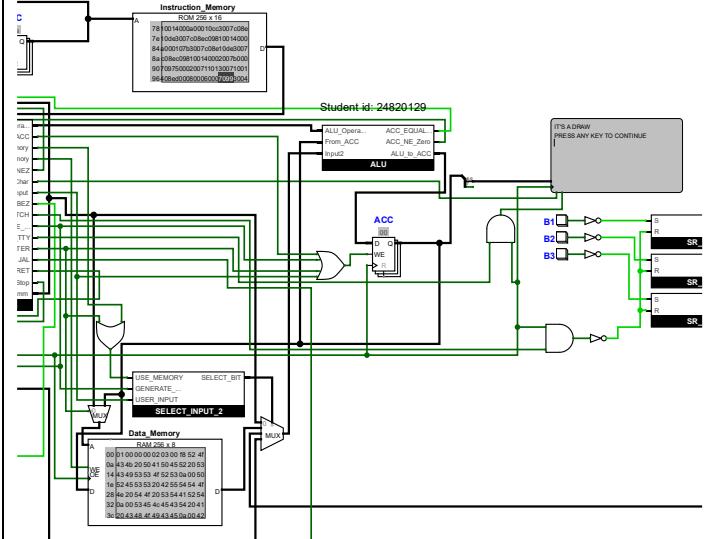
Testing game outputs

Check	Game state	Scores and round count	Expected final screen(screenshot)	Actual final screen(screenshot)
Player wins 2-0	BO3	Game Count (position 00): 02 Player Score (position 01): 02 Computer Score (position 02): 00 BO3 Flag (position 03): 01	YOU WON OVERALL PRESS ANY KEY TO CONTINUE	

Player wins 2-1	BO3	Game Count (position 00): 03 Player Score (position 01): 02 Computer Score (position 02): 01 BO3 Flag (position 03): 01	YOU WON OVERALL PRESS ANY KEY TO CONTINUE	
Computer wins 2-0	BO3	Game Count (position 00): 02 Player Score (position 01): 00 Computer Score (position 02): 02 BO3 Flag (position 03): 01	YOU LOST OVERALL PRESS ANY KEY TO CONTINUE	

Computer wins 2-1	BO3	Game Count (position 00): 03 Player Score (position 01): 01 Computer Score (position 02): 02 BO3 Flag (position 03): 01	YOU LOST OVERALL PRESS ANY KEY TO CONTINUE	
Draw	BO3	Game Count (position 00): 03 Player Score (position 01): 01 Computer Score (position 02): 01 BO3 Flag (position 03): 01	IT'S A DRAW PRESS ANY KEY TO CONTINUE	

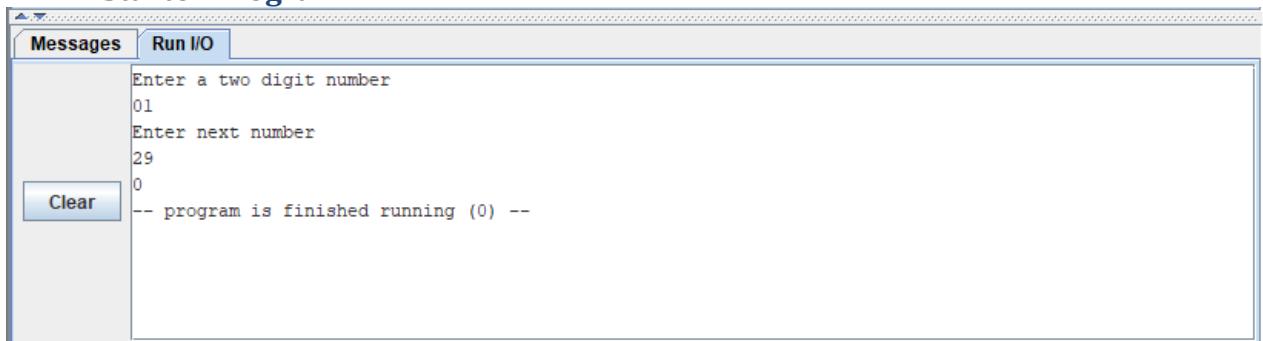
Player wins 1-0	BO1	Game Count (position 00): 01 Player Score (position 01): 01 Computer Score (position 02): 00 BO3 Flag (position 03): 00	YOU WON OVERALL PRESS ANY KEY TO CONTINUE	
Computer wins 1-0	BO1	Game Count (position 00): 01 Player Score (position 01): 00 Computer Score (position 02): 01 BO3 Flag (position 03): 00	YOU LOST OVERALL PRESS ANY KEY TO CONTINUE	

Draw	BO1	Game Count (position 00): 01 Player Score (position 01): 00 Computer Score (position 02): 00 BO3 Flag (position 03): 00	IT'S A DRAW PRESS ANY KEY TO CONTINUE	
------	-----	--	--	--

2. RISC-V Assembly Language Programming

2.1 Test the existing program with minor adjustments

2.1.1 Starter Program



```

Messages Run I/O
Enter a two digit number
01
Enter next number
29
0
-- program is finished running (0) --

```

2.1.2 Expected Result

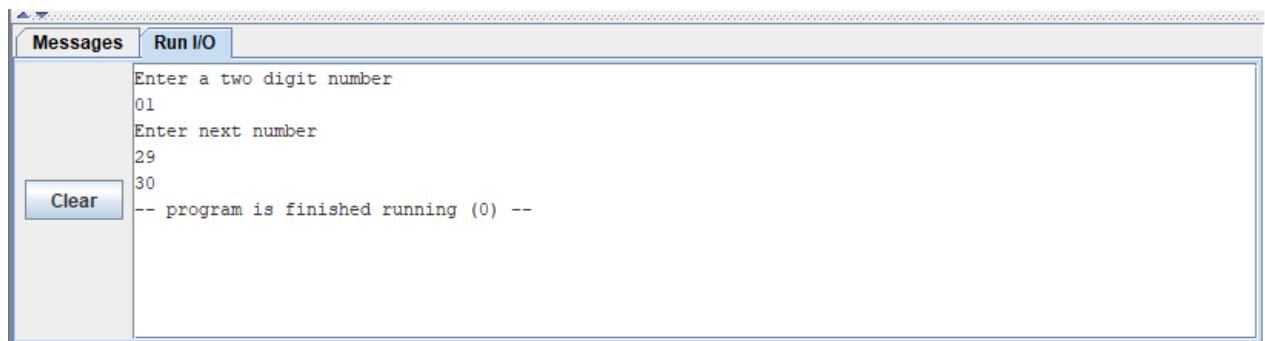
The expected output of the program is decimal 30.

2.1.3 Program code

```
## add the two values together and store in s2  
add s2,s0,sl
```

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

2.1.4 Result



The program outputs the correct total of decimal 30 indicating a successful test.

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 totalMsg: .string "The total is: "  
7
```

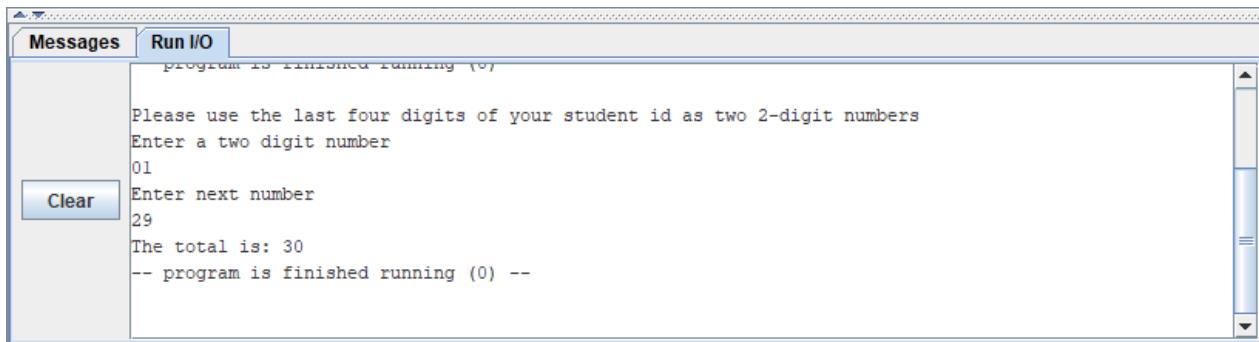
The new message outputs “The total is:” to the console

```
## add the two values together and store in s2  
add s2,s0,sl  
  
#output total message  
addi a7, zero, 4  
la a0, totalMsg  
ecall
```

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

The code under the comment “output total message” outputs the totalMsg string from the .data memory followed by the total of the two values.

2.2.2 Results



2.2.3 Explanation

Clear messages prevent user error and help turn a confusing UI into something that is easier to use for the average person.

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"
totalMsg: .string "The total is: "
outputOrMsg: .string "\nThe result of bitwise OR is: "
outputNandMsg: .string "\nThe bitwise of bitwise NAND is: "
```

.text

```
#outputmessage 1 to the console
addi a7, zero, 4
la a0, enterMsg1
ecall
```

```
# 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
#bitwise OR on s0 and s1, stored in s3
or s3,s0,s1
#bitwise NAND on s0 and s1, stored in s4
and s4,s0,s1
not s4,s4

#output total message
addi a7, zero, 4
la a0, totalMsg
ecall

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

#output the OR message and result
addi a7, zero, 4
la a0, outputOrMsg
ecall

add a0, s3, zero
addi a7, zero, 1
ecall

#putput the NAND message an result
addi a7, zero, 4
la a0, outputNandMsg
ecall

add a0, s4, zero
addi a7, zero, 1
ecall

addi a7, zero, 10
ecall

```

2.3.2 Expected Result

Value in s0	Value in s1	Bitwise Operation	Expected Result in decimal signed 2's compliment	Expected output to console
01	29	OR	29	The result of bitwise OR is: 29
01	29	NAND	-2	The bitwise of bitwise NAND is: -2

2.3.3 Result

```

Messages Run I/O
program is finished running (0)

Please use the last four digits of your student id as two 2-digit numbers
Enter a two digit number
01
Enter next number
29
The total is: 30
The result of bitwise OR is: 29
The bitwise of bitwise NAND is: -2
-- program is finished running (0) --

```

The result of a bitwise OR on 01 and 29 (29) was correctly output to the console alongside the message; similarly, the result of a bitwise NAND on 01 and 29 (-2) was also output correctly.

2.3.4 Explanation

Bitwise OR:

For the bitwise OR the built in RISC-V “or” instruction was applied to registers s0 and s1, the result was stored in s3.

Bitwise NAND:

For the bitwise NAND, RISV-V does not contain a built-in instruction. Using Boolean algebra, a bitwise NAND can be applied by combining a bitwise AND with a bitwise NOT, both of which are built into RISC-V.

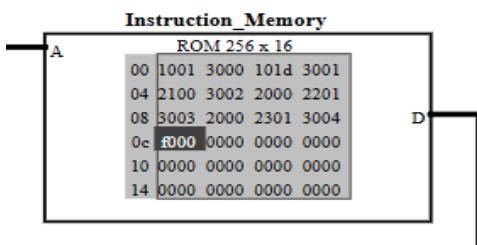
First the AND instruction was applied to s0 and s1, being stored in s4. Next, the NOT instruction was applied to s4, being stored in s4.

2.3.5 Comparison

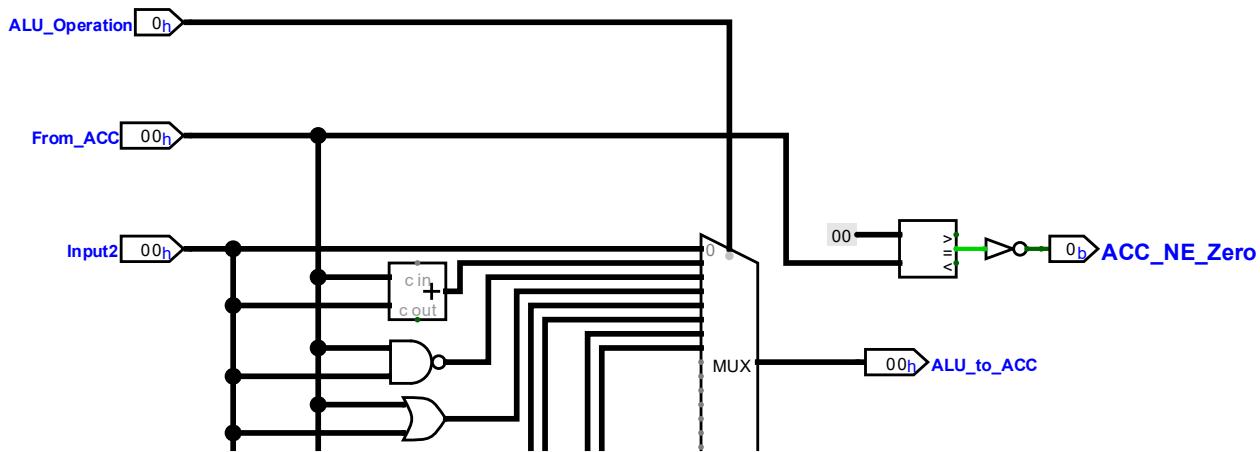
Bitwise operations in RISCV RARS:

0x00400048	0x00940933	add x18,x8,x9	40: add s2,s0,s1
0x0040004c	0x009469b3	or x19,x8,x9	42: or s3,s0,s1
0x00400050	0x00947a33	and x20,x8,x9	44: and s4,s0,s1
0x00400054	0xffffa4a13	xori x20,x20,0xffffffff	45: not s4,s4

Bitwise operation assembly code in Logisim:



Bitwise instruction inside ALU in Logisim:



Comparison of logic operations:

In RISCV RARS, the hex code instruction, e.g. 0x00947a33 for bitwise AND, includes multiple pieces of information: the operation, source registers, and destination in one 32-bit instruction.

Whereas in Logisim, only one bit controls the operation. For example, 2201 is used for a bitwise NAND and a bitwise OR is 2301. The second bit in the instruction is wired to the ALUs MUX; this is the only information that needs to change to differentiate between the two operations.

Comparison of machine code development:

In RISC-V, the assembly language is automatically converted by the assembler into hex instructions. In contrast, in Logisim, the hex instructions were written manually by analysing the MUX, ALU, and decoder to develop the machine code.

Checking for correct results:

In RISC-V, verification was text-based; the result was checked by comparing the system output to the expected result. Whereas in Logisim, the result was verified by looking at the value stored in the data memory and checking it against the expected result.

2.4 Adding a subroutine

2.4.1 Subroutine 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"
totalMsg: .string "The total is: "
outputOrMsg: .string "\nThe result of bitwise OR is: "
outputNandMsg: .string "\nThe result of bitwise NAND is: "
```

.text

main:

```
#output message 1 to the console
li a7, 4
```

```

la a0, enterMsg1
ecall

# output message 2 to the console
li a7, 4
la a0, enterMsg2
ecall

# read an integer from keyboard input and store in s0
li a7, 5
ecall
mv s0, a0

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

li a7, 4
la a0, enterMsg3
ecall

# then receive the input and store in s1
li a7, 5
ecall
mv s1,a0

#perform operations

## add the two values together and store in s2
add s2,s0,s1
#bitwise OR on s0 and s1, stored in s3
or s3,s0,s1
#bitwise NAND on s0 and s1, stored in s4
and s4,s0,s1
not s4,s4

#output total message
la a0, totalMsg

# output the value from s2
mv a1, s2
#store return address at ra and jump to print functtion
jal ra, printMsgAndValue

#output the OR message and result
la a0, outputOrMsg
#output value from s3
mv a1, s3
#store return address at ra and jump to print functtion
jal ra, printMsgAndValue

#output the NAND message an result
la a0, outputNandMsg
#output value from s4

```

```

mv a1,s4
#store return address at ra and jump to print function
jal ra, printMsgAndValue

#end program
li a7, 10
ecall

#subroutine for outputting a message and value

printMsgAndValue:

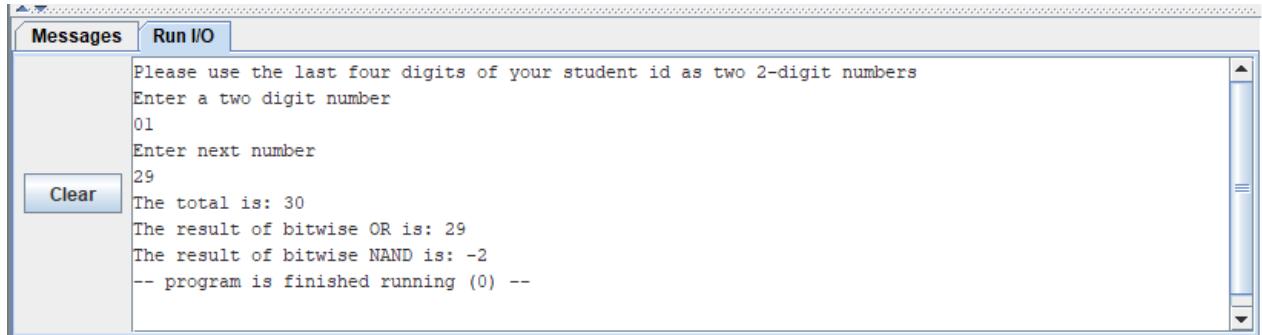
#output message stored in a0
li a7,4
ecall

#output the value stored in a1
mv a0,a1
li a7,1
ecall

ret

```

2.4.2 Result



2.4.3 Explanation

Subroutines serve many purposes in programming, helping keep code: reusable, readable and maintainable.

Reusability: Without subroutines, you would have to rewrite the same printing code every time you wanted to display a message. This is repetitive and inefficient. By using a subroutine, you write the code once and can use it as many times as you need.

Readability: Subroutines keep the main part of your program clean. Instead of clogging the main loop with low level details like register moves and system calls, you move that logic into a separate block. This allows a developer to look at the main program and easily understand the overall flow.

Maintainability: Subroutines make updating your code much faster. For example, if you wanted to add a new line after every message, you would only need to change the code in one place. Without a subroutine, you would have to find and edit every single print command in your entire program.

JAL and RET in RISC-V

JAL (jump and link): This instruction is used to jump to the subroutine defined in the instruction. When it is called it saves the current position in the program (return address) in the register defined in the instruction (the link register, in RISC-V RARS ra). Without saving the return address the program would not know where to jump back to.

RET (return): This instruction is used at the end of a subroutine. It looks at the return address (inside ra) and jumps back to it allowing the program to continue where it left off.

I used the knowledge I gained in section 1.8 when adding JAL to my simulation to write this section, this research helped me to understand the “magic” behind subroutines and specifically JAL.

2.5 Swapping Numbers

2.5.1 Program Code

```

38
39 #if s0>s1 set t1 to 1
40 sgt tl,s0,s1
41 #branch to sorted if t1 does not equal zero
42 bnez tl, sorted
43
44 #swap the values
45 add s2, s0, zero
46 add s0, s1, zero
47 add s1, s2, zero
48
49 #subroutine for when numbers are sorted
50 sorted:
51
52 #output message
53 li a7,4
54 la a0,sortedMsg
55 ecall
56
57 #output s0
58 li a7,1
59 mv a0,s0
60 ecall
61
62 #output character
63 li a7,11
64 li a0,45
65 ecall
66
67 #output s1
68 li a7,1
69 mv a0,s1
70 ecall
71

```

2.5.2 Expected Result

There will be 3 test cases for the branch instruction; when, s0 is smaller than s1 the swap instructions should be carried out and then the sorted subroutine will run as this is next in the program, when s0 is bigger the bnez instruction will branch to the sorted subroutine; skipping the swap and when both numbers are equal the swap should be carried out and then the sorted subroutine will run.

Test	s0	s1	First value in text output	Second value in text output
s0 smaller	01	29	29	01
s0 bigger	29	01	29	01
s0 = s1	01	01	01	01

2.5.3 Result

First number smaller:

```
Please use the last four digits of your student id as two 2-digit numbers
Enter a two digit number
01
Enter next number
29
Here are the sorted values: 29-1
The swap instructions have been carried out as expected.
```

First number bigger:

```
Please use the last four digits of your student id as two 2-digit numbers
Enter a two digit number
29
Enter next number
01
Here are the sorted values: 29-1
The program branched the sorted subroutine, skipping the swap instructions.
```

Both numbers equal:

```
Please use the last four digits of your student id as two 2-digit numbers
Enter a two digit number
01
Enter next number
01
Here are the sorted values: 1-1
The program branched the sorted subroutine, skipping the swap instructions.
```

2.5.4 Alternative approach

Research uncovered swapping two variable using XOR (Azad, 2007).

```
x = x xor y
y = x xor y
x = x xor y
```

XOR method implemented:

```
#if s0>s1 set t1 to 1
sgt t1,s0,s1
#branch to sorted if t1 does not equal zero
bnez t1, sorted

#swap the values using XOR
xor s0,s0,s1
xor s1,s0,s1
xor s0,s0,s1

#subroutine for when numbers are sorted
sorted:

#output message
li a7,4
la a0,sortedMsg
ecall
```

2.5.5 Explanation

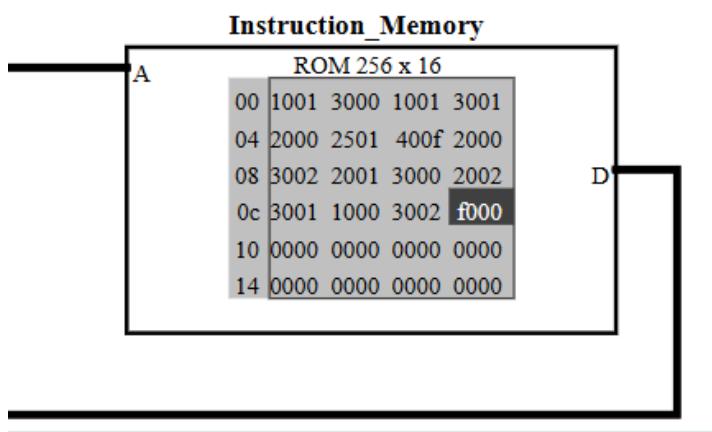
Both methods achieve the same result; however, the XOR method removes the need for a temporary register, I prefer the XOR method for this reason. This research helped me understand how bitwise operations can be used to optimize programs and in this specific case memory use.

2.5.6 Comparison

Swapping instructions in RISCV RARS:

0x00400050	0x00944433	xor x8, x8, x9	45: xor s0, s0, s1
0x00400054	0x009444b3	xor x9, x8, x9	46: xor s1, s0, s1
0x00400058	0x00944433	xor x8, x8, x9	47: xor s0, s0, s1

Swapping instructions in Logisim:



Comparison of swapping operations:

In the CPU simulation, copying a value from one address to another requires a slow, multistep process: loading the value to the ACC, storing it at a temp address, loading the second value to the original address and finally saving the temporary value to complete the swap.

In RISCV RARS the swaps are performed within the CPU's registers “load and store instructions access memory, while other instructions work with CPU registers” (Green 2024). Once the values have been loaded into the registers, the swapping operations happen instantly inside the processor, whereas, in Logisim the CPU which must repeatedly access slow memory for each step of the swap.

Comparison of efficiency:

The approach used in RISCV RARS is significantly more efficient, reducing 16 separate instructions in the Logisim simulation into 3 register operations inside RISV RARS. By minimizing usage of slow data memory, the execution speed and code length in RISV RARS is more preferential.

Value of research:

Research of the Load-Store architecture was valuable as it improved my understanding of the bottleneck that frequently accessing slow memory over register can cause.

2.6 Counting down with a loop

2.6.1 Loop Code

```

37
38    loop:
39
40    #call subroutine to output value
41    jal ra,printF
42    #subtract 1 from s0 and store in s0
43    addi s0,s0,-1
44    #set t1 to 1 if s0 is greater than s1
45    sgt t1,s0,s1
46    #branch to label loop if t1 is not zero
47    bnez t1,loop
48
49    loopdone:
50
51    #call subroutine to output value
52    jal,ra,printF
53    #end program
54    li a7,10
55    ecall
56
57
58
59    printf:
60    #output s0
61    li a7,1
62    mv a0,s0
63    ecall
64    #output whitespace
65    li a7,11
66    li a0,32
67    ecall
68
69    ret
70
71
--
```

2.6.2 Results

First number smaller:

```

Please use the last four digits of your student id as two 2-digit numbers
Enter a two digit number
01
Enter next number
29
29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
-- program is finished running (0) --
```

First number bigger:

```

Please use the last four digits of your student id as two 2-digit numbers
Enter a two digit number
29
Enter next number
01
29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
-- program is finished running (0) --
```

Both numbers equal:

```

Please use the last four digits of your student id as two 2-digit numbers
Enter a two digit number
01
Enter next number
01
1 0
-- program is finished running (0) --

```

2.6.3 Alternative approach

```
#subroutine for when numbers are sorted
sorted:
```

```
#countdown loop
loop:
```

```
#branch to loopdone if s0>s1
blt s0,s1,loopdone
#call subroutine to output value
jal ra,printF
#subtract 1 from s0 and store in s0
addi s0,s0,-1
#branch unconditionally to loop
b loop
```

```
loopdone:
```

```
#end program
li a7,10
ecall
```

First number smaller:

Please use the last four digits of your student id as two 2-digit numbers
Enter a two digit number
01
Enter next number
29
29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
-- program is finished running (0) --

First number bigger:

program is finished running (0)
Please use the last four digits of your student id as two 2-digit numbers
Enter a two digit number
29
Enter next number
01
29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
-- program is finished running (0) --

Both numbers equal:

```

Messages Run I/O
program is finished running (0)

Please use the last four digits of your student id as two 2-digit numbers
Enter a two digit number
01
Enter next number
01
1
-- program is finished running (0) --

```

2.6.4 Explanation

SGT/BNEZ loop

This loop checks the condition at the end, meaning the loop will always run at least once. When the inputs are equal, the loop runs once, printing the number and decrementing it. Afterwards, the condition evaluates to false causing the program to fall through to the loopdone section. This triggers the final print, sending the countdown out of range. The loop functions correctly in all other scenarios.

BLT/B loop

This loop checks condition at the beginning before any code is executed. This fixes the bug found in the previous loop. By using a blt instruction, the program immediately branches to the exit if the number is already too low. The unconditional branch at the bottom simply loops until the condition is true. Because the check happens first, and the final print statement was removed, the countdown always stays the specified range, even when inputs are equal.

2.7 Making use of arrays

2.7.1 Pseudocode

variable map
 s11 for array index counter
 s0 for user input
 s1 for user input
 s2 base address
 s3 total items (so output loop knows when to stop)
 t0 value to be added

Use .space to reserve space in memory for the array

Use .align to format memory correctly

Load array pointer to s2

Get user input for 2 numbers between 00 and 99

Sort the numbers

Use first loop (countdown loop) to store the integers in the array:

Branch to first loop end if s0<s1

JAL to input array

Addi -1 to s0

Branch unconditionally to first loop

first loop end:

save array index counter (total items) to s3

reset array index counter

second loop:

beq if s11=s3 branch to loop done

JAL output array

Unconditional branch second loop

Loop done:

Ends program#

Input array:

Multiply array index by 4 to get actual index

Add offset to array pointer to get actual memory position

Store s0 in array

Addi 1 to s11 (index counter)

Return

Output array:

Multiply array index by 4 to get actual index

Add offset to array pointer to get actual memory position

Output value at the address with whitespace

Add 1 to s11

return

2.7.2 Expected Results

When the program has finished running, the array in the data segment will store numbers 29 down to 1 in their hexadecimal equivalent in sequence. Register t1 will contain the memory address of the last value in the array. Register s1 will hold the lower limit value, when s0 is less than the value(0) in s1 the second loop will begin. Register s2 holds the base address of the memory, registers s3 and s11 hold the total number of items in the array showing the index counter correctly reached the end of the array. Register a0 will contain 0x20, the ASCII code for a whitespace which is the character for the last syscall executed.

2.7.3 Result

Run I/O of completed program:

The screenshot shows the Run I/O window with the 'Run I/O' tab selected. The window displays the following text:

```
-- program is finished running (0) --
Please use the last four digits of your student id as two 2-digit numbers
Enter a two digit number
29
Clear
Enter next number
01
29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
-- program is finished running (0) --
```

Data Segment of completed program:

The screenshot shows the Data Segment window with the following details:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x61656c50	0x75206573	0x74206573	0x6c206568	0x20747361	0x72756f66	0x67696420	0x20737469
0x10010020	0x752066ef	0x2072756f	0x64757473	0x20746e65	0x1206469	0x7742073	0x2d3220ef	0x65676964
0x10010040	0x756e2074	0x7265626d	0x000a2073	0x65746e45	0x20612072	0x206f7774	0x69676964	0x756e2074
0x10010060	0x726562ed	0x6e45000a	0x20726574	0x7478656e	0xd756e20	0x20726562	0x6548000a	0x61206572
0x10010080	0x74206572	0x73206568	0x6574726f	0x61762064	0x7365756c	0x0000203a	0x0000001d	0x0000001c
0x100100a0	0x0000001b	0x0000001a	0x00000019	0x00000018	0x00000017	0x00000016	0x00000015	0x00000014
0x100100c0	0x00000013	0x00000012	0x00000011	0x00000010	0x0000000f	0x0000000e	0x0000000d	0x0000000c
0x100100e0	0x0000000b	0x0000000a	0x00000009	0x00000008	0x00000007	0x00000006	0x00000005	0x00000004
0x10010100	0x00000003	0x00000002	0x00000001	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Below the table, there are buttons for navigation and filtering: <| |>, 0x10010000 (.data), Hexadecimal Addresses, Hexadecimal Values, ASCII.

Registers of completed program:

Registers	Floating Point	Control and Status	
Name	Number	Value	
zero	0	0x00000000	
ra	1	0x00400084	
sp	2	0x7ffffefffc	
gp	3	0x10008000	
tp	4	0x00000000	
t0	5	0x00000000	
t1	6	0x10010108	
t2	7	0x00000000	
s0	8	0x00000000	
s1	9	0x00000001	
a0	10	0x00000020	
a1	11	0x00000000	
a2	12	0x00000000	
a3	13	0x00000000	
a4	14	0x00000000	
a5	15	0x00000000	
a6	16	0x00000000	
a7	17	0x0000000a	
s2	18	0x10010098	
s3	19	0x0000001d	
s4	20	0x00000000	
s5	21	0x00000000	
s6	22	0x00000000	
s7	23	0x00000000	
s8	24	0x00000000	
s9	25	0x00000000	
s10	26	0x00000000	
s11	27	0x0000001d	
t3	28	0x00000000	
t4	29	0x00000000	
t5	30	0x00000000	
t6	31	0x00000000	
pc		0x00400090	

End of program analysis shows results are as expected.

2.7.4 Explanation

In high level languages like C++ and Python, an array is a data structure which contains elements that can be read using a key or index; however, in RISC-V an array is simply a pre-allocated block of memory.

Using an Array in RISC-V

Defining an Array:

To define an array, you must give it a name and size in the data segment.

```
.align 2
#100 bytes for storage of 100 integers
emptyArray: .space 400
```

In this example the name/label of the array is “emptyArray” and the directive “.space 400” allocates 400 contiguous bytes of memory for the array. The .align 2 directive is used to “Pad the location counter (in the current subsection)

to a particular storage boundary”(Free Software Foundation 2025), this ensures the array starts at a 4 word boundary allowing the CPU to read and write to the array without erroring.

Addresses:

To access specific items in the array this formula can be used:

$$\text{Element Address} = \text{Base Address} + \text{Index} * \text{Size(bytes)}$$

To implement this in RISCV RARS, the base address must be saved to a register.

```
main:
```

```
#get base address of emptyArray
la,s2,emptyArray
```

In this snippet the base address of the array is saved at register s2.

```
outputArray:
#multiply array index by 4
slli t1,s11,2
#add offset to base address
add t1,s2,t1
#output value at the address
lw a0,0(t1)
li a7,1
ecall
```

This is then used in the output array subroutine to calculate the value at the given index.

Loop and Index Counter:

To know the length of the array, a register was used to count the elements as they were stored.

```

firstLoopEnd:

#save total amount of items
mv s3,s11

#reset index
li s11,0

```

As each element was added to the array, s11 was incremented by 1. Once the loop had finished this value was saved to s3 which served as the amount of the elements in the loop

```

secondLoop:

#branch to loop done if index counter = total items
beq s11,s3,loopDone

```

This was used in the second part of the loop here to know when to stop the output loop as the end of the array had been reached.

Value of Research:

The research into the .align 2 directive showed how CPUs process data in 4-byte chunks, rather than single bytes. This took the 'magic' out of arrays in high-level languages, as I now have a deep understanding of the underlying mechanics that are abstracted away.

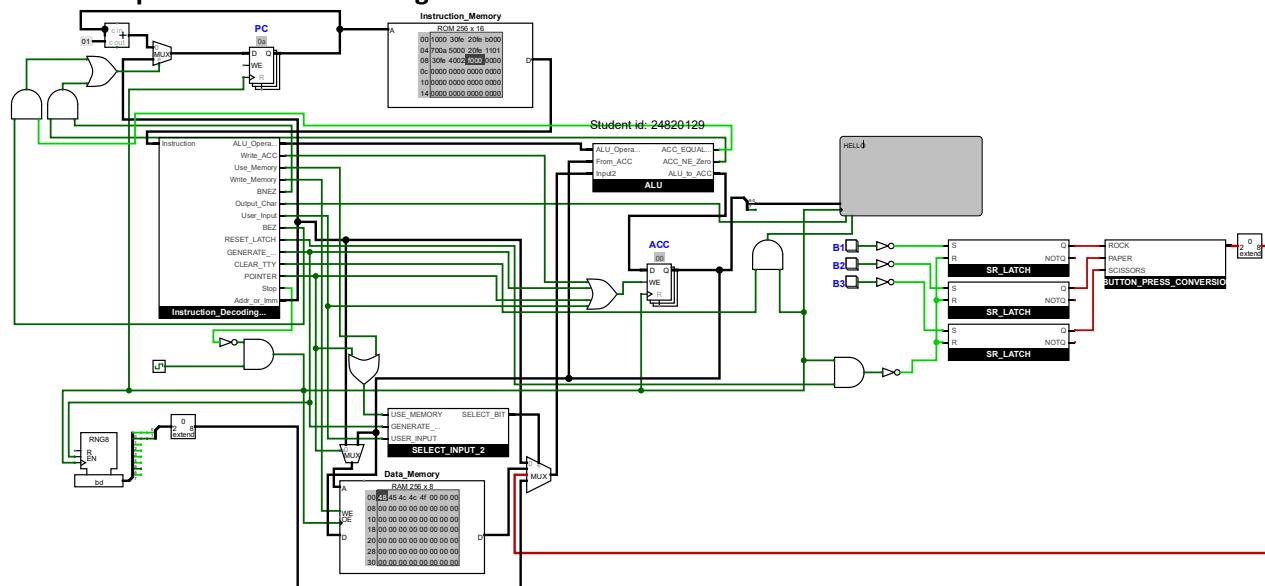
2.7.5 Comparison

Adding an Array to the Logisim simulation:

Adding pointers

To allow for arrays to be used in Logisim, functionality would have to be added to allow the output from the ACC to be the address input for the data memory. This would require adding a MUX before the address input which would decide whether address was used from the instruction or the ACC.

Pointer implementation inside Logisim from task 1.8:



Software implementation

The software implementation of Arrays would require manually (i.e the programmer) setting aside data to be used for the array. Mapping the data memory would be imperative and could sound something like “addresses 1e – 28 are used for emptyArray” in a comment.

Example of Array implementation in Logisim:

```
# Address e7-fe: emptyArray  
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

2.8 Designing a game

2.8.1 Stage 1 – Menu, Character Selection and Enemy Spawning

General Game Scope:

The chosen game for this stage is a text-based RPG, the game will allow the player to: select a character/build, fight enemies and fight a boss thus completing the game.

The game will use .word arrays to store the different attributes of 3 character and enemy types and will implement a game loop with multiple battles to simulate an RPG feel.

Initial Pseudocode/Plan:

Variable map

pointer for currentPlayer array stored in s0
pointer for currentEnemy array stored in s1

.data

All player and enemy arrays will follow the format

label: .word health, damage, healing, name

Knight: .word

Wizard: .word

Musketeer: .word

Goblin: .word

Golem: .word

Witch: .word

currentEnemy: .word

currentPlayer: .word

For all names, as .word struct only stores 4 bites they will be defined as strings and the array will store the pointer to them.

```

E.G:
knight_str: .string "Knight"
-----



main:

la s0, currentPlayer
la s1, currentEnemy

output to system "Castle Defence 2025"
output to system "1 - Start Game"
output to system "2 - Exit Game"

li t1 1,2

beq t0,t1(1) to chooseCharacter
beq t0,t1(2) to exit
else output to system "Invalid input"
b main
-----


chooseCharacter:

output to system"Choose your build:"
output to system"1 - Knight - Attributes: Health - 3, Damage - 2, Healing - 1"
output to system"2 - Musketeer - Attributes: Health - 1, Damage - 3, Healing - 2"
output to system"3 - Wizard - Attributes: Health - 2, Damage - 1, Healing - 3"

get user input

user input in t0

li t1 , 1,2,3

beq t1 = 1 to selectKnight
beq t1 = 2 to SelectMusk
beq t1 = 3 to selectWiz
-----


gameLoop:

#print currentEnemy name alongside spawn messgae
lw t1, 12(s1)
output to system "An enemy " t1 "has spawned"
-----


setCurrentPlayer:

#t1 is pointer to character array (knight)

#load health multiplier

```

```

li t5,50

copy health (Word 0)
lw t4, 0(t1)
mul t4,t5,t4
sw t4, 0(s0)

#load damage multiplier
li t5,20

copy damage (Word 1)
lw t4, 4(t1)
mul t4,t5,t4
sw t4, 4(s0)

#load healing multiplier
li t5,20

copy heal (Word 2)
lw t4, 8(t1)
mul t4,t5,t4
sw t4, 8(s0)

copy name address (Word 3)
lw t4, 12(t1)
sw t4, 12(s0)

ret

```

setCurrentEnemy

#t1 is pointer to enemy array (witch)

```

#load health multiplier
li t5,50

copy health (Word 0)
lw t4, 0(t1)
mul t4,t5,t4
sw t4, 0(s1)

#load damage multiplier
li t5,20

copy damage (Word 1)
lw t4, 4(t1)
mul t4,t5,t4
sw t4, 4(s1)

#load healing multiplier
li t5,10

copy heal (Word 2)
lw t4, 8(t1)
mul t4,t5,t4

```

```
sw t4, 8($1)

copy name address (Word 3)
lw t4, 12($1)
sw t4, 12($1)

ret
```

```
getEnemySpawnID:
```

```
li a7,42
li a0, 0
li a1, 3

t1 = 0,1,2

beq a0 = 0 selectGoblin
beq a0 = 1 selectGolem
beq a0 = 2 SelectWitch
```

```
selectKnight:
```

```
la t1, knight
jal setCurrentPlayer
b getEnemySpawnID
```

```
selectMusk:
```

```
la t1, musketeer
jal setCurrentPlayer
b getEnemySpawnID
```

```
selectWiz:
```

```
la t1, wizard
jal setCurrentPlayer
b getEnemySpawnID
```

```
selectGoblin:
```

```
la t1, goblin
JAL setCurrentEnemy
b gameLoop
```

```
selectGolem:
```

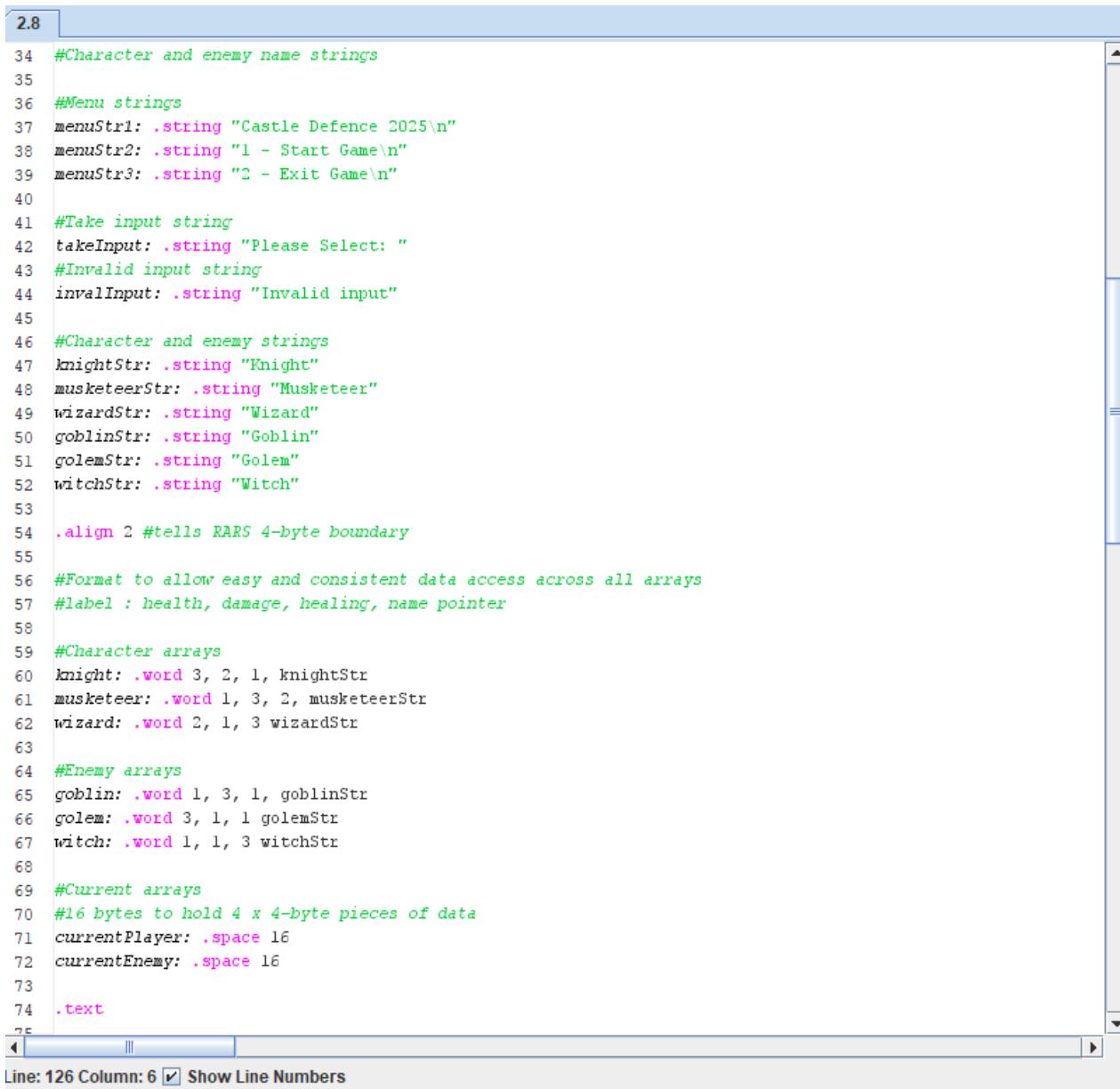
```
la t1, orc
JAL setCurrentEnemy
b gameLoop
```

```
selectWitch:  
  
la t1, witch  
JAL setCurrentEnemy  
b gameLoop
```

```
-----  
exit:
```

Memory and Array Implementation:

.data section:



```
2.8  
34 #Character and enemy name strings  
35  
36 #Menu strings  
37 menuStr1: .string "Castle Defence 2025\n"  
38 menuStr2: .string "1 - Start Game\n"  
39 menuStr3: .string "2 - Exit Game\n"  
40  
41 #Take input string  
42 takeInput: .string "Please Select: "  
43 #Invalid input string  
44 invalidInput: .string "Invalid input"  
45  
46 #Character and enemy strings  
47 knightStr: .string "Knight"  
48 musketeerStr: .string "Musketeer"  
49 wizardStr: .string "Wizard"  
50 goblinStr: .string "Goblin"  
51 golemStr: .string "Golem"  
52 witchStr: .string "Witch"  
53  
54 .align 2 #tells RARS 4-byte boundary  
55  
56 #Format to allow easy and consistent data access across all arrays  
57 #label : health, damage, healing, name pointer  
58  
59 #Character arrays  
60 knight: .word 3, 2, 1, knightStr  
61 musketeer: .word 1, 3, 2, musketeerStr  
62 wizard: .word 2, 1, 3 wizardStr  
63  
64 #Enemy arrays  
65 goblin: .word 1, 3, 1, goblinStr  
66 golem: .word 3, 1, 1 golemStr  
67 witch: .word 1, 1, 3 witchStr  
68  
69 #Current arrays  
70 #16 bytes to hold 4 x 4-byte pieces of data  
71 currentPlayer: .space 16  
72 currentEnemy: .space 16  
73  
74 .text  
75
```

Line: 126 Column: 6 Show Line Numbers

The character and enemies' attributes are stored in fixed-length templates. They are never changed directly, their values are copied into the currentPlayer/currentEnemy when the game is running. Due to RISC-V architecture, .word

is limited to 4 bytes; therefore, each array contains a pointer to the name string and not the actual string, to ensure that each array remains a consistent size in memory. Menu strings and input/error messages are also present to prompt the user on what to do/what went wrong.

Testing main:

Code for main:

```
main:  
  
#load pointers to currentPlayer and currentEnemy to s0 and s1 respectively  
  
la s0, currentPlayer  
la s1, currentEnemy  
  
#output menu strings and take input to system  
  
li a7,4  
  
la a0,menuStr1  
ecall  
la a0,menuStr2  
ecall  
la a0,menuStr3  
ecall  
la a0,takeInput  
ecall  
  
#get user input  
li a7,5  
ecall  
mv t0,a0  
  
#validate user input  
  
#if input 1 or 2 branch respectively  
li t1,1  
beq t1,t0,chooseCharacter  
li t1,2  
beq t1,t0,exit  
  
#else return to main  
li a7,4  
la a0,invalInput  
ecall  
b main
```

Code for chooseCharacter and exit:

```
#sub routines  
  
chooseCharacter:  
  
li a7,1  
li a0,1  
ecall  
  
exit:  
li a7,10  
ecall
```

Input	Expected Run I/O	Actual Run I/O	Comment
1	Output 1 and then exit	<pre>Castle Defence 2025 1 - Start Game 2 - Exit Game Please Select: 1 1 -- program is finished running (0)</pre>	Test successful
2	Exit	<pre>Castle Defence 2025 1 - Start Game 2 - Exit Game Please Select: 2 -- program is finished running (0) --</pre>	Test successful
Hello	Error	Runtime error: 0x00400048: invalid input integer	Test result as expected as system is expecting integer, will add proper input validation later in development.
4	Output invalid input message and then loop to menu again	<pre>Castle Defence 2025 1 - Start Game 2 - Exit Game Please Select: 4 Invalid input Castle Defence 2025 1 - Start Game 2 - Exit Game Please Select: 2</pre>	Test successful.

Choose Character and Enemy Selection:

.data section with character selection strings added:

```

#-----
#chooseCharacter strings
chooseBuildStr: .string "\nChoose your build\n\n"

#Choose Knight Strings
chooseKnightStr1: .string "1 - Knight\n"
chooseKnightStr2: .string "Attributes: Health Damage Healing\n"
chooseKnightStr3: .string "            3      2      1      \n\n"
#Choose Musketeer Strings
chooseMusketeerStr1: .string "2 - Musketeer\n"
chooseMusketeerStr2: .string "Attributes: Health Damage Healing\n"
chooseMusketeerStr3: .string "            1      3      2      \n\n"
#Choose Wizard Strings
chooseWizardStr1: .string "3 - Wizard\n"
chooseWizardStr2: .string "Attributes: Health Damage Healing\n"
chooseWizardStr3: .string "            2      1      3      \n\n"
#-----
```

Code for implementing character selection:

```

chooseCharacter:

#print character selection information
jal ra chooseCharacterStrings

#output take input string and then take input
la a0, takeInput
li a7,4
ecall
li a7,5
ecall
#move user input for character selection to t0
mv t0,a0
#compare user input with value in t1 and branch appropriately
li t1,1
beq t1,t0,selectKnight
li t1,2
beq t1,t0,selectMusketeer
li t1,3
beq t1,t0,selectWizard
#else output inval message and loop to chooseCharacter
li a7,4
la a0,invalInput
ecall
b chooseCharacter
```

chooseCharacterStrings subroutine:

```

#-----#
#function to print all strings for chooseCharacter without cluttering
chooseCharacterStrings:

#load print string syscall to a7
li a7,4
#load pointers to a0 and print
la a0,chooseBuildStr
ecall

la a0,chooseKnightStr1
ecall

la a0,chooseKnightStr2
ecall

la a0,chooseKnightStr3
ecall

la a0,chooseMusketeerStr1
ecall

la a0,chooseMusketeerStr2
ecall

la a0,chooseMusketeerStr3
ecall

la a0,chooseWizardStr1
ecall

la a0,chooseWizardStr2
ecall

la a0,chooseWizardStr3
ecall
#return to choose character
ret

```

Inside chooseCharacter a function called chooseCharacterStrings was created to keep the code clean and maintainable. Input is then taken from the user and stored in t0 for the branching logic.

selectKnight (example):

```

selectKnight:
la t1,knight
mv t6,s0
jal ra copyFromArray
#branch to getEnemySpawnID for enemy spawning
b getEnemySpawnID

```

copyFromArray:

```

copyFromArray:

#t1 is pointer to selected character array

#load health multiplier
li t5,50
#copy health and save in currentXX array (s0)
lw t4,0(t1)
mul t4,t5,t4
sw t4,0(t6)

#load damage multiplier
li t5,20
#copy damage and save in currentXX array (s0)
lw t4,4(t1)
mul t4,t5,t4
sw t4,4(t6)

#load healing multiplier
li t5,20
#copy healing and save in currentXX array (s0)
lw t4,8(t1)
mul t4,t5,t4
sw t4,8(t6)

#copy name address and save in currentXX array (s0)
lw t4,12(t1)
sw t4,12(t6)

ret

```

getEnemySpawnID:

```

#function to get random enemy spawn
getEnemySpawnID:

#get a random number within the range 0 and 2
li a7,42
li a0,0
li a1,3
ecall

#check random number and branch appropriately

mv t0,a0
li tl,0
beq tl,t0,selectGoblin
li tl,1
beq tl,t0,selectGolem
li tl,2
beq tl,t0,selectWitch

#else try again

b getEnemySpawnID

```

For user input 1, which equates to selectKnight, the selectKnight function is called. Inside the pointer to the array (base address) is passed to t1 and the pointer to the array currentPlayer is passed to t6 for use inside copyFromArray. copyFromArray retrieves each attribute from the necessary character array and saves it in the currentPlayer array at the given offset, according to the format defined in the comments in the .data section, multiplying each attribute by the respective multiplier. Finally, a branch to getEnemySpawnID selects the first enemy the player will battle using random number generation.

selectGoblin (example):

```
selectGoblin:
la t1,goblin
mv t6,s1
jal ra copyFromArray
b gameLoop
```

This function loads the pointer to the base address of the goblin array to t1 and moves the pointer of the currentEnemy array to t6 for use inside the copyFromArray.

Testing character selection and enemy spawning:

Character selection:

Before execution of copyFromArray for character selection:

The screenshot shows a debugger interface with several windows:

- Assembly Window:** Displays assembly code with highlighted instructions. The highlighted instruction is `lw t4,4(t1)`.
- Registers Window:** Shows the state of various registers. The register `t4` has a value of `1`.
- Memory Dump Window:** Shows the memory dump for the `0x10010000 (.data)` segment. It includes columns for Address, Value (+0), Value (+4), Value (+8), Value (+c), Value (+10), Value (+14), Value (+18), and Value (+1c). The value for `t4` at address `0x10010180` is `1`.
- Messages Window:** Displays the message "Please Select: 2".

As character 2(musketeer) was selected by the user, this input has been moved to t0. The program then correctly branches to selectMuskeeter, loading the pointer to the array in t1.

After execution of copyFromArray:

The screenshot shows the GDB-like interface with several panes:

- Registers** pane: Shows registers with values like zero=0, ra=1, sp=2, gp=3, tp=4, t0=5, t1=6, t2=7, s0=8, s1=9, a0=10, a1=11, a2=12, a3=13, a4=14, a5=15, a6=16, a7=17, s2=18, s3=19, s4=20, s5=21, s6=22, s7=23, s8=24, s9=25, s10=26, s11=27, t3=28, t4=29, t5=30, t6=31, pc=4194512.
- Text Segment** pane: Shows assembly code for the Text Segment. A specific instruction at address 0x004000e4 is highlighted: `0x03df0eb3 mul x29,x30,x29`.
- Data Segment** pane: Shows memory dump starting at address 0x100101c0. The value at address 0x100101f0 is highlighted as 268501051.
- Messages** pane: Displays the message "Please Select: 2".

After the copyFromArray function has been executed, values are stored in the data segment at addresses 0x100101F0 to 0x10010200. The data is in the correct format with the pointer to the string in the last address indicating a successful test.

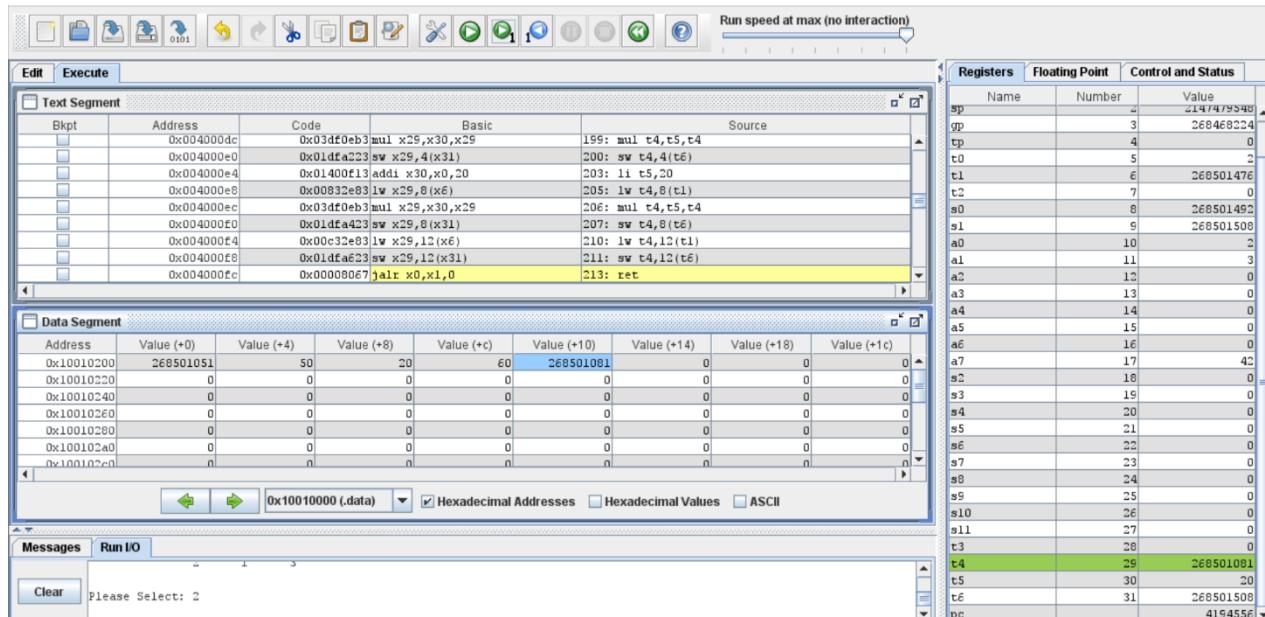
Enemy Spawning:

Before execution of copyFromArray:

This screenshot is similar to the previous one but before the copyFromArray function is executed. The registers and memory dump are identical, except for the value at address 0x100101f0 which is now 0.

The randomly generated number 2 is stored in t0 for use inside copyFromArray, which then branched to selectWitch storing the pointer to the witch array in t1 and the pointer to the currentEnemyArray in t6.

After execution of copyFromArray:



After the copyFromArray function has been executed, values are stored in the data segment at addresses 0x10010204 to 0x10010210. The data is in the correct format with the pointer to the string in the last address meaning a successful test.

2.8.2 Stage 2 – First Battle

After identifying a random number is necessary more than once for combat, getEnemySpawnID was refactored to getRandom.

getRandom:

```

224 getRandom:
225
226 #get a random number within the range 0 and t2
227 #t2 is upper range of random
228 li a7,42
229 li a0,0
230 mv a1,t2
231 ecall
232
233 #check random number and branch appropriately
234
235 mv t0,a0
236 li t1,0
237 beq t1,t0,selectGoblin
238 li t1,1
239 beq t1,t0,selectGolem
240 li t1,2
241 beq t1,t0,selectWitch
242
243 #else try again
244 b getRandom

```

Pseudocode for First Battle:


```

#s3 round count variable
gameLoop:

battleLoop:

endBattleCheck:

    #get healing stat
    lw t1,0($0)
    #get enemy healing stat
    lw t2,0($1)

    blez t1,playerLosesBattle
    blez t2,playerWinsBattle

userTurn:

firstRoundCheck:

    bnez $3,notFirst

firstRound:
    Output to system "A [ENEMY NAME] HAS APPEARED"
    Output to system "What will you do?"
    b notFirst

notFirst:
    output to system "1 - Attack"
    output to system "2 - Heal"
    output to system "Health: [Health value]"

    take user input and store in t0

    mv a0,$0
    mv a1,$1

    li t1,1
    li t2,2

    beq t0,t1,useApplyDmg
    beq t0,t2,useApplyHeal
    else
    print Invalid input message
    b userTurn

useApplyDmg:

    jal ra applyDmg
    jal printUserDmg
    b enemyTurn

useApplyHeal:

    jal ra applyHeal

```

```

jal printApplyHeal
b enemyTurn

enemyTurn:
    get random number and store in t0

    mv a0,s1
    mv a1,s0

    li t1,1
    li t2,2

    beq t0,t1,enemyApplyDmg
    beq t0,t2,enemyApplyHeal

    enemyApplyDmg:

    jal ra applyDmg
    jal printEnemyDmg
    addi s3,s3,1
    b endBattleCheck

    enemyApplyHeal:

    jal ra applyHeal
    jal printEnemyHeal
    addi s3,s3,1
    b endBattleCheck

```

Pseudocode for necessary functions:

```

a0 = attackers address
a1 = target address

applyDmg:

#get attacker damage
lw t2,4(a0)
#get target health
lw t3,0(a1)
#subtract attack from health and update
sub t3,t3,t2
sw t3, 0(a1)

ret

a0 = healers address

applyHeal:

#get healing stat
lw t2,8(a0)
#get healers health

```

```

lw t3,0(a0)
#apply healing and save
add t3,t3,t2
sw t3,0(a0)

ret

printUserDmg:
output to system "You dealt [damage] amount of damage"
output to system "Enemy [enemy] has [enemy health] health left "
ret

printUserHeal:
output to system "You healed for [healing] amount of health"
output to system "Your total health is [health]"
ret

printEnemyDmg:
output to system: "You were dealt [damage] amount of damage by enemy [enemy]"
ret

printEnemyHeal:
output to system: "Enemy [enemy] healed for [healing] amount of healing"
ret

playerWinsBattle:
output to system "You defeated the enemy [enemy]"
exit

playerLosesBattle:
output to system "You were defeated by the enemy [enemy]"
exit

```

Implementation of subroutines/functions:

Firstly, the subroutines that the game loop uses were implemented.

Strings in .data for player/enemy action and win or loss:

```
83 #strings for player win/loss
84
85 playerWinsStr: .string "You defeated the enemy "
86 playerLosesStr: .string "You were defeated by the enemy "
87
88 #
89 #strings for enemy and player actions
90
91 #print user damage
92
93 userDmgStr1: .string "You dealt "
94 userDmgStr2: .string " amount of damage\n"
95 userDmgStr3: .string "Enemy "
96 userDmgStr4: .string " has "
97 userDmgStr5: .string " health left\n"
98
99 #prints user heal
100
101 userHealStr1: .string "You healed for "
102 userHealStr2: .string " amount of health\n"
103 userHealStr3: .string "Your total health is\n"
104
105 #prints enemy damage
106
107 enemyDmgStr1: .string "You were dealt "
108 enemyDmgStr2: .string " amount of damage by enemy\n"
109
110 #prints enemy heal
111
112 enemyHealStr1: .string "Enemy "
113 enemyHealStr2: .string " healed for "
114 enemyHealStr3: .string " amount of healing\n"
115
116 #-----
```

Apply heal and damage functions:

```

#-----  

#functions to apply damage and healing  

#attacker address/healer address = a0  

#target address for attack = a1  

applyDmg:  

#get attacker damage stat  

lw t2,4(a0)  

#get target health  

lw t3,0(a1)  

#subtract attack from health and update  

sub t3,t3,t2  

sw t3, 0(a1)  

ret  

applyHeal:  

#get healing stat  

lw t2,8(a0)  

#get healers health  

lw t3,0(a0)  

#apply healing and update  

#apply healing and update  

add t3,t3,t2  

sw t3,0(a0)  

ret

```

Print functions for user/enemy actions (example printUserDmg):

```
225 printUserDmg:  
226  
227 #print "You dealt [damage] amount of damage"  
228 li a7,4  
229 la a0,userDmgStr1  
230 ecall  
231 #get player damage using offset 4  
232 li a7,1  
233 lw a0,4($0)  
234 ecall  
235  
236 li a7,4  
237 la a0,userDmgStr2  
238 ecall  
239  
240 #print "Enemy [enemy name] has [enemy health] health left"  
241 la a0,userDmgStr3  
242 ecall  
243 #get enemy name pointer using offset 12  
244 lw a0,12($1)  
245 ecall  
246  
247 la a0,userDmgStr4  
248 ecall  
249 #print enemy health value  
#print enemy health value  
li a7,1  
lw a0,0($1)  
ecall  
  
la a0,userDmgStr5  
ecall  
  
ret
```

First battle implementation:

```

193
194 gameLoop:
195
196 endBattleCheck:
197
198 #get user health stat
199 lw t1,0($0)
200 #get enemy health stat
201 lw t2,0($1)
202
203 #branch to player win/loses respectively
204
205 blez t1,userLosesBattle
206 blez t2,userWinsBattle
207
208 #else
209 ret
210
211 battleLoop:
212
213
214 userTurn:
215
216
217 #check if first round
218 firstRoundCheck:
219 bneq $3,notFirst
220
221 #only output these strings if it is the first round of the fight
222
223 firstRound:
224 #prints first round strings
225 jal ra, firstRoundPrint
226
227
228
229 notFirst:
230 #check if enemy or player health is zero
231 jal ra,endBattleCheck
232 #use print function to print strings
233 jal ra,notFirstPrint
234 #take user input and store in t0
235 li a7,5
236 ecall
237 #move user input to safe place
238 mv t0,a0

```

```
238 mv t0,a0
239 #move pointer to array addresses to a0 and a1 for use in apply damage and apply
240 mv a0,s0
241 mv a1,s1
242 #li 1 and 2 for checking user input for branching
243 li t1,1
244 li t2,2
245 #perform branch checks and branch to label otherwise loop
246 beq t0,t1,useApplyDmg
247 beq t0,t2,useApplyHeal
248 #print invalid input message and loop
249 li a7,4
250 la a0,invalInput
251 ecall
252 b userTurn
253
254
255 useApplyDmg:
256 #apply damage and print strings, then branch to enemy turn
257 jal ra applyDmg
258 jal ra printUserDmg
259 b enemyTurn
260
261 useApplyHeal:
262 #apply heal and strings, then branch to enemy turn
263 #apply damage and print strings, then branch to enemy turn
264 jal ra applyDmg
265 jal ra printUserDmg
266 b enemyTurn
267
268 enemyTurn:
269 #get random number 0 or 1 and store in t0
270 li t2,2
271 jal ra getRandom
272 mv t0,a0
273 #move pointers for use in apply damage/heal
274 mv a0,s1
275 mv a1,s0
276 #load immediate 0 and 1 for apply damage/heal
277 li t1,0
278 li t2,1
279 beq t0,t1,enemyApplyDmg
280 beq t0,t2,enemyApplyHeal
```

```

281 enemyApplyDmg:
282 #print and apply damage
283 jal ra applyDmg
284 jal ra printEnemyDmg
285 #increment round count
286 addi s3,s3,1
287 #branch to end battle check
288 b userTurn
289 enemyApplyHeal:
290 jal ra applyHeal
291 jal ra printEnemyHeal
292 addi s3,s3,1
293 b userTurn
294
295 #-----
```

userWinsBattle:

```
#print "You defeated the enemy [enemy name]"
li a7,4
la a0,playerWinsStr
ecall

#get enemy name pointer using offset 12
lw a0,12($l)
ecall

li a7,4
la a0,newLine
ecall

#exit program
b exit
```

userLosesBattle:

```
#print "You were defeated by the enemy [enemy name]"
li a7,4
la a0,playerLosesStr
ecall

#get enemy name pointer using offset 12 of
lw a0,12($l)
ecall

li a7,4
la a0,newLine
ecall

#exit program
b exit
..
```

In the pseudocode, endBattleCheck was being called at the end of the battle. This was causing the user or enemy to carry out their turn, even if the battle should have ended. To fix this bug, the check now happens after a user or enemy performs an action.

```

useApplyDmg:
#apply damage and print strings, then branch to enemy turn
jal ra applyDmg
jal ra printUserDmg
jal ra endBattleCheck
b enemyTurn

useApplyHeal:
#apply heal and strings, then branch to enemy turn
jal ra applyHeal
jal ra printUserHeal
jal ra endBattleCheck
b enemyTurn
enemyApplyDmg:
#print and apply damage
jal ra applyDmg
jal ra printEnemyDmg
jal ra endBattleCheck
#increment round count
addi s3,s3,1
#branch to end battle check
b userTurn
enemyApplyHeal:
jal ra applyHeal
jal ra printEnemyHeal
jal ra endBattleCheck
addi s3,s3,1
b userTurn

```

Testing the battle:

4 user/enemy actions and two battle outcomes are tested in this section. User/enemy heal/damage and player win/loss.

User/Enemy Actions:

User or Enemy	Action Tested	Expected Outcome	Actual Outcome
User	Heal	Print "You healed for [healing] amount of healing" "Your total health is [health]" and then proceed to enemy turn. Health should be old health+healing.	A Goblin HAS APPEARED What will you do? 1 - Attack 2 - Heal Health: 50 Please Select: 2 You healed for 40 amount of health Your total health is 90Enemy Goblin healed for 20 amount of healing

User	Attack	Print "You dealt [damage] amount of damage" "Enemy [enemy name] has [enemy health] remaining" then proceed to enemy turn	What will you do? 1 - Attack 2 - Heal Health: 90 Please Select: 1 You dealt 60 amount of damage Enemy Goblin has 10 health left Enemy Goblin healed for 20 amount of healing
Enemy	Heal	Print "Enemy [enemy name] healed for [healing] amount of healing" then branch to endBattleCheck	A Goblin HAS APPEARED What will you do? 1 - Attack 2 - Heal Health: 50 Please Select: 2 You healed for 40 amount of health Your total health is 90Enemy Goblin healed for 20 amount of healing
Enemy	Attack	Print "You were dealt [damage] amount of damage by enemy" then branch to endBattleCheck	A Witch HAS APPEARED What will you do? 1 - Attack 2 - Heal Health: 50 Please Select: 2 You healed for 40 amount of health Your total health is 90You were dealt 20 amount of damage by enemy

All actions tests were successful in terms of subtracting and adding health, the strings will be reformatted for a more game like feel.

User win/loss:

Win or Loss	Expected Outcome	Actual outcome
Win	Print "You defeated the enemy [enemy name]" then exit	You dealt 60 amount of damage Enemy Golem has -30 health left You defeated the enemy Golem
Loss	Print "You were defeated by [enemy name]" then exit	You healed for 20 amount of health Your total health is 50You were dealt 60 amount of damage by enemy You were defeated by the enemy Goblin -- program is finished running (0) --

Both tests were successful. As the game flow is functional, the appearance was edited.

By adding and removing new line characters, the game now looked like this in the run i/o.

When the game begins a series of new line characters are used to essentially clear the console.

```

#first round strings
firstRoundStr1: .string "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\nA "
firstRoundStr2: .string " HAS APPEARED\n\n"

#not first strings
notFirstStr1: .string "What will you do? \n\n"
notFirstStr2: .string "1 - Attack\n"
notFirstStr3: .string "2 - Heal\n\n"
notFirstStr4: .string "Health: "

```

Run I/O after emendations:

Castle Defence 2025

1 - Start Game
2 - Exit Game

Please Select: 1

Choose your build

1 - Knight
Attributes: Health Damage Healing
3 2 1

2 - Musketeer
Attributes: Health Damage Healing
1 3 2

3 - Wizard
Attributes: Health Damage Healing
2 1 3

Please Select: 2

A Witch HAS APPEARED

What will you do?

1 - Attack
2 - Heal

Health: 50

Please Select: 1

You dealt 60 amount of damage
Enemy Witch has -10 health left

You defeated the enemy Witch

-- program is finished running (0) --

Final comments:

At this point in development, having a health value of -10 shown to the user is acceptable as the only priority right now is core functionality. After the five battle loop and boss battle functionality is implemented bugs like this will be addressed.

2.8.3 Stage 3 – Five Battles and Boss Fight

To implement the boss fight, A battle counter variable will be implemented. If battle counter variable is 5 the boss fight will be initiated, when it reaches 6 the game will end and notify the player that they have beat the game. To implement the five battle loop, since the player has one life, inside userLosesBattle there will be logic to end the game; equally, if the user wins the battle a new enemy will be spawned which will lead back to the battleLoop.

Pseudocode for Five Battles and Boss Fight:

```
#s4 = battle counter

gameLoop:

endBattleCheck:

#get user health stat
lw t1,0($0)
#get enemy health stat
lw t2,0($1)

#branch to player win/loses respectively

blez t1,userLosesBattle
blez t2,userWinsBattle

#else
ret

battleLoop:

userTurn:

#check if first round
firstRoundCheck:
bneq s3,notFirst

#only output these strings if it is the first round of the fight

firstRound:
#prints first round strings
jal ra, firstRoundPrint

notFirst:

#check if enemy or player health is zero
jal ra,endBattleCheck
```

```

#use print function to print strings
jal ra,notFirstPrint
#take user input and store in t0
li a7,5
ecall
#move user input to safe place
mv t0,a0
#move pointer to array addresses to a0 and a1 for use in apply damage and apply heal
mv a0,s0
mv a1,s1
#li 1 and 2 for checking user input for branching
li t1,1
li t2,2
#perform branch checks and branch to label otherwise loop
beq t0,t1,useApplyDmg
beq t0,t2,useApplyHeal
#print invalid input message and loop
li a7,4
la a0,invalInput
ecall
b userTurn

useApplyDmg:
#apply damage and print strings, then branch to enemy turn
jal ra applyDmg
jal ra printUserDmg
jal ra endBattleCheck
b enemyTurn

useApplyHeal:
#apply heal and strings, then branch to enemy turn
jal ra applyHeal
jal ra printUserHeal
jal ra endBattleCheck
b enemyTurn

enemyTurn:
#get random number 0 or 1 and store in t0
li t2,2
jal ra getRandom
mv t0,a0
#move pointers for use in apply damage/heal
mv a0,s1
mv a1,s0
#load immediate 0 and 1 for apply damage/heal
li t1,0
li t2,1
beq t0,t1,enemyApplyDmg
beq t0,t2,enemyApplyHeal

enemyApplyDmg:
#print and apply damage
jal ra applyDmg
jal ra printEnemyDmg
jal ra endBattleCheck

```

```

#increment round count
addi s3,s3,1
#branch to end battle check
b userTurn
enemyApplyHeal:
jal ra applyHeal
jal ra printEnemyHeal
jal ra endBattleCheck
addi s3,s3,1
b userTurn

userWinsBattle:

li a7,4
la a0,newLine
ecall
#print "You defeated the enemy [enemy name]"
li a7,4
la a0,playerWinsStr
ecall

#get enemy name pointer using offset 12
lw a0,12($1)
ecall

li a7,4
la a0,newLine
ecall

#reset round counter
li s3,0
#increment battle counter
addi s4,s4,1
li t1,5
#check if five battles have happened
beq t1,s4,bossBattle
#check if six battles have happened
li t1,6
beq t1,s4,endGameWin

#else

li t2,3
jal getRandom
b spawnEnemy

userLosesBattle:
li a7,4
la a0,newLine
ecall

```

```
#print "You were defeated by the enemy [enemy name]"
li a7,4
la a0,playerLosesStr
ecall

#get enemy name pointer using offset 12 of
lw a0,12($1)
ecall

li a7,4
la a0,newLine
ecall

b endGameLoss
```

Added functions:

bossBattle:

```
la t1, slime
la t6, s1
jal ra, copyFromArray
b userTurn

b userTurn
```

endGameWin:

```
output to system "Well done you won the game"
b exit
```

endGameLoss:

```
output to system "Well done you lost the game"
b exit
```

Implementation of Pseudocode:

```
userWinsBattle:

li a7,4
la a0,newLine
ecall
#print "You defeated the enemy [enemy name]"
li a7,4
la a0,playerWinsStr
ecall

#get enemy name pointer using offset 12
lw a0,12($1)
ecall|
```

```
li a7,4
la a0,newLine
ecall

#reset round counter
li s3,0
#increment battle counter
addi s4,s4,1
li t1,5
#check if five battles have happened
beq t1,s4,bossBattle
```

```
564 #check if six battles have happened
565 li t1,6
566 beq t1,s4,endGameWin
567
568 #else
569
570 li t2,3
571 jal ra getRandom
572 b spawnEnemy
573
574 userLosesBattle:
575 li a7,4
576 la a0,newLine
577 ecall
578
579 #print "You were defeated by the enemy [enemy name]"
580 li a7,4
581 la a0,playerLosesStr
582 ecall
583
584 #get enemy name pointer using offset 12 of
585 lw a0,12($1)
586 ecall
587
588 li a7,4
```

```

588 li a7,4
589 la a0,newLine
590 ecall
591
592 #exit program
593 b endGameLoss
594 #
#-----#
#functions for boss battle and game win/loss

bossBattle:

la t1,slime
mv t6,sl
jal ra copyFromArray
b userTurn

endGameWin:

li a7,4
la a0,endWinStr
ecall
b exit

endGameLoss:

li a7,4
la a0,endLossStr
ecall
b exit

```

Data segment:

```

145 #Enemy arrays
146 goblin: .word 1, 3, 1, goblinStr
147 golem: .word 3, 1, 1, golemStr
148 witch: .word 1, 1, 3, witchStr
149 slime: .word 3,3,1, slimeStr
150
48 #Character and enemy strings
49 knightStr: .string "Knight"
50 musketeerStr: .string "Musketeer"
51 wizardStr: .string "Wizard"
52 goblinStr: .string "Goblin"
53 golemStr: .string "Golem"
54 witchStr: .string "Witch"
55 slimeStr: .string "Boss Slime"

endWinStr: .string "Well done you won the game"
endLossStr: .string "Well done you lost the game"

```

Comments:

With the checks added to userWinsBattle to spawn the slime boss after 5 battles and end the game after 6, the game will now: take the user through a battle, check the round count variable and spawn a new enemy/boss or end the game if the battle is lost and repeat this cycle until the boss or player is defeated.

2.8.4 Testing

Testing the RPG:

To test the completed the game, the following sections will be checked:

1. Initialisation and Start Menu
2. Build/Character Selection
3. Enemy Spawning
4. Actions
5. Player wins/loses 1 and 5 battles
6. Player wins/loses boss fight
7. Game Balance
8. User Experience

To allow for full image sizes, in each testing stage, one table will be used for text and another table will be used for the related image using the Test I.D.

1. Testing Initialisation and Start Menu:

Initialisation

Test table:

Test I.D	Test	Expected Result	Actual Result	Comment	Fail/pass
1.1	16 bytes of memory is allocated for currentEnemy and currentPlayer data.	In lables tab, currentEnemy and currentPlayer are allocated 16 bytes with no overlap.	currentEnemy is shown in label tab at 0x1003a4 with currentPlayer stored at 0x1003b4. The difference is 16 bytes (0x10).	-	Pass
1.2	Round count (s3) variable and Battle count variable (s4) are set to 0	Value 0 is stored in s3 and s4	Value 0 stored in s3 and s4.	-	Pass
1.3	currentPlayer and currentEnemy array pointers are stored in s0 and s1	Hex address for currentPlayer and currrentEnemy arrays are stored in registers s0 and s1	Pointer to start of currentPlayer array “0x100103a4” is stored in s0, pointer to currentEnemy array “0x100103b4” is stored in s1.	As seen in the images for 1.3, the currentPlayer health value is stored at 0x100103a4, ending at 0x100103ac	Pass

			with the pointer to the string of currentPlayer. The currentEnemy health value is stored at 0x100103b4 with the pointer to the string being stored at 0x100103c0.	
--	--	--	---	--

Image table:

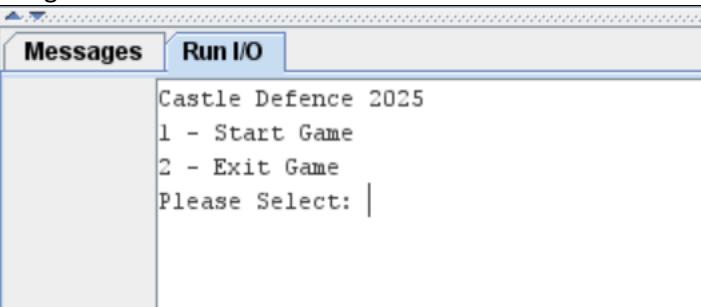
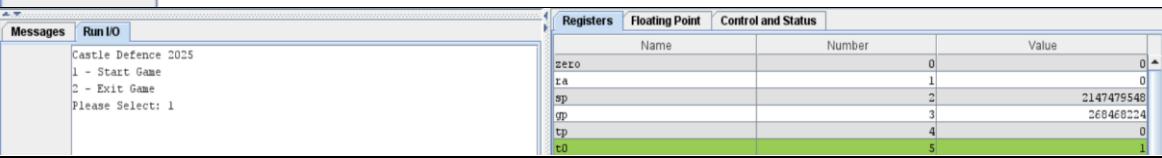
Test I.D	Images																								
1.1	currentPlayer 0x100103a4 currentEnemy 0x100103b4																								
1.2	<p>The screenshot shows the assembly code for the Text Segment. It includes instructions like la s0, currentPlayer, la s1, currentEnemy, addi x19, x0, 0, addi x20, x0, 0, addi x17, x0, 4, addi x10, x0, -19, and ecall. The registers pane shows values for sp, gp, tp, t0, t1, t2, s0, s1, a0 through a5, and s2 through s4. The memory pane shows pointers s0 and s1 pointing to 0x100103a4 and 0x100103b4 respectively.</p>																								
1.3	Pointers in Registers s0 and s1: <table border="1"><tr><td>s0</td><td>8</td><td>0x100103a4</td></tr><tr><td>s1</td><td>9</td><td>0x100103b4</td></tr></table> currentEnemy and currentPlayer in data segment: <table border="1"><tr><td>0x100103a0</td><td>268501087</td><td>50</td><td>60</td><td>40</td><td>268501051</td><td>150</td><td>20</td><td>20</td></tr><tr><td>0x100103c0</td><td>268501075</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	s0	8	0x100103a4	s1	9	0x100103b4	0x100103a0	268501087	50	60	40	268501051	150	20	20	0x100103c0	268501075	0	0	0	0	0	0	0
s0	8	0x100103a4																							
s1	9	0x100103b4																							
0x100103a0	268501087	50	60	40	268501051	150	20	20																	
0x100103c0	268501075	0	0	0	0	0	0	0																	

Start Menu

Test table:

Test I.D	Test	Expected Result	Actual Result	Comment	Fail/pass
1.4	Menu strings output to console correctly	Output to system should be: Castle Defence 2025 1 - Start Game 2 - Exit Game Please Select:	Correct output to system as seen in image 1.4	-	Pass
1.5	User input is stored in t0	Value "1" should be stored in t0	Value 1 stored in t0	-	Pass
1.6	User input 1 branches to chooseCharacter, user input 2 branches to exit	User input 1 should branch to chooseCharacter, input 2 should exit program	Branching happened correctly in both test cases	-	Pass
1.7	User input of 3 (invalid) should output invalid input message and branch to main label	Invalid input string will output to console, the program will then loop to main label	Invalid input strings was output to console, then program looped to main	Add new line character to invalid input string	Pass
1.8	User input is a non integer input	Program should crash.	Program crashed	Add more robust input validation	Pass

Image table:

Test I.D	Images
1.4	
1.5	
1.6	User input 1:

	<p>Messages Run I/O</p> <pre>Castle Defence 2025 1 - Start Game 2 - Exit Game Please Select: 1 Choose your build 1 - Knight Attributes: Health Damage Healing 3 2 1 2 - Musketeer Attributes: Health Damage Healing 1 3 2 3 - Wizard Attributes: Health Damage Healing 2 1 3 Please Select: </pre>
User input 2:	<p>Castle Defence 2025</p> <p>1 - Start Game</p> <p>2 - Exit Game</p> <p>Please Select: 2</p> <p>-- program is finished running (0) --</p>
1.7	<p>Castle Defence 2025</p> <p>1 - Start Game</p> <p>2 - Exit Game</p> <p>Please Select: 3</p> <p>Invalid input</p> <p>Castle Defence 2025</p> <p>1 - Start Game</p> <p>2 - Exit Game</p> <p>Please Select: </p>
1.8	<p>Run I/O</p> <p>Castle Defence 2025</p> <p>1 - Start Game</p> <p>2 - Exit Game</p> <p>Please Select: a</p> <p>line 184: Runtime exception at 0x00400050: invalid integer input (syscall 5)</p>

2. Build/Character Selection

Test table:

Test I.D	Test	Expected Result	Actual Result	Comment	Fail/pass
2.1	User input 1	Pointer address for knight array is moved to t1, pointer address for currentPlayer from s0 is moved to t6.copyFromArray will copy data all data from knight array to currentPlayer array.	Pointers stored correctly and character array copied successfully.	-	Pass
2.2	User input 2	Pointer address for musketeer array is moved to t1, pointer address for currentPlayer from s0 is moved to t6. copyFromArray will copy data all data from knight array to currentPlayer array.	Pointers stored correctly and character array copied successfully.	-	Pass
2.3	User input 3	Pointer address for wizard array is moved to t1, pointer address for currentPlayer from s0 is moved to t6.copyFromArray will copy data all data from knight array to currentPlayer array.	Pointers stored correctly and character array copied successfully.	-	Pass
2.4	User input 4	Invalid input string should output to console, then program should loop to chooseCharacter	Invalid input string output to console program then loops	-	Pass
2.5	User input non integer	Program should crash	Program crashes	Further input validation necessary.	Fail

Image table:

Test I.D	Image						
2.1	<p>Pointer to knight array in t1 and pointer to currentPlayer in t6:</p> <p>Knight array copied to currentPlayer array.</p> <table border="1"> <tr><td>0x100103a0</td><td>268501087</td><td>150</td><td>40</td><td>20</td><td>268501044</td></tr> </table>	0x100103a0	268501087	150	40	20	268501044
0x100103a0	268501087	150	40	20	268501044		
2.2	<p>Pointer to musketeer array in t1 and pointer to currentPlayer in t6:</p> <p>Musketeer array copied to currentPlayer array:</p> <table border="1"> <tr><td>0x100103a0</td><td>268501087</td><td>50</td><td>60</td><td>40</td><td>268501051</td></tr> </table>	0x100103a0	268501087	50	60	40	268501051
0x100103a0	268501087	50	60	40	268501051		
2.3	<p>Pointer to wizard array stored in t6 and currentPlayer stored in t1:</p>						

Registers Floating Point Control and Status

Name	Number	Value
sp	3	0x10000000
gp	4	0x00000000
tp	5	0x00000000
t0	6	0x10010354
t1	7	0x00000000
s0	8	0x10010344
si	9	0x100103b4
a0	10	0x00000000
ai	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000005
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x10010344
pc		0x0040057c

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400558	0xdf030313	addi x6,x6,0xfffffd0	
	0x0040055c	0x0000fb3	add x31,x0,x8	731: mv t6,s0
	0x00400560	0x6ffff0ef	jal x1,0xffffffffe8	732: jal ra copyFromArray
	0x00400564	0x00300393	addi x7,x0,3	734: li t2,3
	0x00400568	0xf9dff0ef	jal x1,0xffffffff9c	735: jal ra getRandom
	0x0040056c	0xfadff0ef	jal x0,0xfffffffffa	736: b spawnEnemy
	0x00400570	0x0fc10317	auipc x6,0x00000fc10	739: la t1,wizard
	0x00400574	0xde430313	addi x6,x6,0xfffffdde4	
	0x00400578	0x00800fb9	add x31,x0,x8	740: mv t6,s0
	0x0040057c	0xf4dff0ef	jal x1,0xffffffff4c	741: jal ra copyFromArray

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x100102c0	0x6cc16568	0xa0a6a874	0x75ef5900	0x6f742072	0x20ec6174	0x6c616568	0x69206874	0x59002073
0x100102e0	0x7720756f	0x20657265	0x6c16564	0x20002074	0x75ef6d5e	0x68200020	0x65ef6d5e	0x20002072
0x10010300	0xc0796c20	0x6de56e65	0x45000a79	0x796d65e6	0x68200020	0x65ef61e5	0x6fe6e2064	0x65761e6d
0x10010320	0x75ef6d61	0xef20746e	0x5682066	0xee96cc1	0x000000a7	0x00000003	0x00000002	0x00000001
0x10010340	0x10010034	0x00000001	0x00000003	0x00000002	0x1001003b	0x00000002	0x00000001	0x00000003
0x10010360	0x10010045	0x00000001	0x00000003	0x00000001	0x1001004c	0x00000003	0x00000001	0x00000001
0x10010380	0x10010053	0x00000001	0x00000001	0x00000003	0x10010059	0x00000003	0x00000003	0x00000001
0x100103a0	0x1001005f	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Labels

Label	Address
enemyRealtStr3	0x1001031f
knight	0x10010344
musketeer	0x10010344
wizard	0x10010354
goblin	0x10010364
golem	0x10010374
witch	0x10010384
slime	0x10010394
currentPlayer	0x100103a4
currentEnemy	0x100103b4

Messages Run I/O

Wizard array copied to currentPlayer array:

0x100103a0	268501087	100	20	60	268501061
------------	-----------	-----	----	----	-----------

2.4

Messages Run I/O

```

l - Knight
Attributes: Health Damage Healing
      3      2      1

2 - Musketeer
Attributes: Health Damage Healing
      1      3      2

3 - Wizard
Attributes: Health Damage Healing
      2      1      3

Please Select: 4
Invalid input

Choose your build

1 - Knight
Attributes: Health Damage Healing
      3      2      1

2 - Musketeer
Attributes: Health Damage Healing
      1      3      2

3 - Wizard
Attributes: Health Damage Healing
      2      1      3

Please Select:

```

2.5 line 184: Runtime exception at 0x00400050: invalid integer input (syscall 5)

3 - Enemy Spawning

Test Table:

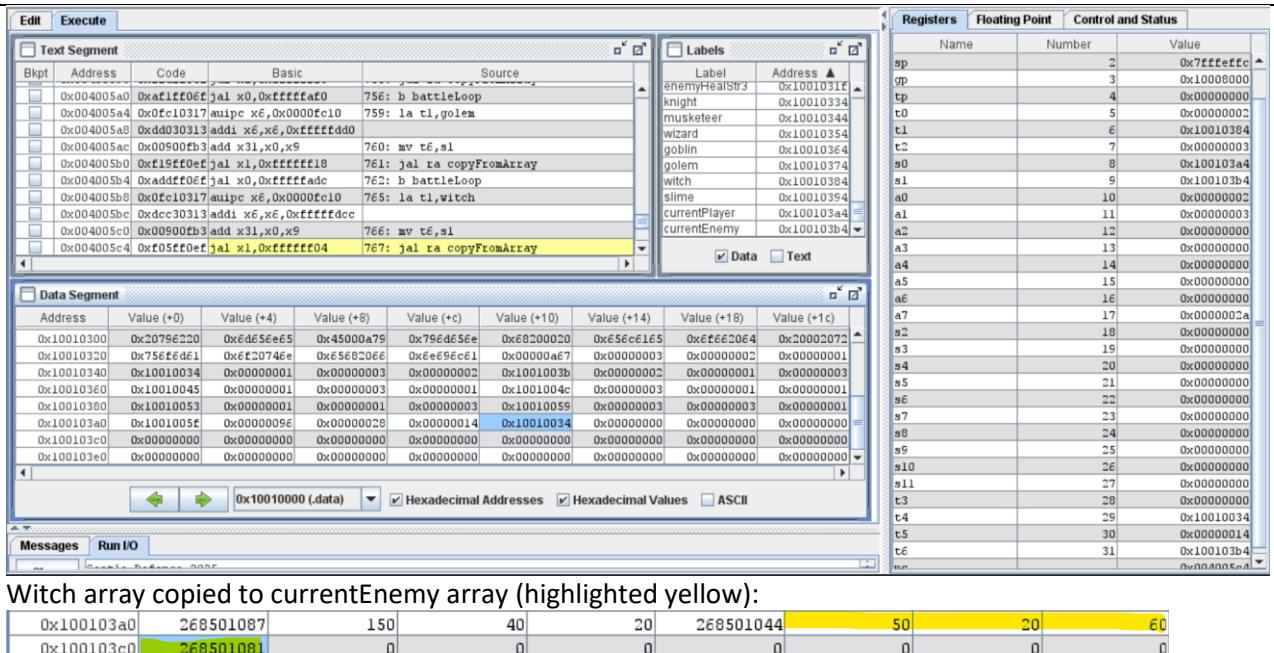
Test I.D	Test	Expected Result	Actual Result	Comment	Fail/pass
3.1	Goblin spawns	Random generation output 0 branches to selectGoblin.	Program functioned as expected.		Pass

		Pointer to goblin array is loaded to t1, currentEnemy pointer is loaded to t6. copyFromArray copies data in goblin array to current enemy array.			
3.2	Golem spawns	Random generation output 1 branches to selectGolem. Pointer to golem array is loaded to t1, currentEnemy pointer is loaded to t6. copyFromArray copies data in golem array to current enemy array.	Program functioned as expected.		Pass
3.3	Witch spawns	Random generation output 2 branches to selectWitch. Pointer to witch array is loaded to t1, currentEnemy pointer is loaded to t6. copyFromArray copies data in witch array to current enemy array.	Program functioned as expected.		Pass
3.4	Random number is invalid	Branch to getRandom will initiate	Program functioned as expected.	This input validation will leave the game in an infinite loop.	Pass

Image Table:

Test	Image

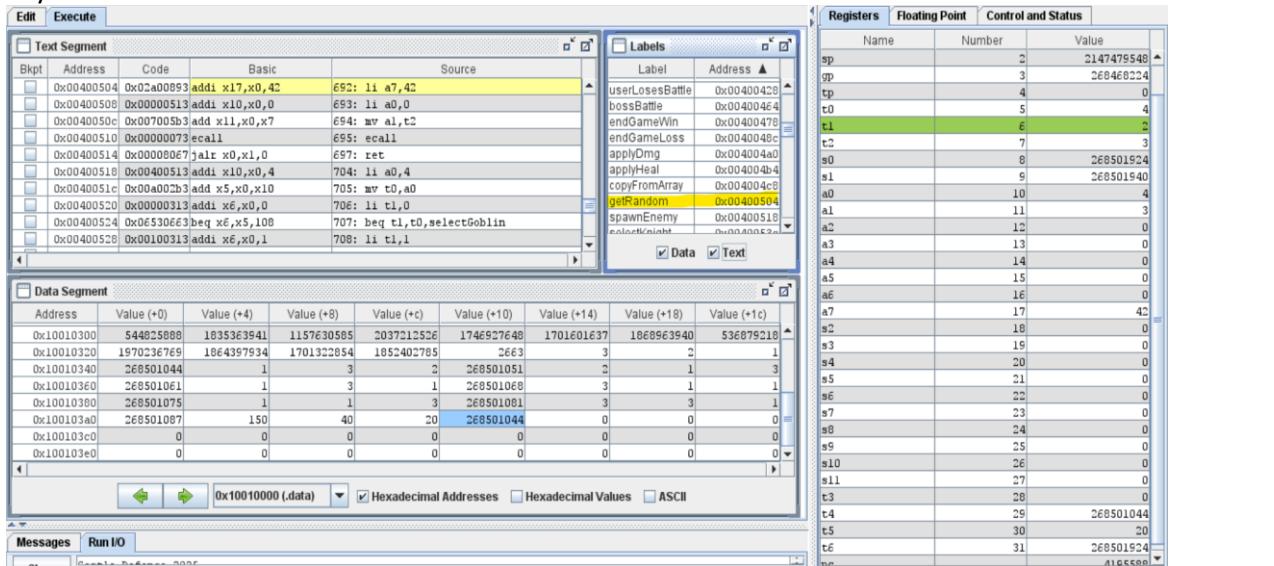
I.D																			
3.1	<p>Pointer to goblin array stored in t1, pointer to currentEnemy stored in t6:</p> <p>Goblin array copied to currentEnemy array (highlighted yellow):</p> <table border="1"> <tr> <td>0x100103a0</td> <td>268501087</td> <td>150</td> <td>40</td> <td>20</td> <td>268501044</td> <td>50</td> <td>60</td> <td>20</td> </tr> <tr> <td>0x100103c0</td> <td>268501088</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> </table>	0x100103a0	268501087	150	40	20	268501044	50	60	20	0x100103c0	268501088	0	0	0	0	0	0	0
0x100103a0	268501087	150	40	20	268501044	50	60	20											
0x100103c0	268501088	0	0	0	0	0	0	0											
3.2	<p>Pointer to golem array stored in t1, pointer to currentEnemy stored in t6:</p> <p>Golem array copied to currentEnemy array (highlighted yellow):</p> <table border="1"> <tr> <td>0x100103a0</td> <td>268501087</td> <td>150</td> <td>40</td> <td>20</td> <td>268501044</td> <td>150</td> <td>20</td> <td>20</td> </tr> <tr> <td>0x100103c0</td> <td>268501075</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> </table>	0x100103a0	268501087	150	40	20	268501044	150	20	20	0x100103c0	268501075	0	0	0	0	0	0	0
0x100103a0	268501087	150	40	20	268501044	150	20	20											
0x100103c0	268501075	0	0	0	0	0	0	0											
3.3	<p>Pointer to witch array stored in t1, pointer to currentEnemy stored in t6:</p>																		



Witch array copied to currentEnemy array (highlighted yellow):

0x100103a0	268501087	150	40	20	268501044	50	20	60
	0x100103c0	268501081	0	0	0	0	0	0

- 3.4 4 is stored in getRandom, which initiates jump to getRandom label. getRandom label address highlighted in yellow:



4 –Actions

For testing player and enemy actions, musketeer and golem are used.

Golem Attributes: 150 health, 20 damage, 20 healing

Musketeer Attributes: 50 health, 60 damage, 40 healing

Test Table:

Test I.D	Test	Expected Result	Actual Result	Comment	Fail/pass
4.1	Player (musketeer) attacks enemy (Golem)	User input 1 is stored in t0, program then branches to useApplyDamage.	Program functioned as described in expected result	-	Pass

		The health value inside currentEnemy will equal 110, branch to enemyTurn is then made.			
4.2	Player (musketeer) heals	User input 2 is stored in t0, program then branches to useApplyHealth. The health value inside currentPlayer will equal 90.	Program functioned as described in expected result	-	Pass
4.3	Enemy (Golem) attack player (musketeer)	Random number 0 is moved to t0, program then branches to enemyApplyDmg. The health value inside currentPlayer will equal 30.	Program functioned as described in expected result	"You were dealt XX damage by enemy" string needs to be edited to print enemy name string after "enemy"	Pass
4.4	Enemy (Golem) heals	Random number 1 is moved to t0, program then branches to enemyApplyHeal. The health value inside currentEnemy will equal 170	Program functioned as described in expected result	-	Pass
4.5	User input is non integer	Program crashes	Program crashed	Robust input validation needed	Fail
4.6	Player gets attacked for more than remaining health	Game should branch to userLosesBattle	Program functioned as expected.	-	Pass
4.7	Enemy gets attacked for more than remaining health	Game should branch to userWinsBattle and spawn new enemy.	Program functioned as expected	Add logic to set health to zero when it goes negative, as an enemy or player cannot have negative health	Pass

Test I.D	Image
----------	-------

4.1

Clear

A Golem HAS APPEARED

What will you do?

1 - Attack
 2 - Heal

Health: 50

Please Select: 2

You healed for 40 amount of health

currentPlayer array:

The screenshot shows a debugger interface with three main panes:

- Registers** pane (left): A table of memory addresses and their corresponding values.
- Stack** pane (middle): A table of memory addresses and their corresponding assembly instructions.
- Data Segment** pane (bottom): A table of memory addresses and their corresponding values.

Registers (Left Pane):

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x10010380	268501075	1	1	3	268501081
0x100103a0	268501087	90	60	40	268501051

Stack (Middle Pane):

Source
4: li a7,4
5: la a0,newLine
6: ecall
7: li a7,4
8: la a0,userHealStr1
9: ecall
10: li a7,1
11: lw a0,8(\$0)
12: ecall
13: li a7,4
14: la a0, userHealStr2
15: ecall
16: li a7,1
17: lw a0,8(\$0)
18: ecall
19: li a7,4
20: la a0,userHealStr3

Data Segment (Bottom Pane):

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x10010380	268501075	1	1	3	268501081
0x100103a0	268501087	90	60	40	268501051

4.2	<p>A Golem HAS APPEARED</p> <p>Clear</p> <p>What will you do?</p> <p>1 - Attack 2 - Heal</p> <p>Health: 50</p> <p>Please Select: 1</p> <p>You dealt 60 amount of damage</p> <p>Enemy Golem has 90 health left</p> <p>currentEnemy (highlighted):</p> <table border="1" data-bbox="219 646 1349 725"> <thead> <tr> <th>Address</th><th>Value (+0)</th><th>Value (+4)</th><th>Value (+8)</th><th>Value (+c)</th><th>Value (+10)</th><th>Value (+14)</th><th>Value (+18)</th><th>Value (+1c)</th></tr> </thead> <tbody> <tr> <td>0x10010380</td><td>268501075</td><td>1</td><td>1</td><td>3</td><td>268501081</td><td>3</td><td>3</td><td>1</td></tr> <tr> <td>0x100103a0</td><td>268501087</td><td>30</td><td>60</td><td>40</td><td>268501051</td><td>90</td><td>20</td><td>20</td></tr> </tbody> </table>	Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)	0x10010380	268501075	1	1	3	268501081	3	3	1	0x100103a0	268501087	30	60	40	268501051	90	20	20
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)																				
0x10010380	268501075	1	1	3	268501081	3	3	1																				
0x100103a0	268501087	30	60	40	268501051	90	20	20																				
4.3	<p>You were dealt 20 amount of damage by enemy</p> <p>currentPlayer:</p> <table border="1" data-bbox="219 815 1349 871"> <tr> <td>0x100103a0</td><td>268501087</td><td>30</td><td>60</td><td>40</td><td>268501051</td></tr> </table>	0x100103a0	268501087	30	60	40	268501051																					
0x100103a0	268501087	30	60	40	268501051																							
4.4	<p>Enemy Golem healed for 20 amount of healing</p> <p>currentEnemy:</p> <table border="1" data-bbox="219 938 1349 983"> <tr> <td>170</td><td>20</td><td>20</td></tr> </table>	170	20	20																								
170	20	20																										
4.5	<p>line 247: Runtime exception at 0x004000a4: invalid integer input (syscall 5)</p>																											
4.6	<p>What will you do?</p> <p>1 - Attack 2 - Heal</p> <p>Health: 10</p> <p>Please Select: 2</p> <p>You healed for 40 amount of health</p> <p>Your total health is 50</p> <p>You were dealt 60 amount of damage by enemy</p> <p>You were defeated by the enemy Goblin</p> <p>Well done you lost the game</p>																											

4.7	<p>What will you do?</p> <p>1 - Attack 2 - Heal</p> <p>Health: 50</p> <p>Please Select: 1</p> <p>You dealt 60 amount of damage</p> <p>Enemy Golem has -30 health left</p> <p>You defeated the enemy Golem</p>
-----	---

5 - Player wins/loses 1 and 5 battles

Test I.D	Test	Expected Result	Actual Result	Comment	Pass/Fail
5.1	Player wins 1 battle	playerWinsStr will be output to console, the round counter in s3 should be reset. The battle counter in s4 should be incremented by 1 to 1. Another battle should be initiated.	Program behaved as described.	-	Pass.
5.2	Enemy wins 1 battle	playerLosesStr will be output to console, which will then branch to endGameLoss which prints endLossStr to console and ends the program.	Program behaved as described.	Insert new line character after "You were defeated by enemy Goblin"	Pass
5.3	Player wins 5 battles	userWinsBattle will branch to bossBattle which sets currencyEnemy to "Boss Slime". The battle loop functions as normal.	Program behaved as described.	-	Pass

Test I.D	Image
5.1	<p>playerWinsStr: A Witch HAS APPEARED</p> <p>What will you do?</p> <p>1 - Attack 2 - Heal</p> <p>Health: 50</p> <p>Please Select: 1</p> <p>You dealt 400 amount of damage</p> <p>Enemy Witch has -350 health left</p> <p>You defeated the enemy Witch</p> <p>Next battle initiated: A Golem HAS APPEARED</p> <p>What will you do?</p> <p>1 - Attack 2 - Heal</p> <p>Health: 50</p> <p>Please Select: </p> <p>Values in registers:</p>

Registers	Floating Point	Control and Status	
Name	Number	Value	
zero	0	0	
ra	1	4194464	
sp	2	2147479548	
gp	3	268468224	
tp	4	0	
t0	5	1	
t1	6	50	
t2	7	150	
s0	8	268501924	
s1	9	268501940	
a0	10	268501373	
a1	11	3	
a2	12	0	
a3	13	0	
a4	14	0	
a5	15	0	
a6	16	0	
a7	17	5	
s2	18	0	
s3	19	0	
s4	20	1	
s5	21	0	
s6	22	0	
s7	23	0	
s8	24	0	
s9	25	0	
s10	26	0	
s11	27	0	
t3	28	-350	
t4	29	268501075	
t5	30	20	

5.2 What will you do?

1 - Attack
2 - Heal

Health: 10

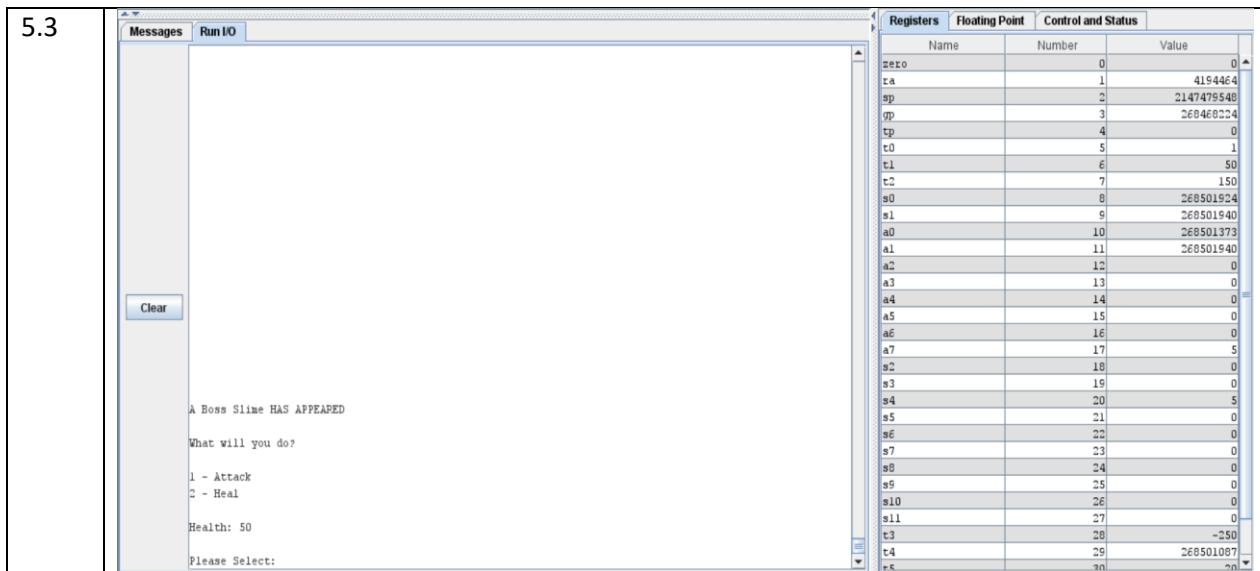
Please Select: 2

You healed for 40 amount of health

Your total health is 50

You were dealt 60 amount of damage by enemy

You were defeated by the enemy Goblin
Well done you lost the game
-- program is finished running (0) --



6 - Player wins/loses boss fight

Test I.D	Test	Expected Result	Actual Result	Comment	Pass/Fail
6.1	Player wins boss fight	battleLoop branches to userWinsBattle. userWinsBattle uses beq comparison to check if this is the sixth battle, this is true, it then branches to endGameWin which prints the endWinStr to console and branches to exit.	Program behaved as expected	Tidy strings for better user experience	Pass
6.2	Player loses boss fight	battleLoop branches to userLosesBattle, which branches to endGameLoss. the endLossStr is output to the console and branches to exit.	Program behaved as expected	Tidy strings for better user experience	Pass

Test I.D	Image

6.1	<pre>A Boss Slime HAS APPEARED What will you do? 1 - Attack 2 - Heal Health: 50 Please Select: 1 You dealt 60 amount of damage Enemy Boss Slime has 90 health left You were dealt 60 amount of damage by enemy You were defeated by the enemy Boss Slime Well done you lost the game -- program is finished running (0) --</pre>
6.2	<pre>What will you do? 1 - Attack 2 - Heal Health: 50 Please Select: 1 You dealt 60 amount of damage Enemy Boss Slime has -30 health left You defeated the enemy Boss Slime Well done you won the game -- program is finished running (0) --</pre>

7 – Balancing

Turns to kill table

To address game balancing in a mathematical manor, Google Gemini 2.5 was used to uncover the turns to kill method. One standard method game developers use in modern, turn-based RPG games is using a “turns to kill” formula. In this RPG there are three factors to calculating turns to kill:

1. Target Health
2. Attacker Damage
3. Target Heals

Considering the probability of a player attacking or healing is 50%, this formula was used to determine the actual average turns to kill.

$$\text{Turns to kill} = (\text{Target Health}/\text{Attacker Damage})/0.5$$

Turns to kill table:

Player character	Enemy character	Player Health	Player Damage	Enemy Health	Enemy Damage	Turns to kill player	Turns to kill enemy
Knight	Goblin	150	40	50	60	5.0	2.5
Knight	Golem	150	40	150	20	15.0	7.5
Knight	Witch	150	40	50	20	15.0	2.5
Knight	Slime	150	40	150	60	5.0	7.5
Musketeer	Goblin	50	60	50	60	1.6	1.6
Musketeer	Golem	50	60	150	20	5.0	5.0
Musketeer	Witch	50	60	50	20	5.0	1.6
Musketeer	Slime	50	60	150	60	1.6	5.0
Wizard	Goblin	100	20	50	60	3.3	5.0
Wizard	Golem	100	20	150	20	10.0	15.0
Wizard	Witch	100	20	50	20	10.0	5.0
Wizard	Slime	100	20	150	60	3.3	15.0

Comments:

The knight has an extremely high turns to kill compared to the rest of the characters indicating his high health stat mixed with medium damage is overpowered, the turns to kill enemy seems balanced.

The musketeer has an extremely low TTK on average (3.3) indicating the health stat is too low, the turns to kill enemy is also high indicating the enemies require slightly more health, specifically the goblin and witch which die in one hit.

The wizard has a fairly high turns to kill, indicating his health and healing are balanced however his turns to kill enemy are far higher on average indicating he needs a small damage buff.

The goblin has an extremely low average TTK and an extremely high TTK enemy which indicates it requires a small health buff to survive more than one hit.

The golem, like the knight, has an extremely high and a low TTK enemy on average. This shows the golem needs a health reduction and slight damage increase.

The witch is completely underpowered compared to all characters, requiring a health and damage increase to be a viable enemy.

The slime is completely overpowered, with low TTK's and high TTK enemies. A health and damage reduction is needed to bring the slime to a balanced position.

Implementation of findings

The current multiplication of attributes system does not create a fair environment, to address this, in the next stage a base value for each character will be implemented. For example, the wizard has a base of 100 health (like every other character), the health attribute of 2 will add 20 points to his health bringing each character closer together but

still giving a reason for the user to choose a different build.

8 – User experience

To test for user experience, the game will be played 15 times total (5 plays for each character). The following criteria will be marked 1(poor) to 10(amazing) for each character, there will also be non-character specific criteria to be marked at the end of the 15 games.

Character specific:

1. Does the character gameplay suit the character name (does the musketeer feel high damage low defence)?
2. Do you always get a fair chance to win the battle (no one shot kills)?
3. Does healing feel useful?
4. Does the character have a fair balance between its strong and weak stat e.g the musketeer is high dps but also just enough health to stay alive?
5. Is the boss fight winnable?
6. Does each turn feel useful?
7. Does each battle feel different/engaging?
8. Is it possible to recover from an early heavy hit?
9. Is the character fun?

Non character specific:

1. Is the menu easy to use?
2. Is the character selection screen clear?
3. Are strings easy to read? Are strings displayed for long enough
4. Does the random enemy spawning and turn feel random?
5. Do battles flow well?
6. Is the game difficult enough?
7. Do attributes feel equally as powerful or is one clearly better than the rest?
8. Is the game replayable?

Knight

Criteria	Rating
Gameplay suit name?	5 – felt tanky
Fair chance to win?	8 – unfair to enemy
Is healing useful?	4 – didn't have to use often as high health
Fair balance between strength and weakness?	5 – felt like weakness was too strong
Boss fight winnable?	1 – no
Does each turn feel important?	2 – especially against golem
Each battle feel different/engaging?	1 – no
Able to recover from a big hit?	5 – yes, too easily
Fun?	3 – not really

Musketeer

Criteria	Rating
Gameplay suit name?	5 – very high damage, too overpowered
Fair chance to win?	1 – too overpowered
Is healing useful?	7
Fair balance between strength and weakness?	1
Boss fight winnable?	10 – won all boss fights
Does each turn feel important?	2 – damage too high
Each battle feel different/engaging?	1 – just spammed attacks
Able to recover from a big hit?	5 – too easy
Fun?	5 – just spammed attack

Wizard

Criteria	Rating
Gameplay suit name?	1 – doesn't feel very magical
Fair chance to win?	1 – didn't win a game
Is healing useful?	7 – very high healing stat
Fair balance between strength and weakness?	1 – too little damage
Boss fight winnable?	1 – no
Does each turn feel important?	5 – yes but because underpowered
Each battle feel different/engaging?	5
Able to recover from a big hit?	1 – too little damage
Fun?	5 – yes because underpowered

Non-Character Specific Table

Criteria	Rating
Menu easy to use?	8
Character selection clear?	9
Are strings easy to read and well timed?	7 – add press enter to continue at end of a battle Health string when enemy appears is not obvious the players health is being shown

	A Boss Slime HAS APPEARED What will you do? 1 - Attack 2 - Heal Health: 30 Please Select: 2 You healed for 20 amount of health Your total health is 50 You were dealt 60 amount of damage by enemy
Does random feel random?	10
Do battle flow well?	8 – will add press enter to continue
Is the game difficult enough?	4 – too easy to kill normal enemies, boss fight too hard
Do attributes feel balanced or is one clearly the best?	4 – healing feels underpowered
Is the game replayable?	5 – yes because quick, no because boring

Conclusion of Testing

The core mechanics of the game work perfectly, however there are many balancing and user experience issues. This is how I plan to address these issues highlighted during testing.

List of amendments to be made in next section:

Game Mechanics

1. Change attribute balancing using base stat + multiplier.
2. Nerf the slime boss.
3. Add an XP system that increases attributes over time.
4. Buff healing.
5. Add a min/max to healing.
6. Add a “Dodge” action, when used attack will miss.

User Experience

1. Make all strings more immersive.
2. Add press c to continue, so user can move at their own pace.
3. Make the wizard strings feel more like a wizard.
4. Change battle screen to be clearer and more immersive.
5. Input validation

2.8.5 Final iteration of the RPG – Game Mechanics and User Experience

Game Mechanics

List of amendments plan:

1. Change attribute balancing using base stat + multiplier.

Using the TTK formula ($\text{Turns to kill} = (\text{Target Health}/\text{Attacker Damage})/0.5$), we get the desired “Target health” and “Attacker Damage”. If we set TTK = 5, Target Health/Attacker Damage = 2.5; meaning, the Health stat needs to be 2.5x the attacker damage for an average TTK of 5.

Base Stats:

Health 100 (multiplier by point = 10) max health stat 130

Damage 40 (multiplier by point = 4) max damage stat 52

Health 20 (multiplier by point = 6) max healing stat 38

With these stats, the game is well balanced, no character can get one shot killed or heal an entire attacker’s damage from one shot.

2. Nerf the slime boss.

The slime boss is almost impossible to beat for most builds, due to his high damage and health combined. To address this the slime will now be a tanky, low damage boss. The slime will now have a health attribute of 4, damage attribute of 1 and healing attribute of 2.

3. Add an XP system that increases attributes over time.

When a player kills an enemy, they will get 2 XP, when 3 XP is reached the player will be able to upgrade an attribute of their choice. The amount of XP needed to level up will increase by 1 XP per level. Player XP will be stored in s2.

4. Buff healing.

Healing is now more in line with the rest of the attributes and should provide more value to the user.

5. Add a min/max to healing.

A max health stat will be stored in the currentEnemy and player arrays. When healing is used, if player health is greater than max health, player health will be set to max health. When a target takes damage, if their health is less than zero their health will be set to zero.

6. Add a “Dodge” action, when used attack will miss.

User input 3 will use the dodge action, when used if the enemy attacks the attack will deal zero damage. It has a 3-round cooldown.

User Experience

1. Make all strings more immersive.

Strings are currently basic; all strings will be reworked to give the game a more polished feel. Also add strings to display new features like XP.

2. Add press c to continue, so user can move at their own pace.

At given points in the game i.e after a battle, the program will ask for user input to continue.

3. Change battle screen to be clearer.

Battle screen layout and strings will be reworked to narrate the game better alongside providing the player with more and clearer information.

4. Input validation

To check for valid input without crashing the game, we will take user input as an ASCII value and convert to its actual value instead of using the readint syscall.

Final Iteration:

Game Mechanics

1 – Attribute Balancing

new copyFromArray:

```

copyFromArray:

#t1 is pointer to selected character array
#t6 is array to be copied to

#load health multiplier
li t5,10
#copy health and save in currentXX array (s0)
lw t4,0(t1)
mul t4,t5,t4
#add base stat
addi t4,t4,100
sw t4,0(t6)

#load damage multiplier
li t5,4
#copy damage and save in currentXX array (s0)
lw t4,4(t1)
mul t4,t5,t4
#add base stat
addi t4,t4,40
sw t4,4(t6)

#load healing multiplier
li t5,6
#copy healing and save in currentXX array (s0)
lw t4,8(t1)
mul t4,t5,t4
#add base stat
addi t4,t4,20
sw t4,8(t6)

```

Attributes in currentPlayer after Musketeer selection:

0x100103a0	268501087	110	52	32	268501051
------------	-----------	-----	----	----	-----------

2 – Nerf slime boss

```
slime: .word 4, 1, 2, slimeStr
```

3 – XP System

To implement an XP system, 3 Variables were implemented.

S2 = Player xp variable
 S5 = Max xp variable
 S6 = Player Level

In userWinsBattle, the players XP is incremented by 2. The current XP is then checked against the max XP using the “bge” instruction, when current xp >= max xp the program branches to upgradeAttribute which deals with printing all strings and upgrading the attributes inside the currentPlayer array.

userWinsBattle:

```

#DEAL WITH XP
#add 2xp
addi $2,$2,2
#if player xp is greater than or equal or equal to max XP
bge $2,$5,upgradeAttribute

#label to return to after upgrade attribute
afterUpgrade:
#BOSS CHECKS

```

upgradeAttribute:

```

126 #-----
127 upgradeAttribute:
128
129 #add 1 to player level variable
130 addi $6,$6,1
131 #multiply max xp by 2
132 li $0,2
133 mul $5,$5,$0
134 #set xp to zero
135 mv $2,zero
136
137 #ask user what attribute they want to upgrade and branch
138
139 li $7,4
140
141 la $0,upgAttStr1
142 ecall
143
144 li $7,1
145 mv $0,$6
146 ecall
147
148 li $7,4
149
150 la $0,upgAttStr2
151 ecall
152
153 la $0,upgAttStr3
154 ecall
155
156 la $0,upgAttStr4
157 ecall
158
159 la $0,upgAttStr5
160 ecall
161
162 la $0,takeInput
163 ecall
164
165 li $7,5
166 ecall
167 #move user input to t0
168 mv $0,$0
169 #check input and branch appropriately
170 li $1,1
171 beq $1,$0,upgHealth
172 li $1,2
173 beq $1,$0,upgDmg

```

upgradeAttribute strings:

```
964 b upgradeAttribute
965
966 upgHealth:
967
968 #get player max health and add 10
969 addi $7,$7,10
970 #heal player to full
971 sw $7,0($0)
972 #print info to user
973 li $7,4
974 la $0,upgHealthStr
975 ecall
976
977 li $7,1
978 mv $0,$7
979 ecall
980
981 b afterUpgrade
982
983 upgDmg:
984 #get player damage and add 4
985 lw $3,4($0)
986 addi $3,$3,4
987 #save in currentPlayer array
988 sw $3,4($0)
989
990 #print info to user
991 li $7,4
992 la $0,upgDmgStr
993 ecall
994
995 li $7,1
996 lw $0,4($0)
997 ecall
998
```

```

b afterUpgrade

upgHeal:
#get player healing and add 6
lw t3,8($0)
addi t3,t3,6
#save in currentPlayer array
sw t3,8($0)
#print info to user
li a7,4
la a0,upgHealStr
ecall

li a7,1
lw a0,8($0)
ecall

b afterUpgrade

#upgradeAttribute strings

upgAttStr1: .string "\nYou have reached level "
upgAttStr2: .string "\n\nSelect an attribute to upgrade\n\n"
upgAttStr3: .string "1 - Health\n"
upgAttStr4: .string "2 - Damage\n"
upgAttStr5: .string "3 - Healing\n\n"

upgHealthStr: .string "Your health stat is now: "
upgDmgStr: .string "Your damage stat is now: "
upgHealStr: .string "Your healing stat is now: "

```

upgradeAttribute in run I/O:

```

You have reached level 1

Select an attribute to upgrade

1 - Health
2 - Damage
3 - Healing

Please Select: 2
Your damage stat is now: 56

```

Registers s2,s5,s6:

Register s2 has been set to 0 as the player has levelled up, Register s5 has been set to 4 the xp needed for level 2. S6 has been set to 1 as the player is now level 1.

s2	18	0
s3	19	0
s4	20	1
s5	21	4
s6	22	1
s7	23	0

5 – Min/Max healing

Implementing a minimum health of zero is achieved using a branch instruction to skip setting health to zero if health is more than zero.

```
applyDmg:
#get attacker damage stat
lw t2,4(a0)
#get target health
lw t3,0(al)
#subtract attack from health and update
sub t3,t3,t2
#deal with resulting health less than zero
li t6,0
#if 0 is less than health, continue
blt t6,t3,continueDmg
#set t3 to zero if health is below zero
li t3,0
continueDmg:
sw t3,0(al)
ret
```

Implementing a maximum health is achieved by storing player and enemy maximum health stats in registers s7 and s8. These values are passed to t4 when the enemy or player uses the healing/damage action.

```
applyHeal:
#t4 is max healing stat

#get healing stat
lw t2,8(a0)
#get healers health
lw t3,0(a0)
#apply healing and update
add t3,t3,t2
#check if healing is greater than max health
blt t3,t4,continueHeal
mv t3,t4

continueHeal:
sw t3,0(a0)
ret
```

Inside notFirst for player max health:

```
#move pointer to array addresses to a0 and a1 for use in apply damage and apply heal
mv a0,s0
mv a1,s1
mv t4,s7
#li 1 and 2 for checking user input for branching
li t1,1
li t2,2
#perform branch checks and branch to label otherwise loop
beq t0,t1,useApplyDmg
beq t0,t2,useApplyHeal
```

Inside enemyTurn for enemy max health:

```
enemyTurn:
#get random number 0 or 1 and store in t0
li t2,2
jal ra getRandom
mv t0,a0
#move pointers for use in apply damage/heal
mv a0,s1
mv a1,s0
mv t4,s8
```

6 – Dodge action**Pseudocode for dodge action****Dodge**

```
s9 = dodge flag
s10 = dodge cooldown

if user input = 3
branch useDodge

useDodge:
if s10 greater than zero branch noDodge
sets dodgeFlag to 1
set dodge cooldown to 3
b enemyTurn

noDodge:
print cannot use dodge string tell user the value of cooldown
branch to notFirst

enemyApplyDamage:
beq s9,zero, takeDmg
li s9,zero
print dodge strings "dodged enemy attack"
b skipDmg
takedmg:
```

```

jal ra applyDmg
jal ra printEnemyDmg
skipdmg:
jal ra endBattleCheck
#increment round count
addi s3,s3,1

dodgeCooldownDmg:
ble s10,zero,doneDmg
addi s10,s10,-1
doneDmg:
b userTurn

enemyApplyHeal:
beq s9,zero, skipDodgeHeal
print "Dodge failed"
li s9,zero
skipDodgeHeal:
jal ra applyHeal
jal ra printEnemyHeal
jal ra endBattleCheck
addi s3,s3,1

dodgeCooldownHeal:
ble s10,zero,skipCooldownReset
addi s10,s10,-1
doneHeal:
b userTurn

```

Implementation of dodge action

Inside userTurn

```

notFirst:

#check if enemy or player health is zero
jal ra,endBattleCheck
#use print function to print strings
jal ra,notFirstPrint
#take user input and store in t0
li a7,5
ecall
#move user input to safe place
mv t0,a0
#move pointer to array addresses to a0 and a1 for use
mv a0,s0
mv a1,s1
mv t4,s7
#li 1 and 2 for checking user input for branching
li t1,1
li t2,2
li t3,3
#perform branch checks and branch to label otherwise i
beq t0,t1,useApplyDmg
beq t0,t2,useApplyHeal
beq t0,t3,useDodge
#print invalid input message and loop
li a7,4
la a0,invalidInput
ecall
b userTurn
User input 3 = useDodge

```

```

useDodge:
#if branch cooldown is active, no dodge
bgtz s10,noDodge
#set dodge flag to 1
li s9,1
#set cooldown to 3
li s10,3
b enemyTurn

noDodge:
#print strings to tell user dodge cooldown info
li a7,4
la a0,dodgeCooldownStr1
ecall

li a7,1
mv a0,s10
ecall

li a7,4
la a0,dodgeCooldownStr2
ecall

b notFirst

```

```

Inside enemyTurn
enemyApplyDmg:
#check for dodge
beq $9,zero,takeDmg
li $9,0
#print dodged attack string
li a7,4
la a0,dodgeAttackStr
ecall
b skipDmg
#print and apply damage
takeDmg:
mv a0,$1
jal ra applyDmg
jal ra printEnemyDmg
skipDmg:
jal ra endBattleCheck
#increment round count
addi $3,$3,1
#deal with dodge cooldown
jal ra updateCooldown
b userTurn

```

```

enemyApplyHeal:
#check for dodge
beq $9,zero, skipDodgeHeal
#print dodge failed string
li a7,4
la a0,dodgeHealStr
ecall
li $9,0
skipDodgeHeal:
mv a0,$1
jal ra applyHeal
jal ra printEnemyHeal
jal ra endBattleCheck
addi $3,$3,1
#deal with dodge cooldown
jal ra updateCooldown
b userTurn

```

updateCooldown subroutine

```

#deal with dodge cooldown
updateCooldown:
ble $10,zero,updateCooldownDone|
addi $10,$10,-1
updateCooldownDone:
ret

```

Comments:

The updateCooldown subroutine was created instead of checking the cooldown in enemyApplyDmg/Heal to make the code more readable and easier to maintain.

User Experience

1 – Immersive Strings

```
22
23 #Character and enemy name strings
24
25 #Menu strings
26 menuStr1: .string "Castle Defence 2025\n"
27 menuStr2: .string "1 - Start Game\n"
28 menuStr3: .string "2 - Exit Game\n"
29
30
31 #-----
32
33 #Character and enemy strings
34 knightStr: .string "Knight"
35 musketeerStr: .string "Musketeer"
36 wizardStr: .string "Wizard"
37 goblinStr: .string "Goblin"
38 golemStr: .string "Golem"
39 witchStr: .string "Witch"
40 slimeStr: .string "Boss Slime"
41
42 #-----
43 #dodge strings
44
45 dodgeCooldownStr1: .string "Dodge is on cooldown for "
46 dodgeCooldownStr2: .string " rounds\n\n"
47
48 dodgeAttackStr: .string "You dodged the attack\n"
49 dodgeHealStr: .string "\nThe enemy did not attack, dodge ineffective"
50
51 #-----
52 #upgradeAttribute strings
53
54 upgAttStr1: .string "\nYou have reached level "
55 upgAttStr2: .string "\n\nSelect an attribute to upgrade\n\n"
56 upgAttStr3: .string "1 - Health\n"
57 upgAttStr4: .string "2 - Damage\n"
58 upgAttStr5: .string "3 - Healing\n\n"
59
60
61 upgHealthStr: .string "Your max health is now: "
62 upgDmgStr1: .string "You now deal: "
63 upgDmgStr2: .string "damage"
64 upgHealStr: .string "Your healing stat is now: "
65
66 #-----
67 #chooseCharacter strings
68 chooseBuildStr: .string "\nChoose your build\n\n"
69
```



```

endWinStr: .string "\nYou mighty soldier\n\nYou beat the game!"
endLossStr: .string "\nUnlucky brave soldier\n\nYou were defeated"

#-----
#strings for enemy and player actions

#print user damage

userDmgStr1: .string "You dealt "
userDmgStr2: .string " amount of damage\n\n"
userDmgStr3: .string "Enemy "
userDmgStr4: .string " has "
userDmgStr5: .string " health remaining"

#prints user heal

userHealStr1: .string "You healed for "
userHealStr2: .string " health\n\n"
userHealStr3: .string "Your total health is "

#prints enemy damage

enemyDmgStr1: .string "You were dealt "
enemyDmgStr2: .string " amount of damage by enemy\n"

#prints enemy heal

enemyHealStr1: .string "\nEnemy "
enemyHealStr2: .string " healed for "
enemyHealStr3: .string " amount of healing\n"

```

Implementing “Themed” strings makes the game feel like an actual game that is supposed to be played. Strings have consistent spacing using “\n”.

2 – Press any to continue

```

#press any to continue
li a7,4
la a0,continueStr
ecall
la a0,continueBuffer
li a7,8
ecall

jal ra endBattleCheck
b enemyTurn

```

Press any to continue prompts the user to press enter to continue, but any input followed by an enter will result in the program continuing. The input is stored in a buffer in memory to prevent any bugs.

3 – Make battle screen clearer

Battle screen now displays player level and explicitly tells the user that their health is being displayed.

```
A Boss Slime HAS APPEARED!
```

```
What will you do?
```

```
1 - Attack  
2 - Heal  
3 - Dodge
```

```
Your Health: 130
```

```
Your Level: 2
```

4 – Input validation

To implement robust input validation, a sub routine was implemented to take user input and convert the user input from ASCII to an integer. This allows the user to input any character without crashing the program. Considering the game only uses integers 1-9 for user input, the subroutine only accounts for characters 1-9.

As the ReadString syscall only stores the pointer to the string, subtracting 0x30 from a0 would be pointless. Instead an inputBuffer is used, somewhere the string can be stored, we can then access the first byte using the “lbu” instruction (load byte unsigned); subtract 0x30, perform input validation checks etc.

Pseudocode

```
ASCII CONVERSION
```

```
invalidInput:
```

```
"The input is not valid"
```

```
takeInputConversion:
```

```
"Please Select"
```

```
take input as char and store in buffer
```

```
load buffer address to t0 and get first byte  
subtract 0x30
```

```
bgt a0 > 9 invalidInput  
blt a0 > 0 invalidInput
```

```
mv t0,a0
```

```
ret
```

Actual implementation

```

#function to take input as ASCII and convert to integer also validates input

invalidInput:
#print invalid input string
li a7,4
la a0,invalidInput
ecall
#take input and convert
takeInputConversion:
#print please select string
li a7,4
la a0,takeInput
ecall
#take input as string
la a0, inputBuffer
li al,4
li a7,8
ecall
#load the first byte from inputBuffer to t0
la t0,inputBuffer
lbu t0,0(t0)
#subtract 0x30 (48) from a0
addi t0,t0,-48
#check if input is valid
li tl,9
bgtu t0,tl,invalidInput
#move t0 to a0 and return
mv a0,t0
ret

```

Comments

The bgtu pseudo instruction combines bgt/blt into one instruction, by ignoring the signed bit bgtu treats negative numbers as huge positive numbers which triggers the branch.

```

#Take input string
takeInput: .string "Please Select: "
#Invalid input string
invalidInput: .string "Invalid input please select a number\n"
#new line
newLine: .string "\n"
#input out of range string
inputOOR: .string "The input you selected is not a defined option\n"

```

Added inputOOR string is displayed when the input is valid inside takeInputConversion, but not a valid menu option.

```

Castle Defence 2025
1 - Start Game
2 - Exit Game
Please Select: e
Input is not a valid number!
Please Select: r
Input is not a valid number!
Please Select: f
Input is not a valid number!
Please Select: sa
Input is not a valid number!
Please Select: asfInput is not a valid number!
Please Select: dfdInput is not a valid number!
Please Select: 5
Input is not an option!
Castle Defence 2025
1 - Start Game
2 - Exit Game
Please Select:

```

Program no longer crashes when input is invalid.

3. References

Azad, K. (2007) 'Swap two variables using XOR', *BetterExplained*, 16 January. Available at:

<https://betterexplained.com/articles/swap-two-variables-using-xor/> (Accessed: 2 December 2025).

Free Software Foundation (2025) '.align', *Using as: the GNU assembler*. Available at:

<https://sourceware.org/binutils/docs/as/Align.html> (Accessed: 4 December 2025).

Green, W. (2024) 'RISC-V Assembler: Load Store', *Project F*, 15 February. Available at: <https://projectf.io/posts/riscv-load-store/> (Accessed: 2 December 2025).

Green, W. (2024) 'RISC-V Assembler Cheat Sheet', *Project F*, 14 June. Available at: <https://projectf.io/posts/riscv-cheat-sheet/> (Accessed: 6 November 2025).

'Link register' (2025) *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Link_register (Accessed: 14 November 2025).

Pesto (2010) 'When does printf's %s stop printing?', *Stack Overflow*, 28 April. Available at: <https://stackoverflow.com/questions/2726301/when-does-printfs-char-stop-printing> (Accessed: 10 November 2025).

Tihane, F. (2023) 'What is character encoding? Exploring Unicode, UTF-8, ASCII, and more', *Lokalise Blog*, 18 August. Available at: <https://lokalise.com/blog/what-is-character-encoding-exploring-unicode-utf-8-ascii-and-more/> (Accessed: 2 November 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="4.0.0" version="1.0">

```

This file is intended to be loaded by Logisim-evolution
v4.0.0(<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="type" val="output"/>
  </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="(1750,780)" 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="0"/>
  <a name="bit5" val="0"/>
  <a name="bit6" val="0"/>
  <a name="bit7" val="1"/>
  <a name="incoming" val="8"/>
</comp>
<comp lib="0" loc="(2610,860)" name="Bit Extender">
  <a name="in_width" val="2"/>
  <a name="out_width" val="8"/>
  <a name="type" val="zero"/>
</comp>
<comp lib="0" loc="(570,1090)" name="Clock">
  <a name="appearance" val="classic"/>
</comp>
<comp lib="0" loc="(670,510)" name="Constant">
  <a name="width" val="8"/>
</comp>
<comp lib="0" loc="(730,1250)" name="Splitter">
  <a name="fanout" val="8"/>
  <a name="incoming" val="8"/>
</comp>
<comp lib="0" loc="(780,1160)" name="Splitter">
  <a name="facing" val="west"/>
</comp>
<comp lib="0" loc="(840,1160)" name="Bit Extender">
  <a name="in_width" val="2"/>
  <a name="out_width" val="8"/>
  <a name="type" val="zero"/>
</comp>
<comp lib="1" loc="(1120,1090)" name="OR Gate">
  <a name="facing" val="south"/>
</comp>
<comp lib="1" loc="(1540,940)" name="OR Gate">
  <a name="inputs" val="4"/>
</comp>
<comp lib="1" loc="(1820,850)" name="AND Gate">
  <a name="facing" val="north"/>
</comp>
<comp lib="1" loc="(1950,1070)" name="AND Gate"/>
<comp lib="1" loc="(2000,860)" name="NOT Gate"/>
<comp lib="1" loc="(2000,890)" name="NOT Gate"/>
<comp lib="1" loc="(2000,920)" name="NOT Gate"/>
<comp lib="1" loc="(2010,1070)" name="NOT Gate"/>
<comp lib="1" loc="(390,620)" name="OR Gate">
  <a name="facing" val="west"/>
  <a name="inputs" val="3"/>
</comp>
<comp lib="1" loc="(560,630)" name="AND Gate">
  <a name="facing" val="north"/>
</comp>

```

```

<comp lib="1" loc="(590,1050)" name="NOT Gate"/>
<comp lib="1" loc="(620,610)" name="AND Gate">
  <a name="facing" val="north"/>
</comp>
<comp lib="1" loc="(650,1070)" name="AND Gate"/>
<comp lib="2" loc="(1100,1220)" name="Multiplexer">
  <a name="facing" val="south"/>
  <a name="width" val="8"/>
</comp>
<comp lib="2" loc="(1430,1240)" name="Multiplexer">
  <a name="select" val="2"/>
  <a name="selloc" val="tr"/>
  <a name="width" val="8"/>
</comp>
<comp lib="2" loc="(860,540)" name="Multiplexer">
  <a name="select" val="2"/>
  <a name="width" val="8"/>
</comp>
<comp lib="3" loc="(730,500)" name="Adder"/>
<comp lib="4" loc="(1080,1250)" name="RAM">
  <a name="appearance" val="classic"/>
  <a name="asyncread" val="true"/>
  <a name="clearpin" val="true"/>
  <a name="label" val="Data_Memory"/>
  <a name="labelfont" val="SansSerif bold 14"/>
  <a name="labelvisible" val="true"/>
</comp>
<comp lib="4" loc="(1160,490)" name="ROM">
  <a name="appearance" val="classic"/>
  <a name="contents">addr/data: 8 16
8000 1000 3000 3001 3002 3003 3004 3005
3006 3007 1008 3007 c08e 101d 3007 c08e
c098 2000 1803 4068 2001 1802 4068 2002
1802 4068 a000 1034 3007 c08e 1045 3007
c08e 104f 3007 c08e 105a 3007 c08e c098
9000 7028 3005 2004 2905 3006 2006 1103
3006 2006 1a03 3006 a000 2006 1801 4042
2006 1802 404c 107b 3007 c08e 2000 1101
3000 c056 1068 3007 c08e 2001 1101 3001
2000 1101 3000 c056 1071 3007 c08e 2002
1101 3002 2000 1101 3000 c056 2003 1801
4011 a000 1088 3007 c08e 10a0 3007 c08e
c098 2004 1801 7068 2003 1101 3003 c011
2001 2802 4084 2001 2502 4070 1001 407a
a000 10bb 3007 c08e 10de 3007 c08e c098
1001 4000 a000 10cc 3007 c08e 10de 3007
c08e c098 1001 4000 a000 107b 3007 c08e
10de 3007 c08e c098 1001 4000 2007 b000
7097 5000 2007 1101 3007 1001 408e d000
8000 6000 7099 3004 d000
</a>
  <a name="dataWidth" val="16"/>
  <a name="label" val="Instruction_Memory"/>
  <a name="labelfont" val="SansSerif bold 14"/>
  <a name="labelvisible" val="true"/>
</comp>

```

```

<comp lib="4" loc="(1580,870)" name="Register">
    <a name="appearance" val="logisim_evolution"/>
    <a name="label" val="ACC"/>
</comp>
<comp lib="4" loc="(640,1170)" name="Random">
    <a name="appearance" val="logisim_evolution"/>
</comp>
<comp lib="4" loc="(810,330)" name="Register">
    <a name="appearance" val="logisim_evolution"/>
    <a name="label" val="LINK_REG"/>
</comp>
<comp lib="4" loc="(920,510)" name="Register">
    <a name="appearance" val="logisim_evolution"/>
    <a name="label" val="PC"/>
</comp>
<comp lib="5" loc="(1910,800)" name="TTY"/>
<comp lib="5" loc="(1940,860)" name="Button">
    <a name="label" val="B1"/>
</comp>
<comp lib="5" loc="(1940,890)" name="Button">
    <a name="label" val="B2"/>
</comp>
<comp lib="5" loc="(1940,920)" name="Button">
    <a name="label" val="B3"/>
</comp>
<comp lib="8" loc="(1590,665)" name="Text">
    <a name="font" val="SansSerif plain 20"/>
    <a name="text" val="Student id: 24820129"/>
</comp>
<comp loc="(1370,1140)" name="SELECT_INPUT_2"/>
<comp loc="(1710,700)" name="ALU"/>
<comp loc="(2300,1000)" name="SR_LATCH"/>
<comp loc="(2300,860)" name="SR_LATCH"/>
<comp loc="(2300,930)" name="SR_LATCH"/>
<comp loc="(2560,860)" name="BUTTON_PRESS_CONVERSION"/>
<comp loc="(640,540)" name="SELECT_BIT_PC"/>
<comp loc="(990,690)" name="Instruction_Decoding_Unit"/>
<wire from="(1000,1050)" to="(1000,1280)"/>
<wire from="(1000,1050)" to="(1020,1050)"/>
<wire from="(1000,660)" to="(1000,770)"/>
<wire from="(1000,970)" to="(1000,1040)"/>
<wire from="(1010,1160)" to="(1010,1430)"/>
<wire from="(1010,1430)" to="(1380,1430)"/>
<wire from="(1010,650)" to="(1010,820)"/>
<wire from="(1010,820)" to="(1010,990)"/>
<wire from="(1010,820)" to="(1040,820)"/>
<wire from="(1020,1100)" to="(1020,1320)"/>
<wire from="(1020,1100)" to="(1580,1100)"/>
<wire from="(1020,1320)" to="(1080,1320)"/>
<wire from="(1020,640)" to="(1020,660)"/>
<wire from="(1020,660)" to="(1730,660)"/>
<wire from="(1020,830)" to="(1020,1050)"/>
<wire from="(1030,1120)" to="(1030,1160)"/>
<wire from="(1030,1160)" to="(1150,1160)"/>
<wire from="(1030,850)" to="(1030,860)"/>
<wire from="(1030,860)" to="(1370,860)"/>

```

```

<wire from="(1030,870)" to="(1030,1120)"/>
<wire from="(1030,870)" to="(1360,870)"/>
<wire from="(1040,1030)" to="(1040,1200)"/>
<wire from="(1040,1030)" to="(1100,1030)"/>
<wire from="(1040,1200)" to="(1080,1200)"/>
<wire from="(1040,820)" to="(1040,850)"/>
<wire from="(1040,850)" to="(1090,850)"/>
<wire from="(1040,910)" to="(1040,1030)"/>
<wire from="(1040,910)" to="(1340,910)"/>
<wire from="(1050,1300)" to="(1080,1300)"/>
<wire from="(1050,750)" to="(1050,1300)"/>
<wire from="(1060,1180)" to="(1150,1180)"/>
<wire from="(1060,1240)" to="(1060,1340)"/>
<wire from="(1060,1240)" to="(1140,1240)"/>
<wire from="(1060,1340)" to="(1080,1340)"/>
<wire from="(1060,810)" to="(1060,960)"/>
<wire from="(1060,960)" to="(1060,1180)"/>
<wire from="(1060,960)" to="(1490,960)"/>
<wire from="(1070,1230)" to="(1070,1260)"/>
<wire from="(1070,1230)" to="(1100,1230)"/>
<wire from="(1070,1260)" to="(1080,1260)"/>
<wire from="(1080,1010)" to="(1140,1010)"/>
<wire from="(1080,470)" to="(1080,500)"/>
<wire from="(1080,500)" to="(1080,540)"/>
<wire from="(1080,500)" to="(1160,500)"/>
<wire from="(1080,730)" to="(1080,1010)"/>
<wire from="(1090,850)" to="(1090,1190)"/>
<wire from="(1090,850)" to="(1380,850)"/>
<wire from="(1100,1030)" to="(1100,1040)"/>
<wire from="(1100,1220)" to="(1100,1230)"/>
<wire from="(1100,950)" to="(1100,1000)"/>
<wire from="(1110,1170)" to="(1110,1190)"/>
<wire from="(1110,1170)" to="(1140,1170)"/>
<wire from="(1120,1090)" to="(1120,1140)"/>
<wire from="(1120,1140)" to="(1150,1140)"/>
<wire from="(1140,1010)" to="(1140,1040)"/>
<wire from="(1140,1080)" to="(1140,1170)"/>
<wire from="(1140,1080)" to="(1460,1080)"/>
<wire from="(1140,1170)" to="(1140,1240)"/>
<wire from="(1320,1340)" to="(1360,1340)"/>
<wire from="(1330,930)" to="(1330,980)"/>
<wire from="(1330,980)" to="(1480,980)"/>
<wire from="(1340,910)" to="(1340,950)"/>
<wire from="(1340,950)" to="(1490,950)"/>
<wire from="(1350,890)" to="(1350,970)"/>
<wire from="(1350,970)" to="(1800,970)"/>
<wire from="(1360,1230)" to="(1360,1340)"/>
<wire from="(1360,1230)" to="(1390,1230)"/>
<wire from="(1360,870)" to="(1360,930)"/>
<wire from="(1360,930)" to="(1490,930)"/>
<wire from="(1370,1090)" to="(1900,1090)"/>
<wire from="(1370,1140)" to="(1410,1140)"/>
<wire from="(1370,1240)" to="(1370,1370)"/>
<wire from="(1370,1240)" to="(1390,1240)"/>
<wire from="(1370,1370)" to="(2620,1370)"/>
<wire from="(1370,860)" to="(1370,1090)"/>

```

```
<wire from="(1380,1220)" to="(1390,1220)"/>
<wire from="(1380,1250)" to="(1380,1430)"/>
<wire from="(1380,1250)" to="(1390,1250)"/>
<wire from="(1380,850)" to="(1380,1220)"/>
<wire from="(1400,550)" to="(1430,550)"/>
<wire from="(1410,1140)" to="(1410,1220)"/>
<wire from="(1420,710)" to="(1420,920)"/>
<wire from="(1420,920)" to="(1490,920)"/>
<wire from="(1430,1240)" to="(1440,1240)"/>
<wire from="(1430,550)" to="(1430,670)"/>
<wire from="(1440,740)" to="(1440,1240)"/>
<wire from="(1440,740)" to="(1490,740)"/>
<wire from="(1460,1080)" to="(1710,1080)"/>
<wire from="(1460,720)" to="(1460,1080)"/>
<wire from="(1460,720)" to="(1490,720)"/>
<wire from="(1480,690)" to="(1480,700)"/>
<wire from="(1480,700)" to="(1490,700)"/>
<wire from="(1480,980)" to="(1480,1450)"/>
<wire from="(1540,940)" to="(1550,940)"/>
<wire from="(1550,920)" to="(1550,940)"/>
<wire from="(1550,920)" to="(1580,920)"/>
<wire from="(1560,810)" to="(1560,900)"/>
<wire from="(1560,810)" to="(1720,810)"/>
<wire from="(1560,900)" to="(1580,900)"/>
<wire from="(1580,1100)" to="(1850,1100)"/>
<wire from="(1580,940)" to="(1580,1100)"/>
<wire from="(1640,900)" to="(1710,900)"/>
<wire from="(1710,700)" to="(1730,700)"/>
<wire from="(1710,720)" to="(1720,720)"/>
<wire from="(1710,740)" to="(1720,740)"/>
<wire from="(1710,790)" to="(1710,820)"/>
<wire from="(1710,820)" to="(1920,820)"/>
<wire from="(1710,900)" to="(1710,1080)"/>
<wire from="(1710,900)" to="(1730,900)"/>
<wire from="(1720,680)" to="(1720,720)"/>
<wire from="(1720,740)" to="(1720,810)"/>
<wire from="(1730,660)" to="(1730,700)"/>
<wire from="(1730,780)" to="(1730,900)"/>
<wire from="(1730,780)" to="(1750,780)"/>
<wire from="(1770,790)" to="(1910,790)"/>
<wire from="(1800,900)" to="(1800,970)"/>
<wire from="(1820,840)" to="(1820,850)"/>
<wire from="(1820,840)" to="(1930,840)"/>
<wire from="(1840,900)" to="(1840,970)"/>
<wire from="(1840,970)" to="(1850,970)"/>
<wire from="(1850,1050)" to="(1850,1100)"/>
<wire from="(1850,1050)" to="(1900,1050)"/>
<wire from="(1850,800)" to="(1850,970)"/>
<wire from="(1850,800)" to="(1910,800)"/>
<wire from="(1850,970)" to="(1850,1050)"/>
<wire from="(1920,810)" to="(1920,820)"/>
<wire from="(1930,810)" to="(1930,840)"/>
<wire from="(1940,860)" to="(1970,860)"/>
<wire from="(1940,890)" to="(1970,890)"/>
<wire from="(1940,920)" to="(1970,920)"/>
<wire from="(1950,1070)" to="(1980,1070)"/>
```

```
<wire from="(2000,860)" to="(2080,860)"/>
<wire from="(2000,890)" to="(2060,890)"/>
<wire from="(2000,920)" to="(2050,920)"/>
<wire from="(2010,1070)" to="(2030,1070)"/>
<wire from="(2030,1050)" to="(2030,1070)"/>
<wire from="(2030,1050)" to="(2070,1050)"/>
<wire from="(2050,1000)" to="(2080,1000)"/>
<wire from="(2050,920)" to="(2050,1000)"/>
<wire from="(2060,890)" to="(2060,930)"/>
<wire from="(2060,930)" to="(2080,930)"/>
<wire from="(2070,1020)" to="(2070,1050)"/>
<wire from="(2070,1020)" to="(2080,1020)"/>
<wire from="(2070,880)" to="(2070,950)"/>
<wire from="(2070,880)" to="(2080,880)"/>
<wire from="(2070,950)" to="(2070,1020)"/>
<wire from="(2070,950)" to="(2080,950)"/>
<wire from="(2300,1000)" to="(2330,1000)"/>
<wire from="(2300,860)" to="(2340,860)"/>
<wire from="(2300,930)" to="(2320,930)"/>
<wire from="(2320,880)" to="(2320,930)"/>
<wire from="(2320,880)" to="(2340,880)"/>
<wire from="(2330,900)" to="(2330,1000)"/>
<wire from="(2330,900)" to="(2340,900)"/>
<wire from="(2560,860)" to="(2570,860)"/>
<wire from="(2610,860)" to="(2620,860)"/>
<wire from="(2620,860)" to="(2620,1370)"/>
<wire from="(380,1030)" to="(990,1030)"/>
<wire from="(380,560)" to="(380,1030)"/>
<wire from="(380,560)" to="(420,560)"/>
<wire from="(390,540)" to="(390,620)"/>
<wire from="(390,540)" to="(420,540)"/>
<wire from="(440,600)" to="(620,600)"/>
<wire from="(440,620)" to="(560,620)"/>
<wire from="(440,640)" to="(510,640)"/>
<wire from="(510,1450)" to="(1480,1450)"/>
<wire from="(510,640)" to="(510,710)"/>
<wire from="(510,710)" to="(510,1450)"/>
<wire from="(510,710)" to="(660,710)"/>
<wire from="(540,1280)" to="(1000,1280)"/>
<wire from="(540,680)" to="(540,1280)"/>
<wire from="(550,1040)" to="(1000,1040)"/>
<wire from="(550,1040)" to="(550,1050)"/>
<wire from="(550,1050)" to="(560,1050)"/>
<wire from="(560,620)" to="(560,630)"/>
<wire from="(570,1090)" to="(600,1090)"/>
<wire from="(580,680)" to="(580,690)"/>
<wire from="(580,690)" to="(740,690)"/>
<wire from="(590,1050)" to="(600,1050)"/>
<wire from="(600,660)" to="(600,680)"/>
<wire from="(600,680)" to="(1720,680)"/>
<wire from="(620,1100)" to="(620,1220)"/>
<wire from="(620,1100)" to="(730,1100)"/>
<wire from="(620,1220)" to="(640,1220)"/>
<wire from="(620,600)" to="(620,610)"/>
<wire from="(630,1120)" to="(1030,1120)"/>
<wire from="(630,1120)" to="(630,1210)"/>
```

```
<wire from="(630,1210)" to="(640,1210)"/>
<wire from="(640,380)" to="(640,530)"/>
<wire from="(640,380)" to="(810,380)"/>
<wire from="(640,530)" to="(660,530)"/>
<wire from="(640,540)" to="(790,540)"/>
<wire from="(640,660)" to="(640,670)"/>
<wire from="(640,670)" to="(720,670)"/>
<wire from="(650,1070)" to="(730,1070)"/>
<wire from="(650,470)" to="(1080,470)"/>
<wire from="(650,470)" to="(650,490)"/>
<wire from="(650,490)" to="(690,490)"/>
<wire from="(660,530)" to="(660,710)"/>
<wire from="(670,510)" to="(690,510)"/>
<wire from="(720,1250)" to="(730,1250)"/>
<wire from="(720,530)" to="(720,650)"/>
<wire from="(720,530)" to="(820,530)"/>
<wire from="(720,650)" to="(1010,650)"/>
<wire from="(720,660)" to="(1000,660)"/>
<wire from="(720,660)" to="(720,670)"/>
<wire from="(730,1070)" to="(730,1100)"/>
<wire from="(730,1100)" to="(1020,1100)"/>
<wire from="(730,500)" to="(750,500)"/>
<wire from="(730,600)" to="(730,1070)"/>
<wire from="(730,600)" to="(740,600)"/>
<wire from="(740,400)" to="(740,600)"/>
<wire from="(740,400)" to="(810,400)"/>
<wire from="(740,600)" to="(920,600)"/>
<wire from="(740,640)" to="(1020,640)"/>
<wire from="(740,640)" to="(740,690)"/>
<wire from="(750,1170)" to="(760,1170)"/>
<wire from="(750,1180)" to="(760,1180)"/>
<wire from="(750,360)" to="(750,500)"/>
<wire from="(750,360)" to="(810,360)"/>
<wire from="(750,500)" to="(760,500)"/>
<wire from="(760,500)" to="(760,520)"/>
<wire from="(760,520)" to="(820,520)"/>
<wire from="(760,670)" to="(1430,670)"/>
<wire from="(760,670)" to="(760,690)"/>
<wire from="(760,690)" to="(770,690)"/>
<wire from="(780,1160)" to="(800,1160)"/>
<wire from="(790,540)" to="(790,580)"/>
<wire from="(790,580)" to="(840,580)"/>
<wire from="(800,440)" to="(800,540)"/>
<wire from="(800,440)" to="(880,440)"/>
<wire from="(800,540)" to="(820,540)"/>
<wire from="(840,1160)" to="(1010,1160)"/>
<wire from="(840,560)" to="(840,580)"/>
<wire from="(860,540)" to="(920,540)"/>
<wire from="(870,360)" to="(880,360)"/>
<wire from="(880,360)" to="(880,440)"/>
<wire from="(920,580)" to="(920,600)"/>
<wire from="(980,540)" to="(1080,540)"/>
<wire from="(990,1000)" to="(1100,1000)"/>
<wire from="(990,1000)" to="(990,1030)"/>
<wire from="(990,690)" to="(1480,690)"/>
<wire from="(990,710)" to="(1420,710)"/>
```

```

<wire from="(990,730)" to="(1080,730)"/>
<wire from="(990,750)" to="(1050,750)"/>
<wire from="(990,770)" to="(1000,770)"/>
<wire from="(990,790)" to="(1710,790)"/>
<wire from="(990,810)" to="(1060,810)"/>
<wire from="(990,830)" to="(1020,830)"/>
<wire from="(990,850)" to="(1030,850)"/>
<wire from="(990,870)" to="(1030,870)"/>
<wire from="(990,890)" to="(1350,890)"/>
<wire from="(990,910)" to="(1040,910)"/>
<wire from="(990,930)" to="(1330,930)"/>
<wire from="(990,950)" to="(1100,950)"/>
<wire from="(990,970)" to="(1000,970)"/>
<wire from="(990,990)" to="(1010,990)"/>
</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="type" val="output"/>
  </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="type" val="output"/>
</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="type" val="output"/>
</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="type" val="output"/>
</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="type" val="output"/>
</comp>
<comp lib="0" loc="(600,350)" name="Pin">
<a name="appearance" val="NewPins"/>
<a name="facing" val="west"/>
<a name="label" val="User_Input"/>
<a name="type" val="output"/>
</comp>
<comp lib="0" loc="(600,380)" name="Pin">
<a name="appearance" val="NewPins"/>
<a name="facing" val="west"/>
<a name="label" val="BEZ"/>
<a name="type" val="output"/>
</comp>
<comp lib="0" loc="(600,410)" name="Pin">
<a name="appearance" val="NewPins"/>
<a name="facing" val="west"/>
<a name="label" val="RESET_LATCH"/>
<a name="type" val="output"/>
</comp>
<comp lib="0" loc="(600,440)" name="Pin">
<a name="appearance" val="NewPins"/>
<a name="facing" val="west"/>
<a name="label" val="GENERATE_RANDOM"/>
<a name="type" val="output"/>
</comp>
<comp lib="0" loc="(600,470)" name="Pin">
<a name="appearance" val="NewPins"/>
<a name="facing" val="west"/>
<a name="label" val="CLEAR_TTY"/>
<a name="type" val="output"/>

```

```

</comp>
<comp lib="0" loc="(600,500)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="label" val="POINTER"/>
  <a name="type" val="output"/>
</comp>
<comp lib="0" loc="(600,530)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="label" val="JAL"/>
  <a name="type" val="output"/>
</comp>
<comp lib="0" loc="(600,560)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="label" val="RET"/>
  <a name="type" val="output"/>
</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="type" val="output"/>
</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="radix" val="16"/>
  <a name="type" val="output"/>
  <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="radix" val="16"/>
  <a name="type" val="output"/>
  <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="(350,280)" to="(500,280)"/>
<wire from="(350,290)" to="(490,290)"/>
<wire from="(350,300)" to="(480,300)"/>
<wire from="(350,310)" to="(470,310)"/>
<wire from="(350,320)" to="(460,320)"/>
<wire from="(350,330)" to="(450,330)"/>
<wire from="(350,340)" to="(440,340)"/>
<wire from="(350,350)" to="(430,350)"/>
<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="(430,350)" to="(430,560)"/>
<wire from="(430,560)" to="(600,560)"/>
<wire from="(440,340)" to="(440,530)"/>
<wire from="(440,530)" to="(600,530)"/>
<wire from="(450,330)" to="(450,500)"/>
<wire from="(450,500)" to="(600,500)"/>
<wire from="(460,320)" to="(460,470)"/>
<wire from="(460,470)" to="(600,470)"/>
<wire from="(470,310)" to="(470,440)"/>
<wire from="(470,440)" to="(600,440)"/>
<wire from="(480,300)" to="(480,410)"/>
<wire from="(480,410)" to="(600,410)"/>
<wire from="(490,290)" to="(490,380)"/>
<wire from="(490,380)" to="(600,380)"/>
<wire from="(500,280)" to="(500,350)"/>
<wire from="(500,350)" to="(600,350)"/>
<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="(210,110)" name="Pin">
  <a name="appearance" val="New Pins"/>
  <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="New Pins"/>
  <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="(210,30)" name="Pin">
  <a name="appearance" val="New Pins"/>
  <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="(330,370)" name="Constant">
  <a name="value" val="0x4"/>
  <a name="width" val="3"/>
</comp>
<comp lib="0" loc="(410,550)" name="Bit Extender">
  <a name="in_width" val="1"/>
  <a name="out_width" val="8"/>
</comp>
<comp lib="0" loc="(420,420)" 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="(470,460)" name="Splitter">
  <a name="bit1" val="0"/>
  <a name="bit10" val="1"/>
  <a name="bit11" val="1"/>
  <a name="bit12" val="1"/>
  <a name="bit13" val="1"/>
  <a name="bit14" val="1"/>
  <a name="bit15" val="1"/>
  <a name="bit2" val="0"/>
  <a name="bit3" val="0"/>
  <a name="bit4" val="0"/>
  <a name="bit5" val="0"/>
  <a name="bit6" val="0"/>
  <a name="bit7" val="0"/>
  <a name="bit8" val="1"/>
  <a name="bit9" val="1"/>
  <a name="incoming" val="16"/>
</comp>
<comp lib="0" loc="(680,120)" name="Constant">

```

```

<a name="value" val="0x0"/>
<a name="width" val="8"/>
</comp>
<comp lib="0" loc="(680,180)" name="Constant">
  <a name="value" val="0x0"/>
  <a name="width" val="8"/>
</comp>
<comp lib="0" loc="(680,270)" name="Pin">
  <a name="appearance" val="New Pins"/>
  <a name="facing" val="west"/>
  <a name="label" val="ALU_to_ACC"/>
  <a name="labelfont" val="SansSerif bold 12"/>
  <a name="radix" val="16"/>
  <a name="type" val="output"/>
  <a name="width" val="8"/>
</comp>
<comp lib="0" loc="(810,130)" name="Pin">
  <a name="appearance" val="New Pins"/>
  <a name="facing" val="west"/>
  <a name="label" val="ACC_EQUAL_ZERO"/>
  <a name="type" val="output"/>
</comp>
<comp lib="0" loc="(810,190)" name="Pin">
  <a name="appearance" val="New Pins"/>
  <a name="facing" val="west"/>
  <a name="label" val="ACC_NE_Zero"/>
  <a name="type" val="output"/>
</comp>
<comp lib="1" loc="(390,270)" name="NAND Gate">
  <a name="size" val="30"/>
  <a name="width" val="8"/>
</comp>
<comp lib="1" loc="(390,310)" name="OR Gate">
  <a name="size" val="30"/>
  <a name="width" val="8"/>
</comp>
<comp lib="1" loc="(800,190)" 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="(350,550)" name="Comparator"/>
<comp lib="3" loc="(350,600)" name="Subtractor"/>
<comp lib="3" loc="(350,660)" name="Divider"/>
<comp lib="3" loc="(360,420)" name="Comparator">
  <a name="mode" val="unsigned"/>
</comp>
<comp lib="3" loc="(390,220)" name="Adder"/>
<comp lib="3" loc="(400,360)" name="Shifter">
  <a name="shift" val="lr"/>
</comp>
<comp lib="3" loc="(750,130)" name="Comparator"/>
<comp lib="3" loc="(750,190)" name="Comparator">
  <a name="mode" val="unsigned"/>
</comp>

```

```

<comp loc="(520,490)" name="HEX_ASCII_OUTPUT"/>
<wire from="(210,110)" to="(300,110)"/>
<wire from="(210,190)" to="(240,190)"/>
<wire from="(210,30)" to="(590,30)"/>
<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="(240,350)"/>
<wire from="(240,320)" to="(360,320)"/>
<wire from="(240,350)" to="(240,430)"/>
<wire from="(240,350)" to="(360,350)"/>
<wire from="(240,430)" to="(240,560)"/>
<wire from="(240,430)" to="(320,430)"/>
<wire from="(240,560)" to="(240,610)"/>
<wire from="(240,560)" to="(310,560)"/>
<wire from="(240,610)" to="(240,670)"/>
<wire from="(240,610)" to="(310,610)"/>
<wire from="(240,670)" to="(310,670)"/>
<wire from="(300,110)" to="(300,210)"/>
<wire from="(300,110)" to="(630,110)"/>
<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="(300,410)"/>
<wire from="(300,300)" to="(360,300)"/>
<wire from="(300,410)" to="(300,490)"/>
<wire from="(300,410)" to="(320,410)"/>
<wire from="(300,490)" to="(300,540)"/>
<wire from="(300,540)" to="(300,590)"/>
<wire from="(300,540)" to="(310,540)"/>
<wire from="(300,590)" to="(300,650)"/>
<wire from="(300,590)" to="(310,590)"/>
<wire from="(300,650)" to="(310,650)"/>
<wire from="(330,370)" to="(360,370)"/>
<wire from="(330,680)" to="(330,690)"/>
<wire from="(330,690)" to="(360,690)"/>
<wire from="(350,550)" to="(370,550)"/>
<wire from="(350,600)" to="(540,600)"/>
<wire from="(360,350)" to="(360,360)"/>
<wire from="(360,410)" to="(370,410)"/>
<wire from="(360,610)" to="(360,690)"/>
<wire from="(360,610)" to="(550,610)"/>
<wire from="(370,410)" to="(370,420)"/>
<wire from="(370,420)" to="(380,420)"/>
<wire from="(390,220)" to="(400,220)"/>
<wire from="(390,270)" to="(420,270)"/>
<wire from="(390,310)" to="(440,310)"/>
<wire from="(400,200)" to="(400,220)"/>
<wire from="(400,200)" to="(570,200)"/>
<wire from="(400,360)" to="(450,360)"/>
<wire from="(410,550)" to="(530,550)"/>
<wire from="(420,210)" to="(420,270)"/>

```

```

<wire from="(420,210)" to="(570,210)"/>
<wire from="(420,420)" to="(460,420)"/>
<wire from="(440,220)" to="(440,310)"/>
<wire from="(440,220)" to="(570,220)"/>
<wire from="(450,230)" to="(450,360)"/>
<wire from="(450,230)" to="(570,230)"/>
<wire from="(460,240)" to="(460,420)"/>
<wire from="(460,240)" to="(570,240)"/>
<wire from="(470,460)" to="(520,460)"/>
<wire from="(490,250)" to="(490,440)"/>
<wire from="(490,250)" to="(570,250)"/>
<wire from="(490,450)" to="(500,450)"/>
<wire from="(500,260)" to="(500,450)"/>
<wire from="(500,260)" to="(570,260)"/>
<wire from="(510,270)" to="(510,450)"/>
<wire from="(510,270)" to="(570,270)"/>
<wire from="(510,450)" to="(530,450)"/>
<wire from="(520,280)" to="(520,440)"/>
<wire from="(520,280)" to="(570,280)"/>
<wire from="(520,440)" to="(540,440)"/>
<wire from="(520,460)" to="(520,490)"/>
<wire from="(530,290)" to="(530,430)"/>
<wire from="(530,290)" to="(570,290)"/>
<wire from="(530,430)" to="(550,430)"/>
<wire from="(530,450)" to="(530,550)"/>
<wire from="(540,440)" to="(540,600)"/>
<wire from="(550,430)" to="(550,610)"/>
<wire from="(590,30)" to="(590,190)"/>
<wire from="(610,270)" to="(680,270)"/>
<wire from="(630,110)" to="(630,140)"/>
<wire from="(630,140)" to="(630,200)"/>
<wire from="(630,140)" to="(710,140)"/>
<wire from="(630,200)" to="(710,200)"/>
<wire from="(680,120)" to="(710,120)"/>
<wire from="(680,180)" to="(710,180)"/>
<wire from="(750,130)" to="(810,130)"/>
<wire from="(750,190)" to="(770,190)"/>
<wire from="(800,190)" to="(810,190)"/>
</circuit>
<circuit name="HEX_ASCII_OUTPUT">
  <a name="appearance" val="logisim_evolution"/>
  <a name="circuit" val="HEX_ASCII_OUTPUT"/>
  <a name="circuitnamedboxfixedsize" val="true"/>
  <a name="simulationFrequency" val="2.0"/>
  <comp lib="0" loc="(260,430)" name="Pin">
    <a name="appearance" val="NewPins"/>
    <a name="label" val="HEX_TO_BE_CONVERTED"/>
    <a name="radix" val="16"/>
    <a name="width" val="8"/>
  </comp>
  <comp lib="0" loc="(280,430)" name="Splitter">
    <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="(360,220)" name="Constant">
  <a name="value" val="0x9"/>
  <a name="width" val="4"/>
</comp>
<comp lib="0" loc="(530,210)" name="Constant">
  <a name="value" val="0x30"/>
  <a name="width" val="8"/>
</comp>
<comp lib="0" loc="(560,210)" name="Constant">
  <a name="value" val="0x57"/>
  <a name="width" val="8"/>
</comp>
<comp lib="0" loc="(660,320)" 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="(660,460)" 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="(810,380)" name="Splitter">
  <a name="bit1" val="0"/>
  <a name="bit10" val="1"/>
  <a name="bit11" val="1"/>
  <a name="bit12" val="1"/>
  <a name="bit13" val="1"/>
  <a name="bit14" val="1"/>
  <a name="bit15" val="1"/>
  <a name="bit2" val="0"/>
  <a name="bit3" val="0"/>
  <a name="bit4" val="0"/>
  <a name="bit5" val="0"/>
  <a name="bit6" val="0"/>
  <a name="bit7" val="0"/>
  <a name="bit8" val="1"/>
  <a name="bit9" val="1"/>
  <a name="facing" val="west"/>
  <a name="incoming" val="16"/>
</comp>
<comp lib="0" loc="(850,380)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="label" val="HEX_VAL_ASCII_OUTPUT"/>
  <a name="radix" val="16"/>
  <a name="type" val="output"/>
  <a name="width" val="16"/>
</comp>
<comp lib="1" loc="(500,260)" name="OR Gate"/>
<comp lib="1" loc="(500,500)" name="OR Gate"/>
<comp lib="2" loc="(600,260)" name="Multiplexer">

```

```

<a name="width" val="8"/>
</comp>
<comp lib="2" loc="(600,500)" name="Multiplexer">
    <a name="width" val="8"/>
</comp>
<comp lib="3" loc="(430,260)" name="Comparator">
    <a name="mode" val="unsigned"/>
    <a name="width" val="4"/>
</comp>
<comp lib="3" loc="(430,500)" name="Comparator">
    <a name="mode" val="unsigned"/>
    <a name="width" val="4"/>
</comp>
<comp lib="3" loc="(740,320)" name="Adder"/>
<comp lib="3" loc="(740,460)" name="Adder"/>
<wire from="(260,430)" to="(280,430)"/>
<wire from="(300,250)" to="(300,320)"/>
<wire from="(300,250)" to="(390,250)"/>
<wire from="(300,320)" to="(300,410)"/>
<wire from="(300,320)" to="(620,320)"/>
<wire from="(300,420)" to="(300,460)"/>
<wire from="(300,460)" to="(300,490)"/>
<wire from="(300,460)" to="(620,460)"/>
<wire from="(300,490)" to="(390,490)"/>
<wire from="(360,220)" to="(360,270)"/>
<wire from="(360,270)" to="(360,510)"/>
<wire from="(360,270)" to="(390,270)"/>
<wire from="(360,510)" to="(390,510)"/>
<wire from="(430,260)" to="(440,260)"/>
<wire from="(430,270)" to="(440,270)"/>
<wire from="(430,500)" to="(440,500)"/>
<wire from="(430,510)" to="(440,510)"/>
<wire from="(440,240)" to="(440,260)"/>
<wire from="(440,240)" to="(450,240)"/>
<wire from="(440,270)" to="(440,280)"/>
<wire from="(440,280)" to="(450,280)"/>
<wire from="(440,480)" to="(440,500)"/>
<wire from="(440,480)" to="(450,480)"/>
<wire from="(440,510)" to="(440,520)"/>
<wire from="(440,520)" to="(450,520)"/>
<wire from="(500,260)" to="(500,290)"/>
<wire from="(500,290)" to="(580,290)"/>
<wire from="(500,500)" to="(500,530)"/>
<wire from="(500,530)" to="(580,530)"/>
<wire from="(530,210)" to="(530,270)"/>
<wire from="(530,270)" to="(530,510)"/>
<wire from="(530,270)" to="(570,270)"/>
<wire from="(530,510)" to="(570,510)"/>
<wire from="(560,210)" to="(560,250)"/>
<wire from="(560,250)" to="(560,490)"/>
<wire from="(560,250)" to="(570,250)"/>
<wire from="(560,490)" to="(570,490)"/>
<wire from="(580,280)" to="(580,290)"/>
<wire from="(580,520)" to="(580,530)"/>
<wire from="(600,260)" to="(680,260)"/>
<wire from="(600,500)" to="(690,500)"/>

```

```

<wire from="(660,320)" to="(690,320)"/>
<wire from="(660,460)" to="(690,460)"/>
<wire from="(680,260)" to="(680,330)"/>
<wire from="(680,330)" to="(700,330)"/>
<wire from="(690,310)" to="(690,320)"/>
<wire from="(690,310)" to="(700,310)"/>
<wire from="(690,450)" to="(690,460)"/>
<wire from="(690,450)" to="(700,450)"/>
<wire from="(690,470)" to="(690,500)"/>
<wire from="(690,470)" to="(700,470)"/>
<wire from="(740,320)" to="(780,320)"/>
<wire from="(740,460)" to="(780,460)"/>
<wire from="(780,320)" to="(780,390)"/>
<wire from="(780,390)" to="(790,390)"/>
<wire from="(780,400)" to="(780,460)"/>
<wire from="(780,400)" to="(790,400)"/>
<wire from="(810,380)" to="(850,380)"/>
</circuit>
<circuit name="USER_INPUT">
  <a name="appearance" val="logisim_evolution"/>
  <a name="circuit" val="USER_INPUT"/>
  <a name="circuitnamedboxfixedsize" val="true"/>
  <a name="simulationFrequency" val="2.0"/>
  <comp lib="0" loc="(500,280)" name="Pin">
    <a name="appearance" val="New Pins"/>
    <a name="label" val="B1"/>
  </comp>
  <comp lib="0" loc="(500,310)" name="Pin">
    <a name="appearance" val="New Pins"/>
    <a name="label" val="B2"/>
  </comp>
  <comp lib="0" loc="(500,340)" name="Pin">
    <a name="appearance" val="New Pins"/>
    <a name="label" val="B3"/>
  </comp>
  <comp lib="0" loc="(790,310)" name="Splitter">
    <a name="facing" val="west"/>
    <a name="fanout" val="3"/>
    <a name="incoming" val="3"/>
  </comp>
  <comp lib="0" loc="(820,310)" name="Pin">
    <a name="appearance" val="New Pins"/>
    <a name="facing" val="west"/>
    <a name="label" val="BUTTON_OUTPUT"/>
    <a name="type" val="output"/>
    <a name="width" val="3"/>
  </comp>
  <wire from="(500,280)" to="(540,280)"/>
  <wire from="(500,310)" to="(630,310)"/>
  <wire from="(500,340)" to="(540,340)"/>
  <wire from="(540,220)" to="(540,280)"/>
  <wire from="(540,220)" to="(760,220)"/>
  <wire from="(540,340)" to="(540,390)"/>
  <wire from="(540,390)" to="(770,390)"/>
  <wire from="(630,300)" to="(630,310)"/>
  <wire from="(630,300)" to="(750,300)"/>

```

```

<wire from="(750,300)" to="(750,330)"/>
<wire from="(750,330)" to="(770,330)"/>
<wire from="(760,220)" to="(760,320)"/>
<wire from="(760,320)" to="(770,320)"/>
<wire from="(770,340)" to="(770,390)"/>
<wire from="(790,310)" to="(820,310)"/>
</circuit>
<circuit name="SELECT_INPUT_2">
  <a name="appearance" val="logisim_evolution"/>
  <a name="circuit" val="SELECT_INPUT_2"/>
  <a name="circuitnamedboxfixedsize" val="true"/>
  <a name="simulationFrequency" val="2.0"/>
  <comp lib="0" loc="(260,260)" name="Pin">
    <a name="appearance" val="New Pins"/>
    <a name="label" val="USE_MEMORY"/>
  </comp>
  <comp lib="0" loc="(260,290)" name="Pin">
    <a name="appearance" val="New Pins"/>
    <a name="label" val="GENERATE_RANDOM"/>
  </comp>
  <comp lib="0" loc="(260,320)" name="Pin">
    <a name="appearance" val="New Pins"/>
    <a name="label" val="USER_INPUT"/>
  </comp>
  <comp lib="0" loc="(460,280)" name="Splitter">
    <a name="facing" val="west"/>
  </comp>
  <comp lib="0" loc="(650,280)" name="Pin">
    <a name="appearance" val="New Pins"/>
    <a name="facing" val="west"/>
    <a name="label" val="SELECT_BIT"/>
    <a name="type" val="output"/>
    <a name="width" val="2"/>
  </comp>
  <comp lib="1" loc="(380,260)" name="OR Gate"/>
  <comp lib="1" loc="(380,320)" name="OR Gate"/>
  <wire from="(260,240)" to="(260,260)"/>
  <wire from="(260,240)" to="(330,240)"/>
  <wire from="(260,280)" to="(260,290)"/>
  <wire from="(260,280)" to="(330,280)"/>
  <wire from="(260,290)" to="(260,300)"/>
  <wire from="(260,300)" to="(330,300)"/>
  <wire from="(260,320)" to="(260,340)"/>
  <wire from="(260,340)" to="(330,340)"/>
  <wire from="(380,260)" to="(430,260)"/>
  <wire from="(380,320)" to="(430,320)"/>
  <wire from="(430,260)" to="(430,290)"/>
  <wire from="(430,290)" to="(440,290)"/>
  <wire from="(430,300)" to="(430,320)"/>
  <wire from="(430,300)" to="(440,300)"/>
  <wire from="(460,280)" to="(650,280)"/>
</circuit>
<circuit name="SR_LATCH">
  <a name="appearance" val="logisim_evolution"/>
  <a name="circuit" val="SR_LATCH"/>
  <a name="circuitnamedboxfixedsize" val="true"/>

```

```

<a name="simulationFrequency" val="2.0"/>
<comp lib="0" loc="(160,200)" name="Pin">
  <a name="appearance" val="New Pins"/>
  <a name="label" val="S"/>
</comp>
<comp lib="0" loc="(160,310)" name="Pin">
  <a name="appearance" val="New Pins"/>
  <a name="label" val="R"/>
</comp>
<comp lib="0" loc="(500,220)" name="Pin">
  <a name="appearance" val="New Pins"/>
  <a name="facing" val="west"/>
  <a name="label" val="Q"/>
  <a name="type" val="output"/>
</comp>
<comp lib="0" loc="(500,290)" name="Pin">
  <a name="appearance" val="New Pins"/>
  <a name="facing" val="west"/>
  <a name="label" val="NOTQ"/>
  <a name="type" val="output"/>
</comp>
<comp lib="1" loc="(400,220)" name="NAND Gate"/>
<comp lib="1" loc="(400,290)" name="NAND Gate"/>
<wire from="(160,200)" to="(340,200)"/>
<wire from="(160,310)" to="(340,310)"/>
<wire from="(320,240)" to="(320,260)"/>
<wire from="(320,240)" to="(340,240)"/>
<wire from="(320,260)" to="(410,260)"/>
<wire from="(330,250)" to="(330,270)"/>
<wire from="(330,250)" to="(410,250)"/>
<wire from="(330,270)" to="(340,270)"/>
<wire from="(400,220)" to="(410,220)"/>
<wire from="(400,290)" to="(410,290)"/>
<wire from="(410,220)" to="(410,250)"/>
<wire from="(410,220)" to="(500,220)"/>
<wire from="(410,260)" to="(410,290)"/>
<wire from="(410,290)" to="(500,290)"/>
</circuit>
<circuit name="BUTTON_PRESS_CONVERSION">
  <a name="appearance" val="logisim_evolution"/>
  <a name="circuit" val="BUTTON_PRESS_CONVERSION"/>
  <a name="circuitnamedboxfixedsize" val="true"/>
  <a name="simulationFrequency" val="2.0"/>
  <comp lib="0" loc="(370,350)" name="Pin">
    <a name="appearance" val="New Pins"/>
    <a name="label" val="ROCK"/>
  </comp>
  <comp lib="0" loc="(370,400)" name="Pin">
    <a name="appearance" val="New Pins"/>
    <a name="label" val="PAPER"/>
  </comp>
  <comp lib="0" loc="(370,440)" name="Pin">
    <a name="appearance" val="New Pins"/>
    <a name="label" val="SCISSORS"/>
  </comp>
  <comp lib="0" loc="(620,360)" name="Splitter">

```

```

<a name="facing" val="west"/>
</comp>
<comp lib="0" loc="(660,360)" name="Pin">
  <a name="appearance" val="NewPins"/>
  <a name="facing" val="west"/>
  <a name="type" val="output"/>
  <a name="width" val="2"/>
</comp>
<comp lib="1" loc="(530,370)" name="OR Gate"/>
<comp lib="1" loc="(580,420)" name="OR Gate"/>
<wire from="(370,350)" to="(480,350)"/>
<wire from="(370,400)" to="(530,400)"/>
<wire from="(370,440)" to="(470,440)"/>
<wire from="(470,390)" to="(470,440)"/>
<wire from="(470,390)" to="(480,390)"/>
<wire from="(470,440)" to="(530,440)"/>
<wire from="(530,370)" to="(600,370)"/>
<wire from="(580,420)" to="(590,420)"/>
<wire from="(590,380)" to="(590,420)"/>
<wire from="(590,380)" to="(600,380)"/>
<wire from="(620,360)" to="(660,360)"/>
</circuit>
<circuit name="SELECT_BIT_PC">
  <a name="appearance" val="logisim_evolution"/>
  <a name="circuit" val="SELECT_BIT_PC"/>
  <a name="circuitnamedboxfixedsize" val="true"/>
  <a name="simulationFrequency" val="2.0"/>
  <comp lib="0" loc="(320,250)" name="Pin">
    <a name="appearance" val="NewPins"/>
    <a name="label" val="JAL_BEZ_BNEZ"/>
  </comp>
  <comp lib="0" loc="(320,300)" name="Pin">
    <a name="appearance" val="NewPins"/>
    <a name="label" val="RET"/>
  </comp>
  <comp lib="0" loc="(620,260)" name="Splitter">
    <a name="facing" val="west"/>
  </comp>
  <comp lib="0" loc="(660,260)" name="Pin">
    <a name="appearance" val="NewPins"/>
    <a name="facing" val="west"/>
    <a name="label" val="MUX_IN"/>
    <a name="type" val="output"/>
    <a name="width" val="2"/>
  </comp>
  <wire from="(320,250)" to="(590,250)"/>
  <wire from="(320,300)" to="(590,300)"/>
  <wire from="(590,250)" to="(590,270)"/>
  <wire from="(590,270)" to="(600,270)"/>
  <wire from="(590,280)" to="(590,300)"/>
  <wire from="(590,280)" to="(600,280)"/>
  <wire from="(620,260)" to="(660,260)"/>
</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 may include more subheadings.

5.1 Tasks 2.1 to 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"
totalMsg: .string "The total is: "
outputOrMsg: .string "\nThe result of bitwise OR is: "
outputNandMsg: .string "\nThe bitwise of bitwise NAND is: "

.text

#outputmessage 1 to the console
addi a7, zero, 4
la a0, enterMsg1
ecall

# 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
#bitwise OR on s0 and s1, stored in s3
or s3,s0,s1
#bitwise NAND on s0 and s1, stored in s4
and s4,s0,s1
not s4,s4

#output total message
```

```

addi a7, zero, 4
la a0, totalMsg
ecall

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

#output the OR message and result
addi a7, zero, 4
la a0, outputOrMsg
ecall

add a0, s3, zero
addi a7, zero, 1
ecall

#putput the NAND message an result
addi a7, zero, 4
la a0, outputNandMsg
ecall

add a0, s4, zero
addi a7, zero, 1
ecall

```

```

addi a7, zero, 10
ecall

```

5.2 Task 2.4 Including a subroutine

```
.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"
totalMsg: .string "The total is: "
outputOrMsg: .string "\nThe result of bitwise OR is: "
outputNandMsg: .string "\nThe result of bitwise NAND is: "

```

```
.text
```

```
main:
```

```

#output message 1 to the console
li a7, 4
la a0, enterMsg1
ecall

```

```

# output message 2 to the console
li a7, 4
la a0, enterMsg2

```

```

ecall

# read an integer from keyboard input and store in s0
li a7, 5
ecall
mv s0, a0

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

li a7, 4
la a0, enterMsg3
ecall

# then receive the input and store in s1
li a7, 5
ecall
mv s1,a0

#perform operations

## add the two values together and store in s2
add s2,s0,s1
#bitwise OR on s0 and s1, stored in s3
or s3,s0,s1
#bitwise NAND on s0 and s1, stored in s4
and s4,s0,s1
not s4,s4

#output total message
la a0, totalMsg

# output the value from s2
mv a1, s2
#store return address at ra and jump to print function
jal ra, printMsgAndValue

#output the OR message and result
la a0, outputOrMsg
#output value from s3
mv a1, s3
#store return address at ra and jump to print function
jal ra, printMsgAndValue

#output the NAND message and result
la a0, outputNandMsg
#output value from s4
mv a1,s4
#store return address at ra and jump to print function
jal ra, printMsgAndValue

```

```

#endif program
li a7, 10
ecall

#subroutine for outputting a message and value

printMsgAndValue:

#output message stored in a0
li a7,4
ecall

#output the value stored in a1
mv a0,a1
li a7,1
ecall

ret

```

5.3 Task 2.5 Swapping numbers

```

.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"
sortedMsg: .string "Here are the sorted values: "

.text

main:

#output message 1 to the console
li a7, 4
la a0, enterMsg1
ecall

# output message 2 to the console
li a7, 4
la a0, enterMsg2
ecall

# read an integer from keyboard input and store in s0
li a7, 5
ecall
mv s0, a0

# output the text asking for the next number to the console
li a7, 4
la a0, enterMsg3
ecall

```

```
# then receive the input and store in s1
li a7, 5
ecall
mv s1,a0
```

```
#if s0>s1 set t1 to 1
sgt t1,s0,s1
#branch to sorted if t1 does not equal zero
bnez t1, sorted
```

```
#swap the values using XOR
xor s0,s0,s1
xor s1,s0,s1
xor s0,s0,s1
```

```
#subroutine for when numbers are sorted
sorted:
```

```
#output message
li a7,4
la a0,sortedMsg
ecall
```

```
#output s0
li a7,1
mv a0,s0
ecall
```

```
#output character
li a7,11
li a0,45
ecall
```

```
#output s1
li a7,1
mv a0,s1
ecall
```

```
#exit
li a7,10
ecall
```

5.4 Task 2.6 Loop

```
.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"
sortedMsg: .string "Here are the sorted values: "
```

```
.text
```

```
main:
```

```

#output message 1 to the console
li a7, 4
la a0, enterMsg1
ecall

# output message 2 to the console
li a7, 4
la a0, enterMsg2
ecall

# read an integer from keyboard input and store in s0
li a7, 5
ecall
mv s0, a0

# output the text asking for the next number to the console
li a7, 4
la a0, enterMsg3
ecall

# then receive the input and store in s1
li a7, 5
ecall
mv s1,a0

#if s0>s1 set t1 to 1
sgt t1,s0,s1
#branch to sorted if t1 does not equal zero
bnez t1, sorted

#swap the values using XOR
xor s0,s0,s1
xor s1,s0,s1
xor s0,s0,s1

#subroutine for when numbers are sorted
sorted:

loop:
    #call subroutine to output value
    jal ra,printf
    #subtract 1 from s0 and store in s0
    addi s0,s0,-1
    #set t1 to 1 if s0>s1
    sgt t1,s0,s1
    #branch to label loop if t1 is not zero
    bnez,t1,loop

loopdone:

```

```
#call subroutine to output value
jal ra,printf
```

```
#end program
li a7,10
ecall
```

```
printf:
#output s0
li a7,1
mv a0,s0
ecall
#output whitespace
li a7,11
li a0,32
ecall
```

```
ret
```

5.5 Task 2.7 Array

```
#variable map
# s11 for array index counter
# s0 for user input
# s1 for user input
# s2 base address
# s3 total items (so output loop knows when to stop)
```

```
.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"
sortedMsg: .string "Here are the sorted values: "
.align 2
#100 bytes for storage of 100 integers
emptyArray: .space 400
```

```
.text
```

```
main:
```

```
#get base address of emptyArray
la,s2,emptyArray
```

```
#output message 1 to the console
li a7, 4
la a0, enterMsg1
ecall
```

```
# output message 2 to the console
```

```
li a7, 4
la a0, enterMsg2
ecall
```

```
# read an integer from keyboard input and store in s0
li a7, 5
ecall
mv s0, a0
```

```
# output the text asking for the next number to the console
li a7, 4
la a0, enterMsg3
ecall
```

```
# then receive the input and store in s1
li a7, 5
ecall
mv s1,a0
```

```
#if s0>s1 set t1 to 1
sgt t1,s0,s1
#branch to sorted if t1 does not equal zero
bnez t1, sorted
```

```
#swap the values using XOR
xor s0,s0,s1
xor s1,s0,s1
xor s0,s0,s1
```

```
#subroutine for when numbers are sorted
sorted:
```

```
#first loop which stores the countdown values in the array
firstLoop:
```

```
#branch to reset index counter if all values in range have been stored
blt s0,s1,firstLoopEnd
#jump to input array
jal ra,inputArray
#subtract 1 from s0 and store in s0
addi s0,s0,-1
#branch unconditionally to first loop
b firstLoop
```

```
firstLoopEnd:
```

```
#save total amount of items
mv s3,s11
```

```
#reset index
```

```
li $11,0

secondLoop:

#branch to loop done if index counter = total items
beq $11,$3,loopDone
#jump to output array
jal ra,outputArray
#branch unconditionally
b secondLoop
```

```
loopDone:
#end program
li a7,10
ecall
```

```
inputArray:
#multiply array index by 4
slli $11,$11,2
#add offset to base address
add $11,$2,$11
#store value at the address
sw $0, 0($11)
#increment counter
addi $11,$11,1
```

```
ret
```

```
outputArray:
#multiply array index by 4
slli $11,$11,2
#add offset to base address
add $11,$2,$11
#output value at the address
lw a0,0($11)
li a7,1
ecall
#output whitespace
li a7,11
li a0,32
ecall
#increment counter
addi $11,$11,1
```

```
ret
```

5.6 Task 2.8 Creating a console based game

5.6.1 Stage 1

```
#RPG GAME
```

```

#User starts in menu and is asked to select a character from: knight, wizard,
musketeer

#User has 3 attributes: health,damage and healing
#3 = highest attribute 1 = lowest attribute
#index 1 is health, index 2 damage, index 3 healing
#Knight: 3,2,1
#Musketeer: 1,3,2
#Wizard: 2,1,3
#Attributes are stored in .word arrays

#After user selects character and enters the game, a random enemy spawns from: goblin,
golem, witch
#3 = highest attribute 1 = lowest attribute
#index 1 is health, index 2 damage, index 3 healing
#Goblin; 1,3,1
#Golem: 3,1,1
#Witch: 1,1,3

#After 5 fights, Slime boss appears
#index 1 is health, index 2 damage, index 3 healing
#Slime: 3,3,3

#-----
-----

#Variable map
# s0 = currentPlayer pointer
# s1 = currentEnemy pointer

#-----
-----

.data

#Character and enemy name strings

#Menu strings
menuStr1: .string "Castle Defence 2025\n"
menuStr2: .string "1 - Start Game\n"
menuStr3: .string "2 - Exit Game\n"

#-----
-----

#Character and enemy strings
knightStr: .string "Knight"
musketeerStr: .string "Musketeer"
wizardStr: .string "Wizard"
goblinStr: .string "Goblin"
golemStr: .string "Golem"
witchStr: .string "Witch"

```

```

#-----
-----

#chooseCharacter strings
chooseBuildStr: .string "\nChoose your build\n\n"

#Choose Knight Strings
chooseKnightStr1: .string "1 - Knight\n"
chooseKnightStr2: .string "Attributes: Health Damage Healing\n"
chooseKnightStr3: .string "           3     2     1     \n\n"
#Choose Musketeer Strings
chooseMusketeerStr1: .string "2 - Musketeer\n"
chooseMusketeerStr2: .string "Attributes: Health Damage Healing\n"
chooseMusketeerStr3: .string "           1     3     2     \n\n"
#Choose Wizard Strings
chooseWizardStr1: .string "3 - Wizard\n"
chooseWizardStr2: .string "Attributes: Health Damage Healing\n"
chooseWizardStr3: .string "           2     1     3     \n\n"

#-----
-----


#Take input string
takeInput: .string "Please Select: "
#Invalid input string
invalidInput: .string "Invalid input\n"

#-----
-----


.align 2 #tells RARS 4-byte boundary

#Format to allow easy and consistent data access across all arrays
#label : health, damage, healing, name pointer

#Character arrays
knight: .word 3, 2, 1, knightStr
musketeer: .word 1, 3, 2, musketeerStr
wizard: .word 2, 1, 3, wizardStr

#Enemy arrays
goblin: .word 1, 3, 1, goblinStr
golem: .word 3, 1, 1, golemStr
witch: .word 1, 1, 3, witchStr

#Current arrays
#16 bytes to hold 4 x 4-byte pieces of data
currentPlayer: .space 16
currentEnemy: .space 16

.text
#-----
-----
```

```

main:

#load pointers to currentPlayer and currentEnemy to s0 and s1 respectively

la s0, currentPlayer
la s1, currentEnemy

#output menu strings and take input to system

li a7,4

la a0,menuStr1
ecall
la a0,menuStr2
ecall
la a0,menuStr3
ecall
la a0,takeInput
ecall

#get user input
li a7,5
ecall
mv t0,a0

#validate user input

#if input 1 or 2 branch respectively
li t1,1
beq t1,t0,chooseCharacter
li t1,2
beq t1,t0,exit

#else return to main
li a7,4
la a0,invalInput
ecall
b main

#-----
-----

gameLoop:

b exit

#-----
-----


chooseCharacter:

#print character selection information
jal ra chooseCharacterStrings

```

```

#output take input string and then take input
la a0, takeInput
li a7,4
ecall
li a7,5
ecall
#move user input for character selection to t0
mv t0,a0
#compare user input with value in t1 and branch appropriately
li t1,1
beq t1,t0,selectKnight
li t1,2
beq t1,t0,selectMusketeer
li t1,3
beq t1,t0,selectWizard
#else output inval message and loop to chooseCharacter
li a7,4
la a0,invalInput
ecall
b chooseCharacter

```

copyFromArray:

```

#t1 is pointer to selected character array

#load health multiplier
li t5,50
#copy health and save in currentXX array (s0)
lw t4,0(t1)
mul t4,t5,t4
sw t4,0(t6)

#load damage multiplier
li t5,20
#copy damage and save in currentXX array (s0)
lw t4,4(t1)
mul t4,t5,t4
sw t4,4(t6)

#load healing multiplier
li t5,20
#copy healing and save in currentXX array (s0)
lw t4,8(t1)
mul t4,t5,t4
sw t4,8(t6)

#copy name address and save in currentXX array (s0)
lw t4,12(t1)
sw t4,12(t6)

ret

```

```

#-----
-----#functions
#-----
-----

#get function to get random enemy spawn
getEnemySpawnID:

#get a random number within the range 0 and 2
li a7,42
li a0,0
li a1,3
ecall

#check random number and branch appropriately

mv t0,a0
li t1,0
beq t1,t0,selectGoblin
li t1,1
beq t1,t0,selectGolem
li t1,2
beq t1,t0,selectWitch

#else try again

b getEnemySpawnID

#-----
-----#functions to load Character array pointers to t1 for copyFromArray

selectKnight:
la t1,knight
mv t6,s0
jal ra copyFromArray
#branch to getEnemySpawnID for enemy spawning
b getEnemySpawnID

selectMuskeeter:
la t1,muskeeter
mv t6,s0
jal ra copyFromArray
#branch to getEnemySpawnID for enemy spawning
b getEnemySpawnID

selectWizard:
la t1,wizard
mv t6,s0
jal ra copyFromArray
#branch to getEnemySpawnID for enemy spawning
b getEnemySpawnID

```

```

#-----
-----

#functions to load enemy array pointers to t1 for set copyFromArray

selectGoblin:
la t1,goblin
mv t6,s1
jal ra copyFromArray
b gameLoop

selectGolem:
la t1,golem
mv t6,s1
jal ra copyFromArray
b gameLoop

selectWitch:
la t1,witch
mv t6,s1
jal ra copyFromArray
b gameLoop

#-----
-----

#function to print all strings for chooseCharacter without cluttering
chooseCharacterStrings:

#load print string syscall to a7
li a7,4
#load pointers to a0 and print
la a0,chooseBuildStr
ecall

la a0,chooseKnightStr1
ecall

la a0,chooseKnightStr2
ecall

la a0,chooseKnightStr3
ecall

la a0,chooseMusketeerStr1
ecall

la a0,chooseMusketeerStr2
ecall

la a0,chooseMusketeerStr3
ecall

la a0,chooseWizardStr1
ecall

```

```

la a0,chooseWizardStr2
ecall

la a0,chooseWizardStr3
ecall
#return to choose character
ret

#-----
-----
```

#function to exit
exit:
li a7,10
ecall

5.6.2 Stage 2

#RPG GAME

```

#User starts in menu and is asked to select a character from: knight, wizard,
musketeer

#User has 3 attributes: health,damage and healing
#3 = highest attribute 1 = lowest attribute
#index 1 is health, index 2 damage, index 3 healing
#Knight: 3,2,1
#Musketeer: 1,3,2
#Wizard: 2,1,3
#Attributes are stored in .word arrays

#After user selects character and enters the game, a random enemy spawns from: goblin,
golem, witch
#3 = highest attribute 1 = lowest attribute
#index 1 is health, index 2 damage, index 3 healing
#Goblin; 1,3,1
#Golem: 3,1,1
#Witch: 1,1,3

#After 5 fights, Slime boss appears
#index 1 is health, index 2 damage, index 3 healing
#Slime: 3,3,3

#-----
-----
```

#Variable map
s0 = currentPlayer pointer
s1 = currentEnemy pointer
#s3 = round count variable

```

#-----
-----
```

.data


```

firstRoundStr2: .string " HAS APPEARED\n\n"

#not first strings
notFirstStr1: .string "What will you do? \n\n"
notFirstStr2: .string "1 - Attack\n"
notFirstStr3: .string "2 - Heal\n\n"
notFirstStr4: .string "Health: "
#-----
-----#
#strings for player win/loss

playerWinsStr: .string "You defeated the enemy "
playerLosesStr: .string "You were defeated by the enemy "

#-----
-----#
#strings for enemy and player actions

#print user damage

userDmgStr1: .string "You dealt "
userDmgStr2: .string " amount of damage\n"
userDmgStr3: .string "Enemy "
userDmgStr4: .string " has "
userDmgStr5: .string " health left\n"

#prints user heal

userHealStr1: .string "You healed for "
userHealStr2: .string " amount of health\n"
userHealStr3: .string "Your total health is "

#prints enemy damage

enemyDmgStr1: .string "You were dealt "
enemyDmgStr2: .string " amount of damage by enemy\n"

#prints enemy heal

enemyHealStr1: .string "Enemy "
enemyHealStr2: .string " healed for "
enemyHealStr3: .string " amount of healing\n"

#-----
-----#
.align 2 #tells RARS 4-byte boundary

#Format to allow easy and consistent data access across all arrays
#label : health, damage, healing, name pointer

#Character arrays
knight: .word 3, 2, 1, knightStr
musketeer: .word 1, 3, 2, musketeerStr
wizard: .word 2, 1, 3, wizardStr

#Enemy arrays

```

```

goblin: .word 1, 3, 1, goblinStr
golem: .word 3, 1, 1, golemStr
witch: .word 1, 1, 3, witchStr

#Current arrays
#16 bytes to hold 4 x 4-byte pieces of data
currentPlayer: .space 16
currentEnemy: .space 16

.text

#-----
-----

main:

#load pointers to currentPlayer and currentEnemy to s0 and s1 respectively

la s0, currentPlayer
la s1, currentEnemy

#output menu strings and take input to system

li a7,4

la a0,menuStr1
ecall
la a0,menuStr2
ecall
la a0,menuStr3
ecall
la a0,takeInput
ecall

#get user input
li a7,5
ecall
mv t0,a0

#validate user input

#if input 1 or 2 branch respectively
li t1,1
beq t1,t0,chooseCharacter
li t1,2
beq t1,t0,exit

#else return to main
li a7,4
la a0,invalInput
ecall
b main

#-----
-----

```

```

gameLoop:

endBattleCheck:

#get user health stat
lw t1,0($0)
#get enemy health stat
lw t2,0($1)

#branch to player win/loses respectively

blez t1,userLosesBattle
blez t2,userWinsBattle

#else
ret

battleLoop:

userTurn:

#check if first round
firstRoundCheck:
bneq $3,notFirst

#only output these strings if it is the first round of the fight

firstRound:
#prints first round strings
jal ra, firstRoundPrint


notFirst:
#check if enemy or player health is zero
jal ra,endBattleCheck
#use print function to print strings
jal ra,notFirstPrint
#take user input and store in t0
li a7,5
ecall
#move user input to safe place
mv t0,a0
#move pointer to array addresses to a0 and a1 for use in apply damage and apply heal
mv a0,s0
mv a1,s1
#li 1 and 2 for checking user input for branching
li t1,1
li t2,2
#perform branch checks and branch to label otherwise loop
beq t0,t1,useApplyDmg
beq t0,t2,useApplyHeal
#print invalid input message and loop
li a7,4

```

```

la a0,invalInput
ecall
b userTurn

useApplyDmg:
#apply damage and print strings, then branch to enemy turn
jal ra applyDmg
jal ra printUserDmg
jal ra endBattleCheck
b enemyTurn

useApplyHeal:
#apply heal and strings, then branch to enemy turn
jal ra applyHeal
jal ra printUserHeal
jal ra endBattleCheck
b enemyTurn

enemyTurn:
#get random number 0 or 1 and store in t0
li t2,2
jal ra getRandom
mv t0,a0
#move pointers for use in apply damage/heal
mv a0,s1
mv a1,s0
#load immediate 0 and 1 for apply damage/heal
li t1,0
li t2,1
beq t0,t1,enemyApplyDmg
beq t0,t2,enemyApplyHeal

enemyApplyDmg:
#print and apply damage
jal ra applyDmg
jal ra printEnemyDmg
jal ra endBattleCheck
#increment round count
addi s3,s3,1
#branch to end battle check
b userTurn
enemyApplyHeal:
jal ra applyHeal
jal ra printEnemyHeal
jal ra endBattleCheck
addi s3,s3,1
b userTurn

-----
-----

chooseCharacter:
#print character selection information

```

```

jal ra chooseCharacterStrings

#output take input string and then take input
la a0, takeInput
li a7,4
ecall
li a7,5
ecall
#move user input for character selection to t0
mv t0,a0
#compare user input with value in t1 and branch appropriately
li t1,1
beq t1,t0,selectKnight
li t1,2
beq t1,t0,selectMusketeer
li t1,3
beq t1,t0,selectWizard
#else output inval message and loop to chooseCharacter
li a7,4
la a0,invalInput
ecall
b chooseCharacter

```

```

#-----
-----#
#functions
#-----
-----#

```

```

#gameLoop print functions

firstRoundPrint:
#print first round strings
li a7,4
la a0,firstRoundStr1
ecall
#get name of enemy and print
lw a0,12($1)
ecall

la a0,firstRoundStr2
ecall

ret

```

```

notFirstPrint:

#print notFirst strings
li a7,4
la a0,notFirstStr1
ecall
la a0,notFirstStr2
ecall
la a0,notFirstStr3
ecall

```

```

la a0,notFirstStr4
ecall

#load player health value
li a7,1
lw a0,0($0)
ecall

li a7,4
la a0,newLine
ecall

#print take input
li a7,4
la a0,newLine
ecall
la a0,takeInput
ecall

ret
#-----
-----

#print functions for user and enemy damage/heal

#print user damage

printUserDmg:

li a7,4
la a0,newLine
ecall
#print "You dealt [damage] amount of damage"

la a0,userDmgStr1
ecall
#get player damage using offset 4
li a7,1
lw a0,4($0)
ecall

li a7,4
la a0,userDmgStr2
ecall

#print "Enemy [enemy name] has [enemy health] health left"
la a0,userDmgStr3
ecall
#get enemy name pointer using offset 12
lw a0,12($1)
ecall

la a0,userDmgStr4
ecall
#print enemy health value
li a7,1

```

```

lw a0,0($1)
ecall

li a7,4
la a0,userDmgStr5
ecall

li a7,4
la a0,newLine
ecall

ret

#print user heal

printUserHeal:

li a7,4
la a0,newLine
ecall

#print "You healed for [healing] amount of health"

li a7,4
la a0,userHealStr1
ecall
#get healing using offset 8
li a7,1
lw a0,8($0)
ecall

li a7,4
la a0, userHealStr2
ecall

#print "Your total health is [health]"

la a0,userHealStr3
ecall
#get total health using offset 0
li a7,1
lw a0,0($0)
ecall

li a7,4
la a0,newLine
ecall

ret

#print enemy damage

printEnemyDmg:

```

```

li a7,4
la a0,newLine
ecall
#print "You were dealt [damage] amount of damage by enemy"
li a7,4
la a0,enemyDmgStr1
ecall

#get enemy damage stat using offset 4 of
li a7,1
lw a0,4($1)
ecall

li a7,4
la a0,enemyDmgStr2
ecall

li a7,4
la a0,newLine
ecall

ret

printEnemyHeal:

li a7,4
la a0,newLine
ecall
#print "Enemy [enemy name] healed for [healing] amount of healing"
li a7,4
la a0,enemyHealStr1
ecall

#get enemy name pointer using offset 12
lw a0,12($1)
ecall

la a0,enemyHealStr2
ecall

#get enemy healing stat using offset 8
li a7,1
lw a0,8($1)
ecall

li a7,4
la a0,enemyHealStr3
ecall

li a7,4
la a0,newLine
ecall

```

```

ret

userWinsBattle:

li a7,4
la a0,newLine
ecall
#print "You defeated the enemy [enemy name]"
li a7,4
la a0,playerWinsStr
ecall

#get enemy name pointer using offset 12
lw a0,12($1)
ecall

li a7,4
la a0,newLine
ecall

#exit program
b exit

userLosesBattle:
li a7,4
la a0,newLine
ecall

#print "You were defeated by the enemy [enemy name]"
li a7,4
la a0,playerLosesStr
ecall

#get enemy name pointer using offset 12 of
lw a0,12($1)
ecall

li a7,4
la a0,newLine
ecall

#exit program
b exit
#-----
-----#
#functions to apply damage and healing

#attacker address/healer address = a0
#target address for attack = a1

applyDmg:

#get attacker damage stat
lw t2,4(a0)

```

```

#get target health
lw t3,0(a1)
#subtract attack from health and update
sub t3,t3,t2
sw t3, 0(a1)

ret

applyHeal:

#get healing stat
lw t2,8(a0)
#get healers health
lw t3,0(a0)
#apply healing and update
add t3,t3,t2
sw t3,0(a0)
ret

-----
-----
```

#function to copy from one array to another

```

copyFromArray:

#t1 is pointer to selected character array
#t6 is array to be copied to

#load health multiplier
li t5,50
#copy health and save in currentXX array (s0)
lw t4,0(t1)
mul t4,t5,t4
sw t4,0(t6)

#load damage multiplier
li t5,20
#copy damage and save in currentXX array (s0)
lw t4,4(t1)
mul t4,t5,t4
sw t4,4(t6)

#load healing multiplier
li t5,20
#copy healing and save in currentXX array (s0)
lw t4,8(t1)
mul t4,t5,t4
sw t4,8(t6)

#copy name address and save in currentXX array (s0)
lw t4,12(t1)
sw t4,12(t6)

ret

```

```

#-----
-----

#function to get random enemy spawn
getRandom:

#get a random number within the range 0 and t2
#t2 is upper range of random
li a7,42
li a0,0
mv a1,t2
ecall

ret

#-----
-----


#check random number and branch appropriately
spawnEnemy:

mv t0,a0
li t1,0
beq t1,t0,selectGoblin
li t1,1
beq t1,t0,selectGolem
li t1,2
beq t1,t0,selectWitch

#else try again

b getRandom

#-----
-----


#functions to load Character array pointers to t1 for copyFromArray

selectKnight:
la t1,knight
mv t6,s0
jal ra copyFromArray
#branch to getEnemySpawnID for enemy spawning
li t2,3
jal ra getRandom
b spawnEnemy

selectMusketeer:
la t1,musketeer
mv t6,s0
jal ra copyFromArray
#branch to getEnemySpawnID for enemy spawning
li t2,3
jal ra getRandom

```

```

b spawnEnemy

selectWizard:
la t1,wizard
mv t6,s0
jal ra copyFromArray
#branch to getEnemySpawnID for enemy spawning
li t2,3
jal ra getRandom
b spawnEnemy

#-----
-----

#functions to load enemy array pointers to t1 for set copyFromArray

selectGoblin:
la t1,goblin
mv t6,s1
jal ra copyFromArray
b battleLoop

selectGolem:
la t1,golem
mv t6,s1
jal ra copyFromArray
b battleLoop

selectWitch:
la t1,witch
mv t6,s1
jal ra copyFromArray
b battleLoop

#-----
-----

#function to print all strings for chooseCharacter without cluttering
chooseCharacterStrings:

#load print string syscall to a7
li a7,4
#load pointers to a0 and print
la a0,chooseBuildStr
ecall

la a0,chooseKnightStr1
ecall

la a0,chooseKnightStr2
ecall

la a0,chooseKnightStr3
ecall

la a0,chooseMusketeerStr1
ecall

```

```

la a0,chooseMusketeerStr2
ecall

la a0,chooseMusketeerStr3
ecall

la a0,chooseWizardStr1
ecall

la a0,chooseWizardStr2
ecall

la a0,chooseWizardStr3
ecall
#return to choose character
ret

#-----
-----
```

```

#function to exit
exit:
li a7,10
ecall
```

5.6.3 Stage 3

#RPG GAME

#User starts in menu and is asked to select a character from: knight, wizard, musketeer

#User has 3 attributes: health,damage and healing
#3 = highest attribute 1 = lowest attribute
#index 1 is health, index 2 damage, index 3 healing
#Knight: 3,2,1
#Musketeer: 1,3,2
#Wizard: 2,1,3
#Attributes are stored in .word arrays

#After user selects character and enters the game, a random enemy spawns from: goblin, golem, witch
#3 = highest attribute 1 = lowest attribute
#index 1 is health, index 2 damage, index 3 healing

```

#goblin: 1,3,1
#golem: 3,1,1
#witch: 1,1,3

#After 5 fights, Slime boss appears
#index 1 is health, index 2 damage, index 3 healing
$slime: 3,3,3

#-----
-----

#Variable map
# s0 = currentPlayer pointer
# s1 = currentEnemy pointer
# s3 = round count variable
# s4 = battle counter

#-----
-----


.data

#Character and enemy name strings

#Menu strings
menuStr1: .string "Castle Defence 2025\n"
menuStr2: .string "1 - Start Game\n"
menuStr3: .string "2 - Exit Game\n"

#-----
-----


#Character and enemy strings
knightStr: .string "Knight"
musketeerStr: .string "Musketeer"
wizardStr: .string "Wizard"
goblinStr: .string "Goblin"
golemStr: .string "Golem"
witchStr: .string "Witch"
slimeStr: .string "Boss Slime"

#-----
-----


#chooseCharacter strings
chooseBuildStr: .string "\nChoose your build\n\n"

#Choose Knight Strings
chooseKnightStr1: .string "1 - Knight\n"
chooseKnightStr2: .string "Attributes: Health Damage Healing\n"
chooseKnightStr3: .string "          3      2      1      \n\n"
#Choose Musketeer Strings
chooseMusketeerStr1: .string "2 - Musketeer\n"

```

```

chooseMuskeeterStr2: .string "Attributes: Health Damage Healing\n"
chooseMuskeeterStr3: .string " 1   3   2 \n\n"
#Choose Wizard Strings
chooseWizardStr1: .string "3 - Wizard\n"
chooseWizardStr2: .string "Attributes: Health Damage Healing\n"
chooseWizardStr3: .string " 2   1   3 \n\n"

#-----
-----

#Take input string
takeInput: .string "Please Select: "
#Invalid input string
invalInput: .string "Invalid input\n"
#new line
newLine: .string "\n"

#-----
-----

#gameLoop strings

#first round strings
firstRoundStr1: .string "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\nA "
firstRoundStr2: .string " HAS APPEARED\n\n"

#not first strings
notFirstStr1: .string "What will you do? \n\n"
notFirstStr2: .string "1 - Attack\n"
notFirstStr3: .string "2 - Heal\n\n"
notFirstStr4: .string "Health: "
#-----
-----

#strings for player win/loss and end game win/loss

playerWinsStr: .string "You defeated the enemy "
playerLosesStr: .string "You were defeated by the enemy "

endWinStr: .string "Well done you won the game"
endLossStr: .string "Well done you lost the game"

#-----
-----

#strings for enemy and player actions

#print user damage

userDmgStr1: .string "You dealt "
userDmgStr2: .string " amount of damage\n\n"
userDmgStr3: .string "Enemy "
userDmgStr4: .string " has "
userDmgStr5: .string " health left"

#prints user heal

userHealStr1: .string "You healed for "

```

```

userHealStr2: .string " amount of health\n\n"
userHealStr3: .string "Your total health is "

#prints enemy damage

enemyDmgStr1: .string "You were dealt "
enemyDmgStr2: .string " amount of damage by enemy\n"

#prints enemy heal

enemyHealStr1: .string "Enemy "
enemyHealStr2: .string " healed for "
enemyHealStr3: .string " amount of healing\n"

#-----
-----.
.align 2 #tells RARS 4-byte boundary

#Format to allow easy and consistent data access across all arrays
#label : health, damage, healing, name pointer

#Character arrays
knight: .word 3, 2, 1, knightStr
musketeer: .word 1, 3, 2, musketeerStr
wizard: .word 2, 1, 3, wizardStr

#Enemy arrays
goblin: .word 1, 3, 1, goblinStr
golem: .word 3, 1, 1, golemStr
witch: .word 1, 1, 3, witchStr
slime: .word 3,3,1, slimeStr

#Current arrays
#16 bytes to hold 4 x 4-byte pieces of data
currentPlayer: .space 16
currentEnemy: .space 16

.text

#-----
-----

main:

#load pointers to currentPlayer and currentEnemy to s0 and s1 respectively

la s0, currentPlayer
la s1, currentEnemy
li s3,0
li s4,0

#output menu strings and take input to system

li a7,4

la a0,menuStr1

```

```

ecall
la a0,menuStr2
ecall
la a0,menuStr3
ecall
la a0,takeInput
ecall

#get user input
li a7,5
ecall
mv t0,a0

#validate user input

#if input 1 or 2 branch respectively
li t1,1
beq t1,t0,chooseCharacter
li t1,2
beq t1,t0,exit

#else return to main
li a7,4
la a0,invalInput
ecall
b main

#-----
-----

gameLoop:

endBattleCheck:

#get user health stat
lw t1,0($0)
#get enemy health stat
lw t2,0($1)

#branch to player win/loses respectively

blez t1,userLosesBattle
blez t2,userWinsBattle

#else
ret

battleLoop:

userTurn:

#check if first round
firstRoundCheck:

```

```

bnez s3,notFirst

#only output these strings if it is the first round of the fight

firstRound:
#prints first round strings
jal ra, firstRoundPrint

notFirst:

#check if enemy or player health is zero
jal ra,endBattleCheck
#use print function to print strings
jal ra,notFirstPrint
#take user input and store in t0
li a7,5
ecall
#move user input to safe place
mv t0,a0
#move pointer to array addresses to a0 and a1 for use in apply damage and apply heal
mv a0,s0
mv a1,s1
#li 1 and 2 for checking user input for branching
li t1,1
li t2,2
#perform branch checks and branch to label otherwise loop
beq t0,t1,useApplyDmg
beq t0,t2,useApplyHeal
#print invalid input message and loop
li a7,4
la a0,invalInput
ecall
b userTurn

useApplyDmg:
#apply damage and print strings, then branch to enemy turn
jal ra applyDmg
jal ra printUserDmg
jal ra endBattleCheck
b enemyTurn

useApplyHeal:
#apply heal and strings, then branch to enemy turn
jal ra applyHeal
jal ra printUserHeal
jal ra endBattleCheck
b enemyTurn

enemyTurn:
#get random number 0 or 1 and store in t0
li t2,2
jal ra getRandom
mv t0,a0

```

```

#move pointers for use in apply damage/heal
mv a0,s1
mv a1,s0
#load immediate 0 and 1 for apply damage/heal
li t1,0
li t2,1
beq t0,t1,enemyApplyDmg
beq t0,t2,enemyApplyHeal

enemyApplyDmg:
#print and apply damage
jal ra applyDmg
jal ra printEnemyDmg
jal ra endBattleCheck
#increment round count
addi s3,s3,1
#branch to end battle check
b userTurn
enemyApplyHeal:
jal ra applyHeal
jal ra printEnemyHeal
jal ra endBattleCheck
addi s3,s3,1
b userTurn

-----
-----
```

chooseCharacter:

```

#print character selection information
jal ra chooseCharacterStrings

#output take input string and then take input
la a0, takeInput
li a7,4
ecall
li a7,5
ecall
#move user input for character selection to t0
mv t0,a0
#compare user input with value in t1 and branch appropriately
li t1,1
beq t1,t0,selectKnight
li t1,2
beq t1,t0,selectMusketeer
li t1,3
beq t1,t0,selectWizard
#else output inval message and loop to chooseCharacter
li a7,4
la a0,invalInput
ecall
b chooseCharacter
```

```

#-----
-----#
#functions
#-----
-----

#gameLoop print functions

firstRoundPrint:
#print first round strings
li a7,4
la a0,firstRoundStr1
ecall
#get name of enemy and print
lw a0,12($1)
ecall

la a0,firstRoundStr2
ecall

ret

notFirstPrint:

#print notFirst strings
li a7,4
la a0,notFirstStr1
ecall
la a0,notFirstStr2
ecall
la a0,notFirstStr3
ecall
la a0,notFirstStr4
ecall

#load player health value
li a7,1
lw a0,0($0)
ecall

li a7,4
la a0,newLine
ecall

#print take input
li a7,4
la a0,newLine
ecall
la a0,takeInput
ecall

ret
#-----
-----#


#print functions for user and enemy damage/heal

```

```

#print user damage

printUserDmg:

li a7,4
la a0,newLine
ecall
#print "You dealt [damage] amount of damage"

la a0,userDmgStr1
ecall
#get player damage using offset 4
li a7,1
lw a0,4($0)
ecall

li a7,4
la a0,userDmgStr2
ecall

#print "Enemy [enemy name] has [enemy health] health left"
la a0,userDmgStr3
ecall
#get enemy name pointer using offset 12
lw a0,12($1)
ecall

la a0,userDmgStr4
ecall
#print enemy health value
li a7,1
lw a0,0($1)
ecall

li a7,4
la a0,userDmgStr5
ecall

li a7,4
la a0,newLine
ecall

ret

#print user heal

printUserHeal:

li a7,4
la a0,newLine
ecall

#print "You healed for [healing] amount of health"

```

```

li a7,4
la a0,userHealStr1
ecall
#get healing using offset 8
li a7,1
lw a0,8($0)
ecall

li a7,4
la a0, userHealStr2
ecall

#print "Your total health is [health]"

la a0,userHealStr3
ecall
#get total health using offset 0
li a7,1
lw a0,0($0)
ecall

li a7,4
la a0,newLine
ecall

ret

#print enemy damage

printEnemyDmg:

li a7,4
la a0,newLine
ecall
#print "You were dealt [damage] amount of damage by enemy"
li a7,4
la a0,enemyDmgStr1
ecall

#get enemy damage stat using offset 4 of
li a7,1
lw a0,4($1)
ecall

li a7,4
la a0,enemyDmgStr2
ecall

li a7,4
la a0,newLine
ecall

ret

```

```

printEnemyHeal:

li a7,4
la a0,newLine
ecall
#print "Enemy [enemy name] healed for [healing] amount of healing"
li a7,4
la a0,enemyHealStr1
ecall

#get enemy name pointer using offset 12
lw a0,12($1)
ecall

la a0,enemyHealStr2
ecall

#get enemy healing stat using offset 8
li a7,1
lw a0,8($1)
ecall

li a7,4
la a0,enemyHealStr3
ecall

li a7,4
la a0,newLine
ecall

ret

userWinsBattle:

li a7,4
la a0,newLine
ecall
#print "You defeated the enemy [enemy name]"
li a7,4
la a0,playerWinsStr
ecall

#get enemy name pointer using offset 12
lw a0,12($1)
ecall

li a7,4
la a0,newLine
ecall

#reset round counter
li s3,0
#increment battle counter
addi s4,s4,1

```

```

li t1,5
#check if five battles have happened
beq t1,s4,bossBattle
#check if six battles have happened
li t1,6
beq t1,s4,endGameWin

#else

li t2,3
jal ra getRandom
b spawnEnemy

userLosesBattle:
li a7,4
la a0,newLine
ecall

#print "You were defeated by the enemy [enemy name]"
li a7,4
la a0,playerLosesStr
ecall

#get enemy name pointer using offset 12 of
lw a0,12($1)
ecall

li a7,4
la a0,newLine
ecall

#exit program
b endGameLoss
#-----
-----

#functions for boss battle and game win/loss

bossBattle:
la t1,slime
mv t6,$1
jal ra copyFromArray
b userTurn

endGameWin:

li a7,4
la a0,endWinStr
ecall
b exit

endGameLoss:
li a7,4
la a0,endLossStr
ecall

```

```

b exit

-----
#functions to apply damage and healing

#attacker address/healer address = a0
#target address for attack = a1

applyDmg:

#get attacker damage stat
lw t2,4(a0)
#get target health
lw t3,0(a1)
#subtract attack from health and update
sub t3,t3,t2
sw t3, 0(a1)

ret

applyHeal:

#get healing stat
lw t2,8(a0)
#get healers health
lw t3,0(a0)
#apply healing and update
add t3,t3,t2
sw t3,0(a0)
ret

#
-----

#function to copy from one array to another

copyFromArray:

#t1 is pointer to selected character array
#t6 is array to be copied to

#load health multiplier
li t5,50
#copy health and save in currentXX array (s0)
lw t4,0(t1)
mul t4,t5,t4
sw t4,0(t6)

#load damage multiplier
li t5,20
#copy damage and save in currentXX array (s0)
lw t4,4(t1)
mul t4,t5,t4
sw t4,4(t6)

```

```

#load healing multiplier
li t5,20
#copy healing and save in currentXX array (s0)
lw t4,8(t1)
mul t4,t5,t4
sw t4,8(t6)

#copy name address and save in currentXX array (s0)
lw t4,12(t1)
sw t4,12(t6)

ret

#-----
-----

#function to get random enemy spawn
getRandom:

#get a random number within the range 0 and t2
#t2 is upper range of random
li a7,42
li a0,0
mv a1,t2
ecall

ret

#-----
-----

#check random number and branch appropriately
spawnEnemy:

mv t0,a0
li t1,0
beq t1,t0,selectGoblin
li t1,1
beq t1,t0,selectGolem
li t1,2
beq t1,t0,selectWitch

#else try again

b getRandom

#-----
-----

#functions to load Character array pointers to t1 for copyFromArray

selectKnight:
la t1,knight
mv t6,s0

```

```

jal ra copyFromArray
#branch to getEnemySpawnID for enemy spawning
li t2,3
jal ra getRandom
b spawnEnemy

selectMusketeer:
la t1,musketeer
mv t6,s0
jal ra copyFromArray
#branch to getEnemySpawnID for enemy spawning
li t2,3
jal ra getRandom
b spawnEnemy

selectWizard:
la t1,wizard
mv t6,s0
jal ra copyFromArray
#branch to getEnemySpawnID for enemy spawning
li t2,3
jal ra getRandom
b spawnEnemy

#-----
-----

#functions to load enemy array pointers to t1 for set copyFromArray

selectGoblin:
la t1,goblin
mv t6,s1
jal ra copyFromArray
b battleLoop

selectGolem:
la t1,golem
mv t6,s1
jal ra copyFromArray
b battleLoop

selectWitch:
la t1,witch
mv t6,s1
jal ra copyFromArray
b battleLoop

#-----
-----

#function to print all strings for chooseCharacter without cluttering
chooseCharacterStrings:

#load print string syscall to a7
li a7,4
#load pointers to a0 and print
la a0,chooseBuildStr

```

```

ecall

la a0,chooseKnightStr1
ecall

la a0,chooseKnightStr2
ecall

la a0,chooseKnightStr3
ecall

la a0,chooseMusketeerStr1
ecall

la a0,chooseMusketeerStr2
ecall

la a0,chooseMusketeerStr3
ecall

la a0,chooseWizardStr1
ecall

la a0,chooseWizardStr2
ecall

la a0,chooseWizardStr3
ecall
#return to choose character
ret

#-----
-----
```

#function to exit
exit:
li a7,10
ecall

5.4.5

#RPG GAME

```
#-----
-----
```

#Variable map
s0 = currentPlayer pointer
s1 = currentEnemy pointer
s2 = XP variable
s3 = round count variable
s4 = battle counter
s5 = max XP variable
s6 = player level

```
# s7 = player max health  
# s8 = currentEnemy max health  
# s9 = dodge flag  
# s10 = dodge cooldown
```

```
#-----
```

```
.data
```

```
#Character and enemy name strings
```

```
#Menu strings
```

```
menuStr1: .string "Castle Defence 2025\n"  
menuStr2: .string "1 - Start Game\n"  
menuStr3: .string "2 - Exit Game\n"
```

```
#-----
```

```
#Character and enemy strings
```

```
knightStr: .string "Knight"  
musketeerStr: .string "Musketeer"  
wizardStr: .string "Wizard"  
goblinStr: .string "Goblin"  
golemStr: .string "Golem"  
witchStr: .string "Witch"  
slimeStr: .string "Boss Slime"
```

```
#-----
```

```
#dodge strings
```

```
dodgeCooldownStr1: .string "Dodge is on cooldown for "  
dodgeCooldownStr2: .string " rounds\n\n"
```

```
dodgeAttackStr: .string "You dodged the attack\n"
```

```
dodgeHealStr: .string "\nThe enemy did not attack, dodge ineffective"
```

```
#-----
```

```
#upgradeAttribute strings
```

```
upgAttStr1: .string "\nYou have reached level "  
upgAttStr2: .string "\n\nSelect an attribute to upgrade\n\n"  
upgAttStr3: .string "1 - Health\n"  
upgAttStr4: .string "2 - Damage\n"  
upgAttStr5: .string "3 - Healing\n\n"
```

```
upgHealthStr: .string "Your max health is now: "
```

```
upgDmgStr1: .string "You now deal: "
```

```
upgDmgStr2: .string "damage"
```

```
upgHealStr: .string "Your healing stat is now: "
```

```

#-----
#chooseCharacter strings
chooseBuildStr: .string "\nChoose your build\n\n"

#Choose Knight Strings
chooseKnightStr1: .string "1 - Knight\n"
chooseKnightStr2: .string "Attributes: Health Damage Healing\n"
chooseKnightStr3: .string "      3   2   1  \n\n"

#Choose Musketeer Strings
chooseMusketeerStr1: .string "2 - Musketeer\n"
chooseMusketeerStr2: .string "Attributes: Health Damage Healing\n"
chooseMusketeerStr3: .string "      1   3   2  \n\n"

#Choose Wizard Strings
chooseWizardStr1: .string "3 - Wizard\n"
chooseWizardStr2: .string "Attributes: Health Damage Healing\n"
chooseWizardStr3: .string "      2   1   3  \n\n"

#-----

#Take input string
takeInput: .string "Please Select: "
#Invalid input string
invalidInput: .string "\nInput is not a valid number!\n"
#new line
newLine: .string "\n"
#input out of range string
inputOOR: .string "Input is not an option!\n"
#press enter to continue
continueStr: .string "\nPress enter to continue"

#-----

#gameLoop strings

#first round strings
firstRoundStr1: .string "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\nA "
firstRoundStr2: .string " HAS APPEARED!\n\n"

#not first strings
notFirstStr1: .string "What will you do? \n\n"
notFirstStr2: .string "1 - Attack\n"
notFirstStr3: .string "2 - Heal\n"
notFirstStr4: .string "Your Health: "
notFirstStr5: .string "3 - Dodge\n\n"
notFirstStr6: .string "\nYour Level: "
#-----

#strings for player win/loss and end game win/loss

playerWinsStr: .string "You defeated the enemy "
playerLosesStr: .string "You were defeated by enemy "

endWinStr: .string "\nYou mighty soldier\n\nYou beat the game!"
endLossStr: .string "\nUnlucky brave soldier\n\nYou were defeated"

```

```

#-----
#strings for enemy and player actions

#print user damage

userDmgStr1: .string "You dealt "
userDmgStr2: .string " amount of damage\n\n"
userDmgStr3: .string "Enemy "
userDmgStr4: .string " has "
userDmgStr5: .string " health remaining"

#prints user heal

userHealStr1: .string "You healed for "
userHealStr2: .string " health\n\n"
userHealStr3: .string "Your total health is "

#prints enemy damage

enemyDmgStr1: .string "You were dealt "
enemyDmgStr2: .string " amount of damage by enemy\n"

#prints enemy heal

enemyHealStr1: .string "\nEnemy "
enemyHealStr2: .string " healed for "
enemyHealStr3: .string " amount of healing\n"

#-----
.align 2 #tells RARS 4-byte boundary

#Format to allow easy and consistent data access across all arrays
#label : health, damage, healing, name pointer

#Character arrays
knight: .word 3, 2, 1, knightStr
musketeer: .word 1, 3, 2, musketeerStr
wizard: .word 2, 1, 3, wizardStr

#Enemy arrays
goblin: .word 1, 3, 1, goblinStr
golem: .word 3, 1, 1, golemStr
witch: .word 1, 1, 3, witchStr
slime: .word 4, 1, 2, slimeStr

#Current arrays
#16 bytes to hold 4 x 4-byte pieces of data
currentPlayer: .space 16
currentEnemy: .space 16
#for input validation
inputBuffer: .space 4
continueBuffer: .space 4
.text

```

#-----

main:

#load pointers to currentPlayer and currentEnemy to s0 and s1 respectively

```
la s0, currentPlayer  
la s1, currentEnemy  
li s2,0  
li s3,0  
li s4,0  
#load max xp value  
li s5,2
```

#output menu strings and take input to system

mainLoop:

```
li a7,4
```

```
la a0,menuStr1  
ecall  
la a0,menuStr2  
ecall  
la a0,menuStr3  
ecall
```

```
#get user input  
jal ra takeInputConversion  
mv t0,a0
```

#validate user input

```
#if input 1 or 2 branch respectively  
li t1,1  
beq t1,t0,chooseCharacter  
li t1,2  
beq t1,t0,exit
```

```
#else return to main  
li a7,4  
la a0,inputOOR  
ecall  
b mainLoop
```

#-----

gameLoop:

endBattleCheck:

```
#get user health stat  
lw t1,0(s0)  
#get enemy health stat
```

```

lw t2,0($1)

#branch to player win/loses respectively

blez t1,userLosesBattle
blez t2,userWinsBattle

#else
ret

battleLoop:

userTurn:

#check if first round
firstRoundCheck:
bnez $3,notFirst

#only output these strings if it is the first round of the fight

firstRound:
#prints first round strings
jal ra, firstRoundPrint

notFirst:

#check if enemy or player health is zero
jal ra,endBattleCheck
#use print function to print strings
jal ra,notFirstPrint
#take user input and store in t0
jal ra takeInputConversion
#move user input to safe place
mv t0,a0
#move pointer to array addresses to a0 and a1 for use in apply damage and apply heal
mv a0,s0
mv a1,s1
mv t4,s7
#li 1 and 2 for checking user input for branching
li t1,1
li t2,2
li t3,3
#perform branch checks and branch to label otherwise loop
beq t0,t1,useApplyDmg
beq t0,t2,useApplyHeal
beq t0,t3,useDodge
#print invalid input message and loop
li a7,4
la a0,inputOOR
ecall

```

b userTurn

```
useApplyDmg:  
#apply damage and print strings, then branch to enemy turn  
jal ra applyDmg  
jal ra printUserDmg  
#press any to continue  
li a7,4  
la a0,continueStr  
ecall  
la a0,continueBuffer  
li a7,8  
ecall
```

```
jal ra endBattleCheck  
b enemyTurn
```

```
useApplyHeal:  
#apply heal and strings, then branch to enemy turn  
jal ra applyHeal  
jal ra printUserHeal  
#press any to continue  
li a7,4  
la a0,continueStr  
ecall  
la a0,continueBuffer  
li a7,8  
ecall
```

```
jal ra endBattleCheck  
b enemyTurn
```

```
useDodge:  
#if cooldown is active, no dodge  
bgtz s10,noDodge  
#set dodge flag to 1  
li s9,1  
#set cooldown to 3  
li s10,3  
b enemyTurn
```

```
noDodge:  
#print strings to tell user dodge cooldown info  
li a7,4  
la a0,dodgeCooldownStr1  
ecall
```

```
li a7,1  
mv a0,s10  
ecall
```

```
li a7,4  
la a0,dodgeCooldownStr2
```

```

ecall

#press any to continue
li a7,4
la a0,continueStr
ecall
la a0,continueBuffer
li a7,8
ecall

b notFirst

enemyTurn:
#get random number 0 or 1 and store in t0
li t2,2
jal ra getRandom
mv t0,a0
#move pointers for use in apply damage/heal
mv a1,s0
mv t4,s8
#load immediate 0 and 1 for apply damage/heal
li t1,0
li t2,1
beq t0,t1,enemyApplyDmg
beq t0,t2,enemyApplyHeal

enemyApplyDmg:
#check for dodge
beq s9,zero,takeDmg
li s9,0
#print dodge attack string
li a7,4
la a0,dodgeAttackStr
ecall
b skipDmg
#print and apply damage
takeDmg:
mv a0,s1
jal ra applyDmg
jal ra printEnemyDmg
skipDmg:
#press any to continue
li a7,4
la a0,continueStr
ecall
la a0,continueBuffer
li a7,8
ecall
jal ra endBattleCheck
#increment round count
addi s3,s3,1
#deal with dodge cooldown
jal ra updateCooldown
b userTurn

```

```

enemyApplyHeal:
#check for dodge
beq s9,zero, skipDodgeHeal
#print dodge failed string
li a7,4
la a0,dodgeHealStr
ecall
li s9,0
skipDodgeHeal:
mv a0,s1
jal ra applyHeal
jal ra printEnemyHeal
#press any to continue
li a7,4
la a0,continueStr
ecall
la a0,continueBuffer
li a7,8
ecall
jal ra endBattleCheck
addi s3,s3,1
#endif deal with dodge cooldown
jal ra updateCooldown
b userTurn

```

#-----

```

chooseCharacter:
#print character selection information
jal ra chooseCharacterStrings

#endif output take input string and then take input
jal ra takeInputConversion
#endif move user input for character selection to t0
mv t0,a0
#endif compare user input with value in t1 and branch appropriately
li t1,1
beq t1,t0,selectKnight
li t1,2
beq t1,t0,selectMusketeer
li t1,3
beq t1,t0,selectWizard
#else output inval message and loop to chooseCharacter
li a7,4
la a0,inputOOR
ecall
b chooseCharacter

```

```

#-----
#functions
#-----

#gameLoop print functions

firstRoundPrint:
#print first round strings
li a7,4
la a0,firstRoundStr1
ecall
#get name of enemy and print
lw a0,12($1)
ecall

la a0,firstRoundStr2
ecall

ret

notFirstPrint:

#print notFirst strings
li a7,4
la a0,notFirstStr1
ecall
la a0,notFirstStr2
ecall
la a0,notFirstStr3
ecall
la a0,notFirstStr5
ecall
la a0,notFirstStr4
ecall

#load player health value
li a7,1
lw a0,0($0)
ecall

li a7,4
la a0,notFirstStr6
ecall
#display player level
li a7,1
mv a0,s6
ecall

li a7,4
la a0,newLine
ecall

#print take input

```

```
li a7,4
la a0,newLine
ecall

ret
#-----
#print functions for user and enemy damage/heal

#print user damage

printUserDmg:

li a7,4
la a0,newLine
ecall
#print "You dealt [damage] amount of damage"

la a0,userDmgStr1
ecall
#get player damage using offset 4
li a7,1
lw a0,4($0)
ecall

li a7,4
la a0,userDmgStr2
ecall

#print "Enemy [enemy name] has [enemy health] health left"
la a0,userDmgStr3
ecall
#get enemy name pointer using offset 12
lw a0,12($1)
ecall

la a0,userDmgStr4
ecall
#print enemy health value
li a7,1
lw a0,0($1)
ecall

li a7,4
la a0,userDmgStr5
ecall

li a7,4
la a0,newLine
ecall

ret
```

```

#print user heal

printUserHeal:

li a7,4
la a0,newLine
ecall

#print "You healed for [healing] amount of health"

li a7,4
la a0,userHealStr1
ecall
#get healing using offset 8
li a7,1
lw a0,8(s0)
ecall

li a7,4
la a0, userHealStr2
ecall

#print "Your total health is [health]"

la a0,userHealStr3
ecall
#get total health using offset 0
li a7,1
lw a0,0(s0)
ecall

li a7,4
la a0,newLine
ecall

ret

#print enemy damage

printEnemyDmg:

li a7,4
la a0,newLine
ecall
#print "You were dealt [damage] amount of damage by enemy"
li a7,4
la a0,enemyDmgStr1
ecall

#get enemy damage stat using offset 4 of
li a7,1
lw a0,4(s1)

```

```
ecall  
li a7,4  
la a0,enemyDmgStr2  
ecall
```

```
li a7,4  
la a0,newLine  
ecall
```

```
ret
```

```
printEnemyHeal:
```

```
li a7,4  
la a0,newLine  
ecall  
#print "Enemy [enemy name] healed for [healing] amount of healing"  
li a7,4  
la a0,enemyHealStr1  
ecall
```

```
#get enemy name pointer using offset 12  
lw a0,12($1)  
ecall
```

```
la a0,enemyHealStr2  
ecall
```

```
#get enemy healing stat using offset 8  
li a7,1  
lw a0,8($1)  
ecall
```

```
li a7,4  
la a0,enemyHealStr3  
ecall
```

```
li a7,4  
la a0,newLine  
ecall
```

```
ret
```

```
#-----
```

```
userWinsBattle:
```

```
#battle win strings  
li a7,4  
la a0,newLine  
ecall
```

```

#print "You defeated the enemy [enemy name]"
li a7,4
la a0,playerWinsStr
ecall

#get enemy name pointer using offset 12
lw a0,12($1)
ecall

li a7,4
la a0,newLine
ecall

#reset round counter
li s3,0
#increment battle counter
addi s4,s4,1

#DEAL WITH XP
#add 2xp
addi s2,s2,2
#if player xp is greater than or equal or equal to max XP
bge s2,s5,upgradeAttribute

#label to return to after upgrade attribute
afterUpgrade:
#BOSS CHECKS
#check if five battles have happened
li t1,5
beq t1,s4,bossBattle
#check if six battles have happened
li t1,6
beq t1,s4,endGameWin

#else

li t2,3
jal ra getRandom
b spawnEnemy

userLosesBattle:
li a7,4
la a0,newLine
ecall

#print "You were defeated by the enemy [enemy name]"
li a7,4
la a0,playerLosesStr
ecall

#get enemy name pointer using offset 12 of

```

```

lw a0,12($1)
ecall

li a7,4
la a0,newLine
ecall

#exit program
b endGameLoss
#-----
#functions for boss battle and game win/loss

bossBattle:

la t1,slime
mv t6,s1
jal ra copyFromArray
b userTurn

endGameWin:

li a7,4
la a0,endWinStr
ecall
b exit

endGameLoss:
li a7,4
la a0,endLossStr
ecall
b exit

#-----
#functions to apply damage and healing

#attacker address/healer address = a0
#target address for attack = a1

applyDmg:

#get attacker damage stat
lw t2,4(a0)
#get target health
lw t3,0(a1)
#subtract attack from health and update
sub t3,t3,t2
#deal with resulting health less than zero
li t6,0
#if 0 is less than health, continue
blt t6,t3,continueDmg
#set t3 to zero if health is below zero
li t3,0
continueDmg:

```

```

sw t3,0(a1)
ret

applyHeal:
#t4 is max healing stat

#get healing stat
lw t2,8(a0)
#get healers health
lw t3,0(a0)
#apply healing and update
add t3,t3,t2
#check if healing is greater than max health
blt t3,t4,continueHeal
mv t3,t4

continueHeal:
sw t3,0(a0)
ret

#-----
#function to copy from one array to another

copyFromArray:

#t1 is pointer to selected character array
#t6 is array to be copied to

#load health multiplier
li t5,10
#copy health and save in currentXX array (s0)
lw t4,0(t1)
mul t4,t5,t4
#add base stat
addi t4,t4,100
sw t4,0(t6)

#load damage multiplier
li t5,4
#copy damage and save in currentXX array (s0)
lw t4,4(t1)
mul t4,t5,t4
#add base stat
addi t4,t4,40
sw t4,4(t6)

#load healing multiplier
li t5,6
#copy healing and save in currentXX array (s0)
lw t4,8(t1)
mul t4,t5,t4

```

```
#add base stat  
addi t4,t4,20  
sw t4,8(t6)  
  
#copy name address and save in currentXX array (s0)  
lw t4,12(t1)  
sw t4,12(t6)  
  
ret
```

```
#-----  
#function to get random enemy spawn  
getRandom:
```

```
#get a random number within the range 0 and t2  
#t2 is upper range of random  
li a7,42  
li a0,0  
mv a1,t2  
ecall
```

```
ret
```

```
#-----  
#check random number and branch appropriately  
spawnEnemy:
```

```
mv t0,a0  
li t1,0  
beq t1,t0,selectGoblin  
li t1,1  
beq t1,t0,selectGolem  
li t1,2  
beq t1,t0,selectWitch
```

```
#else try again
```

```
b getRandom
```

```
#-----  
#functions to load Character array pointers to t1 for copyFromArray
```

```
selectKnight:  
la t1,knight  
mv t6,s0  
jal ra copyFromArray  
#save max health  
lw s7, 0(s0)  
#branch to getEnemySpawnID for enemy spawning
```

```
li t2,3
jal ra getRandom
b spawnEnemy

selectMusketeer:
la t1,musketeer
mv t6,s0
jal ra copyFromArray
#save max health
lw s7, 0(s0)
#branch to getEnemySpawnID for enemy spawning
li t2,3
jal ra getRandom
b spawnEnemy
```

```
selectWizard:
la t1,wizard
mv t6,s0
jal ra copyFromArray
#save max health
lw s7, 0(s0)
#branch to getEnemySpawnID for enemy spawning
li t2,3
jal ra getRandom
b spawnEnemy
```

```
#-----
```

```
#functions to load enemy array pointers to t1 for set copyFromArray
```

```
selectGoblin:
la t1,goblin
mv t6,s1
jal ra copyFromArray
lw s8, 0(s1)
b battleLoop
```

```
selectGolem:
la t1,golem
mv t6,s1
jal ra copyFromArray
lw s8, 0(s1)
b battleLoop
```

```
selectWitch:
la t1,witch
mv t6,s1
jal ra copyFromArray
lw s8, 0(s1)
b battleLoop
```

```
#-----#
#function to print all strings for chooseCharacter without cluttering
chooseCharacterStrings:
```

```

#load print string syscall to a7
li a7,4
#load pointers to a0 and print
la a0,chooseBuildStr
ecall

la a0,chooseKnightStr1
ecall

la a0,chooseKnightStr2
ecall

la a0,chooseKnightStr3
ecall

la a0,chooseMusketeerStr1
ecall

la a0,chooseMusketeerStr2
ecall

la a0,chooseMusketeerStr3
ecall

la a0,chooseWizardStr1
ecall

la a0,chooseWizardStr2
ecall

la a0,chooseWizardStr3
ecall
#return to choose character
ret

#-----
upgradeAttribute:

#add 1 to player level variable
addi s6,s6,1
#multiply max xp by 2
li t0,2
mul s5,s5,t0
#set xp to zero
mv s2,zero

#ask user what attribute they want to upgrade and branch

li a7,4

la a0,upgAttStr1
ecall

```

```

li a7,1
mv a0,s6
ecall

li a7,4

la a0,upgAttStr2
ecall

la a0,upgAttStr3
ecall

la a0,upgAttStr4
ecall

la a0,upgAttStr5
ecall

jal ra takeInputConversion
#move user input to t0
mv t0,a0
#check input and branch appropriately
li t1,1
beq t1,t0,upgHealth
li t1,2
beq t1,t0,upgDmg
li t1,3
beq t1,t0,upgHeal

#else
li a7,4
la a0,inputOOR
ecall
b upgradeAttribute

```

upgHealth:

```

#get player max health and add 10
addi s7,s7,10
#heal player to full
sw s7,0($0)
#print info to user
li a7,4
la a0,upgHealthStr
ecall

```

```

li a7,1
mv a0,s7
ecall

```

b afterUpgrade

```

upgDmg:
#get player damage and add 4

```

```
lw t3,4($0)
addi t3,t3,4
#save in currentPlayer array
sw t3,4($0)
```

```
#print info to user
li a7,4
la a0,upgDmgStr1
ecall
```

```
li a7,1
lw a0,4($0)
ecall
```

```
li a7,4
la a0,upgDmgStr2
```

```
b afterUpgrade
```

```
upgHeal:
#get player healing and add 6
lw t3,8($0)
addi t3,t3,6
#save in currentPlayer array
sw t3,8($0)
#print info to user
li a7,4
la a0,upgHealStr
ecall
```

```
li a7,1
lw a0,8($0)
ecall
```

```
b afterUpgrade
```

```
#-----
#deal with dodge cooldown
updateCooldown:
ble s10,zero,updateCooldownDone
addi s10,s10,-1
updateCooldownDone:
ret
#-----
```

```
invalidInput:
#print invalid input string
li a7,4
la a0,invalidInput
ecall
#take input and convert
takeInputConversion:
#print please select string
```

```
li a7,4
la a0,takeInput
ecall
#take input as string
la a0, inputBuffer
li a1,4
li a7,8
ecall
#load the first byte from inputBuffer to t0
la t0,inputBuffer
lbu t0,0(t0)
#subtract 0x30 (48) from a0
addi t0,t0,-48
#check if input is valid
li t1,9
bgt t0,t1,invalidInput
li t1,0
blt t0,t1,invalidInput
#move t0 to a0 and return
mv a0,t0
ret
```

```
#
#-----#
#function to exit
exit:
li a7,10
ecall
```

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.

Number	Tool	Prompt	Output																		
1	Google Gemini 2.5 Pro	I have to create a table to predict the output of 2 operations on 2 operands i need to show the working in binary and then convert to hex, what table headings could i use?	<p>Here are some clear table headings you could use, set up so you can show each operation on a separate row:</p> <ul style="list-style-type: none"> • Operand 1 (Binary) • Operand 2 (Binary) • Operation (e.g., ADD, AND, OR, XOR) • Binary Working (This is where you'd show the column-by-column calculation, carries, etc.) • Binary Result • Hexadecimal Result <hr/> <p style="text-align: center;">7. Example Table Structure:</p> <p>Here's how it would look filled out for operands 1011 and 0101 with ADD and AND operations:</p> <table border="1"> <thead> <tr> <th>Operand 1 (Binary)</th> <th>Operand 2 (Binary)</th> <th>Operation</th> <th>Binary Working</th> <th>Binary Result</th> <th>Hexadecimal Result</th> </tr> </thead> <tbody> <tr> <td>1011</td> <td>0101</td> <td>ADD</td> <td> 1 1 (carries) 1011 + 0101 ---- -- 10000 </td> <td>10000</td> <td>10</td> </tr> <tr> <td>1011</td> <td>0101</td> <td>AND</td> <td> 1011 & 0101
 </td> <td></td> <td></td> </tr> </tbody> </table>	Operand 1 (Binary)	Operand 2 (Binary)	Operation	Binary Working	Binary Result	Hexadecimal Result	1011	0101	ADD	1 1 (carries) 1011 + 0101 ---- -- 10000	10000	10	1011	0101	AND	1011 & 0101 		
Operand 1 (Binary)	Operand 2 (Binary)	Operation	Binary Working	Binary Result	Hexadecimal Result																
1011	0101	ADD	1 1 (carries) 1011 + 0101 ---- -- 10000	10000	10																
1011	0101	AND	1011 & 0101 																		
2	Google Gemini 3	For a text based game with a user playing against the computer (tic tac toe, rock paper scissors) what methods are usually used to find the winner of the game inside the ALU	<p>To understand how a computer finds a winner inside the **ALU (Arithmetic Logic Unit)**, we have to look below the level of high-level code (like Python `if/else` statements) and look at how the processor treats game states as raw numbers and bits.</p> <p>The ALU is the digital circuit inside the CPU that performs integer arithmetic and bitwise logic operations. For games like Tic-Tac-Toe and Rock-Paper-Scissors, the ALU uses three primary methods to determine a winner: **Comparison (Subtraction)**, **Bitwise Masking**, and **Lookup Table Indexing**.</p> <p>Here is how these methods work at the hardware level.</p> <p>---</p> <p>### 1. Rock-Paper-Scissors: Comparison & Modular Arithmetic</p> <p>In Rock-Paper-Scissors, the logic is circular (Rock beats Scissors, Scissors beats Paper, etc.). The ALU handles this by assigning integer values to the choices.</p>																		

		<p>##### The Setup</p> <p>Imagine the computer maps the inputs to registers:</p> <ul style="list-style-type: none"> * **Rock** = 0 * **Paper** = 1 * **Scissors** = 2 <p>##### ALU Method A: The `CMP` Instruction (Subtraction)</p> <p>The most basic method uses the **Compare (`CMP`)** instruction. Internally, the ALU performs a **subtraction** (`Player_Move - Computer_Move`) but discards the result, only keeping the "Flags" (status bits).</p> <ul style="list-style-type: none"> * **Tie Check:** The ALU performs `0 - 0` (Rock vs Rock). The result is 0. The ALU sets the **Zero Flag (ZF)** to 1. The CPU reads this flag to declare a tie. * **Win Check:** The CPU executes a series of conditional jumps based on comparisons. <p>##### ALU Method B: Modular Arithmetic (The Efficient Way)</p> <p>A more elegant way to use the ALU is to calculate the winner mathematically without multiple `if` statements. The ALU can determine the winner using one equation:</p> $\$Result = (Player - Computer + 3) \bmod 3$ <ul style="list-style-type: none"> * **If Result = 0:** It is a Tie. * **If Result = 1:** Player wins. * **If Result = 2:** Computer wins. <p>**Inside the ALU:**</p> <ol style="list-style-type: none"> 1. **ADD/SUB:** It adds 3 to the player's move, then subtracts the computer's move. 2. **DIV:** It performs integer division to get the remainder (Modulo). 3. **CMP:** It compares the remainder to 1 or 2 to set the status flags. <p>---</p> <p>### 2. Tic-Tac-Toe: Bitwise Operations (Bitboards)</p> <p>For grid-based games like Tic-Tac-Toe, checking a winner using `if` statements is slow and inefficient at the assembly level. Instead, the ALU uses **Bitboards**. This is the standard method for board game AI (including Chess).</p> <p>##### The Setup</p> <p>The ALU views the \$3 \$\times\$ 3\$ grid not as an array, but as a **single 9-bit binary number** (stored in a 16-bit or 32-bit register).</p> <ul style="list-style-type: none"> * **Player X's Register:** `000 000 000` (1 represents an X, 0 represents empty/O). * **Winning Masks:** The computer stores 8 "winning patterns" as binary constants. <ul style="list-style-type: none"> * Top Row win: `111 000 000` (Binary) = 448 (Decimal) * Diagonal win: `100 010 001` (Binary) = 273 (Decimal) <p>##### The Operation: `AND` & `CMP`</p> <p>To check if Player X has won, the ALU performs a **Bitwise AND** against the winning masks.</p>
--	--	---

Example Scenario:

Player X has pieces in the top row and bottom right corner.

1. **Player Register:** `111 000 001`

2. **Winning Mask (Top Row):** `111 000 000`

The ALU Steps:

1. **AND Operation:** The ALU takes the Player Register AND the Mask.
* `111 000 001` **AND** `111 000 000` \$\rightarrow\$ Result: `111 000 000`

2. **CMP Operation:** The ALU compares the *Result* against the *Mask*.

* Does Result (`111 000 000`) == Mask (`111 000 000`)?

3. **Zero Flag:** Since they are equal, the subtraction results in zero, setting the **Zero Flag**. The game detects a win.

This is incredibly fast because the ALU checks an entire row, column, or diagonal in a single clock cycle.

3. Lookup Tables (Memory Offset)

Sometimes, rather than calculating the winner, the ALU simply calculates an **address** to look up the answer. This is common in complex text games or optimized board games.

The Setup

Every possible board state is pre-calculated and stored in memory (RAM).

* **State:** Board looks like X-X-X (Top row).

* **Hash/Index:** The board state is converted into a unique integer (e.g., Index 504).

The ALU Operation: `ADD` (Offset Calculation)

1. **Base Address:** The program knows the memory address where the "Results Table" starts (e.g., Address 1000).

2. **Offset:** The current game state is Index 504.

3. **ALU Addition:** `1000 + 504 = 1504`.

4. **Load:** The CPU fetches the value at Address 1504. That value contains the "Winner" data.

Summary of ALU Operations

| Game | Primary ALU Logic | Primary ALU Instructions | Why? |

| --- | --- | --- | --- |

| **Rock Paper Scissors** | Comparison / Modulo | `CMP`, `SUB`, `IDIV` | Simple logic loop or circular math. |

| **Tic-Tac-Toe** | Bitwise Masking | `AND`, `OR`, `XOR` | Checks 3 cells at once; extremely fast. |

| **Complex Logic** | Address Calculation | `ADD`, `SHL` (Shift) | Fast retrieval of pre-calculated results. |

The "Hardware Flags"

Regardless of the method, the final step usually relies on the ALU's **Status Register**:

* **ZF (Zero Flag):** Used to indicate equality (Win detected).

			<p>* **SF (Sign Flag):** Used to indicate who won (Positive = Player 1, Negative = Player 2).</p> <p>**Would you like to see a specific example of the "Bitboard" logic written out in pseudocode or binary to see exactly how the `AND` gate processes it?**</p>																																													
3	Google Gemini 3	<pre>v2.0 raw # 00-07 variables # 00: GAME COUNT # 01: PLAYER SCORE # 02: COMPUTER SCORE # 03: BO3 FLAG # 04: PLAYER INPUT # 05: COMPUTER INPUT # 06: OUTCOME # 07: POINTER 00 00 00 00 00 00 00 00 #"ROCK PAPER SCISSORS\n" 52 4F 43 4B 20 50 41 50 45 52 20 53 43 49 53 53 4F 52 53 0A 00 #"PRESS BUTTON TO START\n" 50 52 45 53 53 20 42 55 54 54 4F 4E 20 54 4F 20 53 54 41 52 54 0A 00 #"SELECT A CHOICE\n" 53 45 4C 45 43 54 20 41 20 43 48 4F</pre>	<p>Based on the raw data provided, the memory addresses are calculated sequentially. The first 8 bytes (to) are reserved for variables, meaning the first string starts at address .</p> <p>Here is the start address for each string to be used for your pointers:</p> <table> <thead> <tr> <th>String Label</th> <th>Hex Address</th> <th>Decimal Address</th> </tr> </thead> <tbody> <tr> <td>ROCK PAPER SCISSORS</td> <td>08</td> <td>8</td> </tr> <tr> <td>PRESS BUTTON TO START</td> <td>1D</td> <td>29</td> </tr> <tr> <td>SELECT A CHOICE</td> <td>34</td> <td>52</td> </tr> <tr> <td>B1: Rock</td> <td>45</td> <td>69</td> </tr> <tr> <td>B2: Paper</td> <td>4F</td> <td>79</td> </tr> <tr> <td>B3: Scissors</td> <td>5A</td> <td>90</td> </tr> <tr> <td>YOU WON</td> <td>68</td> <td>104</td> </tr> <tr> <td>YOU LOST</td> <td>71</td> <td>113</td> </tr> <tr> <td>IT'S A DRAW</td> <td>7B</td> <td>123</td> </tr> <tr> <td>PRESS B1 FOR BEST OF 3</td> <td>88</td> <td>136</td> </tr> <tr> <td>PRESS ANY BUTTON FOR MENU A0</td> <td></td> <td>160</td> </tr> <tr> <td>YOU WON OVERALL</td> <td>BB</td> <td>187</td> </tr> <tr> <td>YOU LOST OVERALL</td> <td>CC</td> <td>204</td> </tr> <tr> <td>PRESS ANY KEY TO CONTINUE</td> <td>DE</td> <td>222</td> </tr> </tbody> </table>	String Label	Hex Address	Decimal Address	ROCK PAPER SCISSORS	08	8	PRESS BUTTON TO START	1D	29	SELECT A CHOICE	34	52	B1: Rock	45	69	B2: Paper	4F	79	B3: Scissors	5A	90	YOU WON	68	104	YOU LOST	71	113	IT'S A DRAW	7B	123	PRESS B1 FOR BEST OF 3	88	136	PRESS ANY BUTTON FOR MENU A0		160	YOU WON OVERALL	BB	187	YOU LOST OVERALL	CC	204	PRESS ANY KEY TO CONTINUE	DE	222
String Label	Hex Address	Decimal Address																																														
ROCK PAPER SCISSORS	08	8																																														
PRESS BUTTON TO START	1D	29																																														
SELECT A CHOICE	34	52																																														
B1: Rock	45	69																																														
B2: Paper	4F	79																																														
B3: Scissors	5A	90																																														
YOU WON	68	104																																														
YOU LOST	71	113																																														
IT'S A DRAW	7B	123																																														
PRESS B1 FOR BEST OF 3	88	136																																														
PRESS ANY BUTTON FOR MENU A0		160																																														
YOU WON OVERALL	BB	187																																														
YOU LOST OVERALL	CC	204																																														
PRESS ANY KEY TO CONTINUE	DE	222																																														

	49 43 45 0A 00	
	#"B1: Rock\n" 42 31 3A 20 52 6F 63 6B 0A 00	
	#"B2: Paper\n" 42 32 3A 20 50 61 70 65 72 0A 00	
	#"B3: Scissors\n" 42 33 3A 20 53 63 69 73 73 6F 72 73 0A 00	
	#"YOU WON\n" 59 4F 55 20 57 4F 4E 0A 00	
	#"YOU LOST\n" 59 4F 55 20 4C 4F 53 54 0A 00	
	#"IT'S A DRAW\n" 49 54 27 53 20 41 20 44 52 41 57 0A 00	
	#"PRESS B1 FOR BEST OF 3\n" 50 52 45 53 53 20 42 31 20 46 4F 52 20 42 45 53 54 20 4F 46 20 33 0A 00	

		<pre> #"PRESS ANY BUTTON FOR MENU\n" 50 52 45 53 53 20 41 4E 59 20 42 55 54 54 4F 4E 20 46 4F 52 20 4D 45 4E 55 0A 00 #"YOU WON OVERALL\n" 59 4F 55 20 57 4F 4E 20 4F 56 45 52 41 4C 4C 0A 00 #"YOU LOST OVERALL\n" 59 4F 55 20 4C 4F 53 54 20 4F 56 45 52 41 4C 4C 0A 00 #"PRESS ANY KEY TO CONTINUE\n" 50 52 45 53 53 20 41 4E 59 20 4B 45 59 20 54 4F 20 43 4F 4E 54 49 4E 55 45 0A 00 Please could you calculate the start address of each string to be used for pointers inside my computer simulation: </pre>	
3	Google Gemini 3	What is the standard practice for game developers to test game	In 2026, balancing a turn-based RPG is a multi-layered process that moves from theoretical math to high-speed automated simulations, and finally to human "feel" testing. Developers generally follow a "Bottom-Up" approach: they define the math first, then verify it with bots, and refine it with players. Here is the industry-standard workflow for testing and achieving game balance.

	<p>balance is a turn based rpg</p>	<p>8. 1. The Mathematical Foundation (The Spreadsheet Phase)</p> <p>Before a single line of combat code is written, designers use Excel or Google Sheets to model the entire game's economy and power curve.</p> <ul style="list-style-type: none"> • Establishing the "Baseline": Developers define a "Standard Unit" (e.g., a Level 1 Soldier). Every other character is balanced relative to this baseline. • TTK (Turns to Kill): This is the most critical metric. Developers decide how many turns a fight should last. <ul style="list-style-type: none"> ◦ Common Mob: 2–3 turns. ◦ Elite Enemy: 5–8 turns. ◦ Boss: 12–20 turns. • The Power Curve: Using formulas like , designers ensure that as players level up, their power grows at a rate that keeps pace with—but eventually exceeds—enemy growth. <p>9. 2. Automated Combat Simulations (The "Bot" Phase)</p> <p>Manual testing is too slow to find every edge case. Modern developers write Python scripts or C# unit tests to run thousands of "headless" battles (battles without graphics) in seconds.</p> <ul style="list-style-type: none"> • Monte Carlo Simulations: The system plays the game 10,000 times using different random seeds (critical hits, misses, damage variance). If the player wins 100% of the time or 0% of the time, the encounter is broken. • AI vs. AI: Developers program a "Simple AI" (always uses strongest attack) and a "Strategic AI" (uses buffs/heals) to fight. If the Strategic AI doesn't perform significantly better than the Simple AI, the tactical depth is too shallow. • Heat Maps of Death: Automated logs track which turn players usually die on. If everyone dies on Turn 2, the "burst damage" is too high. <p>10. 3. Testing for "Dominant Strategies"</p> <p>A game is "unbalanced" if there is one strategy that is always better than others (a Dominant Strategy).</p> <ul style="list-style-type: none"> • The Nash Equilibrium: Developers look for a state where no player can benefit by changing their strategy while others keep theirs unchanged. In RPGs, this means ensuring that "Spamming Fireball" isn't objectively better than using a mix of status effects and physical hits. • Transitivity vs. Intransitivity: * Transitivity: Level 2 is better than Level 1 (standard progression). <ul style="list-style-type: none"> ◦ Intransitivity: Fire beats Grass, Grass beats Water, Water beats Fire (Rock-Paper-Scissors). Testing ensures the "circular" balance holds up across all levels. <p>11. 4. Qualitative Playtesting (The "Feel" Phase)</p> <p>Math can be perfect but the game can still feel "unfair" or "boring." Human testers focus on the emotional arc of a battle.</p> <ul style="list-style-type: none"> • Fairness vs. Randomness: In 2026, "True Randomness" is often avoided in favor of Pseudo-Random Distribution (PRD). If a player has a 90% hit chance and misses twice, they feel cheated. Developers test "Pity Systems" that guarantee a hit after a miss to maintain player engagement. • The "Double/Half" Rule: During early testing, if an ability feels weak, developers don't buff it by 5%—they double it. If it's too strong, they halve
--	------------------------------------	--

it. This "extreme" balancing helps find the "center" much faster than small incremental tweaks.

12. Summary Table: Balancing Metrics

Metric	What it measures	Goal
Turns to Kill (TTK)	Length of encounter	Ensure pacing matches the "vibe" of the area.
Win/Loss Ratio	Difficulty	80–90% win rate for mobs; 30–50% for first-try bosses.
Action Economy	Utility of turns	Ensure "Buffing" or "Defending" is worth the loss of a "Damage" turn.
Resource Attrition	HP/MP drain over time	Ensure players run out of resources just as they reach the end of a dungeon.

Export to Sheets

Would you like me to help you draft a specific **damage formula** or a **stat-scaling spreadsheet template** for a project you're working on?