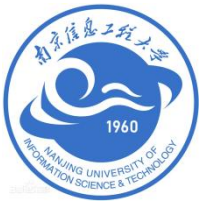

Nanjing University of Information Science and Technology

NUIST

Student Project Report

Programme Module	Computer Organization and System Structure
Title of Work Submitted	Design of a CPU(MIPS, 32bits) in Logisim
Lecturers	Hiuk Jae Shim
Name and ID Number	**Jiarui Huang_202283890036:** Building the main body of the CPU, writing about how to design the CPU, design difficulties, and final article corrections **Kaifeng Jin_202283890029:** Build functions and be responsible for connecting various parts with wires, write how the CPU works, and integrate the content of the paper **Weichen Guo_202283890027:** Search for relevant information, assist in connecting various parts with wires, construct RegisterFile, and conduct final testing on the CPU

Acknowledgement	Thanks to everyone on the team, including Kaifeng Jin，Weichen Guo，and Jiarui Huang. And our team thanks Mr.Shim for his help.

Declaration

I hereby declare that this is my original work produced without the help of any third party unless where referenced within this report.

Student: Kaifeng Jin，Weichen Guo，Jiarui Huang.

Date and Time of Submission: 17th. June. 2024

Table of Contents

Introduction

This project focuses on basic processor design and implements a single clock cycle processor, which means each instruction takes exactly one CPU cycle to finish[1]. It has the functions of ADD, OR, AND, BEQ, SUB, NAND, SLT, NOR, and shift operations in #LAB08 ALU. In addition, the processor can run LW/SW and ADD operations. In summary, we have successfully completed the design of a basic processor and deepened our understanding of datapath during the production process.

Member Assignment

Jiarui Huang_202283890036： Refer to some materials, build the main body of the CPU, and write an explanation of the processor, including the design concepts of each part of the processor, and the difficulties in designing. Then make corrections to the paper.
Kaifeng Jin_202283890029： Search for reference materials online. Build the function section and control， and use lines to connect every part. Also write about the working principle and steps of the processor. Integrate the content of the paper.
Weichen Guo_202283890027： Search for reference materials online. Mainly writing test reports for processors. Mainly build the RegisterFile, assisting in connecting every part.

Processor Explanation

1.How it was designed

ALU Design

Input
1.X and Y:
These are two 32-bit input registers that store the two operands for the ALU (X and Y). Each register has 32 binary bits.
2. opCode:
This is the operation code that determines the operation the ALU will perform. The opCode controls the multiplexer (MUX) to select the appropriate operation.

Calculate Part
1. Arithmetic and Logic Units:
Multiplication: Used to perform multiplication of X and Y. The part has two input, and a c_out to determine the overflow bits, the result will access mux1 and

output in result1. The overflow part will access mux2 and output in result2.

Divider: Used to perform divider of X and Y. The part has two input, and two output. The first output is upper, upper denotes the quotient in a divider operation, which is the integer part of the divider result. The second output is rem, is the leftover part of the divider operation. The upper result will access mux1 and output in result1, the rem result will access mux2 and output in result2.

Adder: Used to perform addition of X and Y. It uses c_in and c_out to handle carry-in and carry-out.

Subtracter: Used to perform subtraction of Y from X.

AND Gate: Performs bit wise AND operation on X and Y.

OR Gate: Performs bit wise OR operation on X and Y.

XOR Gate: Performs bit wise XOR operation on X and Y.

NOT Gate: Performs bit wise NOT operation on X or Y.

2. Multiplexer (MUX):

The MUX selects the output of the ALU based on the value of the opCode. The diagram shows two MUXes, which are used to select result1 and result2.

3. Comparator:

Compares X and Y to determine if they are equal. If they are equal, the output is 1; otherwise, the output is 0.

Output

1. Result1 and result2:

These are the output registers of the ALU, which store the computation results. Depending on the opCode, the MUX selects the corresponding operation result and outputs it to result1 or result2.

2. Equal:

This is the output of the comparator, indicating whether X and Y are equal. If they are equal, this output bit (equal) is 1; otherwise, it is 0.

Process

1. The inputs X and Y are transferred to the ALU through their respective registers.
2. The opCode determines the operation to be performed by the ALU and controls the operation units and the MUX selection.
3. Different operation units in the calculation part perform the corresponding operations based on the values of X and Y.
4. The MUX selects the appropriate calculation result based on the opCode and transfers the result to result1 and result2.
5. If X and Y need to be compared for equality, the comparator outputs the result to equal.Through these steps, this 32-bit ALU can perform various operations and output the corresponding results to the designated registers.

Control design

Instruction decoding logic

1. Signal input

   There are two input, op and func of the circuit. The control circuit unit will send signal to the whole cup by the input of op and func. The op and func are all 6-bit.

2. Signal detector

   The circuit will first determine the op code. The circuit have different option for the input to select which instruction will execute. Op code is mainly for I-type instruction.If the op code is 00, it will access func determine. Use to distinguish different kinds of R-type instruction and send signal to other parts of CPU.

3. Signal decoding

   After detected input code, we use a comparator to decoding the instruction to generate different control signal, the signal will dictate how the subsequent circuit handles the current instruction.

4. I-type instruction generate

   The green lines in the diagram represent the paths of the control signals.Each control signal corresponds to a specific operation, such as:

   LW: Load Word instruction
   SW: Store Word instruction
   BEC: Branch if Equal instruction
   BNE: Branch if Not Equal instruction
   ADDI: Add Immediate instruction

5. R-type instruction generate

   The upper right part of the diagram deals with R-Type instructions, such as ADD and SLT.These instructions use the FUNC field to further specify the exact operation.

6. Special instruction

   SysCall: System call instruction
   These instructions are handled the halt of the counter.

ALU Control Logic

This part will send signal to ALU to dictate which function will execute.

1. Inputs :

   *OP:*This is the opcode input, representing the main operation code for the instruction.
   FUNC: This is the function code input, used mainly for R- Type instructions to specify the exact operation.

2. Logic Gates:

   *AND Gates:*There are AND gates connected to the OP and FUNC signals. These gates help in determining specific operations based on the combination of the OP and FUNC values.
   *OR Gate:*The output from the AND gates is fed into an OR gate.This OR gate combines the signals to determine the correct ALU_OP.

3. MUX (Multiplexer) :

The MUX (Multiplexer) selects between different inputs based on the control signal b. It has two inputs:

Input 0: One of the inputs from the logic gates.

Input 5: Another possible input from the logic gates.

The selection line (b) determines which input is passed through to the output.

4. Output :

ALU_OP: The output of the MUX is the ALU_OP signal. This signal is used to control the operation of the ALU, determining which arithmetic or logic operation to perform.

Detailed Signal F1ow:

The OP input is split and sent to multiple AND gates.

The FUNC input is also split and combined with the OP inputs in the AND gates.

The outputs of the AND gates are combined using an OR gate to generate the appropriate signals.

The MUX then selects the correct signal based on the control input (b).

The selected signal is output as ALU_OP, which controls the ALU' S operation. This setup allows the control unit to generate the correct ALU_ _OP signal based on both the main opcode and the function code, enabling the ALU to perform the required operation for the instruction being executed.

NOTE: To meet the requirement of the assignment, and to simple the design of CPU, the control just open few function in ALU. Although there are sub, mul, div and other functions, we just use the add function and compare function in ALU. Other functions will individual testing in the final part.

Instruction decoding signal

RegDst:

It is used to determine whether RT is used as the target register or RD as the target register

RegWrite:

Determine if Register file needs to write back, R_type, ADDI and LW instruction all need RegWrite signal.

MemToReg:

Determine whether it is necessary to access Memory to take out the data and write it to Register file.

MemToReg:

This is used to determine whether data should be taken from memory and bring to register file.

MemWrite:

This is used to determine whether memory needs to be accessed for writing

AluSrc:

To be fed into Mux to detect the second into ALU is R2 from register file or immediate number from instruction.

Beq:

Used to determine the two number is equal and branch to another instruction.

Bne:

Used to determine the two is not equal and branch to another instruction.

Halt:

Used to determine if the whole circuit should shop.

Register File Design

Register:

1. The register file contains multiple registers, each of which can store data of a fixed width (in this project, the width of each register is 32-bits).
2. Each register has a unique address through which the corresponding register can be accessed.

Determine the location:

1. Enter the address to specify the register to be accessed. There are usually two input address ports, one for read operations and one for write operations.
2. Read Address input: Used to specify which register to read data from.
   RS: The first source register, used to provide data for the first operand.
   Rt: The second source register, which provides the data for the second operand.
3. Write Address input: used to specify which register to write data to.
   Rd: Write operations typically use another address port (such as Rd) to specify the address of the write register. Rd is a destination register that stores the result of an operation or other data that needs to be written.

The use of MUX:

1. The multiplexer is used to select and output specific register data.
2. In the read operation, the Rd and Rs input data to determine which register to pass the data into.
3. During the write operation, select the corresponding register according to the Rd data, and write the data to the target register.

Write back data:

1. Write back The data port is used to receive data that needs to be written to the register.
2. Data is input to the register file through the write data end, and written to the register at the specified address when the clock signal arrives.

Enable signal:

1. Write Enable signal: used to control write operations. If the write enable signal is high (1), data is allowed to be written to the specified register; If the value is low (0), write operations are disabled.
2. Read Enable signal (Read Enable) : control the read operation, usually the read operation is one-straight enabled.

3. The enable signal also requires a MUX to control the output, and the control signal of the enable MUX output is the same as the control signal of the write operation, so that only the value of the target register can be changed when the clock changes, instead of changing the value of all registers.

Clock:

1. The operation of the register file is synchronous clock, that is, the write operation occurs at the rising (or falling) edge of the clock, ensuring that the data is written at a stable time.
2. The clock signal is used to coordinate the timing of the write operation to ensure that the contents of the register are updated at the right time.

2.How to handle difficulties

The difficulties in ALU design

1. The massive wiring arrangement:

   In the entire actual process of ALU, wiring is a very troublesome process. Compared with the 1-bit ALU made last time, the wiring process of 32-bits ALU this time is more complicated. In order to save space, the design of the entire ALU looks more concise, we directly adopt the 32-bits pin. Compared with the last time lab operated each bit and finally integrated it together, avoiding overly complicated wiring, making the whole design look more concise and clear.

2. Two-comparer set:

   In order to make the operation of ALU more rapid, two comparers are designed in ALU to compare data, avoiding the time waste caused by judging whether two data are the same (conditional branch judgment) and whether they are greater than (BEQ and BNE operation) at the same time, making the whole line more efficient.

The difficulties in controller design

1. To meet the requirements by the MIPS instructions:

   In MIPS statements, there are a variety of operations that need to be judged, and MIPS statements are divided into two categories: I-type and R-type. The structural difference between the two is large, which makes it difficult to design the controller to determine the operation. The judgment of the func code with the highest 5-bit op code and the lowest purchase price is still simple, but the location of the source register and the target register of the I-type and R-type are very different. The controller must first determine the input instruction type to determine the correct source register and target register, and the last month bit of I-type is the immediate number. The controller also needs to determine the input instruction type. This caused a lot of difficulties in our design.

2. Decoding

   According to the structure of MIPS statement, instruction input to the controller for decoding needs to be divided into several sections.

The structure of R-type statement is as follows:
op code (6-bits), Rs(5-bits), Rt (5-bits), Rd (5-bits), smart (5-bits), func (6-bits).
The l-type statement structure is as follows:
Op code (6-bits), Rs (5-bits), Rt (5-bits), imme (16-bits).
The controller first divides the input instruction with the splitter, judges different sections, and sends correct control signals to other parts of the CPU. In the process of design, all types of control signals need to be correctly grasped, and each other needs to be taken into account when output.

3. Connect to the ALU part
   Since the control signal of ALU is different from the input instruction signal, the controller needs to map the input instruction. Different operations require different mappings, which greatly increases our workload. In this assignment, in order to simplify the design, we only Only two of the controller's ALU mappings are open: add function and compare function.
4. Connect to other part of the whole CUP
   In addition to add instruction, LW instruction and SW instruction require the controller to send control signals to the ALU. The controller sends control signals to other parts of the CPU, such as whether the register file needs to be read or written, and whether to use main memory operations, these need to be controlled by the controller, so we also need to give the controller a separate design to control. The control signal of the described component, and the two instruction Beq and Bne also require the output result of the ALU, and the controller is not only required. It plays the role of controlling other components, and should also be able to link with other components.

The difficulties in register file design

1. The number of the register:
   We had trouble choosing the number of registers, we didn't know how many registers to use, and based on what we learned: Different ALUs use different registers, but we have to satisfy the minimum number, which is three or three registers of addALU. However, when we use other instructions that only need two registers, there will be a conflict between the two registers, and the machine will not recognize who is the destination. At this time, we choose to use the MUX multiple selector to satisfy the switch destination register, and we also need to return the final value to the register heap, so we also need a return register to store the value. So we chose 8 registers.
2. How to decoding the source address of input instruction and the terminal register to write back:
   At the beginning of the design, we did not know how to read the instructions and how to recognize the instructions, until we remembered that different instructions have different op and the FUNC field allows us to identify different instructions, so we thought of extracting the op and FUNC fields from the instructions. After that, you can successfully identify these different instructions. In writing back to the register, we remembered the solution at that time: it was to write the address

of the register is transmitted to the end, which is then recognized by the data storage and the value is transmitted back to the register pile.

3. The enable signal will make all register change the context it hold:

In the beginning of our design, we found that when we input a write signal into the register pile, all the registers would be rewritten, yet we only need to store our values in the registers we need, which also causes all the values we have stored turn to 0, so we choose to add a multiple selector to limit our target to the target register only when the input returns the signal without changing the value of the other registers.

The difficulties in circuit design

1. Clock sequence:

When we started the test, we found that the clock starting order of PC, register pile, and memory would affect the end result of our experiment. So if we are saving in the register pile and the PC clock switches the level it will cause the register storage to fail, causing us to end up the data was lost, so we forced their clocks to sync and solved the problem.

2. The immediate number of BEQ and BNE:

The last 16 bits of the branch statement are offset, which is used to calculate the destination address to jump to, but the offset setting is given to us on the circuit

The design is difficult because we don't know the current address when the circuit is running, so this makes the calculation of offset difficulty. Therefore, in this action, the last 16 bits are changed directly to the destination address, rather than recalculating the destination with offset in the circuit. The address simplifies the circuit design.

3. Reset the test problem:

When we repeated the test, we found that the PC, the storage heap, etc., needed us to manually reset, which also increased the user's operation and waves. It took unnecessary time, so we designed an automatic reset circuit, and when we execute a new command, these devices will all automatically restore the PC, register pile and so on to the initial settings, which is not only more humane, but also more convenient for the users.

3.How it works

Step1 ADDI

Firstly we enter the ADDI instruction in the instruction memory.

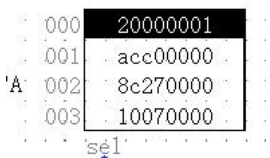

Figure 1 ADDI instruction in the Instruction Memory

Then we see the data has been transferred into ALU.

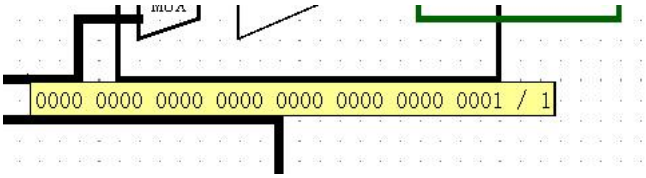

Figure 2 Data in the path

We click the clock button.So this is our add instrument to make Register0 plus 1.That is 1. And remember the data would only be updated when one clock was on the rising edge and the other was on the falling edge.

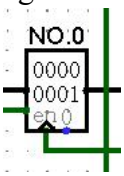

Figure 3 ...0001 is in the Register0

At the same time，the data in R1 and R2 is ...0001.

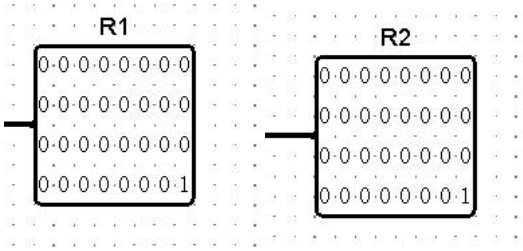

Figure 4 Data in R1 / R2

After completing the first step, we actually found that we have gone from the initial 0001 has become 0010, performed a shift operation.

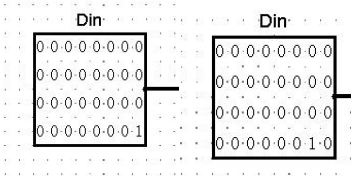

Figure 5 shift op

Step2 LW

It does the LW instruction.

Figure 6 LW instruction in the Instruction Memory

And when we click the two clock buttons again. The data has been into the RAM.

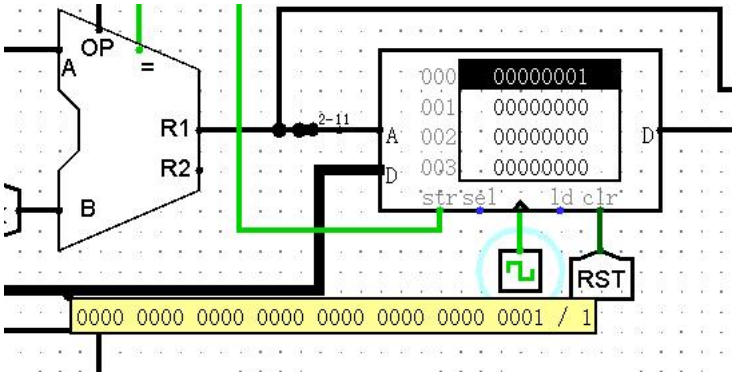

Figure 7 Data in the RAM

And the data has changed,which is ...0000 in R1 and only ...0001 in R2. The data from RS is also from 00000 to 00110.

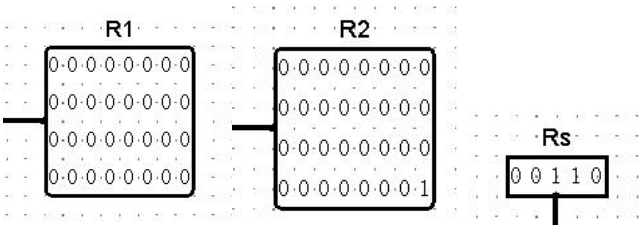

Figure 8 Data in R1 / R2 and RS

Step3 SW

Next, we enter the step for SW instruction.

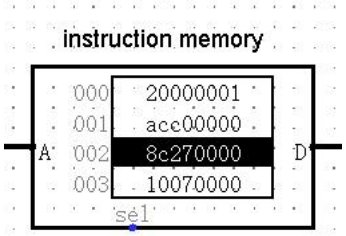

Figure 9 SW instruction in the Instruction Memory

So we repeat the steps for the clock button above. And we will see the initial data ...0001 has been into the Register7.

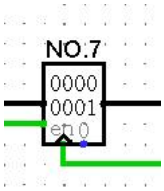

Figure 10 Data in the Register7

At the same time, data from RD and RT is from 00000 to 00111. The data in RS is from 00110 to 00001. This indicates that the CPU has completed the entire load / save

process and stored the data in it.

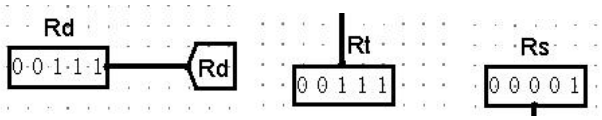

Figure 11 Data in RD / RT /RS

Step4 BEQ

The final step is BEQ. We will determine if the data values in Register1 and 7 are the same.

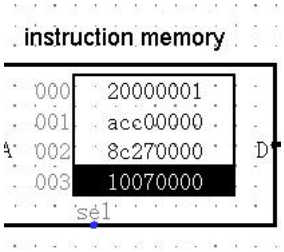

Figure 12 BEQ instruction in the Instruction Memory

At the time, the values from RD / RT / RS change to 00111,00111 and 00000 respectively.

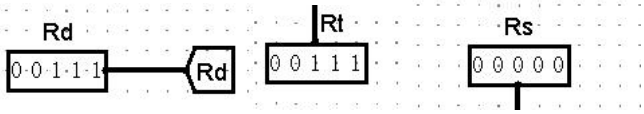

Figure 13 Data in RD / RT / RS

At this time, we find we go on the step which is the same as that we start a new loop. It means the BEQ instruction is success.

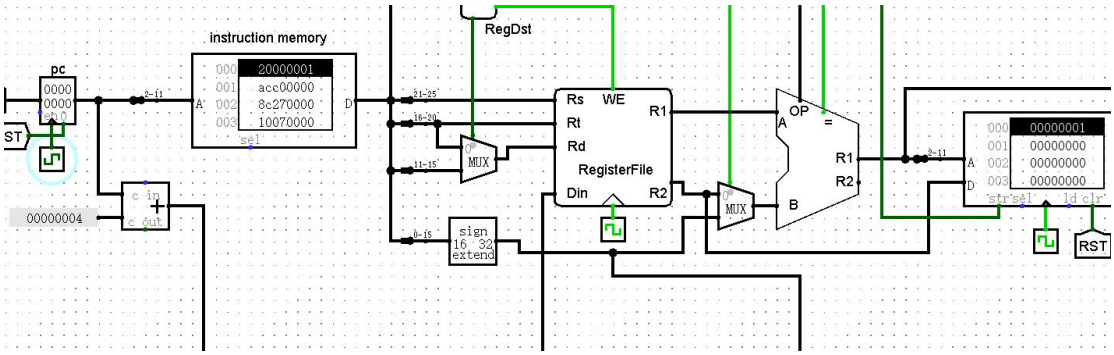

Figure 14 A new loop confirms BEQ

The above are four manual decomposition steps of a loop, including the working processes of ADDI, LW, SW, and BEQ. If we continue, we will see a picture similar to the one below, which means the data is continuously increasing.

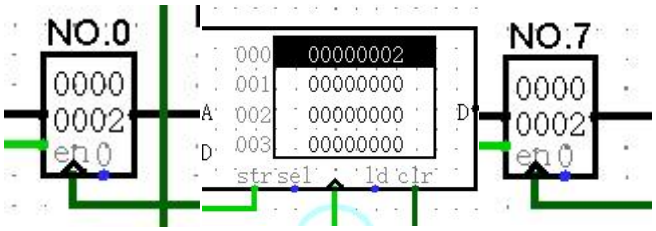

Figure 15 Some future operations

In addition, if we press Ctrl + K in Logisim, we can see that the entire processor is automatically performing calculations through clock adjustment. Because this is a dynamic process, we cannot display it through images. Users can operate and view it themselves through the circle file.

4.How it was tested to see if it works or not

We enter the first instrument is add instrument.

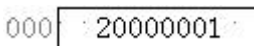

Figure16 ADDI instruction

So this is our add instrument to make Register0 plus 1. Then it is 1.

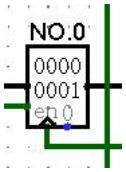

Figure17 Register0

Then we do the SW and LW instrument to set another register have same value for us to compare with itself .So we save it in Register7, its value is same as Register0.

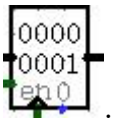

Figure18 Register7 in SW

Then we use BEQ to boolean the value of Register7 and Register0, if they are equal,the loop will continue and keep do the instrument.So we can get the next loop.

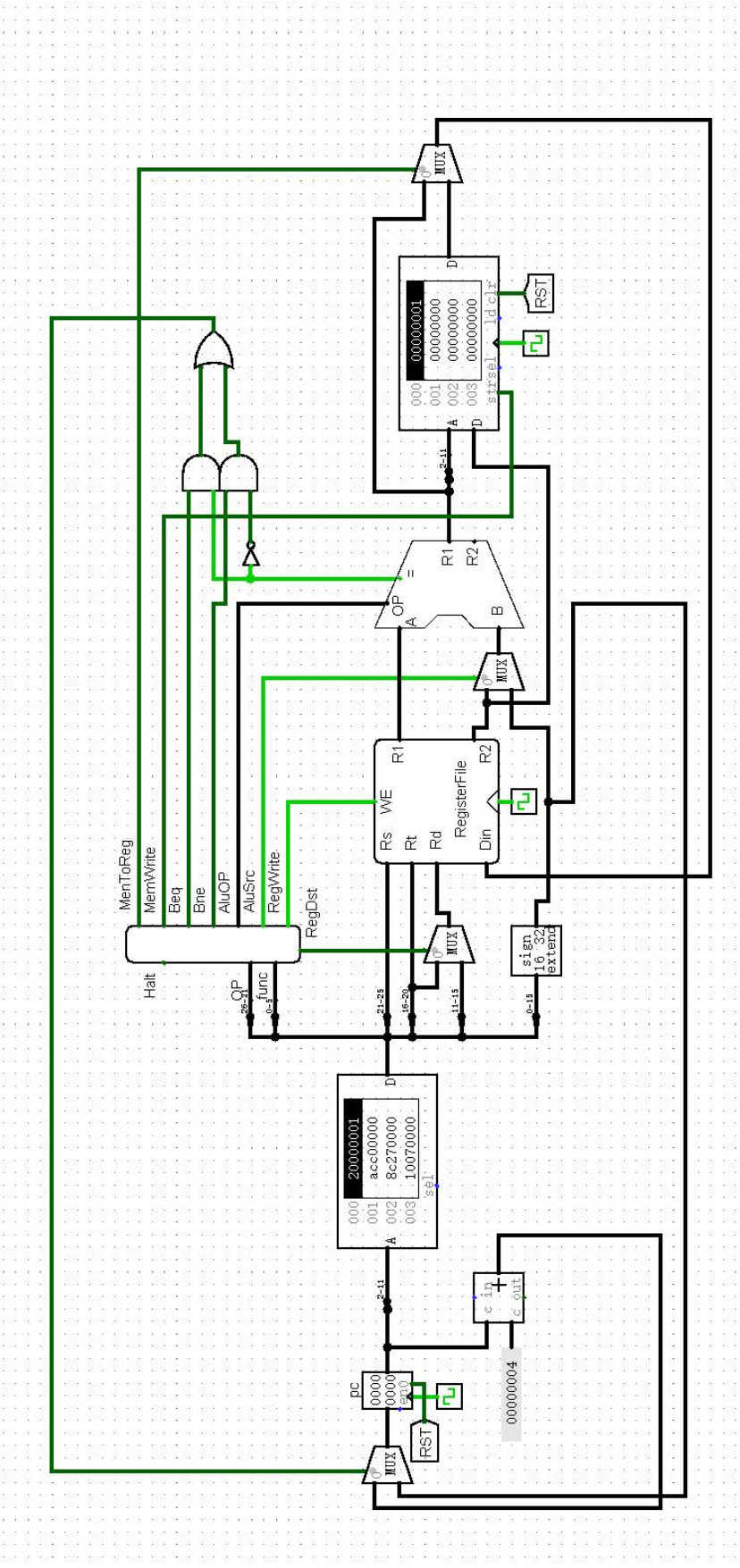

Figure 19 Loop in the processor

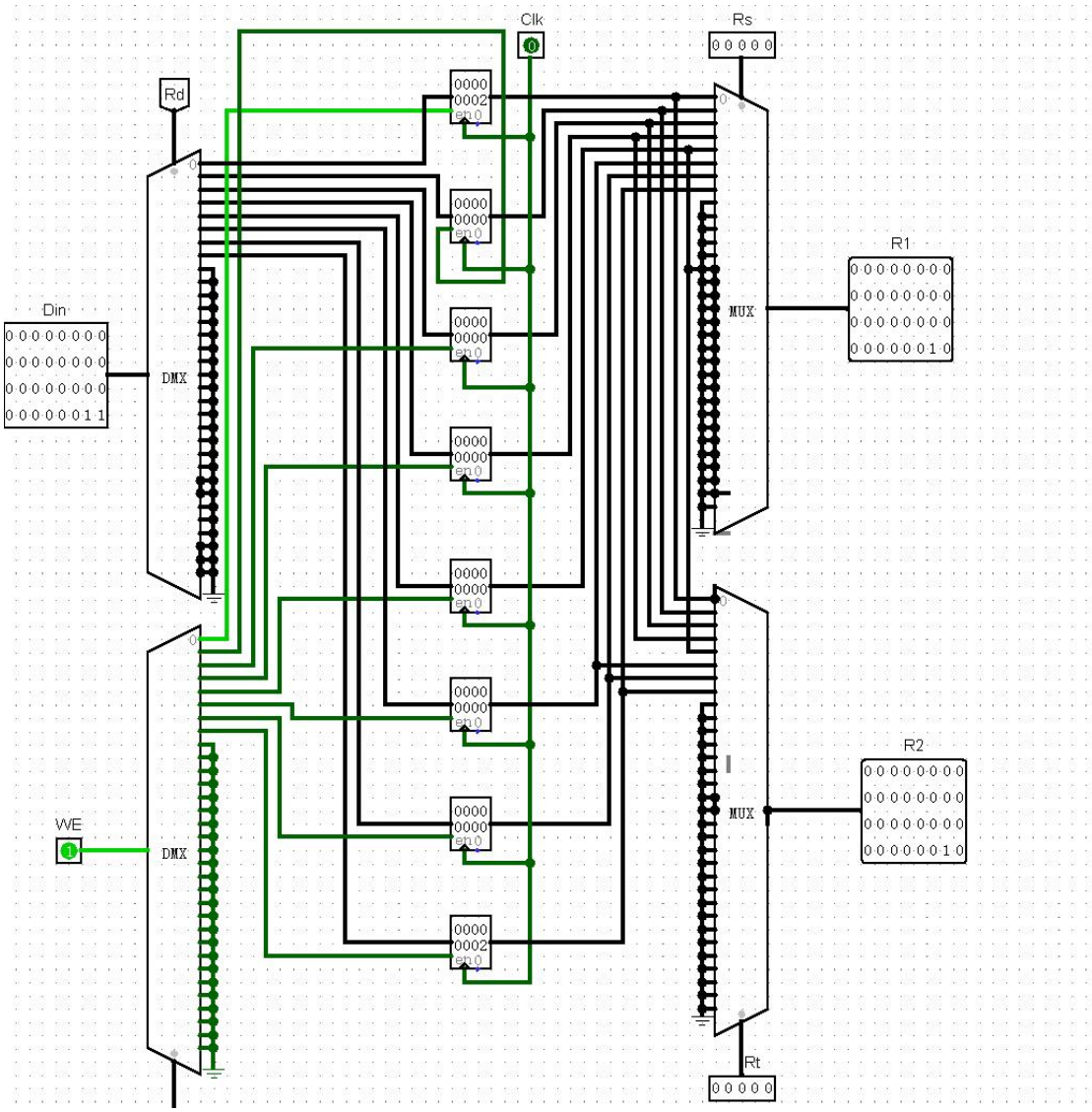

Figure 20 Loop in the RegisterFile

Also you can use RST button to reset all value in the CPU.

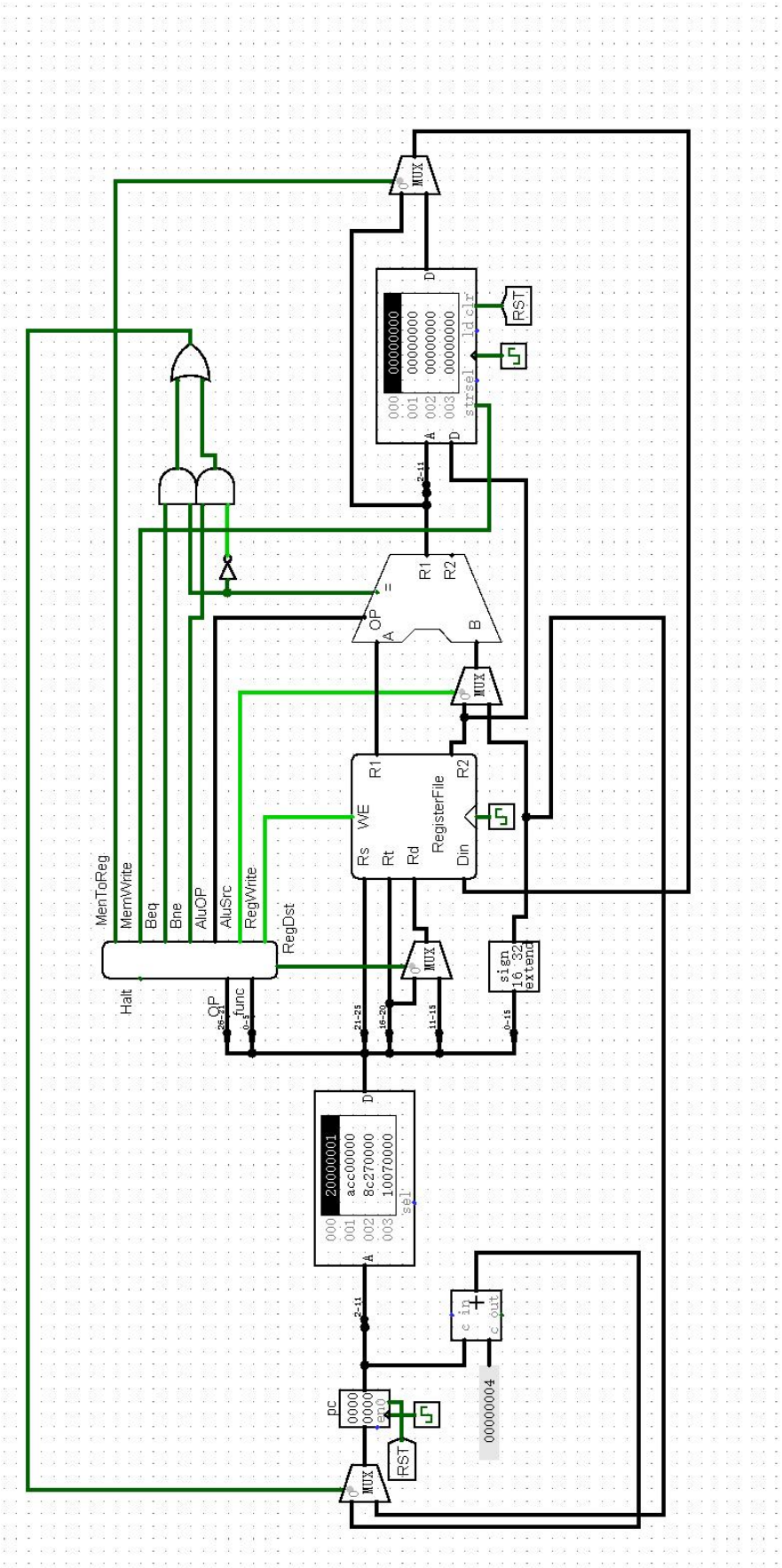

Figure 21 Reset

Processor Description

This is a single clock cycle processor based on the MIPS language. In ALU, it has multiple functions including AND, ADD,OR,NOR,NAND,BEQ,SLT,SUB 。 At present, it can perform simple ADDI and BEQ operations, basic lw, sw operations, loop functions, and shift operations. Because it is not related to the basic requirements of this assignment, we have not yet developed any other operation functions, such as subtraction operation, multiplication operation, division operation and basic AND and OR gates. If necessary, we will reset it in the controller. This CPU does not have a separate output display, but data changes and storage can be viewed in memory and register files.

When users need to check the function, they only need to repeat the operation of a few clock buttons to see how it runs step by step. Then you can also see the changing data in RegisterFile.

If necessary and with more time, we can add more features in the future and make this processor more perfect.

Conclusion

In summary, we have successfully completed this project and created a relatively simple but basic processor that meets the requirements. Meanwhile, during the production process, we logged into the GitHub website and referenced some of the circuits designed by others[1][2]. However, we claim that this is only for reference, and our circuits are still independently completed.

Reference

[1] https://github.com/yuxincs/MIPS-CPU

[2] https://github.com/ztreble/build-a-cpu