METU NORTHERN CYPRUS CAMPUS

EEE 445 COMPUTER ARCHITECTURE I
CNG 331 COMPUTER ORGANISATION

TERM PROJECT: PART #2

FALL 2022

PART #2: Building Memory and Register File.

This part will participate in %10 of the overall course grade (out of 30% which is the percentage of the overall term project). This percent will be divided as follows:

- **30% for functionality of the Register File.**
- **60% for functionality of the memory.**
- **10% for testing your schematic.**

Please Read Chapter 4 from Computer Organization and Design The Hardware/Software Interface 6th ed, Patterson and Hennessy carefully, before starting this project.

2.1 OBJECTIVE

This part consists of three sections. You will learn how to build a register file in the first section. In the second section, you will design instruction memory using ROM and in the third section data memory using RAM. In the end, you will attach the provided ALU from ODTU class to the memory and the register file you are going to design for this assignment.

2.2 PROBLEM DESCRIPTION

This part demonstrates a simple 8-bit CPU using four ALU functions, Register file, RAM, ROM, and control unit. Figure 2.1 shows the final schematic of the simple CPU you will build in this part. PC register will hold the address of the instruction. Every clock cycle, the PC register will be increment by 3. Having a byte addressable memory, 21-bit instruction needs 3 bytes to store. Each byte represents one address which explains why we need to increment PC by three each cycle. The Control Unit will generate the following signals according to the instruction:

1) ALUfunc: used to control the function of the ALU.
2) Mux1: responsible for choosing the value that will be stored to register file.
3) Mux2: responsible for selecting the second source ALU input.
4) Read: enable signal used when reading from memory.
5) Write: enable signal when storing a value in the memory.
6) WE: enable signal used when writing to register file.

Table 2.1: Mux1 control signal

Mux1	Destination register
00	ALU output
01	Memory output
10	Immediate value
11	Don't care

Table 2.2: Mux1 control signal

Mux2	Second source operand
0	Second source register (Rt)
1	Immediate value

Table 2.3: ALU function control signal

ALUfunc	ALU function
00	add
01	and
10	or
11	rotl

Table 2.4: instructions specifications.

Opcode	Mnemonic	Instruction Operation	Assembly Format Code	Machine Code Format	Type
0000	ADD	Rd <= Rt + Rs	ADD rd,rs,rt	0000 ssst ttdd d000 0000 0	R
0100	AND	Rd <= Rt AND Rs	AND rd,rs,rt	0100 ssst ttdd d000 0000 0	R
1000	OR	Rd <= Rt OR Rs	OR rd,rs,rt	1000 ssst ttdd d000 0000 0	R
1100	ROTL	Rd <= ROTL Rt {Rs}	ROTL rd,rs,rt	0100 ssst ttdd d000 0000 0	R
0001	LOAD	Load Rd [Rs+IMM]	LD rd,IMM(rs)	0001 ssst ttdd diii iiii i	I
0010	STORE	Store Rt [Rs+IMM]	ST rt,IMM(rs)	0010 ssst ttdd diii iiii i	I
0011	LOADI	Rd <= IMM	LI rd,IMM	0100 ssst ttdd diii iiii i	I
				s <= Rs address t <= Rt address d <= Rd address i <= Immediate value	

The immediate is 8-bit length stored in the instruction. Since ALU is 8-bit, the maximum immediate that can be used is 8-bit.

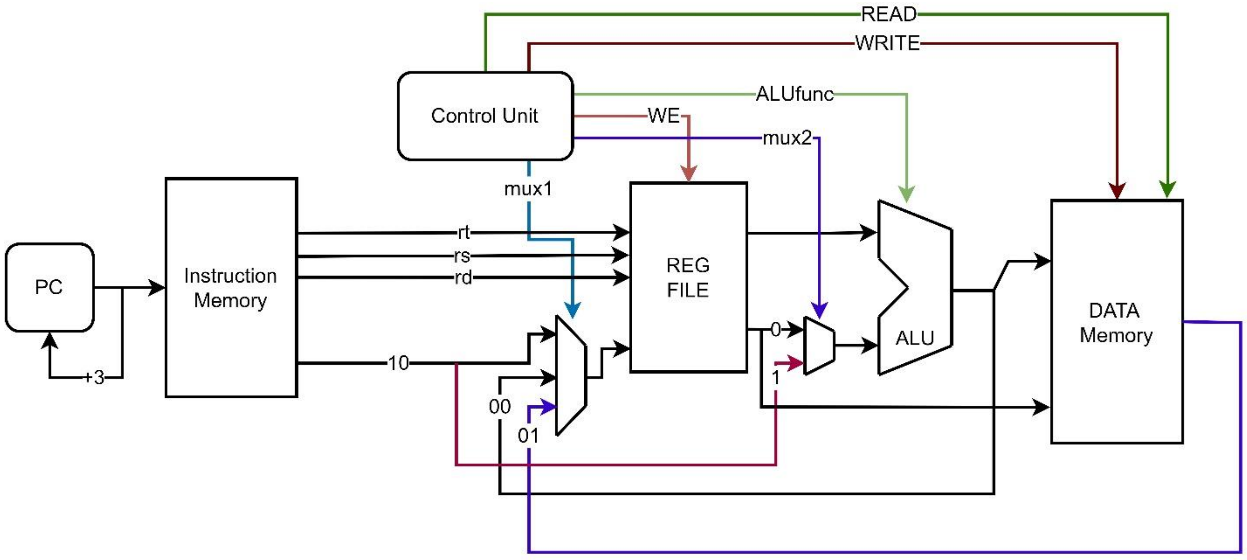


Figure 2.1: Simple CPU schematic.

Each mux has inputs along with their value according to the Control unit table (table 2.5).

Table 2.5: Control unit table

Opcode [3:0]	Alu function (ALUfunc) [1:0]	Register source mux (Mux 1) [1:0]	ALU source mux (Mux 2) [0]	Register Write enable (WE) [0]	Memory write (Write) [0]	Memory Read (Read) [0]	Instruction
0000	00	00	0	1	0	0	ADD
0100	01	00	0	1	0	0	AND
1000	10	00	0	1	0	0	OR
1100	11	00	0	1	0	0	ROTL
0001	00	01	1	1	0	1	LOAD
0010	00	XX	1	0	1	0	STORE
0011	00	10	X	1	0	0	LOADI

After using K-map each control bit will be as follows:

Mux1[0]= opcode[1].opcode[0] ; bit 1 of opcode logically and with bit 0

Mux1[1]= Not(opcode[1]).opcode[0]

Mux2= opcode[1] + opcode[0]

Mux1[1] = opcode[1].opcode[0] ; bit 1 of opcode logically and with bit 0

WE= Not(opcode[1]) + opcode[0]

Mux1[0] = Not(opcode[1]).opcode[0]

MW = opcode[1].Not(opcode[0])

WR= Not(opcode[1]).opcode[0]

Alufunc[0]= opcode[3] opcode[2]

Alufunc[1]= opcode[4] opcode[3]

Instruction format:

We will implement only two formats in this part: R-type and I-type. Instruction length will be 21-bit.

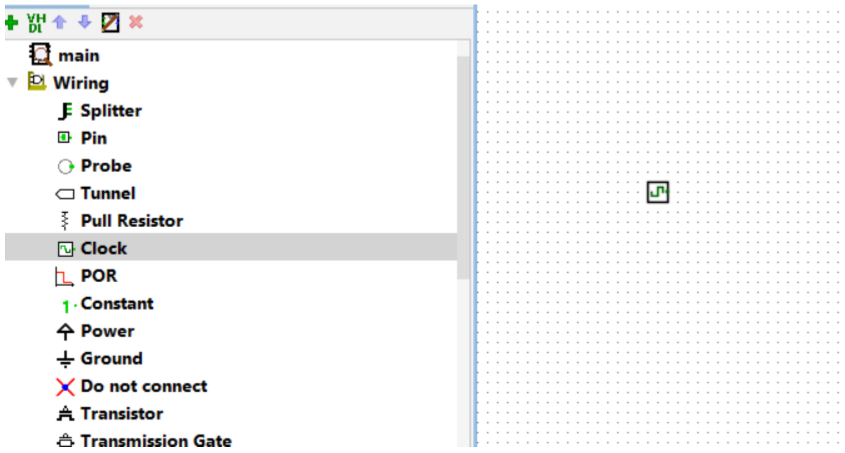
R-type

Opcode [3:0]	Rs [2:0]	rt[2:0]	rd[2:0]	00000000

I-type

Opcode [3:0]	Rs [2:0]	rt[2:0]	rd[2:0]	IMM[7:0]

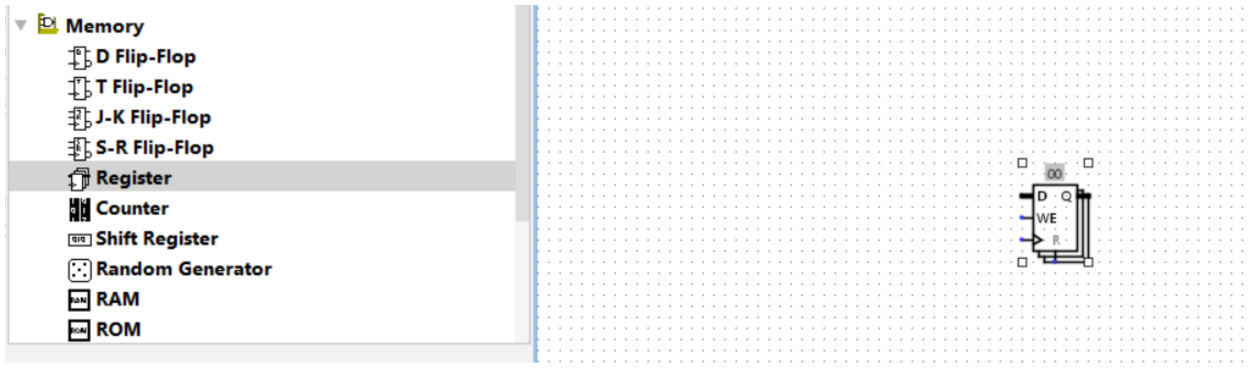
Before you start, in this part, you will need a clock. From the Wiring library, take Clock to use in your design. Note: Logisim calculates your frequency automatically. **Note**: for LI instruction rs and rt are don't care.



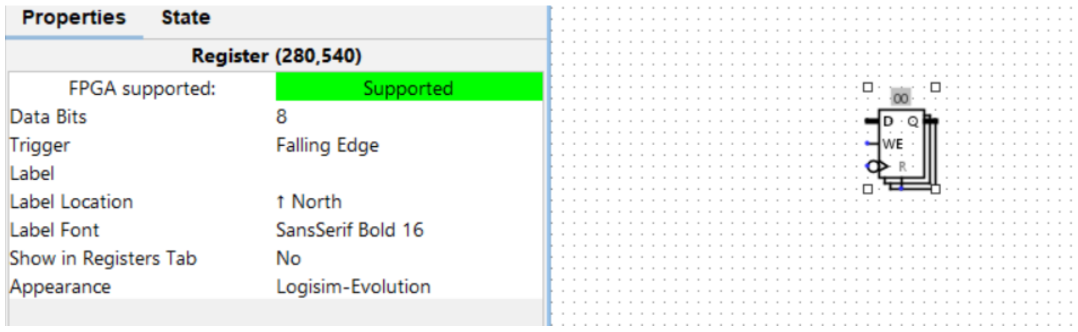
Part 1: building a Register file

This section will go through the steps that will help you build a register file for your term project. Our registers are 8-bit size since ALU is 8-bit functionality. You need to build a register file that holds eight registers. Eight registers need a 3-bit address.

1. Start by dragging a register from Memory library to your canvas.



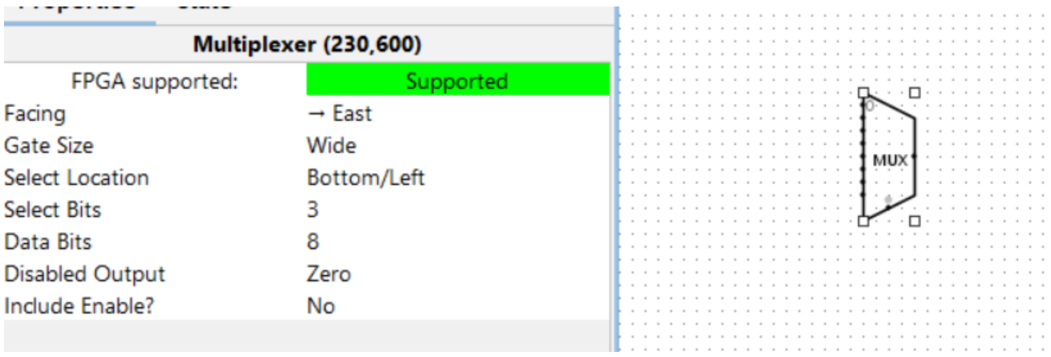
2. You need 8 registers. Each one should have 8 data bits size. Change the properties for your register to match the following figure. Remember to make Trigger sitting to be falling edge.



Register (280,540)

FPGA supported:	Supported
Data Bits	8
Trigger	Falling Edge
Label	
Label Location	↑ North
Label Font	SansSerif Bold 16
Show in Registers Tab	No
Appearance	Logisim-Evolution

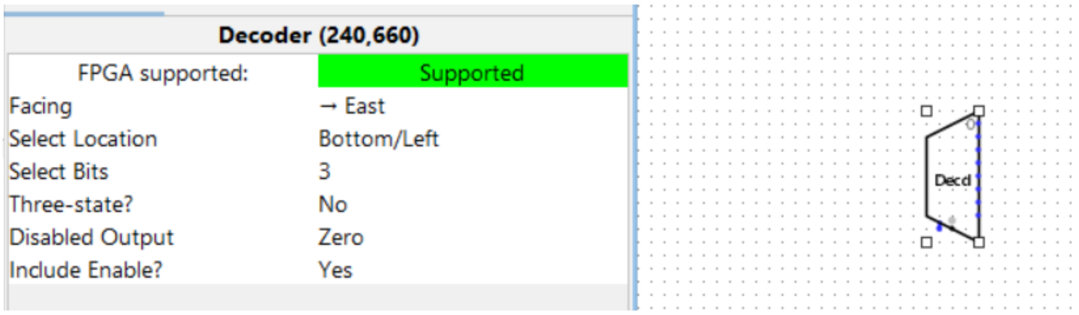
3. You need two mux to select source registers using the address from the instruction format. Change the value of select bits to 3. Do you need Enable?



Multiplexer (230,600)

FPGA supported:	Supported
Facing	→ East
Gate Size	Wide
Select Location	Bottom/Left
Select Bits	3
Data Bits	8
Disabled Output	Zero
Include Enable?	No

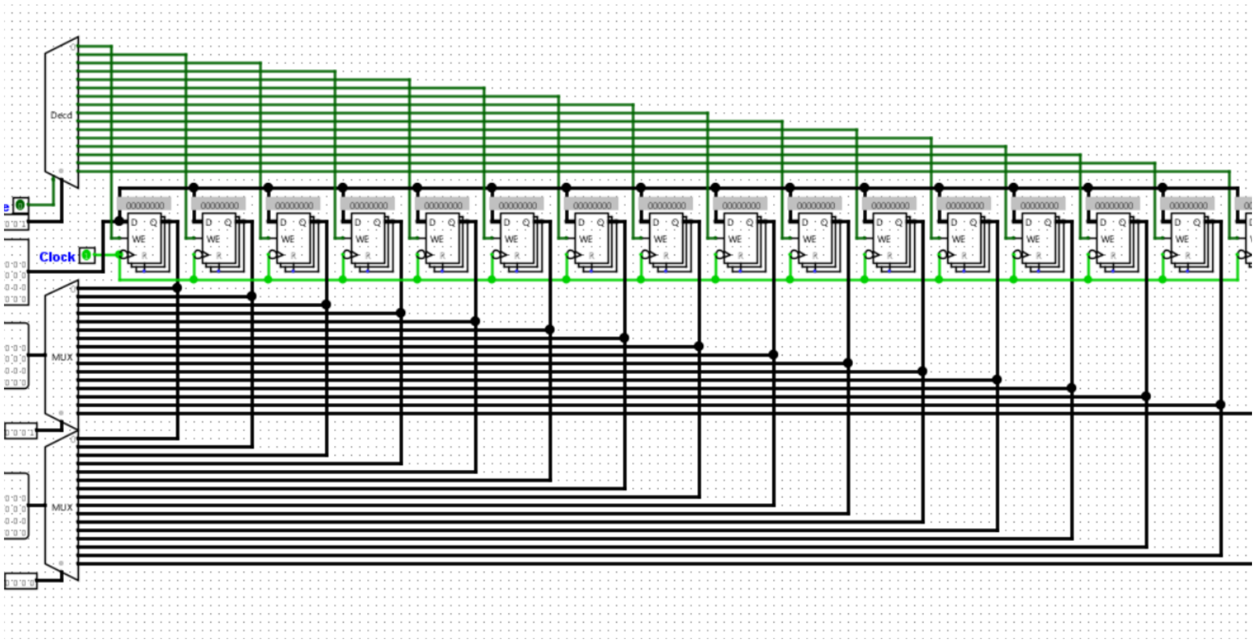
4. One decoder to select the destination register. You need to include enable. Enable signal will represent WE.



Decoder (240,660)

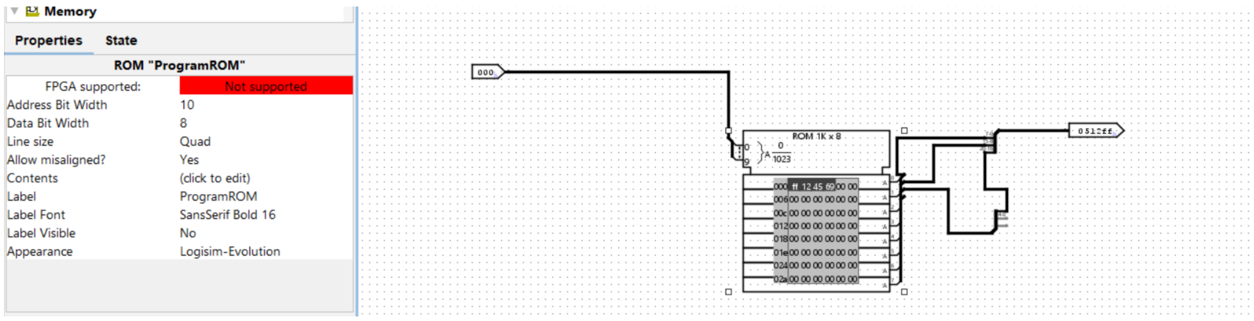
FPGA supported:	Supported
Facing	→ East
Select Location	Bottom/Left
Select Bits	3
Three-state?	No
Disabled Output	Zero
Include Enable?	Yes

5. Connect your registers to the two muxes and the decoder. The decoder will be connected to each WE. Use the same clock for all your system. The decoder will enable one register to write according to the destination address. The following figure shows an example of 32 registers register file. Register file will have 8 registers (r0-r7).



Build 8-reg register file.

Part 2: Building Instruction and Data Memory.
Let's start by building our instruction memory. Logisim Memory library provides a read-only memory, which can be used as instruction memory. If we want to build a ROM with Byte addressing ("Byte addressing" means that each byte in memory is individually addressable, i.e. there is an address x which points to that specific byte.) we need to set the sittings as follow:



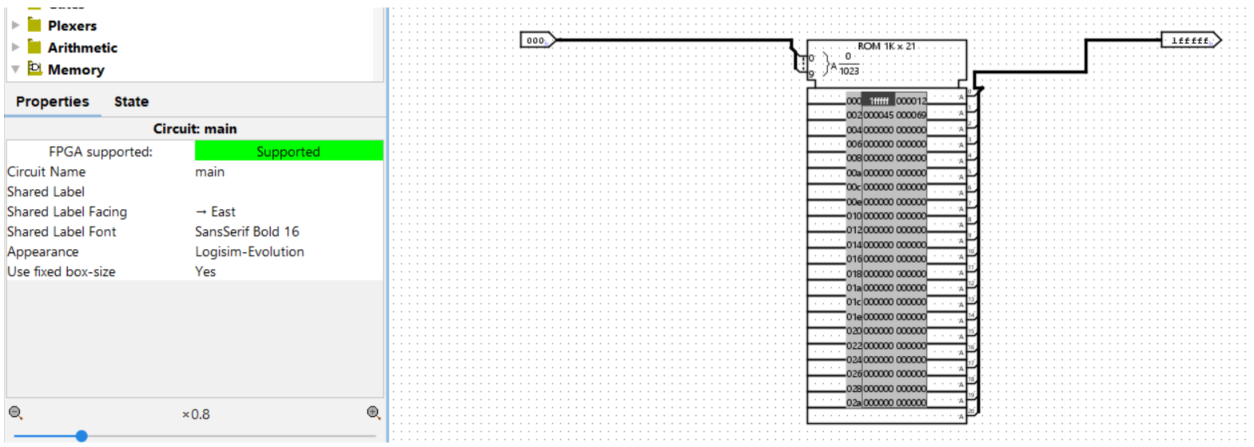
Memory

Properties State

ROM "ProgramROM"

FPGA supported:	Not supported
Address Bit Width	10
Data Bit Width	8
Line size	Quad
Allow misaligned?	Yes
Contents	(click to edit)
Label	ProgramROM
Label Font	SansSerif Bold 16
Label Visible	No
Appearance	Logisim-Evolution

Since we have only a Quad line size, PC should be incremented by four, not three, each cycle. And we will have an empty byte for each instruction. Also, three splitters are needed here. Another way to set the settings is by letting Logisim handle addressing. The following figure shows the properties of a ROM for a 21-bit instruction size with a 10-bit address size. Using this method PC must incremented by 1 each cycle.



Plexers
Arithmetic
Memory

Properties State

Circuit: main

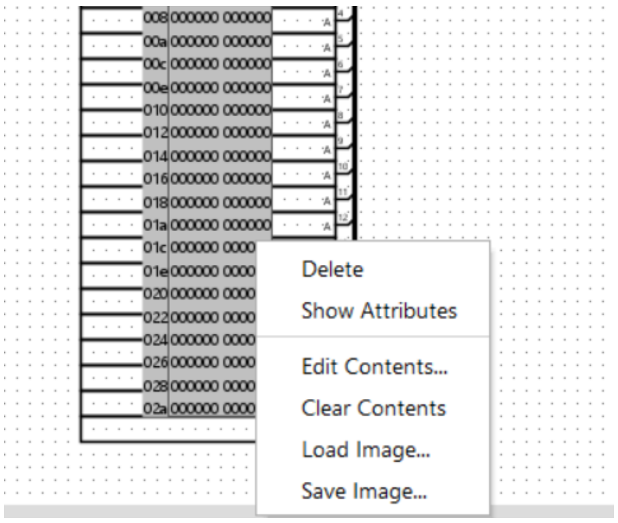
FPGA supported:	Supported
Circuit Name	main
Shared Label	
Shared Label Facing	→ East
Shared Label Font	SansSerif Bold 16
Appearance	Logisim-Evolution
Use fixed box-size	Yes

×0.8

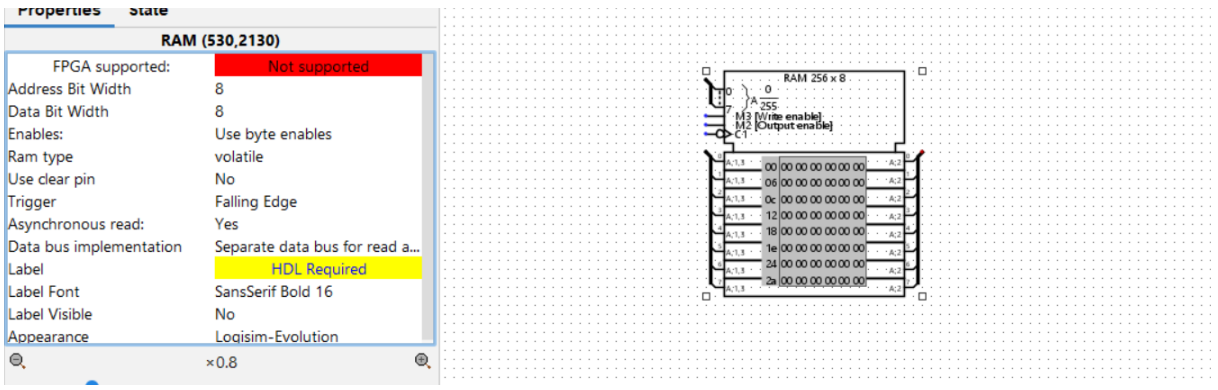
ROM address is 10-bit in length. Every 21 bits will hold one address memory. Logisim will take care of memory alignment, endianness, and address Numbering. Moreover, one instruction will be fetched at each addressed memory, which means we need to increment the PC by one each time.

You must use the second method for this part. And since address is handled by Logisim your PC should be incremented by one each cycle. Each cycle one instruction will be fetched.

You can modify your ROM attributes by right click on it and choose Edit Content.



Finally, we need to build the Data memory. From Memory library choose RAM. The following properties should be set.



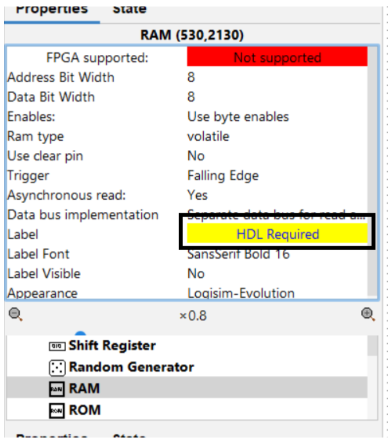
Properties   State

RAM (530,2130)

FPGA supported:	Not supported
Address Bit Width	8
Data Bit Width	8
Enables:	Use byte enables
Ram type	volatile
Use clear pin	No
Trigger	Falling Edge
Asynchronous read:	Yes
Data bus implementation	Separate data bus for read a...
Label	HDL Required
Label Font	SansSerif Bold 16
Label Visible	No
Appearance	Logisim-Evolution

You must connect the address, read/write enables, and Load/store data to your circuit. Try to store values using input pins. Does the RAM store your values as big endian or little endian?

Remember to label all your component in the properties section.

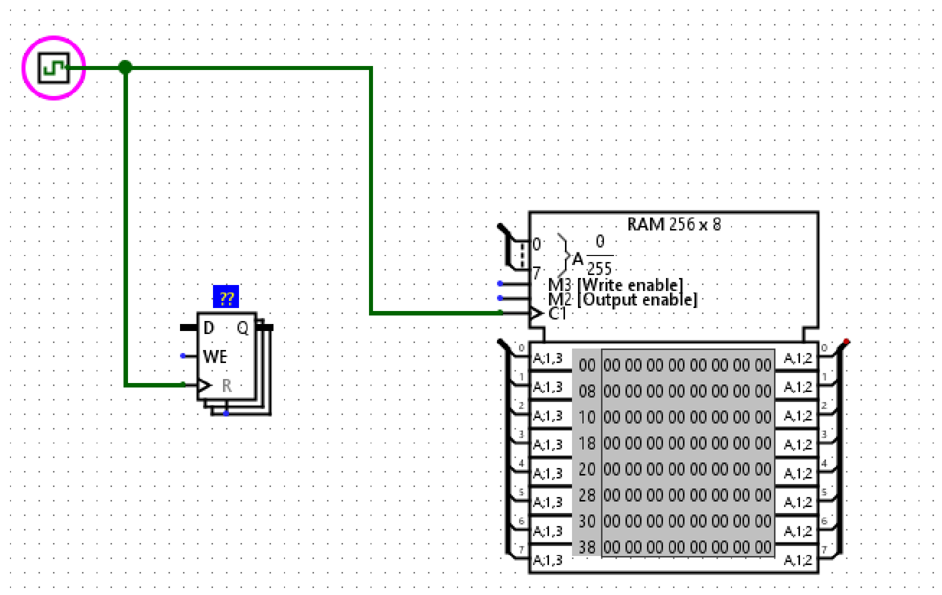


Properties   State

RAM (530,2130)

FPGA supported:	Not supported
Address Bit Width	8
Data Bit Width	8
Enables:	Use byte enables
Ram type	volatile
Use clear pin	No
Trigger	Falling Edge
Asynchronous read:	Yes
Data bus implementation	Separate data bus for read a...
Label	HDL Required
Label Font	SansSerif Bold 16
Label Visible	No
Appearance	Logisim-Evolution

You need to connect your clock signal to each sequential circuit element in your design. Once you use the poke tool, you can press on the clock to add one clock cycle time. As shown in the following figure.



The following table summarizes the bit lengths you need to use in your design.

CPU register bit length	8 bits
Instruction bit length	21 bits
ROM address bit length	21 bits
RAM address bit length	8 bits

In this part, you need to build the schematic shown in figure 2.1. You will find ALU_8.circ on ODTUClass. You need to submit one circuit file named as follows <Your Name>_<Your ID>.circ, e.g., ahmad_2455921.circ.

In addition to your circuit file, you need to submit a report containing a screenshot of your control unit, screenshots of RAM and ROM components with their properties explained, a paragraph explaining the difference between RAM and ROM, and screenshots of the register file and the whole schematic.

ALU uploaded to ODTUClass has two inputs, X and Y, capable of doing all the arithmetic functions listed in Table 2.1. This ALU don't have output flags for this part. The rest of function is same as part 1. ALUfunc input will come from control unit. Whereas X and Y will come from the register file.

For the test part, you need to load image "Test" after downloading it from ODTUClass. In your report, write down the assembly code that this test represents according to this project instruction set architecture ISA, and show the value of each register in register files after running this test.

Please submit one zip file containing your circuit and your report as a PDF to the Term Project Part2 assignment on ODTUClass.

Good Luck ☺