Project Title: Single Cycle 16-bit MIPS-like processor

Course: COE 301 – Computer Organization
Term: 231 – Fall 2023

Done By:

Name	ID
Syed Fareed	202185410
Hammaad Ahmar	201942130
Hasan Alghadaban	202044720

Submission Date: 21/12/2023

1. Introduction

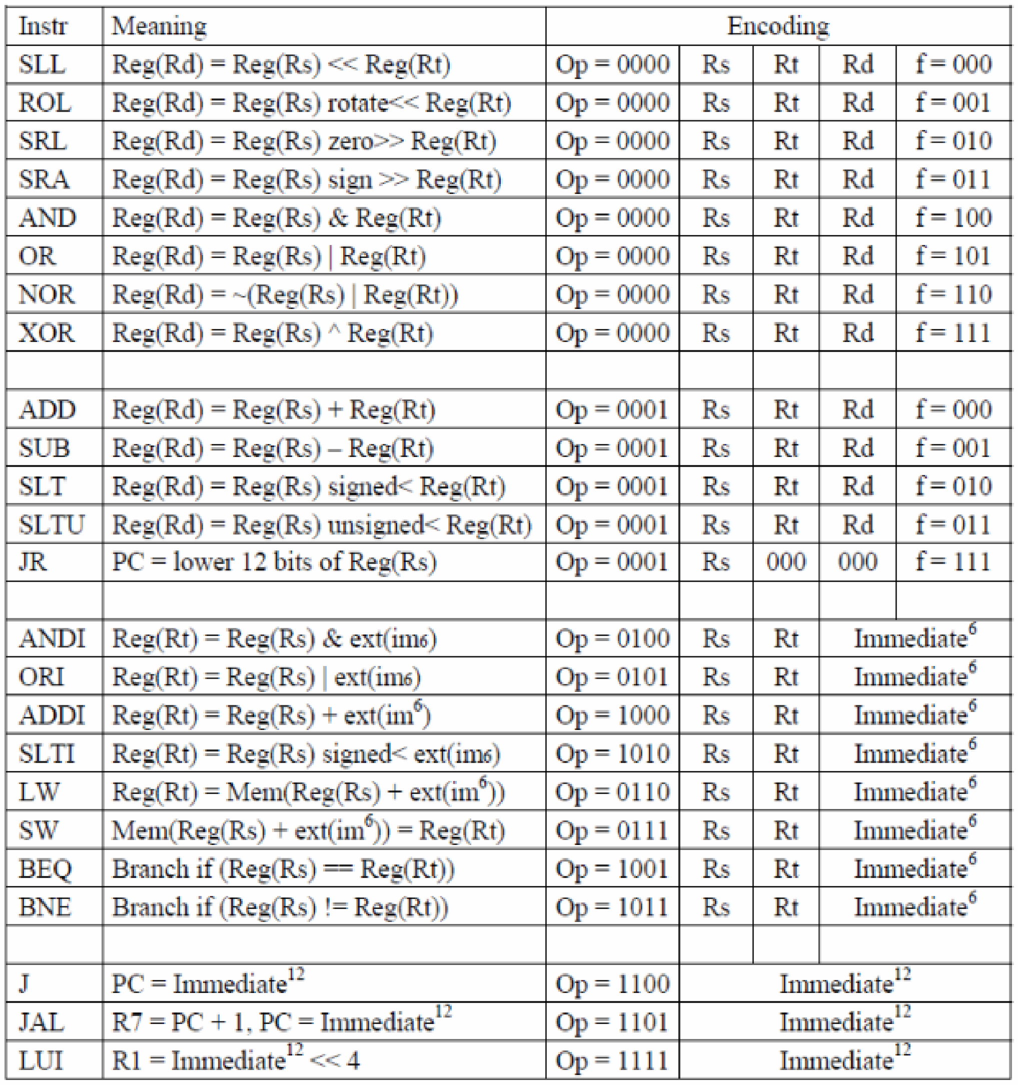
This document presents the development of a 16-bit MIPS like single cycle processor. MIPS is a common design for learning about computers. MIPS stands for Microprocessor without Interlocked Pipeline Stages, which is a way of building a computer's brain that makes it work fast and efficiently. Our project is to create a digital version of such a processor using Logisim, which is a computer program that simulates how processors work. The aim is to understand the basic building blocks of a processor, how they interact, and how to optimize them to work together smoothly.

2. Objectives

This project sets out specific goals to ensure a thorough grasp of the processor design process:

- **Mastering Logisim Simulation:** The first goal is to become skilled at using Logisim. This software is a tool that creates a virtual space where we can build and test a processor without needing physical parts. It helps us visualize and experiment with how different parts of a computer processor work together.
- **Designing a 16-bit Single Cycle MIPS Processor:** The second goal is to design a processor that can handle 16-bit operations, meaning it processes data in chunks of 16 zeros and ones at a time. This small processor should be able to perform basic computing tasks using a set of instructions that we will create. The design process involves deciding how many parts the processor needs, what each part does, and how they all connect.

3. Selected Instruction Set Architecture



Instr	Meaning	Encoding				
SLL	Reg(Rd) = Reg(Rs) << Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 000
ROL	Reg(Rd) = Reg(Rs) rotate<< Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 001
SRL	Reg(Rd) = Reg(Rs) zero>> Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 010
SRA	Reg(Rd) = Reg(Rs) sign >> Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 011
AND	Reg(Rd) = Reg(Rs) & Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 100
OR	Reg(Rd) = Reg(Rs) | Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 101
NOR	Reg(Rd) = ~(Reg(Rs) | Reg(Rt))	Op = 0000	Rs	Rt	Rd	f = 110
XOR	Reg(Rd) = Reg(Rs) ^ Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 111
ADD	Reg(Rd) = Reg(Rs) + Reg(Rt)	Op = 0001	Rs	Rt	Rd	f = 000
SUB	Reg(Rd) = Reg(Rs) – Reg(Rt)	Op = 0001	Rs	Rt	Rd	f = 001
SLT	Reg(Rd) = Reg(Rs) signed< Reg(Rt)	Op = 0001	Rs	Rt	Rd	f = 010
SLTU	Reg(Rd) = Reg(Rs) unsigned< Reg(Rt)	Op = 0001	Rs	Rt	Rd	f = 011
JR	PC = lower 12 bits of Reg(Rs)	Op = 0001	Rs	000	000	f = 111
ANDI	Reg(Rt) = Reg(Rs) & ext(im6)	Op = 0100	Rs	Rt	Immediate^6	
ORI	Reg(Rt) = Reg(Rs) | ext(im6)	Op = 0101	Rs	Rt	Immediate^6	
ADDI	Reg(Rt) = Reg(Rs) + ext(im^6)	Op = 1000	Rs	Rt	Immediate^6	
SLTI	Reg(Rt) = Reg(Rs) signed< ext(im6)	Op = 1010	Rs	Rt	Immediate^6	
LW	Reg(Rt) = Mem(Reg(Rs) + ext(im^6))	Op = 0110	Rs	Rt	Immediate^6	
SW	Mem(Reg(Rs) + ext(im^6)) = Reg(Rt)	Op = 0111	Rs	Rt	Immediate^6	
BEQ	Branch if (Reg(Rs) == Reg(Rt))	Op = 1001	Rs	Rt	Immediate^6	
BNE	Branch if (Reg(Rs) != Reg(Rt))	Op = 1011	Rs	Rt	Immediate^6	
J	PC = Immediate^{12}	Op = 1100	Immediate^{12}			
JAL	R7 = PC + 1, PC = Immediate^{12}	Op = 1101	Immediate^{12}			
LUI	R1 = Immediate^{12} << 4	Op = 1111	Immediate^{12}			

4. Control Unit Signals

Instru--ction	PCSrc	RegDst	ExtOp	Reg Wr	ALU Src	ALU Op	Mem Wr	Mem Rd	WB data
SLL	00	00	DNC	1	1	0000	0	0	10
ROL	00	00	DNC	1	1	0001	0	0	10
SRL	00	00	DNC	1	1	0010	0	0	10
SRA	00	00	DNC	1	1	0011	0	0	10
AND	00	00	DNC	1	1	1100	0	0	10
OR	00	00	DNC	1	1	1101	0	0	10
NOR	00	00	DNC	1	1	1110	0	0	10
XOR	00	00	DNC	1	1	1111	0	0	10
ADD	00	00	DNC	1	1	1000	0	0	10

SUB	00	00	DNC	1	1	1001	0	0	10
SLT	00	00	DNC	1	1	0110	0	0	10
SLTU	00	00	DNC	1	1	0111	0	0	10
JR	11	DNC	DNC	0	DNC	DNC	0	0	DNC
ANDI	00	01	0	1	0	1100	0	0	10
ORI	00	01	0	1	0	1101	0	0	10
LW	00	01	1	1	0	1000	0	1	11
SW	00	01	1	0	0	1000	1	0	DNC
ADDI	00	01	1	1	0	1000	0	0	10
BEQ	{0|zero}	01	1	0	1	1001	0	0	DNC
SLTI	00	01	1	1	0	0110	0	0	10
BNE	{0|~zero}	01	1	0	1	1001	0	0	DNC
J	10	DNC	DNC	0	DNC	DNC	0	0	DNC
JAL	10	11	DNC	1	DNC	DNC	0	0	01
LUI	00	10	DNC	1	DNC	DNC	0	0	00

5. Snapshots of Digital Circuits.

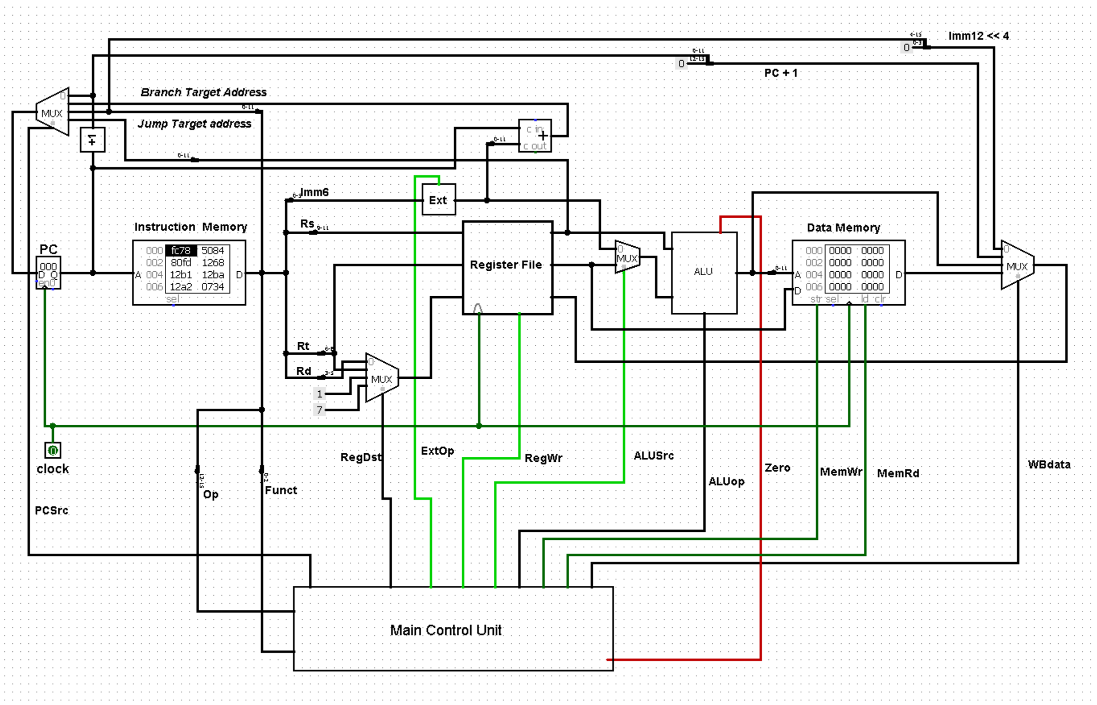


Figure 1. Single-Cycle Processor Design

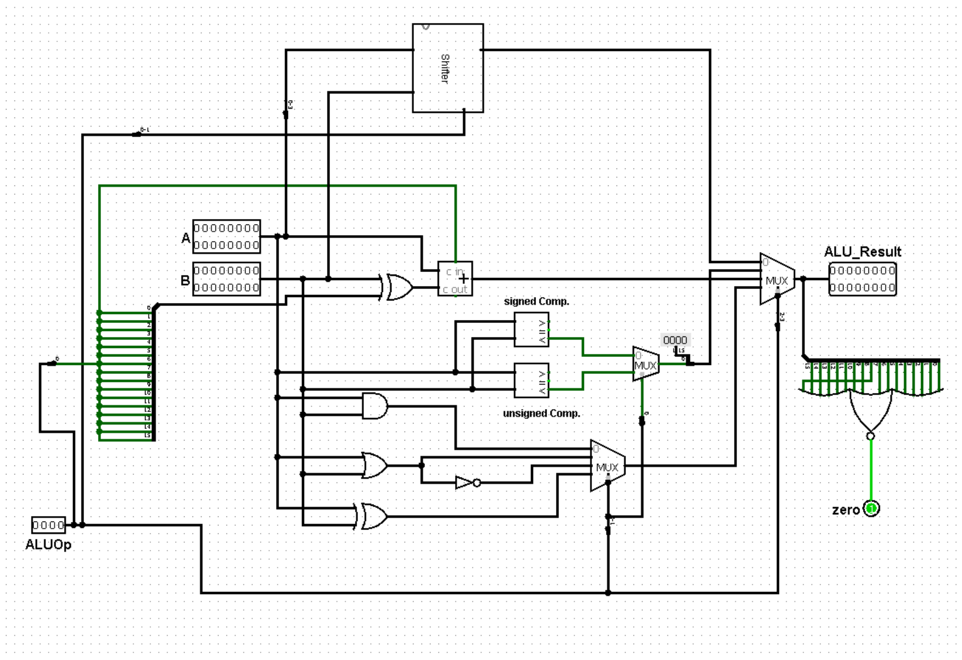


Figure 2. Arithmetic Logic Unit

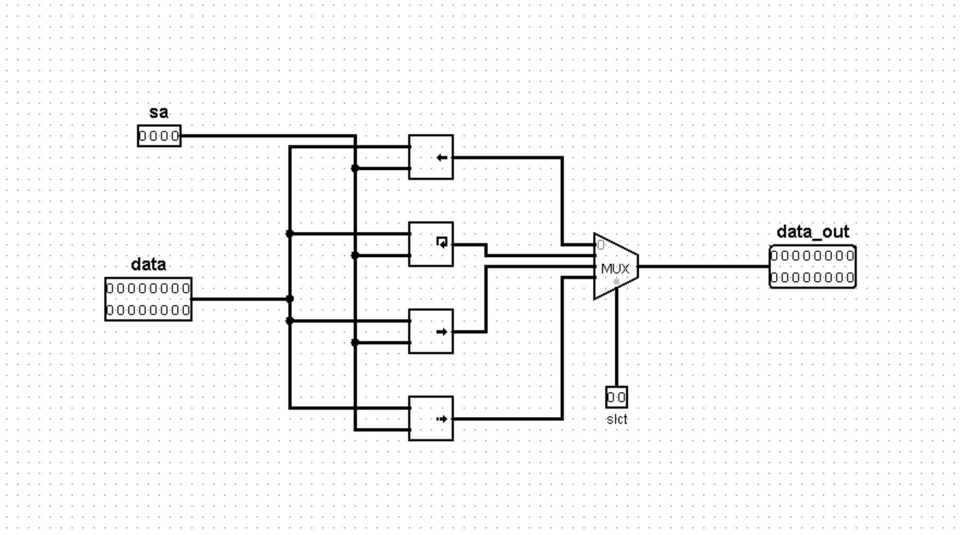


Figure 3: Bit Shifter Component

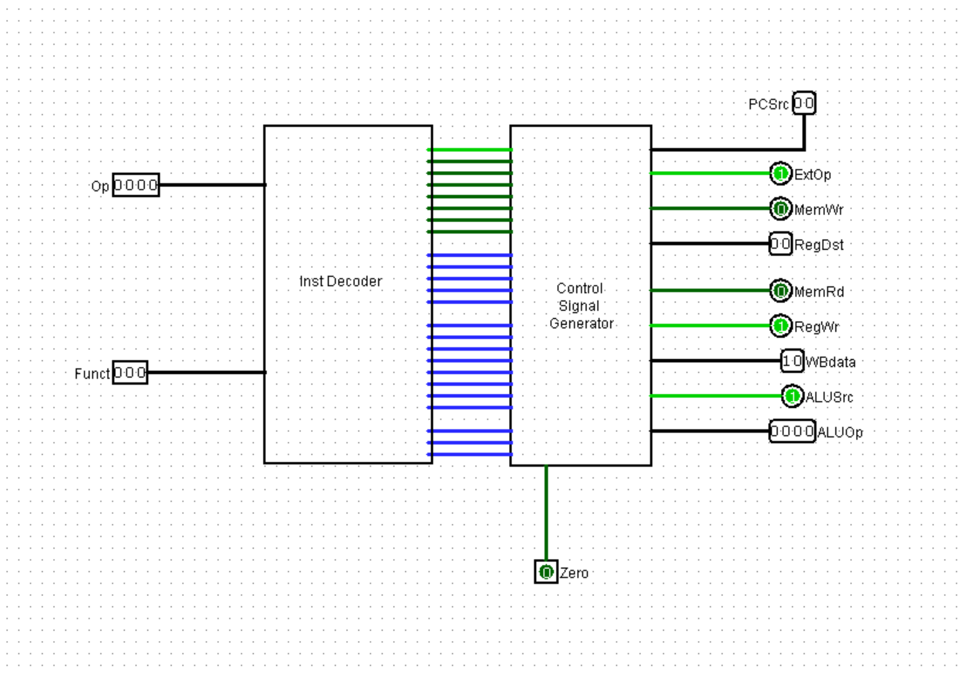


Figure 4: Instruction Decoder and Control Signal Generator

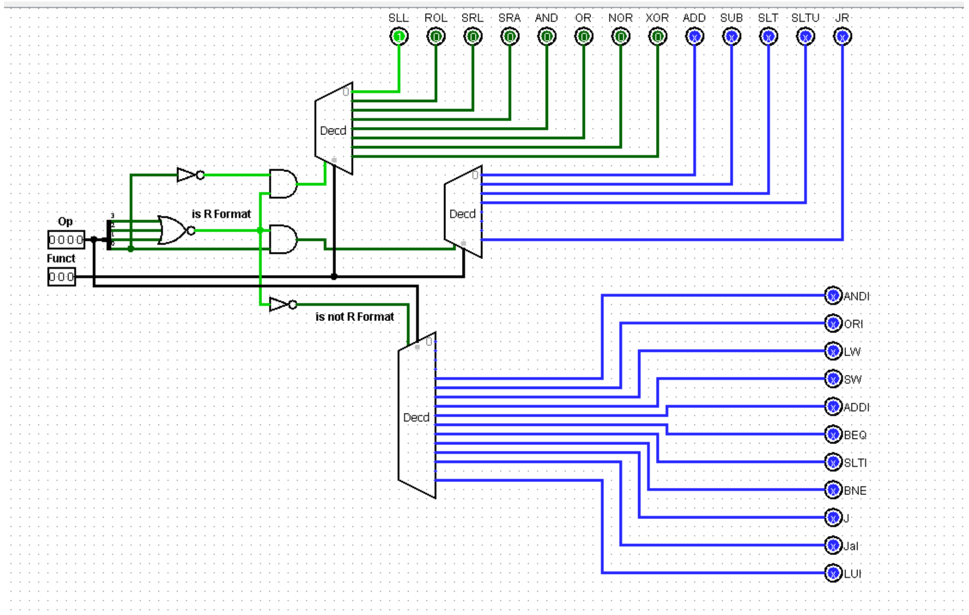


Figure 5: Instruction Decoder

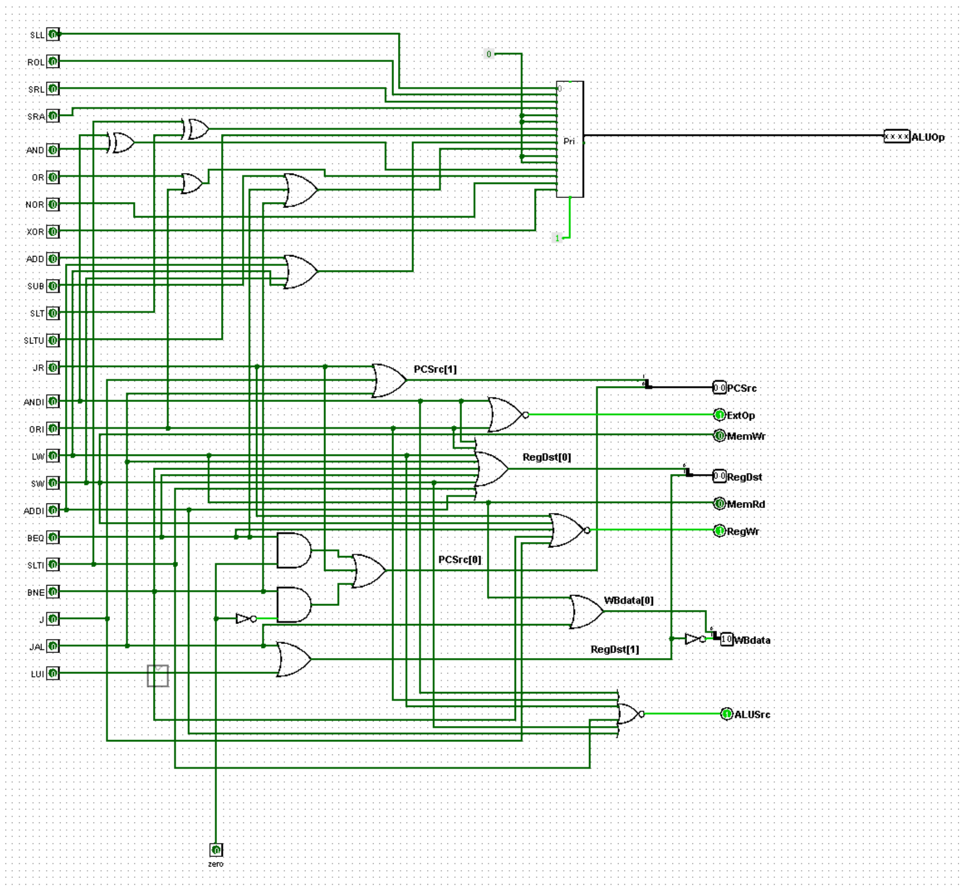


Figure 6: Control Signal Generator

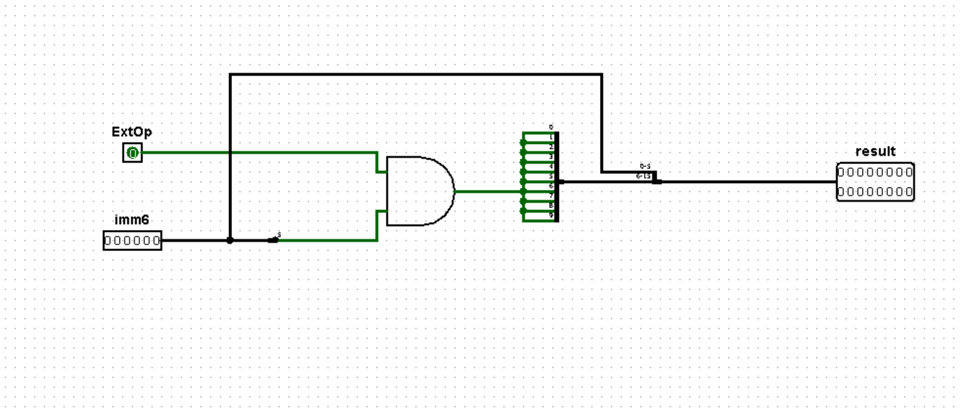


Figure 7: Sign Extension Unit

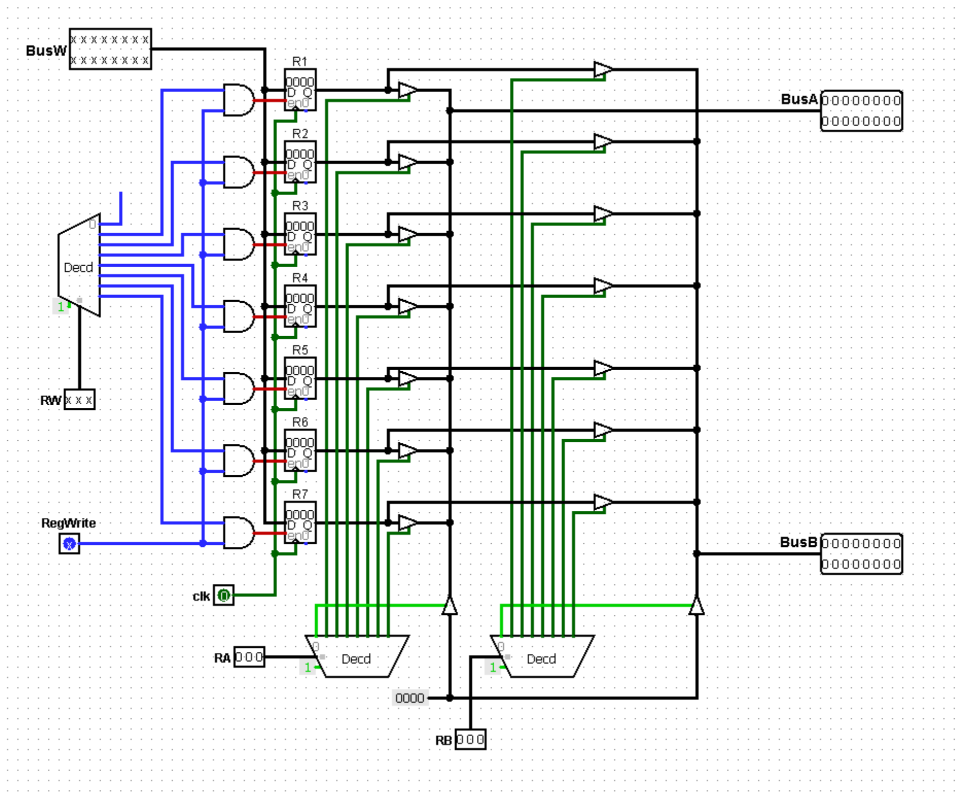


Figure 8: Register File Architecture

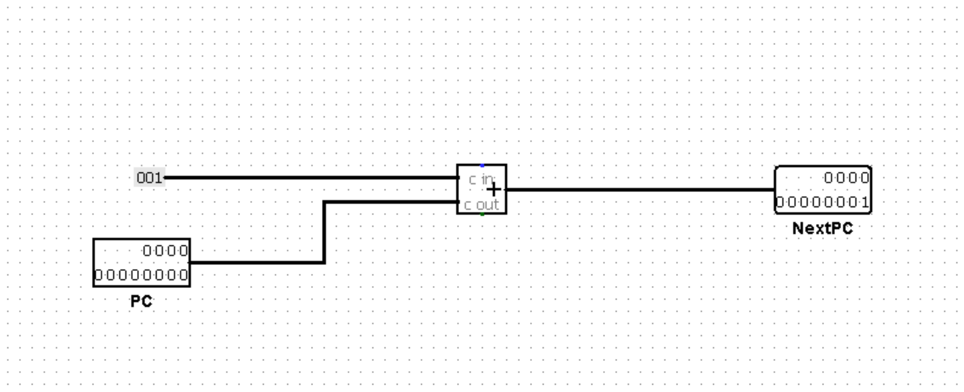
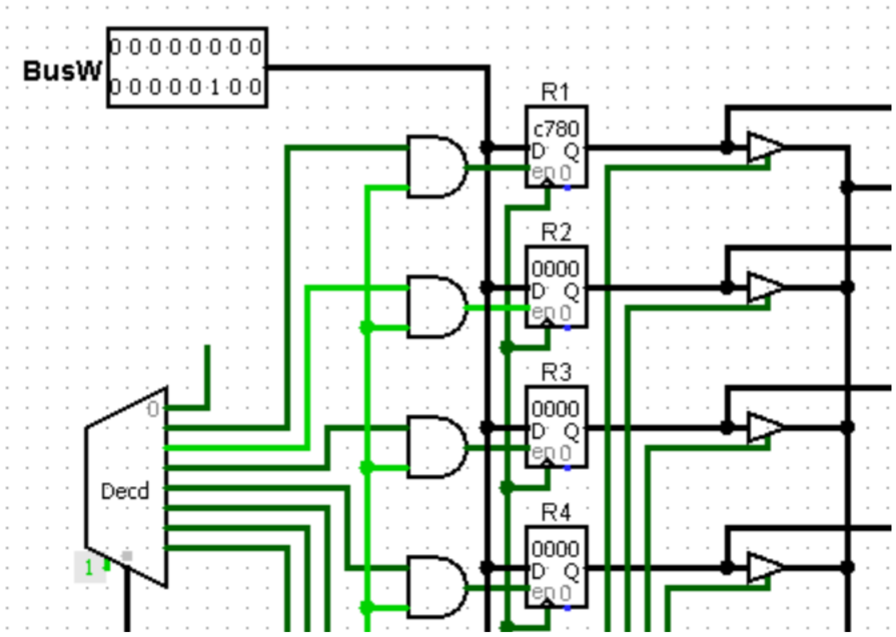


Figure 9: Program Counter (PC) Control

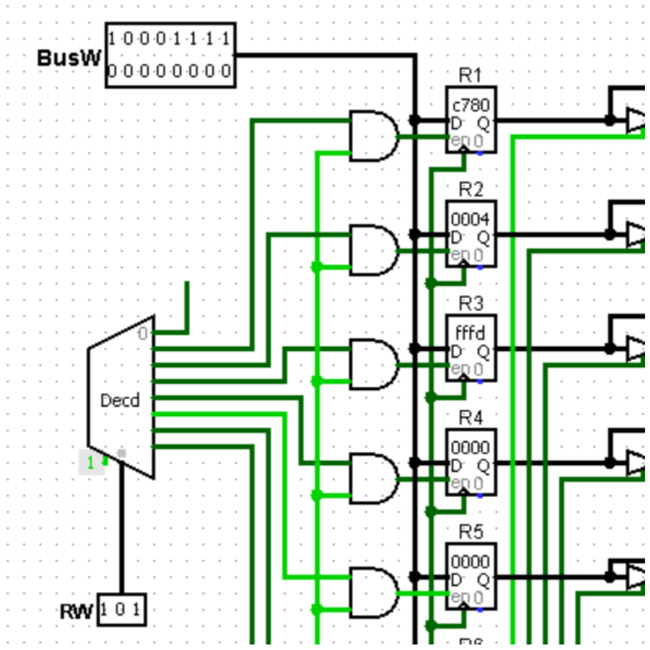
6. Snapshots of Some Simulations

a. Loading a word into register

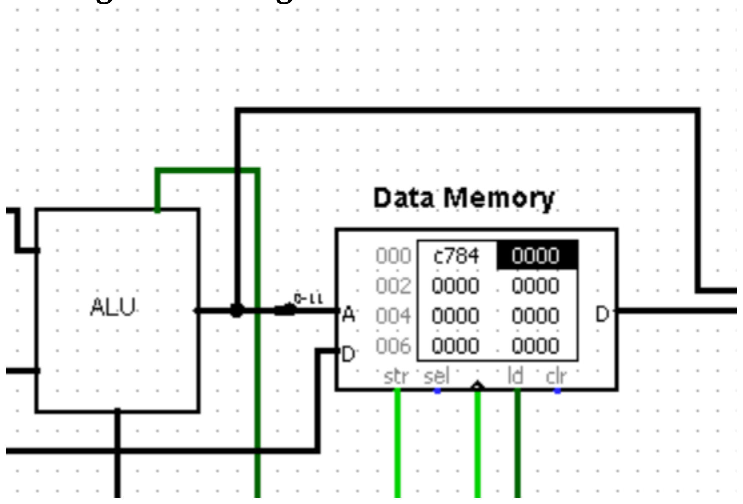


b. Performing Arithmetic

Simulation of the instruction: Instruction: addi r3, r2, r1



c. **Storing and loading words:**



7. Testing

Following test Cases were used to ensure the correctness of the implementation.

a. Initializing Registers (Testing I-Type ALU):

Instruction	Hexadecimal	Expected Result
lui r1, 0x0c78	FC78	r1 = 0xc780
ori r2, r0, 4	5084	r2 = 4 = 0x0004
addi r3, r0, -3	80FD	r3 = -3 = 0xfffd (sign extension)

b. Testing R-Type ALU Instructions (NO RAW hazards – NO Forwarding)

Instruction	Hexadecimal	Expected Result
add r5, r1, r1	1268	r5 = 0x8f00 (carry is ignored)
sub r6, r1, r2	12B1	r6 = 0xc77c
slt r7, r1, r2	12BA	r7 = 1 (true) r1 < 0
sltu r4, r1, r2	12A2	R4 = 1 (true)
and r6, r3, r4	0734	R6 = 0x0001
or r1, r1, r2	044D	R1 = 0xc784
nor r2, r1, r2	0456	R2 = 0x387b
sll r3, r4, r2	0518	R3 = 0x0800
srl r4, r1, r2	0462	R4 = 0x0018
sra r5, r1, r2	046B	R5 = 0xfff8
rol r4, r3, r2	04E1	R4 = 0x0040

c. Testing LW and SW

Instruction	Hexadecimal	Expected Result
sw r1, 0(r0)	7040	MEM[0] = 0xc784
sw r4, 1(r0)	7101	MEM[1] = 0x0040
lw r5, 0(r0)	6140	r5 = MEM[0] = 0xc784

d. Testing Branch and Jump instructions.

Instruction	Hexadecimal	Expected Result
beq r1, r1, +2	9242	branch forward 2 (to bne)
add r5, r2, r2	14A8	should be skipped (r5 not modified)
bne r0, r1, +3	B043	branch forward 3 (to j)
add r6, r2, r2	14B0	should be skipped (r6 not modified)
add r7, r4, r4	1938	should be skipped (r7 not modified)
j 0x20	C020	jump to address 0x20 (ori)
add r5, r2, r2	14A8	should be skipped (r5 not modified)
add r6, r4, r4	1930	should be skipped (r6 not modified)
and r0, r0, r0	0001	NO Operation

e. Testing Jump instructions: Fibonacci Example

Instruction	Hexadecimal	Expected Result
address 0x20:	----	Address of ori = 0x20
ori r1, r0, 5	5045	r1 = 5 (5th Fibonacci element)
jal fib (0x30)	D030	call Fib (r15 = address of or)
or r5, r2, r0	042D	move r5 = r2 (Fib result) = 8
beq r0, r0, 0	9000	branch to self (stop program)
and r0, r0, r0	0004	No operation
. . .		
address 0x30:		Fib starts here:
ori r2, r0, 1	5081	r2 = 1
ori r3, r0, 1	50C1	r3 = 1
add r3, r2, r3	14D8	loop starts here: r3 = r2 + r3
sub r2, r3, r2	1691	r2 = r3 - r2
addi r1, r1, -1	827F	r1 = r1 - 1
bne r1, r0, -3	B07D	branch backward to add if (r1 != 0)
jr r7	1E07	return to caller

8. Work Distribution

Name	Tasks
Syed Fareed	Single Cycle CPU
Hammaad Ahmar	Documenting the Results
Hasan Alghadaban	Attempt to Pipeline the designed CPU

9. Conclusion

In conclusion, the development of a single-cycle 16-bit MIPS-like processor in Logisim has provided a comprehensive hands-on exploration of processor design principles. The project successfully demonstrated the practical implementation of key components, including the instruction set architecture (ISA), control unit, ALU, registers, and memory units, adhering to the MIPS architecture's core principles.

Throughout the design process, challenges were encountered and overcome, contributing to a deeper understanding of hardware intricacies and optimization strategies. Despite its simplified nature, the processor model served as a valuable tool for visualizing and testing theoretical concepts in a practical setting.

This project has underscored the significance of efficient hardware utilization and the complexities involved in creating a functional processor. While the model represents a foundational understanding of processor design, future iterations could explore enhancements such as pipelining or additional instructions to further augment its capabilities.

In summary, the project's successful implementation of a single-cycle processor in Logisim stands as a testament to the comprehension and application of fundamental computer architecture concepts.