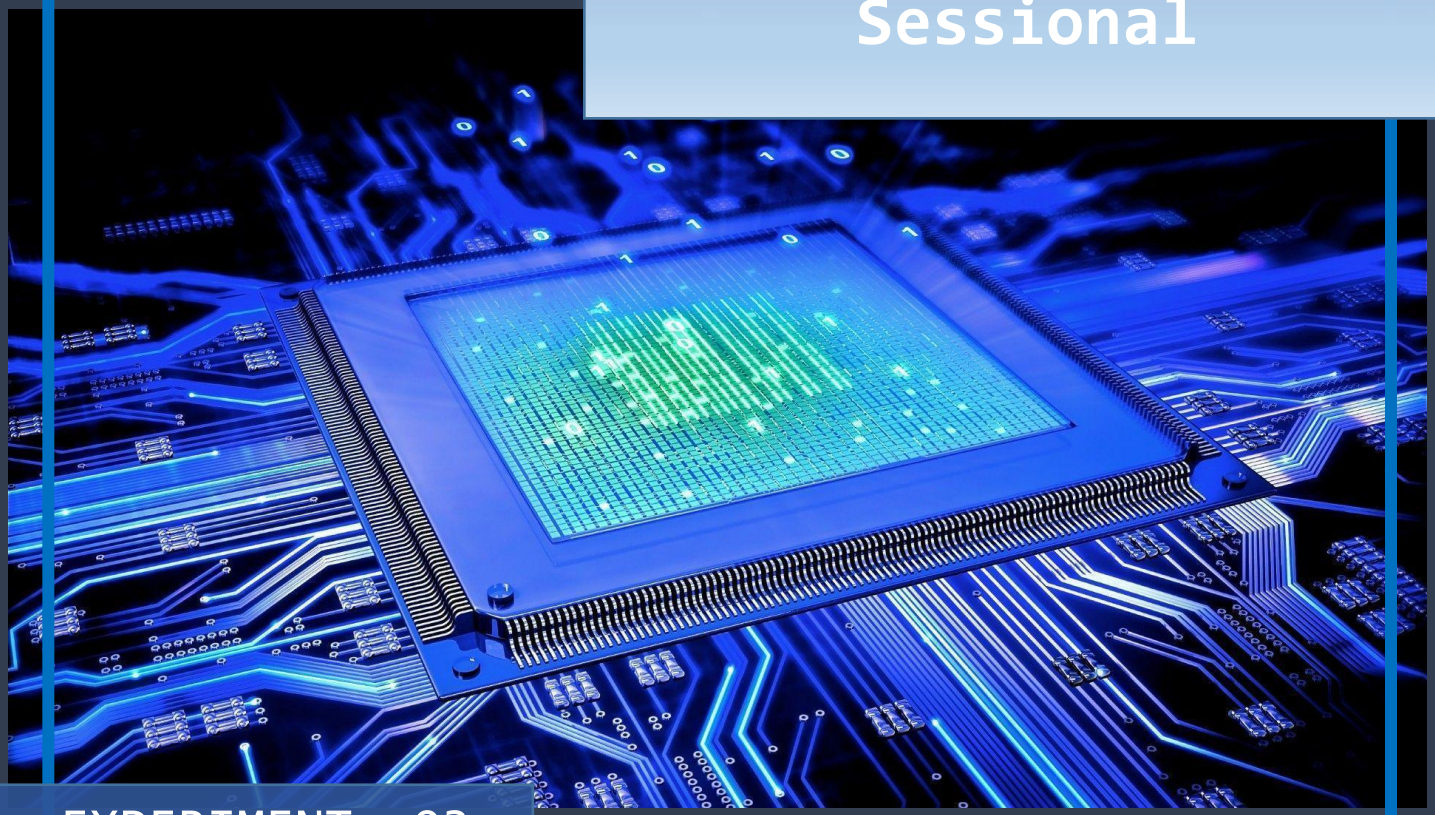
CSE 306

Computer Architecture Sessional



EXPERIMENT: 03

NAME OF EXPERIMENT:

8-BIT MIPS DESIGN AND SIMULATION

GROUP NO: 04

Section: A2

Department: CSE

Group Members: 1705041
1705047
1705049
1705050
1705052

Date of Submission: 20-06-2021

Introduction:

A MIPS processor is one version of a reduced instruction set computer (RISC).Our design is an 8 bit MIPS processor that implements the MIPS instruction set. Each instruction takes 1 clock cycle to be executed.

The MIPS processor has three main components:
· Data path · Controller · ALU control

We will be examining an implementation that includes a subset of the core MIPS instruction set:
- The memory-reference instructions load word (lw) and store word (sw).
- The arithmetic-logical instructions add, sub, AND, OR, and slt.
- The instructions branch equal (beq) and jump (j).

Datapath: Since the processor is an 8 bit processor, the datapath is eight bits wide.

Controller: The MIPS controller is responsible for decoding the fetched instructions from memory and sending the control signal to the data path. The ALU control signals to the ALU control. The Controller is also responsible for the memory writes and program counter.

ALU control: The ALU controller receives ALUOp, two bits, that determine the operation that the ALU needs to carry out. The ALU controller then sends the control signals to the ALU in order for each operation to be carried out. The ALU takes three control signals in order to determine the function the ALU needs to carry.

The length of the clock cycle will be long enough to execute the longest instruction in the MIPS instruction set. Also the clock is edge-triggered.

In examining the implementation, we will have the opportunity to see how the instruction set architecture determines many aspects of the implementation, and how the choice of various implementation strategies affects the clock rate and CPI for the computer.

INSTRUCTION SET DESCRIPTION:

Instruction ID	Instruction	Type	Opcode Number	Opcode	CONTROL BITS (HEX - 16 bits)
A	add	R	2	0010	8 2 2 0
B	addi	I	3	0011	0 2 6 0
C	sub	R	13	1101	8 5 2 0
D	subi	I	4	0100	0 5 6 0
E	and	R	12	1100	8 C 2 0
F	andi	I	1	0001	0 C 6 0
G	or	R	15	1111	8 8 2 0
H	ori	I	0	0000	0 8 6 0
I	sll	R	11	1011	8 0 2 5
J	srl	R	14	1110	8 0 2 1
K	nor	R	6	0110	8 8 3 0
L	sw	I	10	1100	1 2 E 0
M	lw	I	7	0111	3 2 6 0
N	beq	I	5	0101	4 5 0 0
O	bneq	I	9	1001	0 5 0 8
P	j(jump)	J	8	1000	0 0 0 2

High Level Block Diagram:

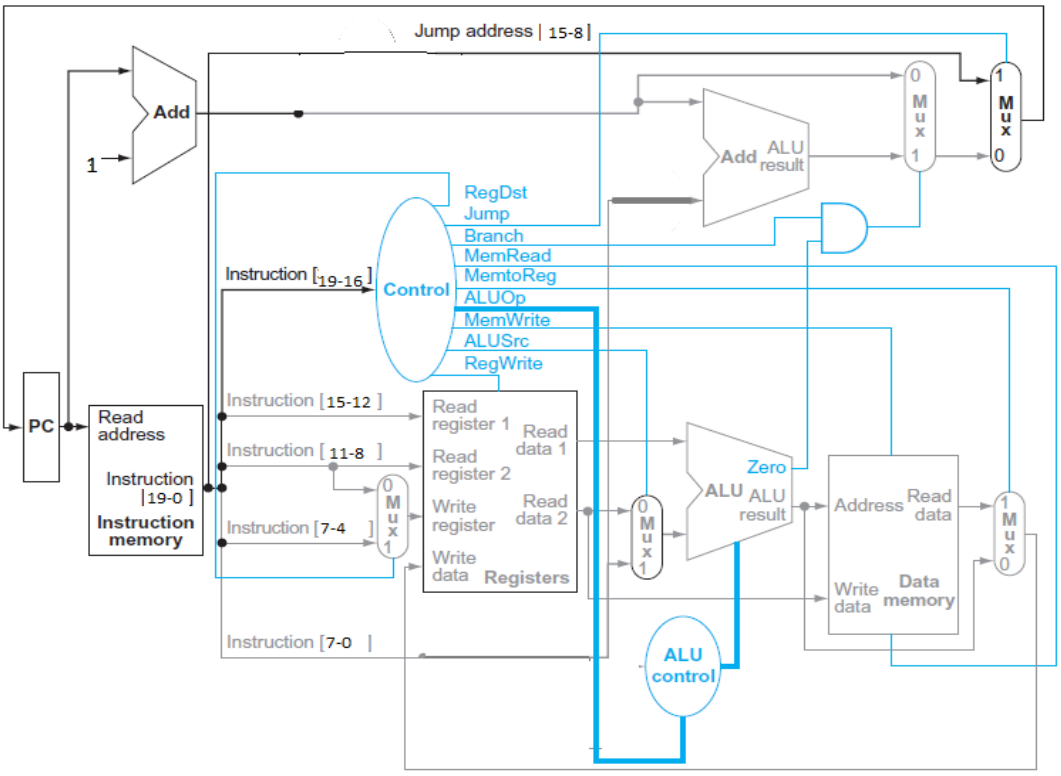


Fig-1: High Level Block Diagram of MIPS

Description:

PC (program counter): Points to the current instruction memory

Instruction Memory: Contains instructions to be executed

Registers: Contains registers used in the operations of the processor ($t0, $t1, $t2, $t3, $t4, $zero, $sp, $temp)

ALU: The central processing unit of the circuit

MUX: Multiplexers; selects the data bits from source 0 if selection bit is 0 and from source 1 if otherwise.

Control: The unit that provides the necessary instructions and selection bits required to perform operations throughout the circuit.

Detailed Circuit Diagram:

Instruction Separator Circuit:

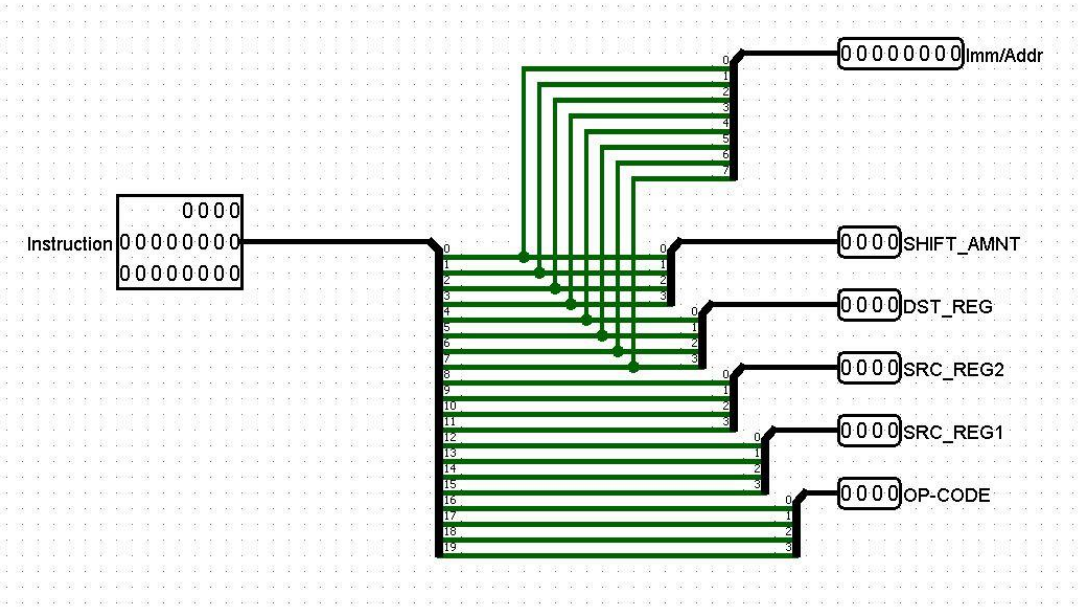


Fig-2.1: Instruction Separator Circuit

Instruction Counter Circuit:

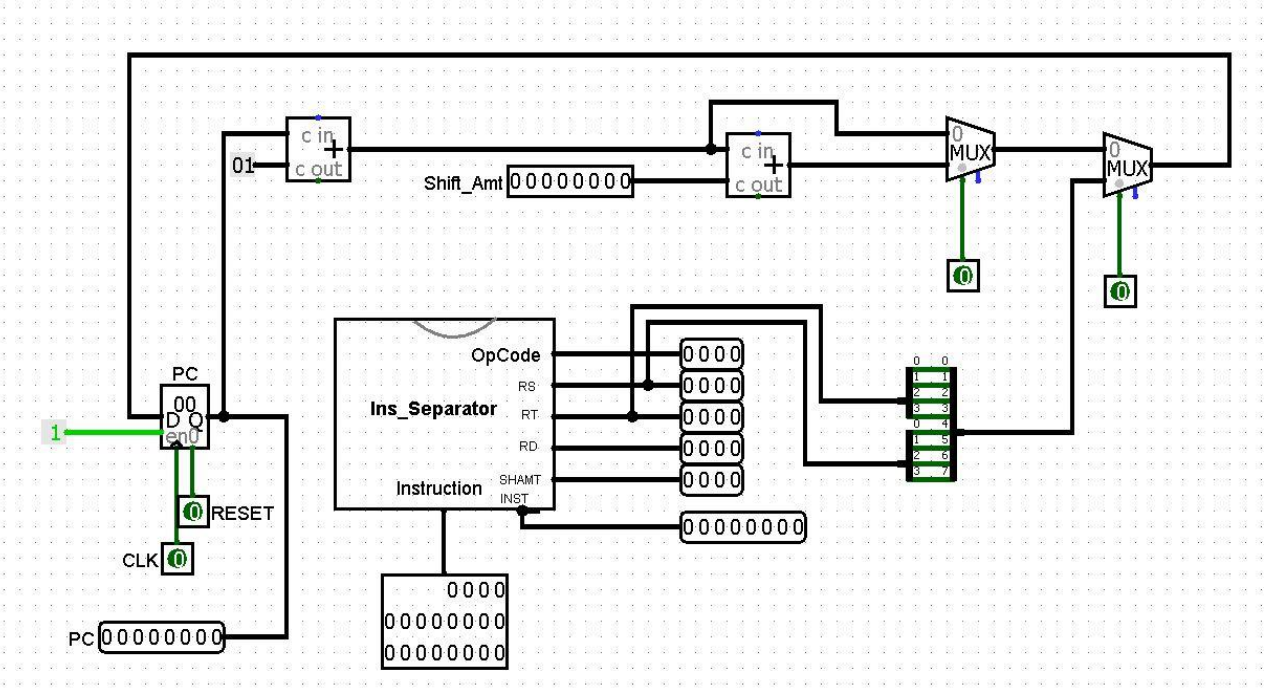


Fig-2.2: Instruction Counter Circuit

Control Circuit:

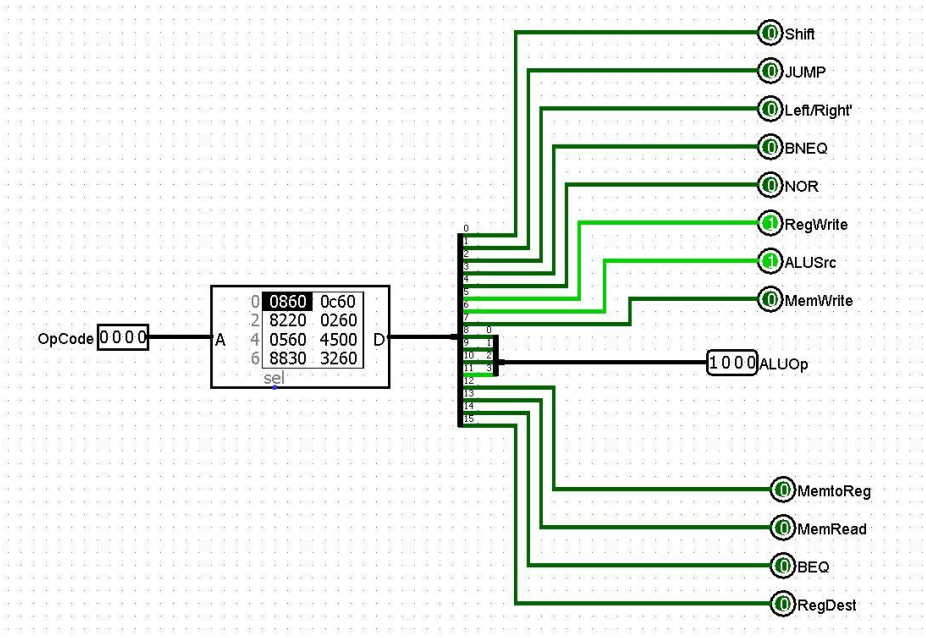


Fig-2.3: Control Circuit

Shifter Circuit:

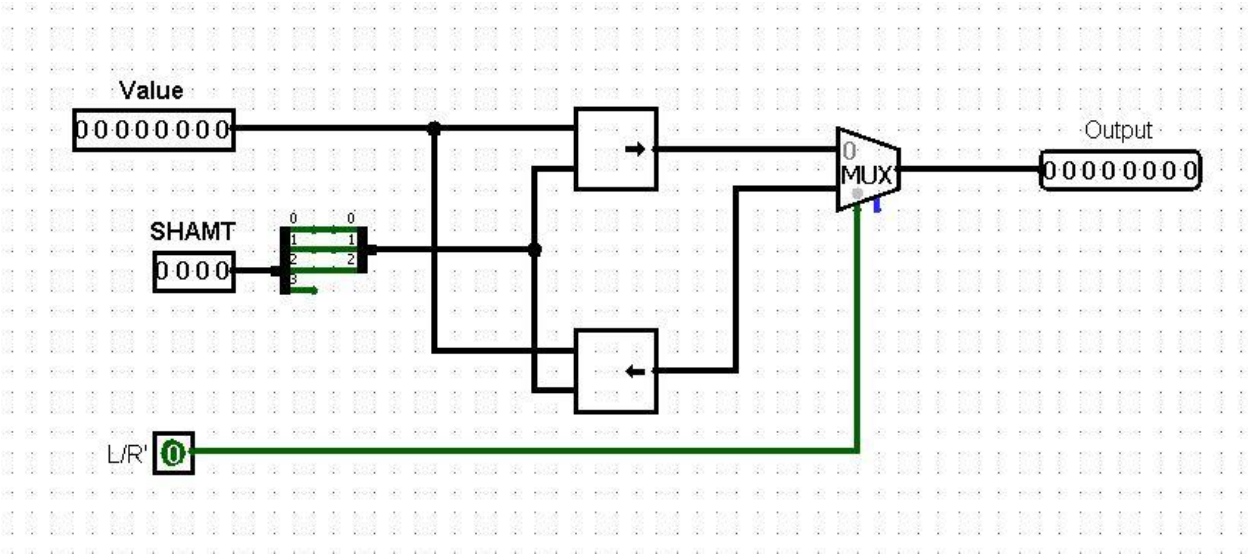


Fig-2.4: Shifter Circuit

Registers:

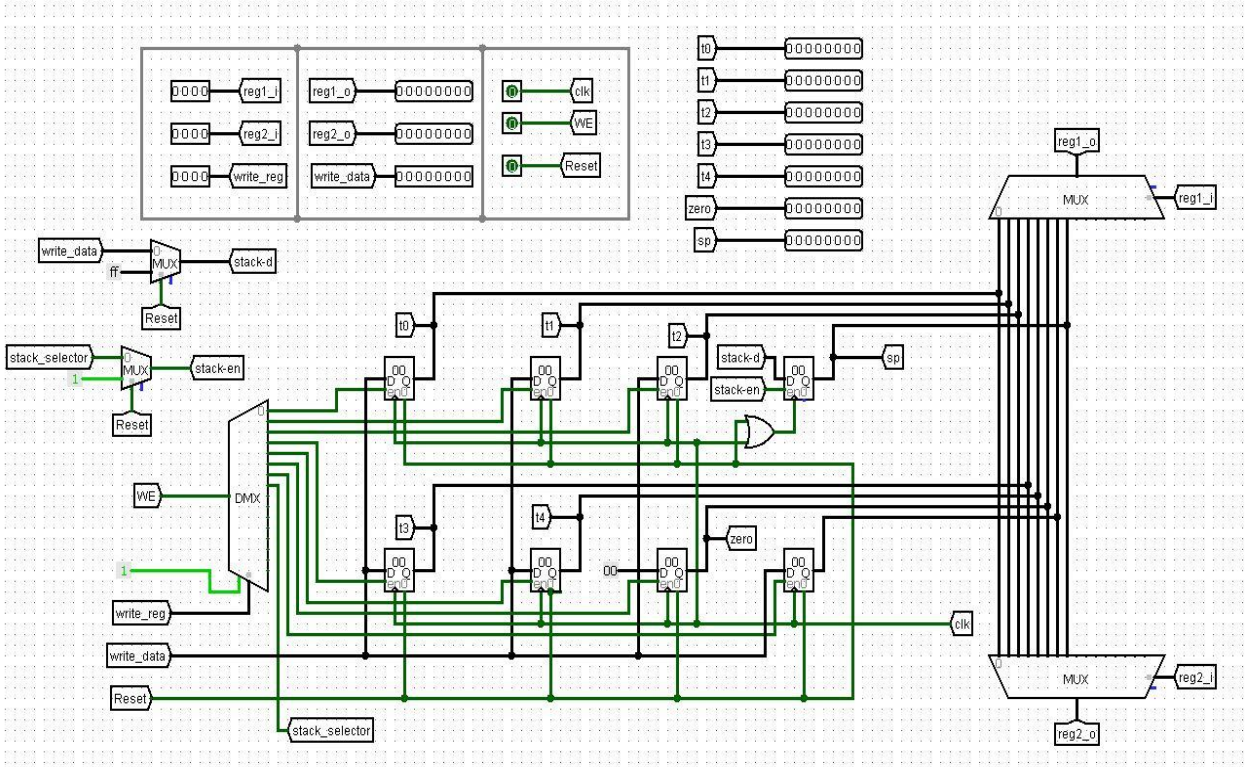


Fig-2.5: Registers Circuit

MIPS (Branch and Shift Part):

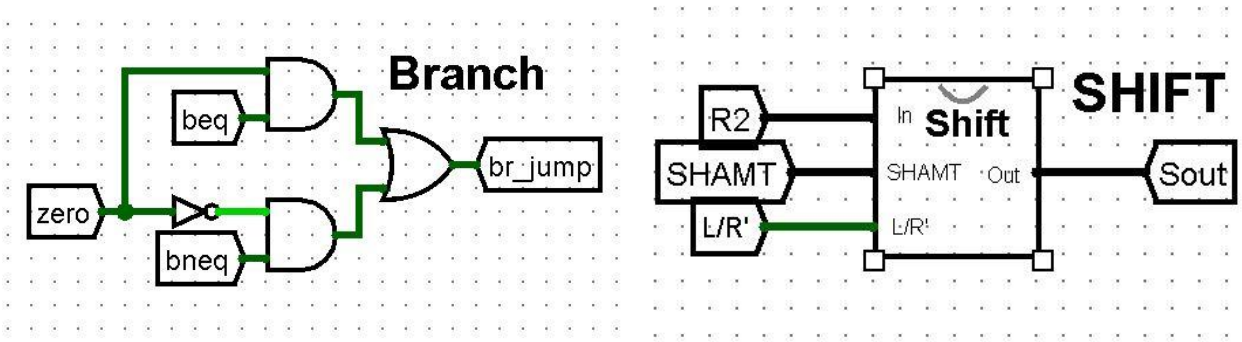


Fig-2.6: MIPS (Branch and Shift Part)

MIPS (Instruction Counter Part):

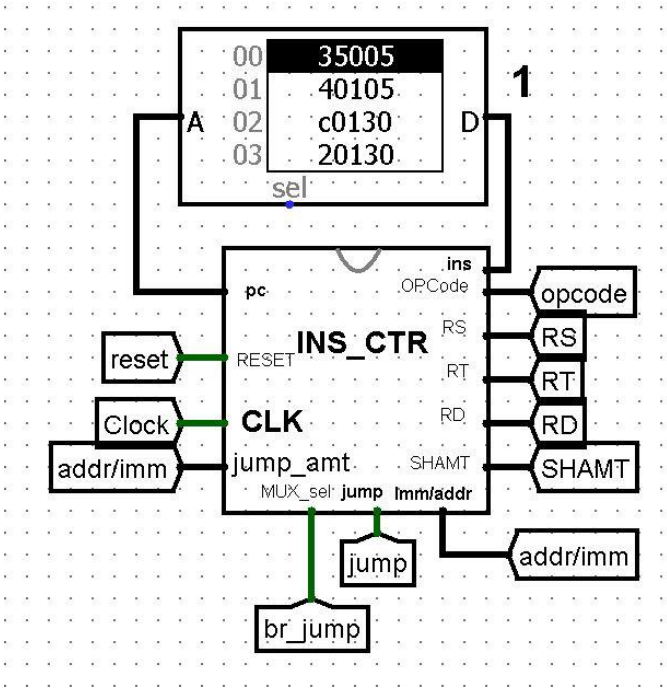


Fig-2.7: MIPS (Instruction Counter Part)

MIPS (ALU Part):

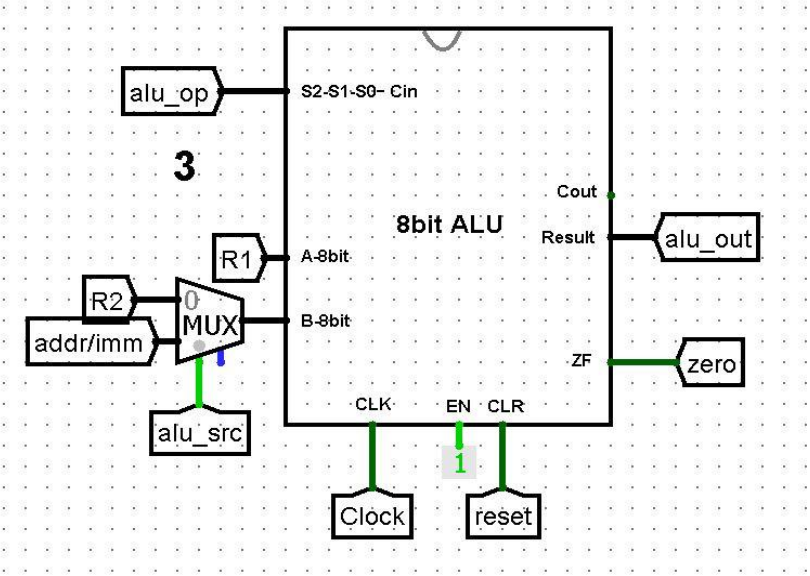


Fig-2.8: MIPS (ALU Part)

MIPS (Control Part):

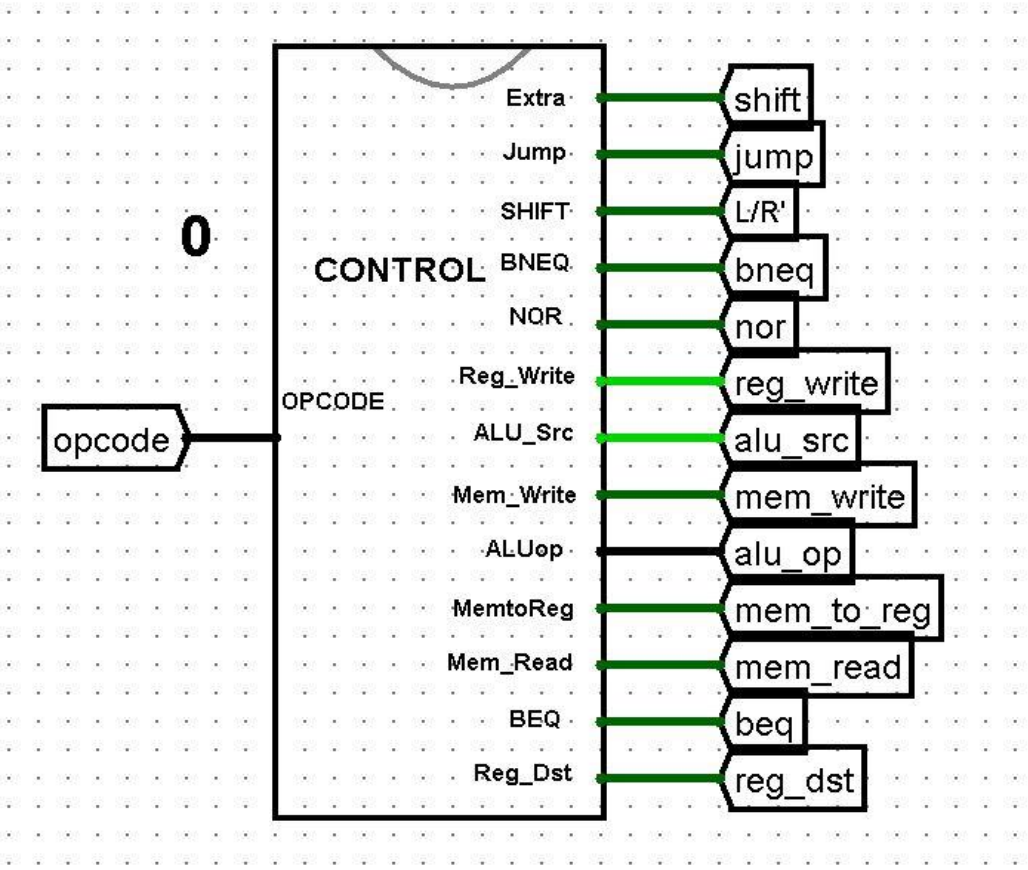


Fig-2.9: MIPS (Control Part)

MIPS (Write Data Selection Part):

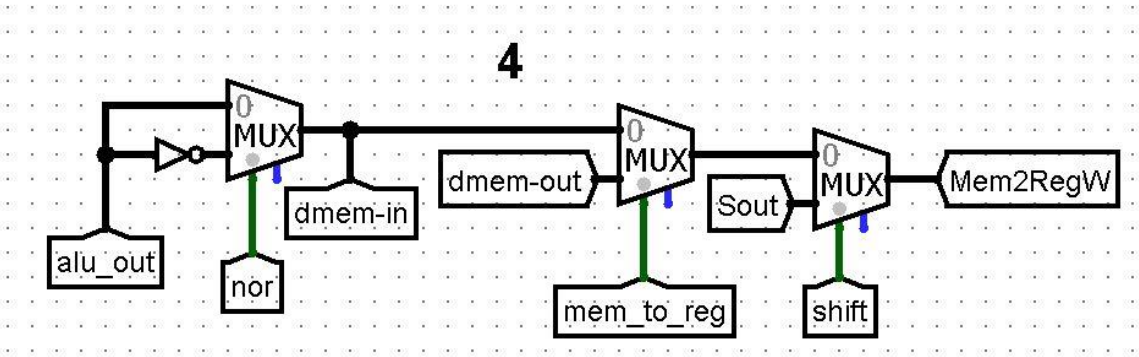


Fig-2.10: MIPS (Write Data Selection Part)

MIPS (Registers Part):

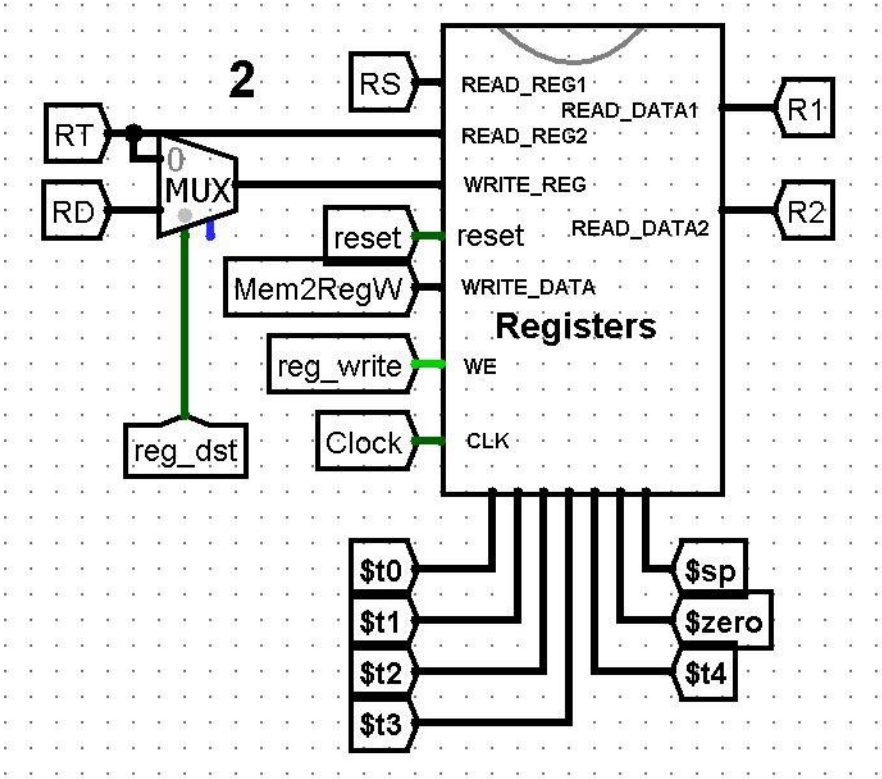


Fig-2.11: MIPS (Registers Part)

MIPS (Data Memory Part):

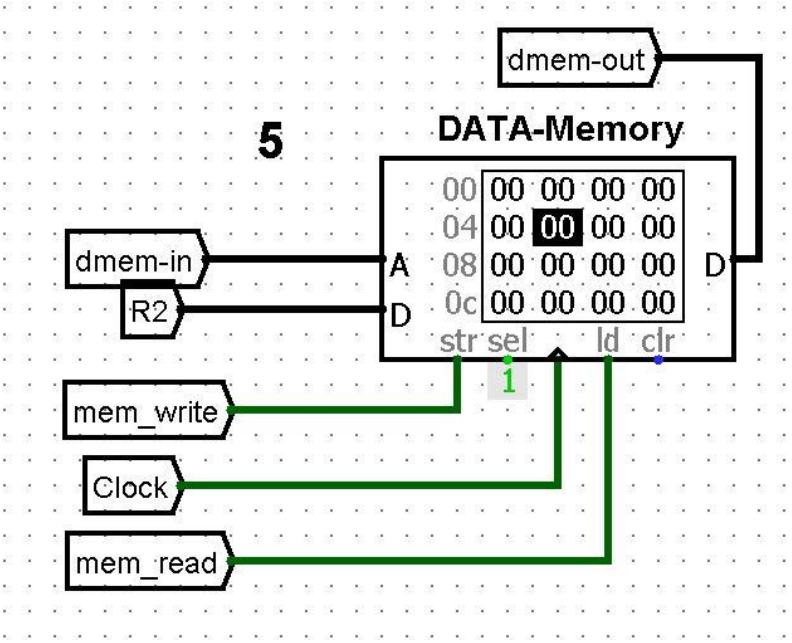


Fig-2.12: MIPS (Data Memory Part)

MIPS (Complete):

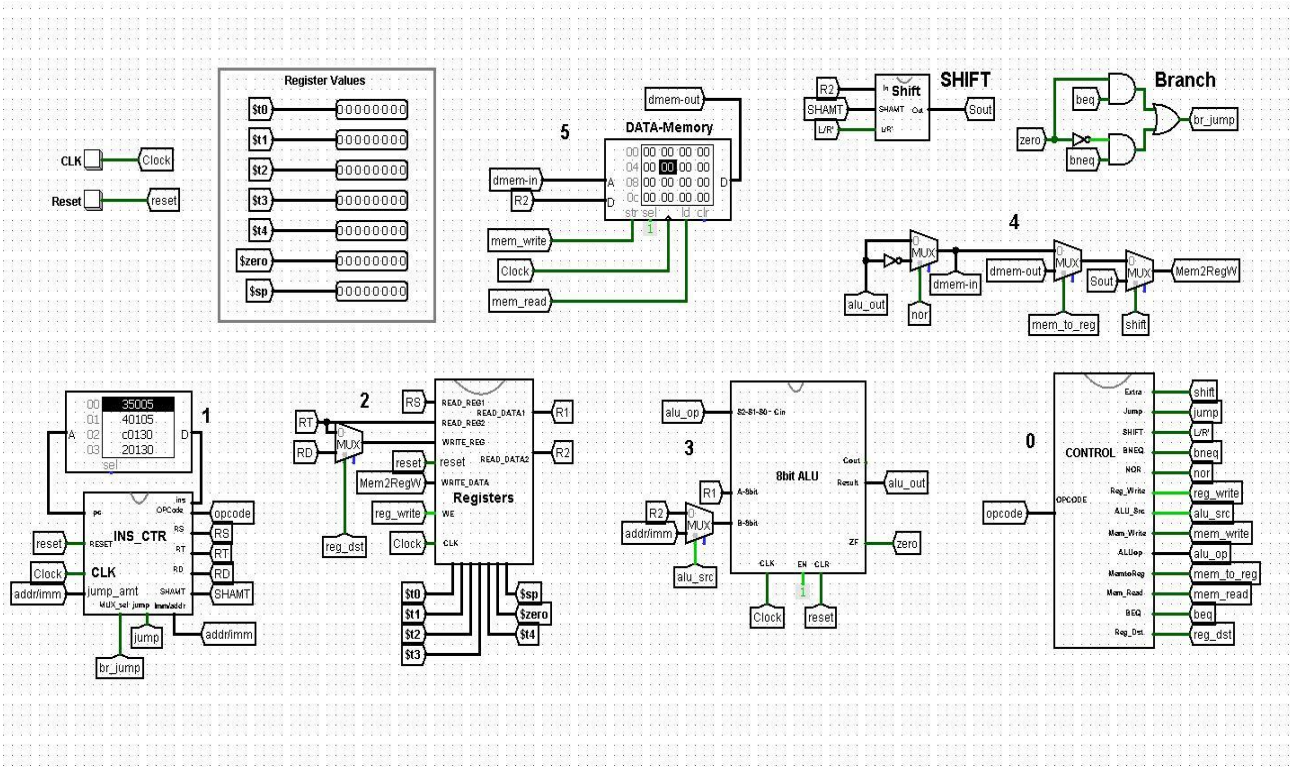


Fig-2.13: MIPS (Complete)

Approach to implement the push and pop instructions:

In case of push or pop operations, the MIPS code is converted into load and store operations by the MIPS to machine code converter. The operation procedure is described below:

push t_i$:

The code is converted into load and add operations as follows:

```
sw $ti, 0($sp)  // value of the register is stored into stack memory
addi $sp, $sp, -1  // stack pointer is decremented by 1
```

push x(t_i$) :

The code is converted into load and add operations as follows:

```
addi $temp, $ti , x    // register temp gets the address of the memory from base
                       // address of $ti and x(by adding them)
lw $temp, 0($temp) // value at location temp is stored into register temp
sw $temp, 0($sp) // value at temp is stored into stack
addi $sp, $sp, -1 // stack pointer is decremented by 1
```

pop t_i$:

The code is converted into load and add operations as follows:

```
lw $ti, 0($sp) // value at top of stack is loaded into register $ti
addi $sp, $sp, 1 // stack pointer is incremented by 1
```

IC Count:

Used IC	Operation	Gates Used	Gates Per IC	IC Count
IC 74LS08	AND	11	4	3
IC 74LS32	OR	12	4	3
IC 74LS04	NOT	17	6	3
IC 74157	MUX (2:1)	69	4	18
Logisim	MUX (16:1)	16		
Logisim	DMUX (1:16)	1		
IC 7480	1-bit Full Adder	8	1	8
(Logisim)	Shifter	2		
IC 74LS83	4 bit Full Adder	4	1	4
(Logisim)	8 bit Register	9		
(Logisim)	RAM (Address bit 8)	1		
(Logisim)	ROM (Address bit 8)	1		
(Logisim)	ROM (Address bit 4)	1		

Total IC used: 39

Simulator Used:

Logisim (Version 2.7.1)

Discussion:

The current MIPS processor is designed as an educational tool for us to further study the MIPS architecture. Firstly, PC registers have updated the values of the instruction memory within one clock cycle. Then with the help of a rom as a control system, we have executed the opcodes of different instructions. Then the bits were selected in order with the required format of the instructions. In the register file, 5 bit addresses are pushed and 32 bit values are found as outputs. With the help of read/write registers, we have read the values from the registers or we have written data on them. Then ALU is used in order to do logical or arithmetic calculations of both values and offset. Moreover a data memory is used to execute load and store instructions. For branch target address and jump instructions, adders are used. As the whole operation is executed in a single architecture structure, some MUX's are used to choose the execution lines as per the instructions.

This design uses 1 clock cycle for all the components. So there are two memories used; one for instructions and another for data memory. This is a great structure in order to learn the single datapath formation. But it is not effective enough to use. In here, to execute load/store instructions, we needed longer time than any other instruction. On the other hand, jump instructions needed less time to execute. But for having a single cycle architecture, all the instructions have to be executed at a similar speed.

Though this architecture is not used in real world, this single cycle MIPS processor is highly helpful on how to implement MIPS instruction sets in a 8 bit design.