**Group 06**

Pratyush Kumar Swain: 230101079

Prayansh Kumar: 230101081

Ritvij Gopal: 230101085

Parv Goyal: 230101124

**OBJECTIVE:**

TO DESIGN A SINGLE CYCLE 32-BIT MIPS (Microprocessor without Interlocked Pipelined Stages) THAT INCLUDES A SUBSET OF CORE MIPS INSTRUCTION SET

**INSTRUCTIONS TO IMPLEMENT:**

1. Memory reference instructions (load, store)
2. ALU instructions (add, sub, and, or, slt (set less than) )
3. Control transfer instructions (beq, jump)
4. Instruction for supporting subroutine (jump, jump and link)

In MIPS, we have 32 registers, each 32-bit wide. There are 3 types of instructions:

1. R-Type instructions-

6-bit opcode, 3 5-bit registers( 2 source registers and 1 destination register ), 5-bit shamt, 6-bit funct

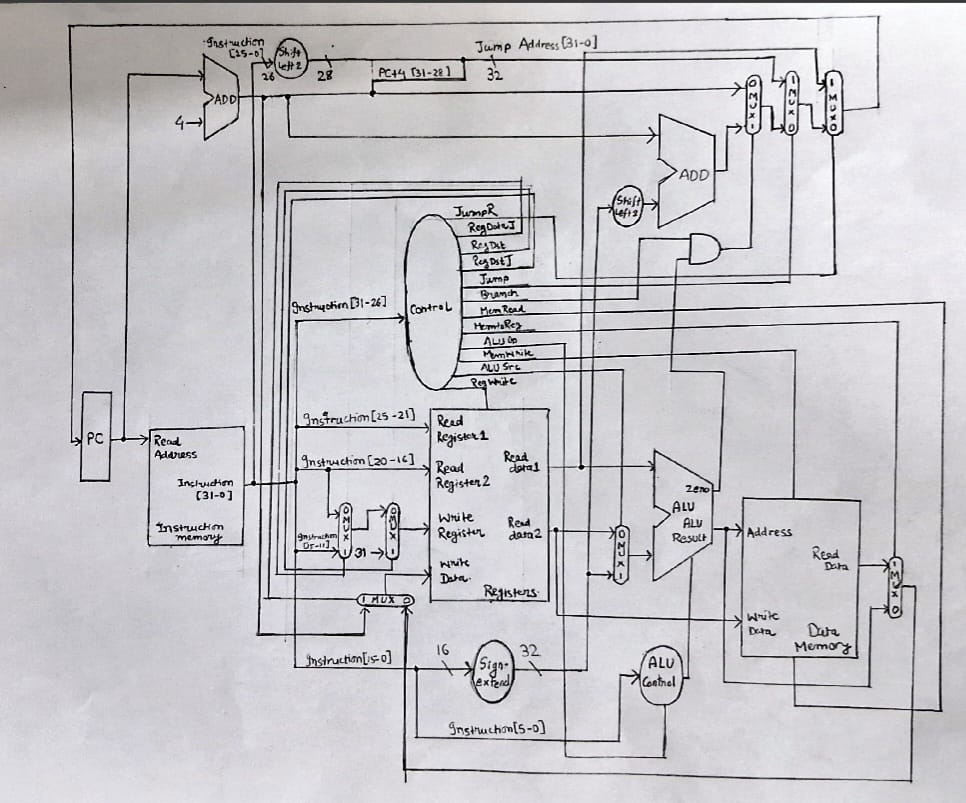
2. I-Type instructions-

6-bit opcode, 2 5-bit registers, 16-bit immediate value

3. J-Type instructions-

6-bit opcode, 26-bit position

**ARCHITECTURE**:



**KEY COMPONENTS:**

**1.Program counter:**

The PC module maintains the address of the current instruction. It has an enable input that allows updating the PC on each clock cycle when enabled. The PC is a 32-bit register with synchronous reset functionality.

**2.Instruction memory:**

The instruction memory module stores the program to be executed. It is implemented as a read-only memory with 256 32-bit locations. In our implementation, we initialize it with a test program that demonstrates all the supported instructions.

**3.Register File:**

The register file consists of 32 general-purpose registers, each 32 bits wide:

• Two read ports that allow reading two registers simultaneously

• One write port for writing data to a register

• Write enable signal to control when writing occurs

• Register $0 is hardwired to zero

**4.ALU:**

The ALU performs operations based on a 4-bit control signal:

• 0000: AND

• 0001: OR

• 0010: ADD

• 0110: SUB

• 0111: SLT (set if less than)

The ALU also outputs a zero flag that indicates whether the result is zero (used for branch decisions).

**5.Data Memory:**

The data memory has 256 32-bit word locations, with separate read and write enable signals. Memory addresses are word-aligned (divided by 4).

**6.Shifters:**

Two types of shifters are used:

• Left shift by 2 bits (multiply by 4) for branch address calculation

• Left shift by 2 bits for jump address calculation

DATA PATH OPERATIONS:

**1.R-Type Instructions:**

1. Fetch instruction from instruction memory using PC

2. Read two source registers (rs, rt) from register file

3. Perform ALU operation based on function code

4. Write result to destination register (rd)

5. Increment PC by 4

**2.Store:**

1. Fetch instruction from instruction memory

2. Read base register (rs) and data register (rt) from register file

3. Sign-extend immediate field

4. Calculate memory address using ALU (base + offset)

5. Write data to memory at the calculated address

6. Increment PC by 4

**3.Load:**

1. Fetch instruction from instruction memory

2. Read base register (rs) from register file

3. Sign-extend immediate field

4. Calculate memory address using ALU (base + offset)

5. Read data from memory at the calculated address

6. Write data to destination register (rt)

7. Increment PC by 4

**4.Branch if equal:**

1. Fetch instruction from instruction memory

2. Read two registers (rs, rt) from register file

3. Compare register values using ALU subtraction

4. Sign-extend and shift immediate field (multiply by 4)

5. If registers are equal (ALU zero flag is set):

• Calculate target address (PC + 4 + shifted immediate)

• Set PC to target address

6. Else: Increment PC by 4

**5.Jump:**

1. Fetch instruction from instruction memory

2. Shift jump target left by 2 bits

3. Combine upper 4 bits of (PC + 4) with shifted target

4. Set PC to jump address

**6.Jump and Link:**

1. Fetch instruction from instruction memory

2. Calculate return address (PC + 4)

3. Write return address to register $31 (ra)

4. Shift jump target left by 2 bits

5. Combine upper 4 bits of (PC + 4) with shifted target

6. Set PC to jump address

**7.Jump register:**

1. Fetch instruction from instruction memory

2. Read register (rs) from register file

3. Set PC to the value in rs

**8.Add immediate:**

1. Fetch instruction from instruction memory

2. Read source register (rs) from register file

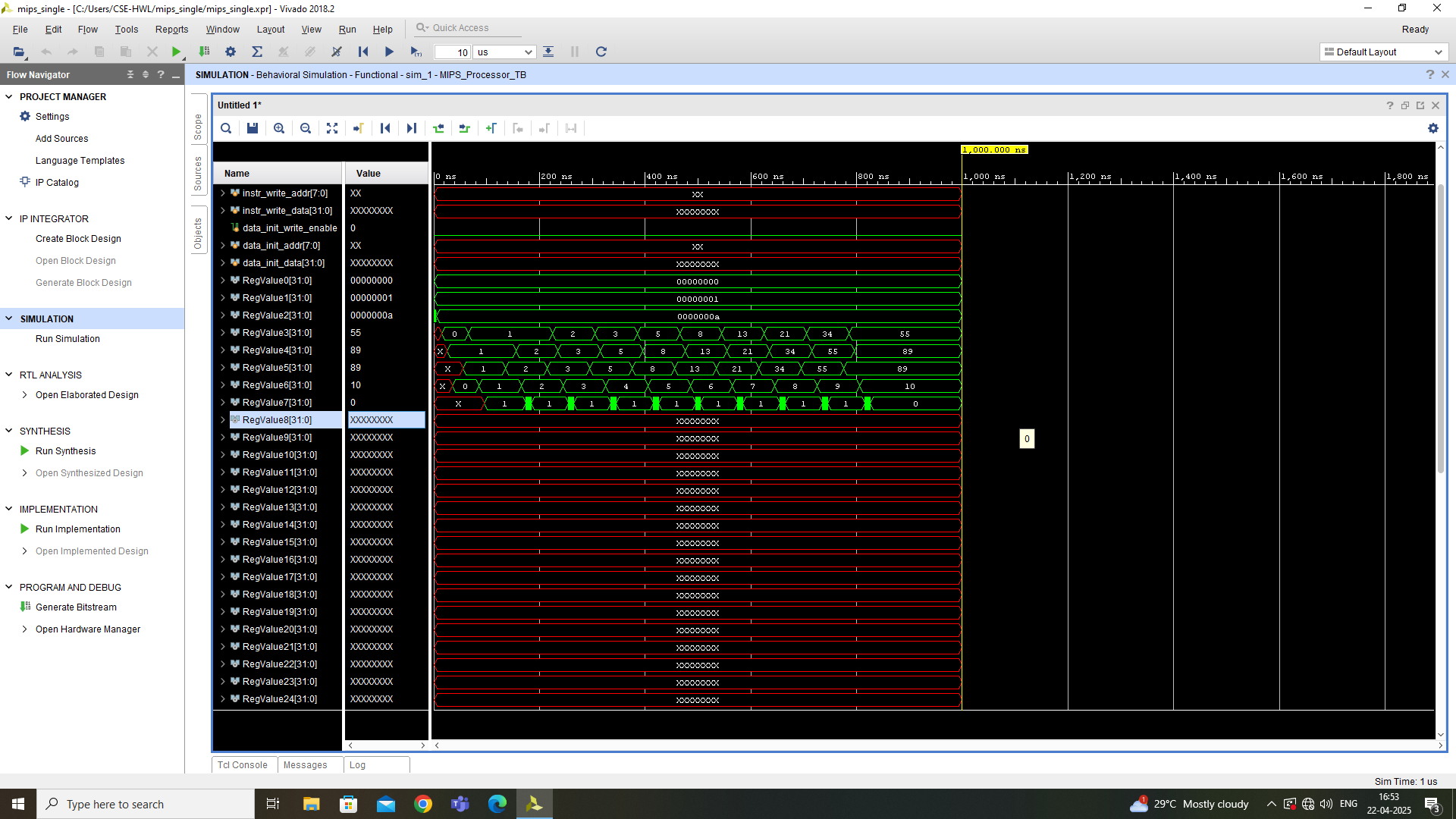
3. Sign-extend immediate field

4. Add register value and sign-extended immediate

5. Write result to destination register (rt)

6. Increment PC by 4

**SIMULATION RESULTS:**

****