BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

CSE306 : COMPUTER ARCHITECTURE SESSIONAL

February 11, 2024

Topic : 4-bit MIPS Design, Simulation, and Implementation

Section : B2
Group : 05

Student ID :
1. 2005091
2. 2005098
3. 20050102
4. 20050109
5. 20050118

1 Introduction

MIPS(Microprocessor without Interlocked Pipelined Stages) is a family of **Reduced Instruction Set Computer (RISC) In- struction Set Architectures(ISA) developed by MIPS com- puter Systems.** MIPS is a modular architecture supporting up to four co-processors (CP 0/1/2/3). In MIPS terminology, CP-0 is the System Control Co-processor (an essential part of the processor thatis implementation defined in MIPS I–V), CP-1 is an optional floating- point unit (FPU) and CP-2/3 are optional implementation-defined co-processors (MIPS III removed CP-3 and reused its opcodes for other purposes). For example, in the PlayStation video game console, CP-2is the Geometry Transformation Engine (GTE), which accelerates the processing of geometry in 3D computer graphics.
Here is an example of sub instruction:

Operation	Instruction	Action
Subtraction	sub $t1,$t2, $t3	$t1 = $t2 - $t3

Table 1: Simple MIPS Instruction

Here, $t1,$t2,$t3 are registers that hold values. The expression is m = a - b , In this example, value of a is stored in $t2 register, valueof b is stored in $t3 and value of m is stored in register $t1.

2 Problem Specification

2.1 Instruction Set

In our Assignment, we are implementing a modified and reduced ver-sion of MIPS instruction set. Here, we are implementing 4-bit MIPS

instruction set.
Instruction set for our MIPS is given below:

Instruction ID	Instruction Type	Instruction
A	Arithmetic	add
B	Arithmetic	addi
C	Arithmetic	sub
D	Arithmetic	subi
E	Logic	and
F	Logic	adi
G	Logic	or
H	Logic	ori
I	Logic	sll
J	Logic	srl
K	Logic	nor
L	Memory	lw
M	Memory	sw
N	Control-conditional	beq
O	Control-conditional	bneq
P	Control-unconditional	j

Table 2: Instruction Set Description

There are 4 formats in our MIPS instruction set. These are :

1. **R type**
2. **S type**
3. **I type**
4. **J type**

Opcode	Src Reg 1	Src Reg 2	Dst Reg
4 bits	4 bits	4 bits	4 bits

Table 3: R type

Opcode	Src Reg 1	Dst Reg	Shamt
4 bits	4 bits	4 bits	4 bits

Table 4: S type

Opcode	Src Reg 1	Src Reg 2/Dst Reg	Addr./Immedi.
4 bits	4 bits	4 bits	4 bits

Table 5: I type

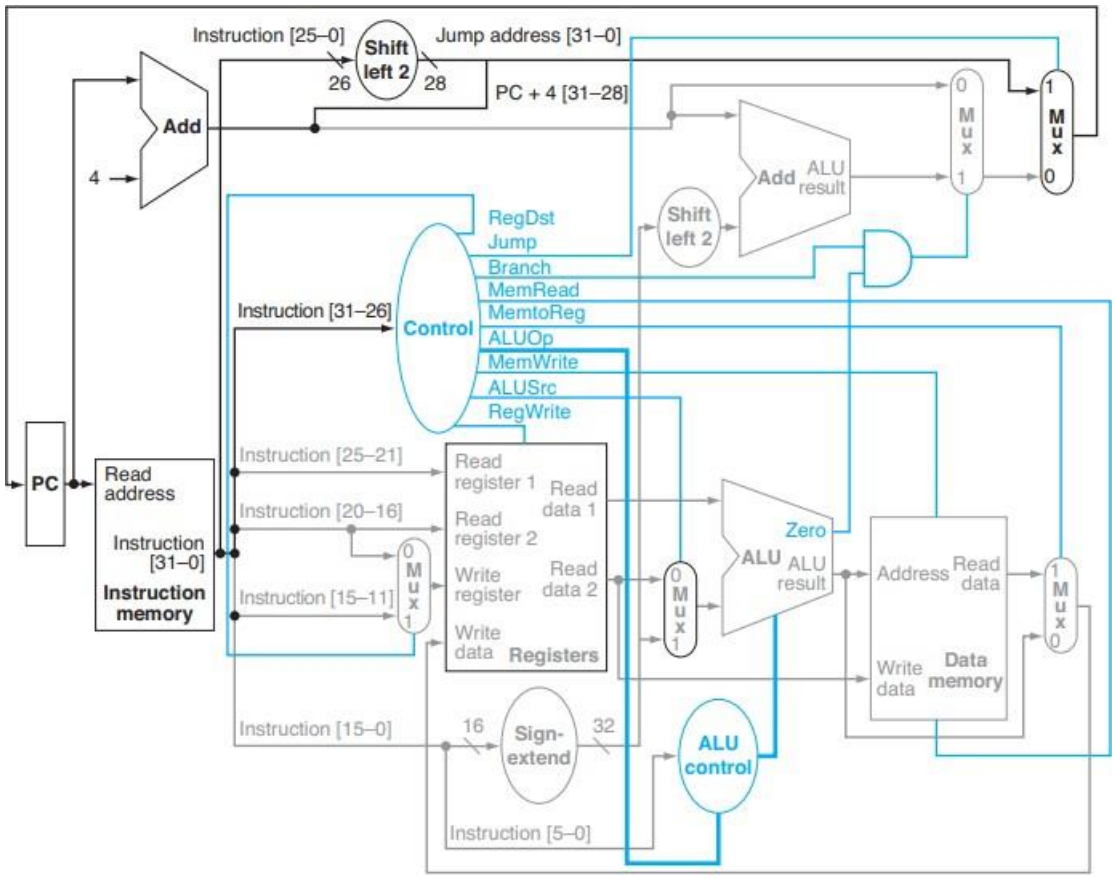
Opcode	Target Jump Address	0
4 bits	8 bits	4 bits

Table 6: J type

Opcode	Instruction Type	Instruction
0000	Memory	lw
0001	Logic	sll
0010	Arithmetic	sub
0011	Logic	ori
0100	Arithmetic	addi
0101	Arithmetic	subi
0110	Logic	srl
0111	Memory	sw
1000	Control-conditional	bneq
1001	Arithmetic	add
1010	Logic	or
1011	Logic	andi
1100	Logic	nor
1101	Control-unconditional	j
1110	Control-conditional	beq
1111	Logic	and

Table 7: Instruction Sequence

3. MIPS Processor Block Diagram:

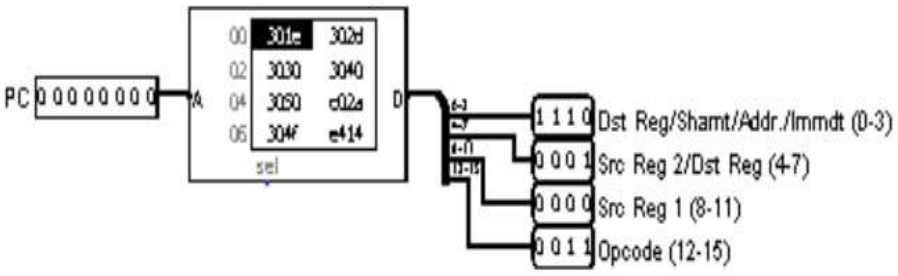


4.Detailed Design Steps:

4.1 Instruction Memory

The instructions are kept in our instruction memory using atMega32A arrays.We received the instructions as input, which we subsequently translated into MIPS instructions code, which produced instructions with 16 bits. These instructions were broken down into four 4-bit values: the destination register, also known as the immediate value, was represented by the first four bits, the first source register by the next four bits, the second source register by the next four bits, and the Opcode, which indicated the type of instruction, by the last four bits. The instruction memory generates a new

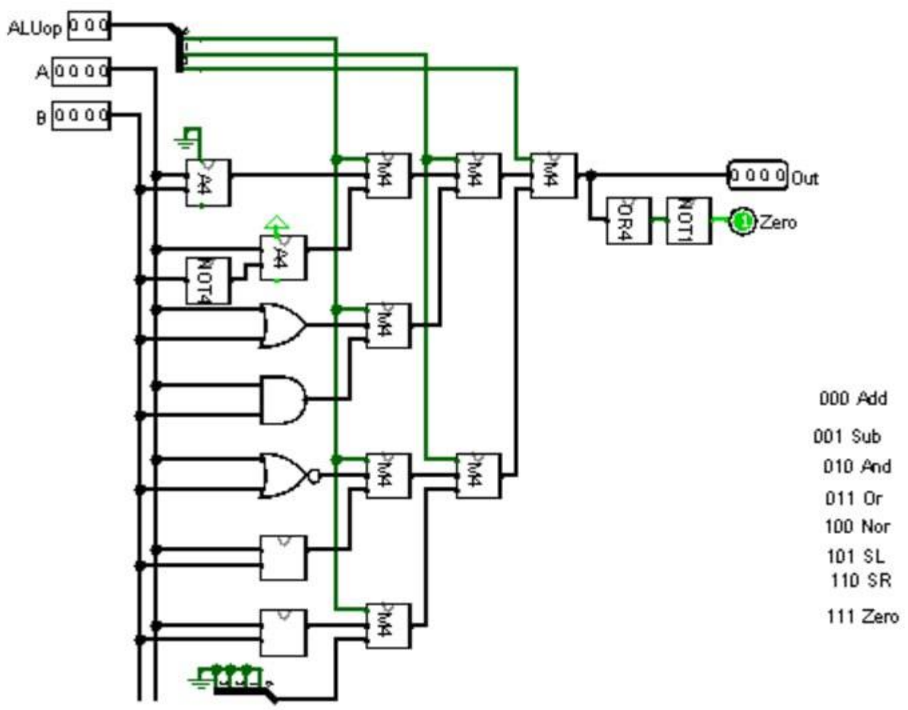
instruction each time the PC value is increased .Since instruction memory is independent of clock, we used one of the pins this Atmega to generate a clock pulse that will be used in devices that are dependent on clock.



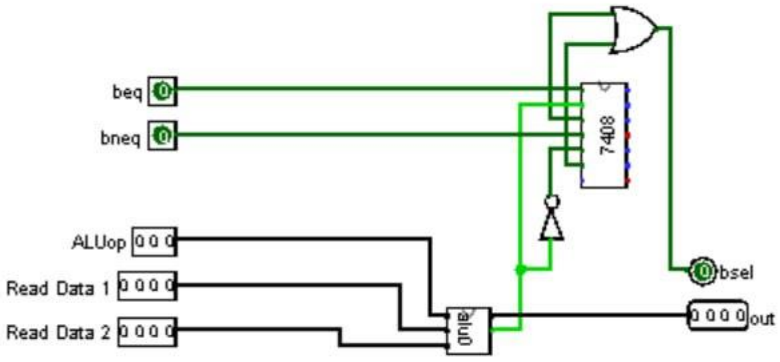
Instruction Memory

4.2 Arithmetic and Logical Unit

The creation of a 4-bit ALU to process arithmetic and logical instructions is our next assignment. Two input values and an ALUOP value are fed into the ALU, which outputs a 4-bit value that can be processed further. In addition, the ALU produces a zero flag to confirm if the output is zero. Our circuit requires the opcode and two 4-bit input values in order to use the ALU. The beq and bneq flags are also inputs. The final output value is produced by the 4-bit ALU, and a bsel flag that indicates whether or not to branch is produced by the zero flag in conjunction with the beq and bneq flags (output from the control unit).



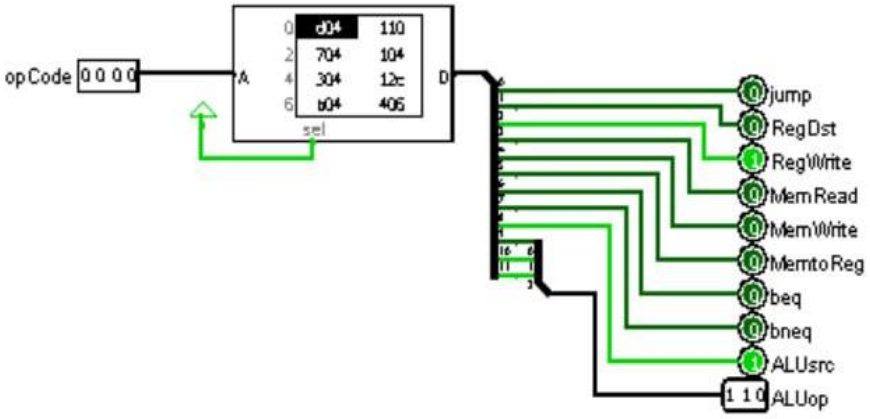
4bit ALU



ALU Branch

4.3 Control Unit

We use the instruction set provided for each group to set the values for our MUX and branch operations in the control unit. In order to accomplish this, we save the hexadecimal values for the various operations in a 16-bit ROM. We utilize the 4-bit OpCode that we receive to choose the appropriate operation from the ROM. In addition to a 3-bit ALUop that regulates the 4-bit ALU's operation, the ROM produces nine selector bits for the operation.



Control Circuit

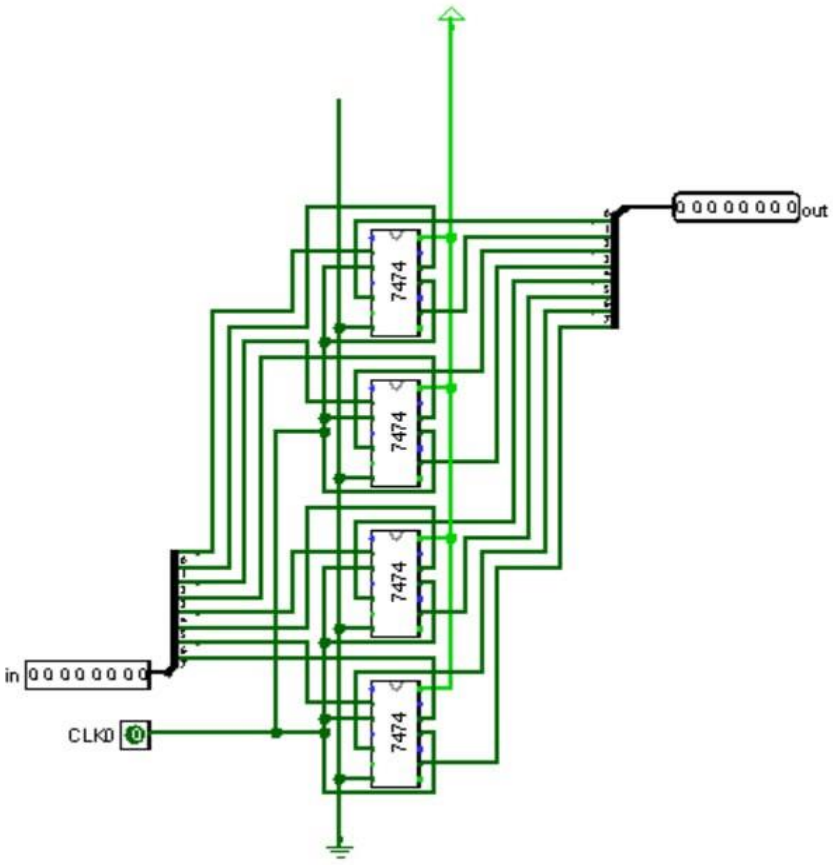
4.4 Program Count Register

This is an 8-bit register with a simple negative edge.

We utilize a PC register to record the value of our PC at any particular moment. The PC value is stored in this register, which has four D-flip flops.

While the current PC instruction is being executed, the PC register outputs the stored value of the PC once per clock cycle. The PC instruction simultaneously transmits the subsequent value for the PC as an input for the PC register. The following number, contingent on the particular instruction being executed, may be PC+1 or PC+ jump amount.

To simulate the behavior of a register, we employed an ATmega.

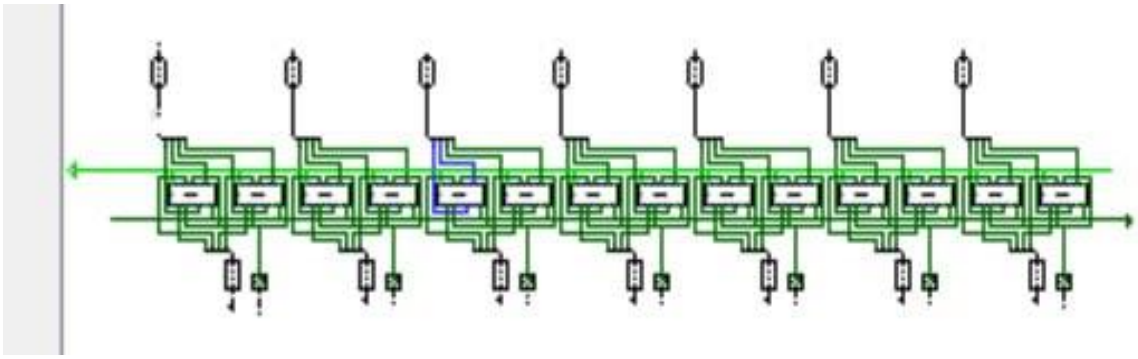


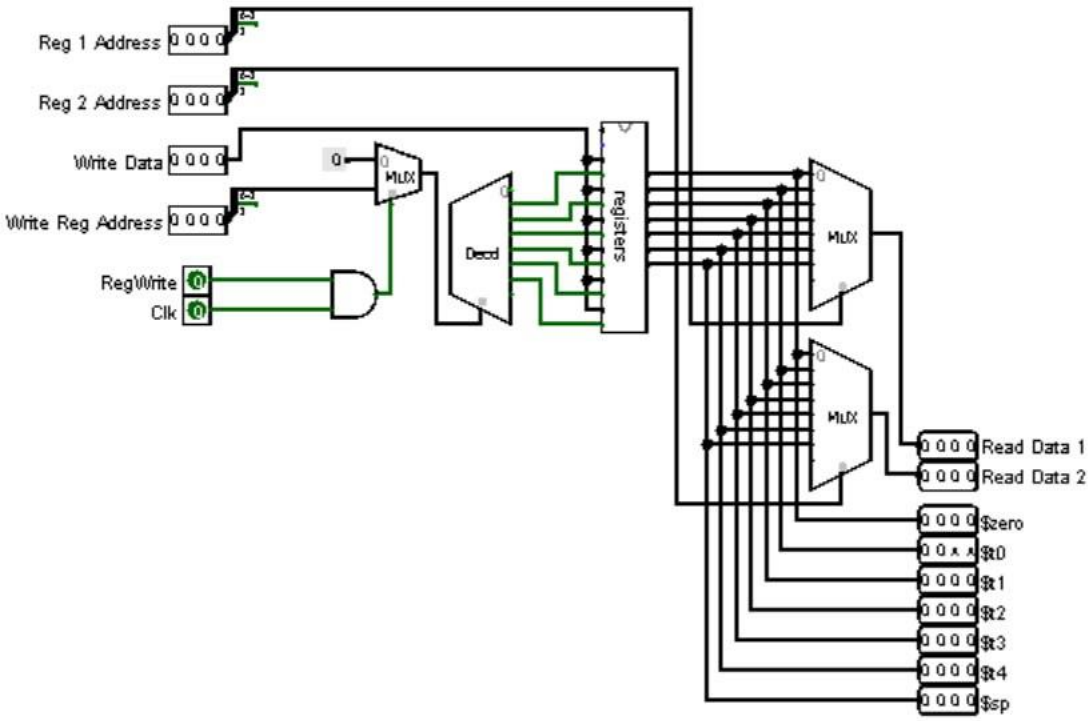
Register 8 bit

4.5 Register File

We first built a file with seven registers in order to implement our register file. We created five temporary registers out of these registers: $t0, $t1, $t2, $t3, and $t4. In addition, we developed the $zero register to handle arithmetic operations with zero value and used the $sp register to handle stack push and pop instructions.

The register file requires the two register addresses as inputs in order to read from or write to a register. When writing to a register, we additionally need to supply an input write data value and the associated write register address. On the other hand, writing in the register file is not necessary if we are carrying out a store word instruction. Thus, in order to decide whether or not to write the value, we use a selection bit called RegWrite as an input. We maintained a pin for the showReg flag, which enables us to stop the register file from operating and display the register's data, one at a time, based on the address that has been supplied.



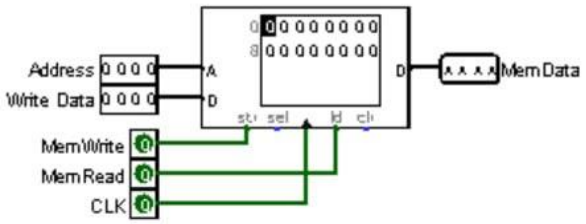


Register File

4.6 Data Memory

Our memory is stored in data memory. Writing to a 16-bit RAM requires a 4-bit memory address and data. With the MemWrite flag from control and the detection of a rising edge in the clock, it is now possible to read from memory in lw instructions and write in memory in store word instructions. Thus, we choose whether to read or write depending on two selector bits. When the MemoryRead flag is

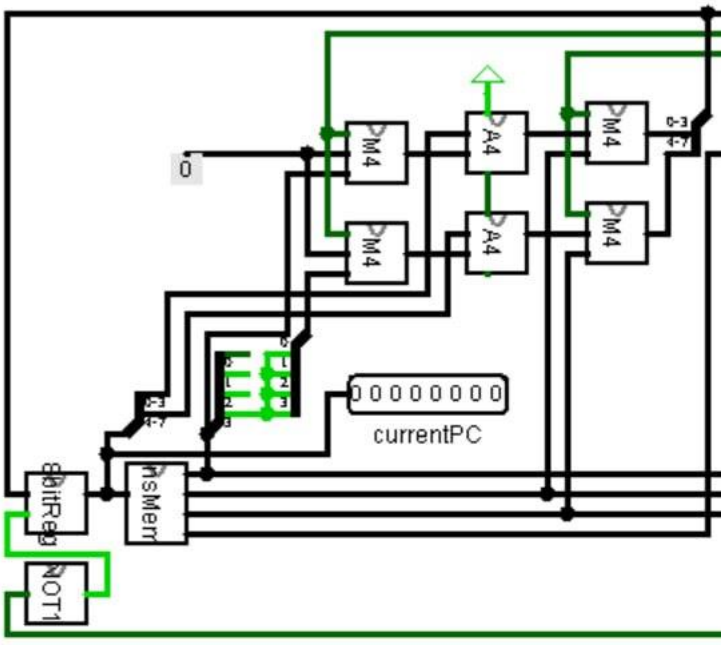
enabled, the output of memory read (lw) is our 4-bit memory data.



Data Memory

4.7 Instruction Fetch Calculation and Additional Multiplexers

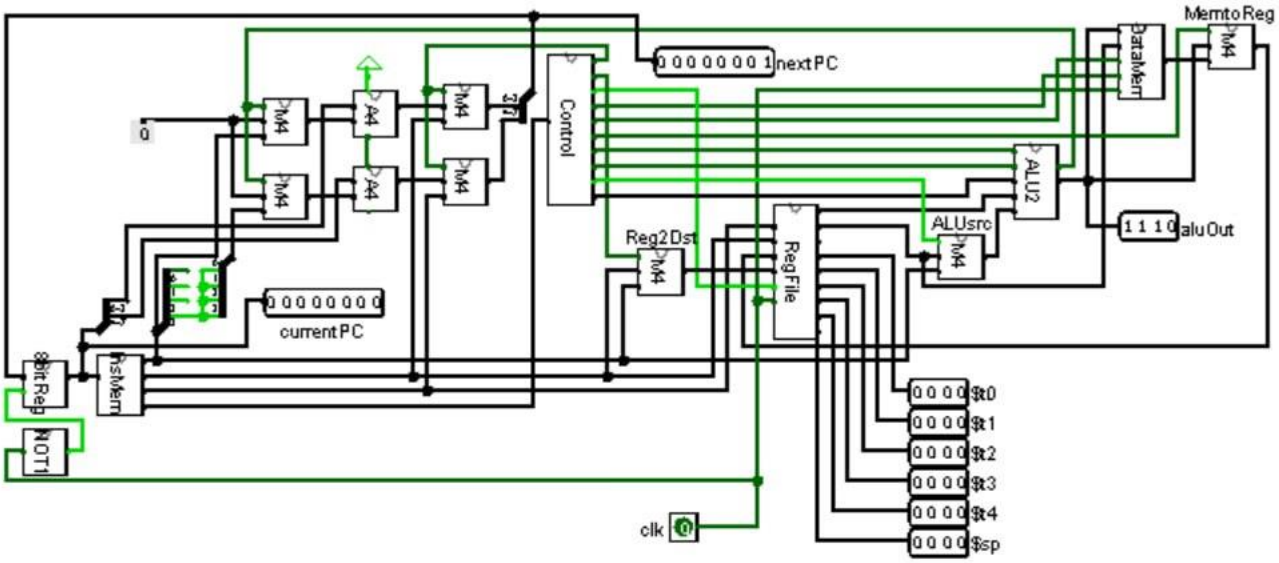
In order to simplify the implementation, we set the value of cin to 1. This allowed us to use only two 4 bit adders to implement the addition. Several multiplexers were used to control the direction of the data to input pins. For our 4 bit PC, we used 7483 Adder to calculate the value of PC. Here, the value of our PC can be either PC+1 or PC+1+branch distance.



MUX	Selector Pin	Input 0	Input 1	Description
Mux1	RegDst	Ins 4:7	Int 0:3	Decide register write address
Mux2	ALUsrc	Register File DH	Ins 0:3	Decides between register data and immediate data
Mux 3	MemtoReg	ALU output	Data Memory output	Sends either ALU output or data memory information
Branch	bsel	GND	Ins 0:3	Sends branch amount to be added
Jump	jump	Pc 0:7	Ins 4:11	Decides next pc based on jump

5 . Diagrams

5.1 Circuit Diagram



MIPS

ATMEGA32A Pin Diagram:

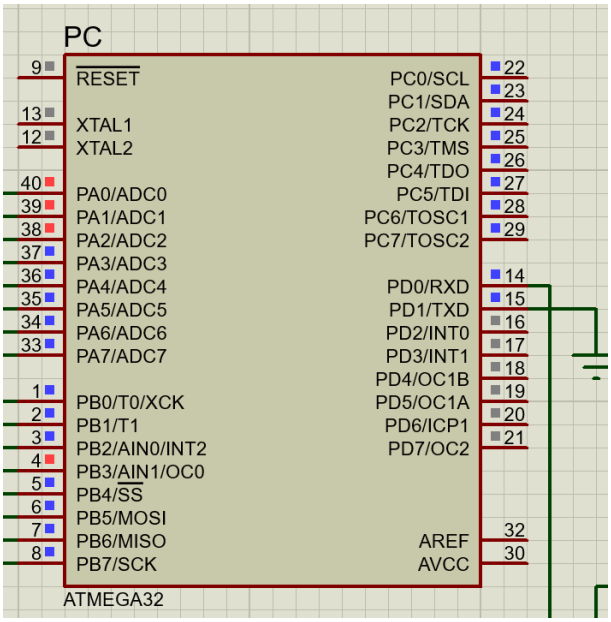


Figure: PC

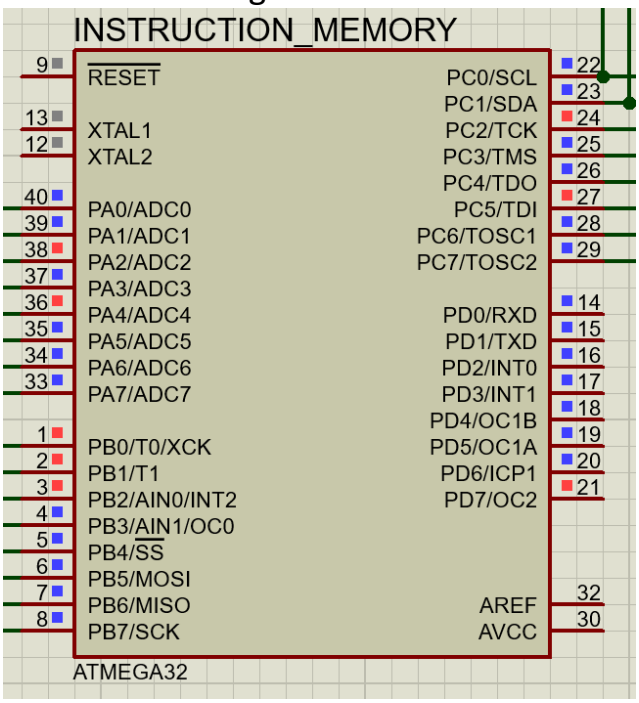


Figure: Instruction Memory

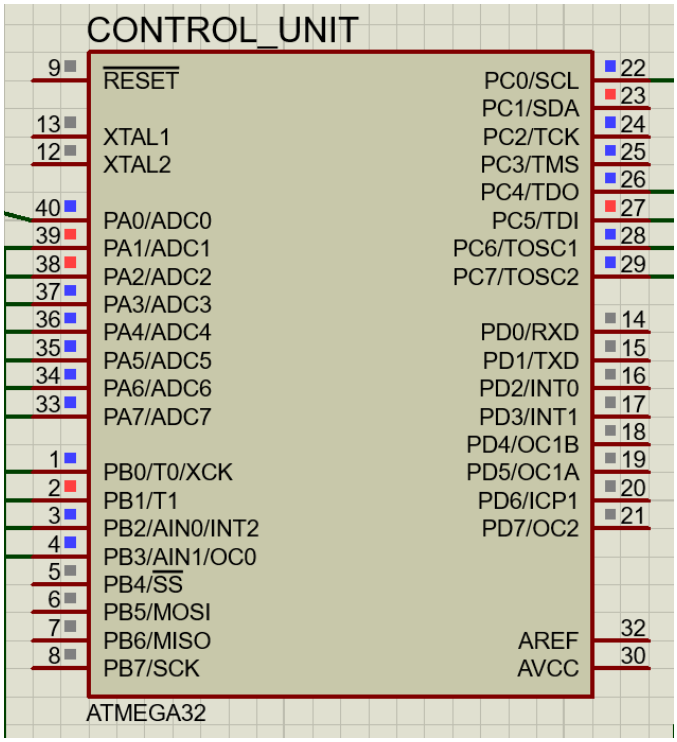


Figure: Control _Unit

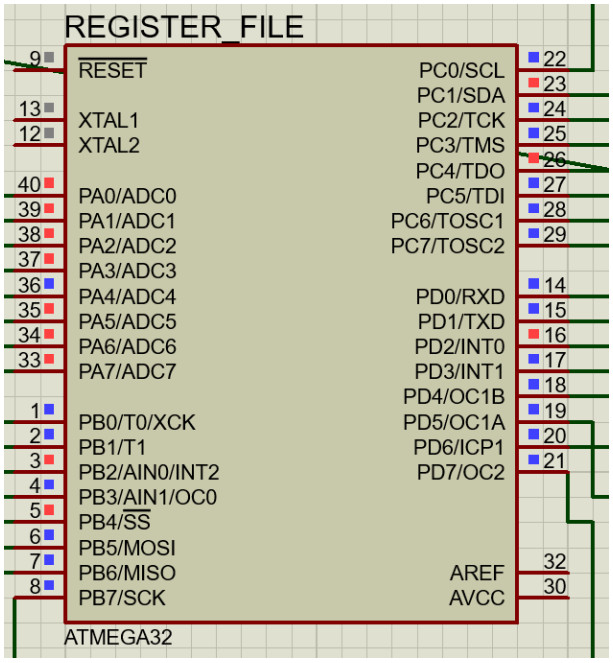


Figure: Register File

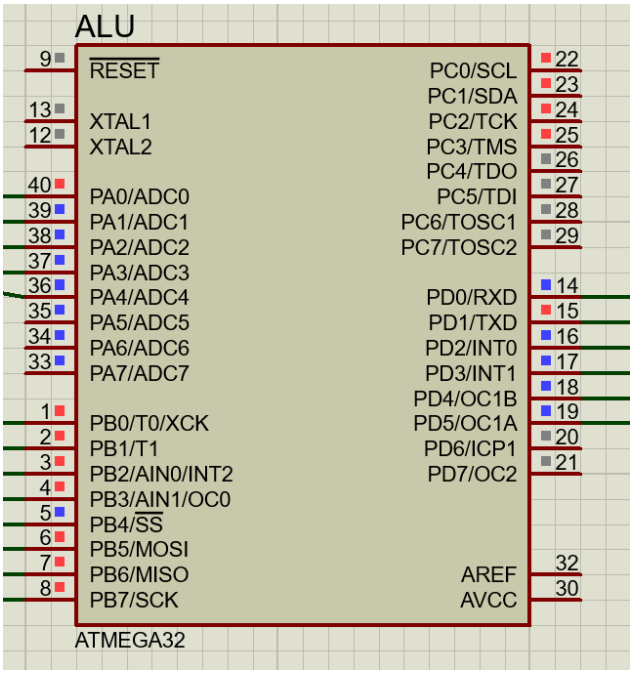


Figure: ALU

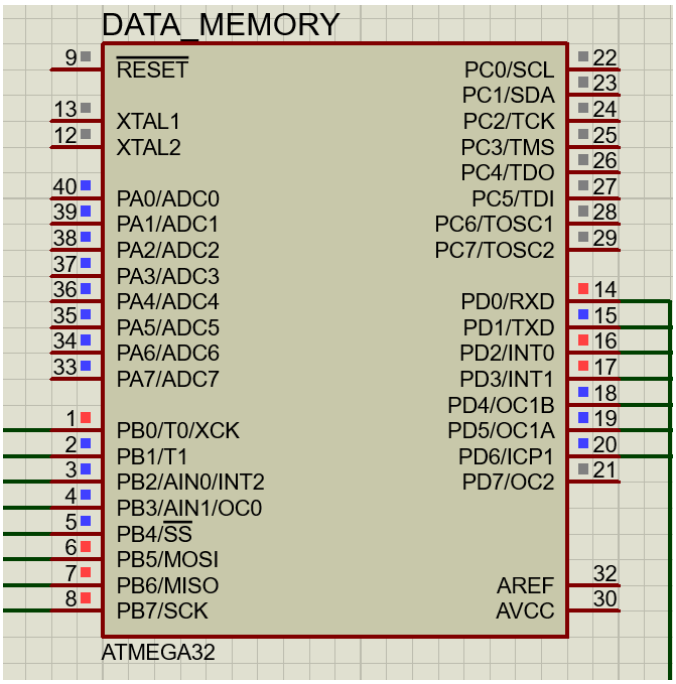


Figure: Data Memory

Required IC's:

IC Number	Count
74LS157(Quad 2 to 1 MUX)	6
74LS83(4 Bit Binary Full Adder)	2
ATMEGA32A (40 pin microprocessor)	6

Simulator: Logisim-win-2.7.1

Discussion:

In this assignment, we designed a 4-bit processor that implements the MIPS instruction set. We employed 7400-library integrated circuits for the software-level circuit implementation. At the hardware level, coding was executed specifically for ATMEGA, ensuring the required functionality. The software implementation was completed without any large setbacks but in the hardware section of the implementation we have faced many obstacles time and time again. We had to spend a lot of time to determine which ports of the Atmega32 to use for which case.

We also faced some problems regarding power distribution in the circuit. Additionally, our breadboards had some problems where the Atmega's were not being set up properly while some ports were shorted. The connections were given very carefully, particularly the jumper wires were checked before using them in the circuit

We adhered to the provided circuit diagram, emphasizing simplicity during the design process. Validation involved testing different inputs and ensuring corresponding outputs matched in our simulation, leading to successful completion.

Ultimately it has been a learning experience and have been a very satisfying experiment.

Contribution Of Each Member:

Activity	Roll
Steps Designing	91,102,109
Optimization and IC minimization	91,118
Block Diagram	98,109
Logisim Implementation	98,102,118
Proteus Implementation	91,102,118
Equipment Management	98,109,118
Hardware Implementation	91,102,109
Hardware Checking and fixing Error	98,102,118
Report Writing	91,98,102,109,118

Figure: Contribution Table