CSE 306
Computer Architecture Sessional

Assignment-3: 4-bit MIPS Design, Simulation, and Implementation

Section - A1

Group - 03

Members of the Group:

i 2005003 - A.H.M Towfique Mahmud

ii 2005004 - Md Zim Mim Siddiqee Sowdha

iii 2005006 - Kowshik Saha Kabya

iv 2005018 - Munzer Mahmood

v 2005019 - Jarin Tasneem

1 Introduction

The MIPS (Microprocessor without Interlocked Pipeline Stages) is a widely used architecture that embodies a RISC (Reduced Instruction Set Computing) approach, renowned for its simplicity, efficiency, and flexibility. In this report, we present the design and implementation of a 4-bit MIPS.

The design of our MIPS computer incorporates five components: PC(Program Counter), Instruction Memory, ALU(Arithmetic Logic Unit, and Data Memory. Each of these components performs their respective tasks and eventually, the whole instruction is processed.

We have used ATMega32 IC to implement the components of MIPS. This has streamlined the process and made the overall circuit more simple and easy to understand.

We have created a 4-bit computer that consists of a 4-bit data bus, a 8-bit address bus, and a 4-bit ALU(Arithmetic Logic Unit). Following the standard design principles, we have included separate instruction and data memory in our design.

2 Instruction Set

2.1 Instruction Set Description

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

Table 1: Instruction Set Description

2.2 Instruction Format

R-type	Opcode	Src Reg-1	Src Reg-2	Dst Reg
	4-bits	4-bits	4-bits	4-bits

S-type	Opcode	Src Reg-1	Dst Reg	Shamt
	4-bits	4-bits	4-bits	4-bits

I-type	Opcode	Src Reg-1	Src Reg-2/Dst Reg	Address/Immediate
	4-bits	4-bits	4-bits	4-bits

J-type	Opcode	Target Jump Address	0
	4-bits	8-bits	4-bits

2.3 Instruction Set Assignment

Serial Number	**Instruction ID**	**Opcode**
0	C	0000
1	L	0001
2	I	0010
3	J	0011
4	M	0100
5	H	0101
6	O	0110
7	E	0111
8	B	1000
9	K	1001
10	F	1010
11	G	1011
12	D	1100
13	N	1101
14	A	1110
15	P	1111

Table 2: Instruction Set Assignment

3 Block Diagram

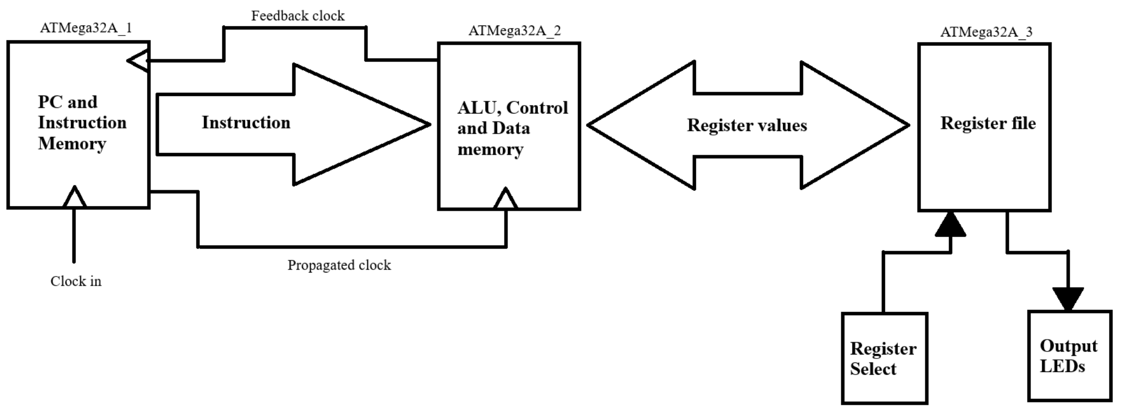

Figure 1: Block Diagram of different components of MIPS

4 Circuit Diagram

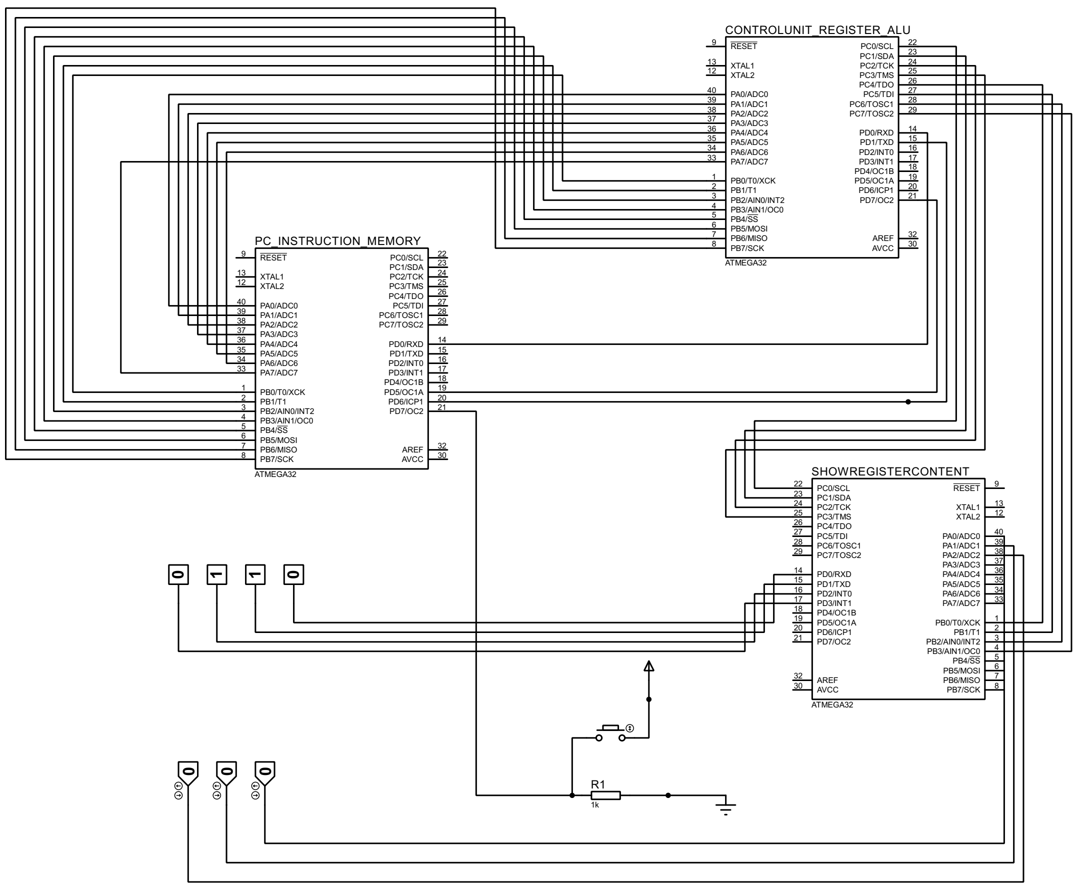

Figure 2: The Complete Diagram of MIPS

5 Description of the Components

5.1 Program Counter

The Program Counter is a counter that keeps track of the exact instruction that is being executed. After every instruction is executed, it is incremented by one pointing to the next instruction to be executed. But when a branching statement comes it is incremented or decremented by offset address based on the condition. Also, when a jump instruction comes, it is changed to instruction where the jump instruction tells it to jump. Every transition of the PC is controlled by an external clock source which is generated by the use of a pulse switch.

5.2 Instruction Memory

Instruction memory is essentially a pre-coded memory that holds all the instructions. All the instructions are burned into the ATMega32 that was dedicated for this purpose. As all of the given instructions are 16 bits, 2 ports of MCU are needed to pass every instruction. We used port A and port B for this purpose. In fact, our PC and instruction memory are merged into one ATMega32 for convenience. When a clock pulse is given to the PC, it increases by 1(except branching and jumping). We stored all the instructions in an array. After the clock pulse is given to the PC, it outputs the instruction stored in the array in the output ports of ATMega32.

5.3 ALU

The ALU unit has been merged with register memory and control unit according to our design choice. It executes the normal work as a normal ALU unit that calculates the arithmetic result of any two register operands. It is also used to calculate the jumping address for both conditional and unconditional jumps.

5.4 Register file

According to our design, we used a one-dimensional array to work as a register file in our second Atmega32 code. The type of the array is unsigned char as each of the register will be of size 8 bit. 4 bit data will be stored in the lower nibble of any register of our choice. The size of this array is 7 as we have a total of seven registers.

5.5 Control unit

This part is responsible for sending necessary control signals for handling different operations.Since we merged this component in the second Atmega32 module, the design became more compact and easy to handle as we can easily manipulate any operation in the corresponding code of Atmega. This made the design very simple and elegant.

6 Writing and Executing a Program in the Designed Machine

We have designed an **assembler** using **flex** and **bison** for converting provided MIPS instructions to executable machine codes. The assembler takes a set of instructions written according to the MIPS instruction set and returns the corresponding set of machine codes for our designed microprocessor. Each MIPS instruction is translated to a **16-bit** machine instruction following the instruction set and instruction format illustrated in the assignment declaration. The derived set of machine codes is introduced as a variable in the driver code for the Instruction Memory module (in our case, an **ATMega32**), and by burning the driver code into the microcontroller unit we store the instructions in the mentioned module.

7 Modules

7.1 Module-1

Here we are referring to each module as an ATmega32. This one is responsible for the merged actions of both program counter and instruction memory. Just like the register file, we are handling the instruction memory by using an array. Also, the input to this module is an 8-bit instruction memory address that we are getting from the second module. Based on that, we are fetching the instruction from the array and sending the 16-bit instruction using two ports,

7.2 Module-2

This part executes the merged actions of the control unit, ALU, and data memory. In this module, all the arithmetic and jump address calculations have been carried out. The result of any arithmetic is stored on the register file module. Any memory access is also carried out in the same way. Also, we are sending the feedback clock and zero flag signal from here to module 1.

7.3 Module-3

This is the register file content module that will show the values stored in any register element.

8 ICs used with their count

IC	Quantity
ATMega32	3
Total	3

9 Software tools used

Simulation Tool: Proteus 8 Professional

Chip Programming Tool: Atmel Studio 7.0, Microchip Studio

10 Discussion

1. We have implemented the circuit in Proteus for software simulation and we have used the built-in ATMega32 IC. Necessary codes for ATMega32 have been written in Atmel Studio 7.0 and Microchip Studio.

2. All of our ATMega32 ICs had been tested before integrating them into our circuit.

3. The circuit has been tested with multiple test cases to ensure that it satisfies the desired outcome.

4. To optimize the number of ICs, we have merged PC with Instruction Memory in a single ATMega32 and Control Unit, ALU, and Data Memory in another ATmega32 IC.

5. Communication between the PC-Instruction Memory module and the Control Unit-ALU-Data Memory unit has been ensured by implementing two generated clocks.

6. Proper delays have been implemented in necessary places to ensure synchronous and correct execution of instructions.

11 Contribution

i. **2005003**

- ATMega32 Driver codes for program counter and instruction memory
- Hardware implementation of program counter and instruction memory module
- Hardware implementation testing and debugging
- Preparation of respective section in the report

ii. **2005004**

- Code for Assembler and Test case generation
- ATMega32 Driver codes for register file
- Software implementation testing and debugging
- Preparation of respective section in the report

iii. **2005006**

- ATMega32 Driver codes for ALU
- Hardware implementation of ALU module
- Software implementation testing and debugging
- Preparation of respective section in the report

iv. **2005018**

- ATMega32 Driver codes for data memory
- Hardware implementation of data memory
- Hardware implementation testing and debugging
- Preparation of respective section in the report

v. **2005019**

- ATMega32 Driver codes for control unit
- Software circuit simulation design
- Hardware implementation testing and debugging
- Preparation of respective section in the report