CSE 306
Computer Architecture Sessional

Assignment-3: 4 bit MIPS Implementation

Section - A2
Group - 02

Members of the group

- 2005035 - Rafiul Islam Nijamy
- 2005036 - Tawhid Muhammad Mubashwir
- 2005037 - Zia Ul Hassan Abdullah
- 2005045 - Prithu Anan
- 2005055 - Ha Meem

Contents

List of Figures

1 Introduction

MIPS (Microprocessor without Interlocked Pipeline Stages) is a Reduced Instruction Set Computing (RISC) architecture that has been utilized extensively in the design of microprocessors and embedded systems because of its simplicity, effectiveness, and adaptability. In this report, we describe the architecture and setup of a 4-bit MIPS computer without pipelining.

Our design consists of 5 major components namely instruction memory, control unit, register file, ALU and data memory. We have implemented these components using ATMega32 micro-controller.

We have created an 4-bit computer that consists of a 4-bit data bus, an 8-bit address bus, and a 4-bit Arithmetic and Logic Unit (ALU). To conform to 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	Opcode
A	Arithmetic	R	add	0001
B	Arithmetic	I	addi	0011
C	Arithmetic	R	sub	1100
D	Arithmetic	I	subi	1011
E	Logic	R	and	1110
F	Logic	I	andi	1001
G	Logic	R	or	0010
H	Logic	I	ori	1101
I	Logic	S	sll	0101
J	Logic	S	srl	0111
K	Logic	R	nor	0000
L	Memory	I	sw	0100
M	Memory	I	lw	1010
N	Control-conditional	I	beq	1000
O	Control-conditional	I	bneq	1111
P	Control-unconditional	J	j	0110

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	Addr./Immdt.
4 bits	4 bits	4 bits	4 bits

- J-Type

Opcode	Target Jump Address	0
4 bits	8 bits	4 bits

3 Complete Block diagram of a 4-bit MIPS processor

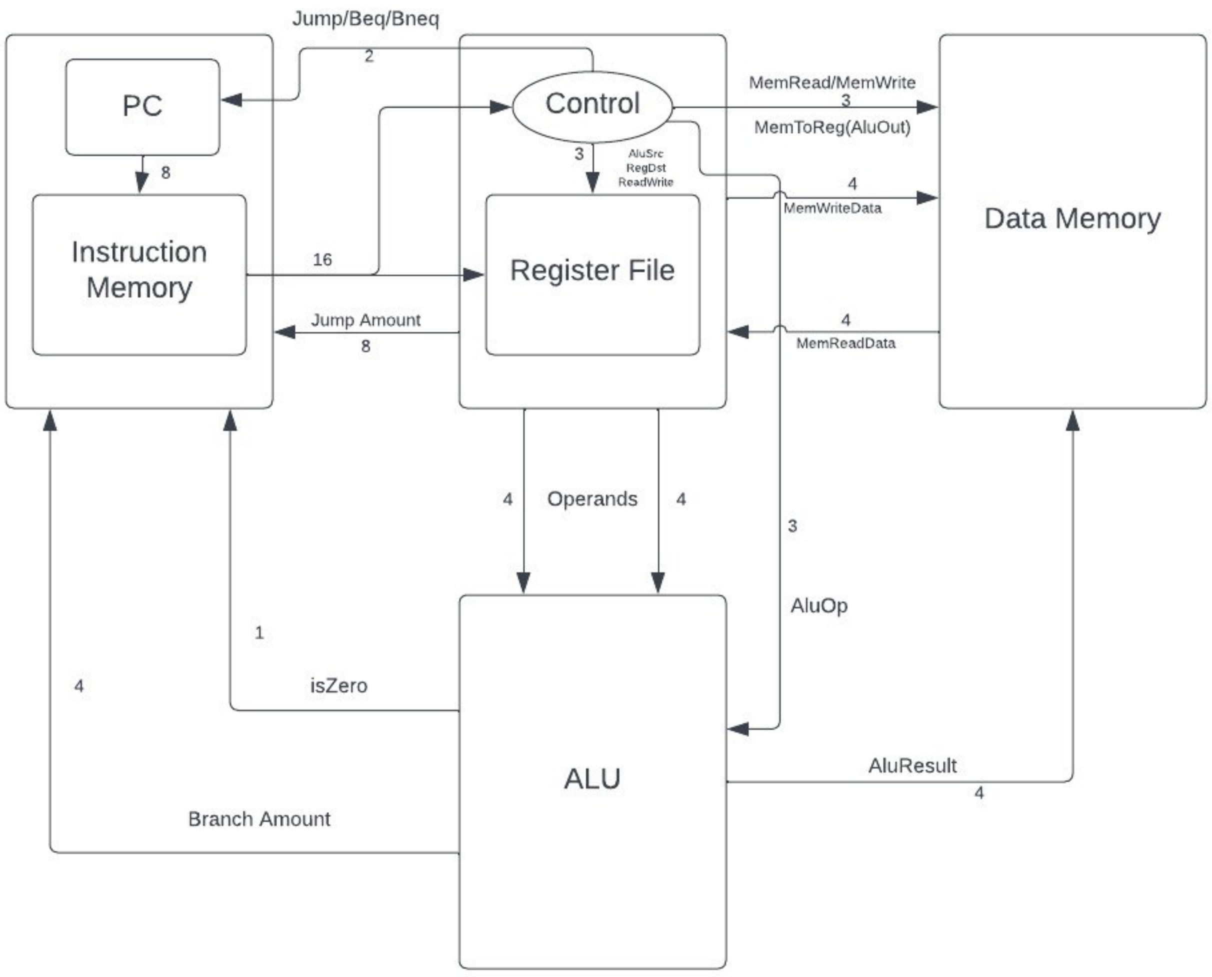

Figure 1: Block Diagram of MIPS implementation

4 Circuit diagram of a 4-bit MIPS processor

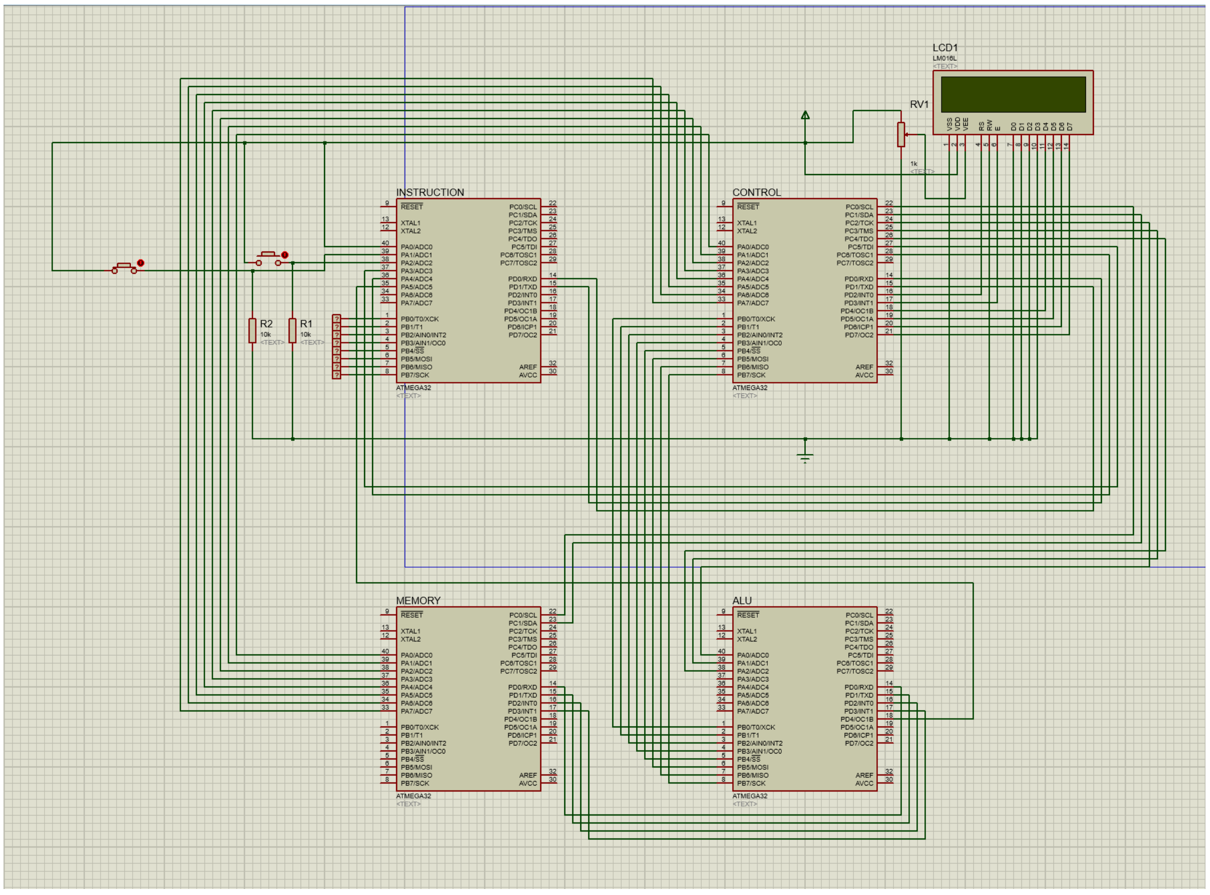

Figure 2: Circuit Diagram of MIPS implementation

5 Details of Implementation

5.1 Instruction

The instruction unit consists of a Program Counter (PC) and an 8-bit Instruction Memory. The program counter points at the beginning of the instruction memory and is incremented by 1 after each instruction unless it receives a jump or branch command from the control unit. In case of jump, the program counter receives 8-bits of jump address and in case of branch, the program counter receives 8 bits of sign extended branch distance. The ALU sends zero flag to the instruction unit to let it decide whether to branch or not.

5.2 Control and Register

This unit receives 16-bit instruction from the instruction unit via UART communication protocol and breaks them up into 4 nibbles. The control unit then decides what control bits are to be sent to the components of the MIPS on the basis of the first nibble which is the opcode. The next 3 nibbles can act as source register, destination register and/or immediate values depending on the instruction type. The control unit determines the type of the instruction through the opcode as well.

5.3 ALU

The ALU performs all sorts of 4-bit arithmetic and logical operations mentioned in the assignment specification. It takes 2 operands as input from the register file and and outputs the result to the data memory. The type of operation to be performed is determined by the incoming control bits from the control unit.

5.4 Data Memory

This unit acts as both a data memory and a stack. This is an essential part of the MIPS especially for lw and sw operations. Push ans pop operations also heavily rely on this unit. Data memory can directly share data with the register file. Whether the data memory will read, write data or send the ALU result to register file is also decided by control bits.

6 Approach to implement the push and pop instructions

Our approach to implementing the push and pop functionalities was to break each of them down to 2 separate instructions.

- Push
  1. Applying sw operation on the given register
  2. Decrementing the value of the stack pointer
- Pop
  1. Incrementing the value of the stack pointer
  2. Applying lw operation on the given register

Additionally we handled the case of push with argument [i.e. push 3,($t2)] by breaking it down to 3 instructions. The additional instruction involved saving the stored value in the stored address in an extra register before then pushing it to the stack.

7 ICs used with their count

IC	Quantity
ATMega32	4
Total	4

Table 2: ICs Used with Quantity

8 Simulator used Along with the Version Number

Proteus 8.9

9 Contribution

- Software:
  - Instruction: 2005037
  - Control and Register File: 2005036, 2005045
  - ALU: 2005055
  - Data Memory: 2005035
- Hardware: 2005035, 2005036, 2005037, 2005045, 2005055
- Report: 2005045, 2005055

10 Discussion

Our implementation of the MIPS architecture heavily relies on the functionalities of the ATMega32 microcontroller. While this may have reduced some level of difficulty from our hardware implementation, it posed a great a challenge from the software side of the task as we had to simulate the entire datapath of MIPS using primitive C code.

One of the noteworthy features of our implementation of MIPS involves serial communication between the the instruction memory and the control and register file unit via the UART protocol. This allowed us to send the entire 16 bit instruction in only two steps using only 2 pins of each microntroller which tremendously reduced the number of required pins for the control unit and allowed us to establish communication with the ALU and data memory unit.

In our efforts to further reduce the required number of ATMega32 pins, we used an LCD screen to display the state of the 6 registers which cost us only 6 pins, whereas we would have needed 24 pins had we used separate LED lights.

Our implementation features both manual mode and auto mode. The auto mode is there to simulate a real computer as it automatically performs all the instructions one by one. The manual mode is used to execute one instruction at a time using a push button which allows us to observe the states of all the registers after each instruction.

We spent a significant amount of time debugging our software, simulating in Proteus for various test cases and running our hardware implementation to ensure that no corners were cut and that both software and hardware are in perfect sync.

Overall, this assignment provided us with the opportunity to dive deeper into the design principles of MIPS architecture and helped us to achieve a better understanding of the instruction cycle. We tried our utmost best to ensure the novelty of our implementation.