CSE306 - Computer Architecture Sessional
Assignment 3

# 8-bit MIPS Design and Simulation


**Submitted by:**

1805038 – Iftekhar E Mahbub Zeeon
1805049 - Somonidro Roy
1805050 - Salman Khondoker
1805053 - HAZ. Sameen Shahgir
1805055 - Shamit Fatin


17 August 2022

# 1. Introduction

Microprocessor without Interlocked Pipe-lined Stages (MIPS) is a Reduced Instruction Set Computer (RISC) Instruction Set Architecture (ISA). In this assignment, we implement a Processor both in software and hardware that operates on a simplified ISA based on MIPS with the 16-bit instructions of the following formats.

- 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 |

Some of the key specifications of this project are:

- 8-bit Data bus
- 16-bit Address bus
- 4-bit ALU
- Support for 16 operations in hardware
- Can simulate stack operations through lw and sw.

# 2. Instruction Set:

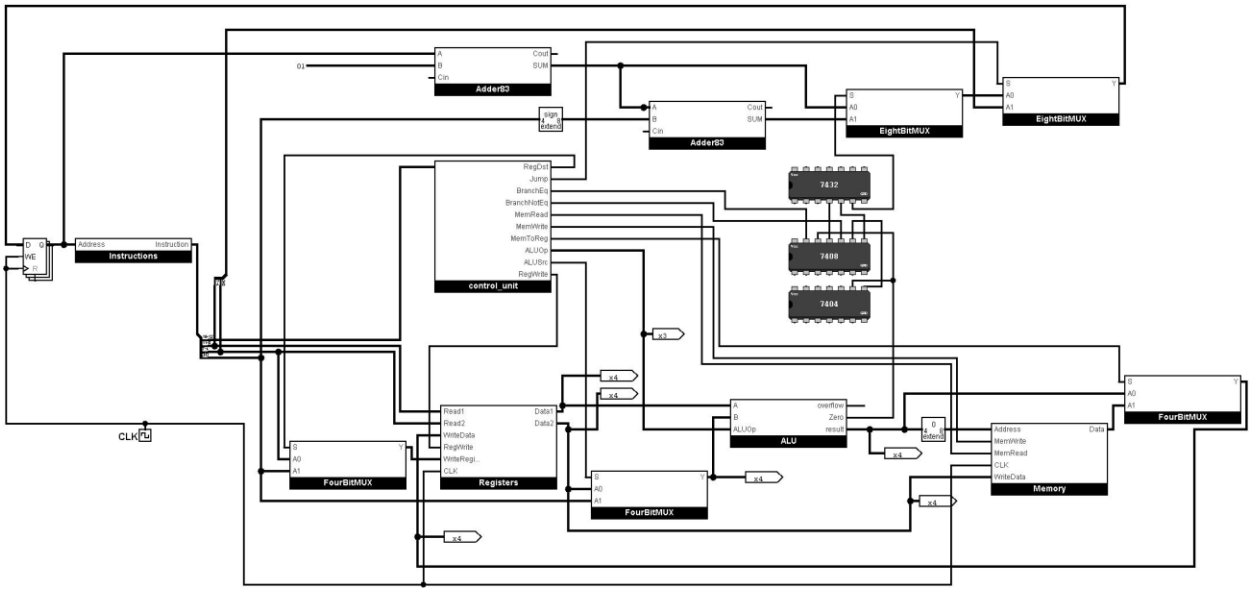
Section A2 Group 3 Instruction Set Assignment: BDEFPIOLHCNJKAGM

## Control Word

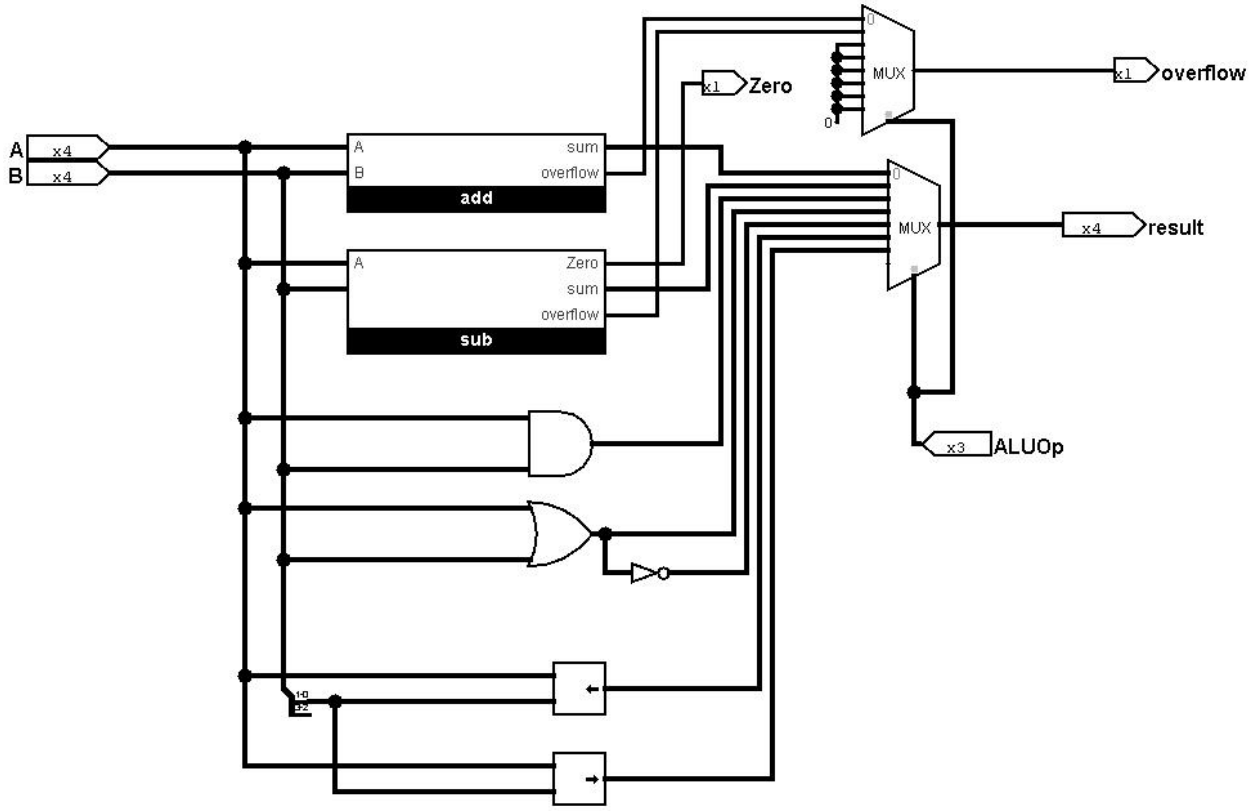
| 11 | 10 | 9 | 8 | 6 | 7 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Reg Dst | Jump | BEQ | BNEQ | MemRead | MemWrite | Mem-toReg | ALU Op(2) | ALU Op(1) | ALU Op(0) | ALU-Src | Reg-Write |

| ID | Name | Type | Opcode | Control Word (Bin) | Control Word (Hex) |
|---|---|---|---|---|---|
| B | addi | I | 0000 | 0000 0000 0011 | 003 |
| D | subi | I | 0001 | 0000 0000 0111 | 007 |
| E | and | R | 0010 | 1000 0000 1001 | 809 |
| F | andi | I | 0011 | 0000 0000 1011 | 00b |
| P | j | J | 0100 | 0100 0000 0000 | 400 |
| I | sll | R | 0101 | 0000 0001 0111 | 017 |
| O | bneq | I | 0110 | 0001 0000 0100 | 104 |
| L | lw | S | 0111 | 0000 1100 0011 | 0c3 |
| H | ori | I | 1000 | 0000 0000 1111 | 00f |
| C | sub | R | 1001 | 1000 0000 0101 | 805 |
| N | beq | I | 1010 | 0010 0000 0100 | 204 |
| J | srl | R | 1011 | 0000 0001 1011 | 01b |
| K | nor | R | 1100 | 1000 0001 0001 | 811 |
| A | add | R | 1101 | 1000 0000 0001 | 801 |
| G | or | R | 1110 | 1000 0000 1101 | 80d |
| M | sw | S | 1111 | 0000 0010 0010 | 022 |

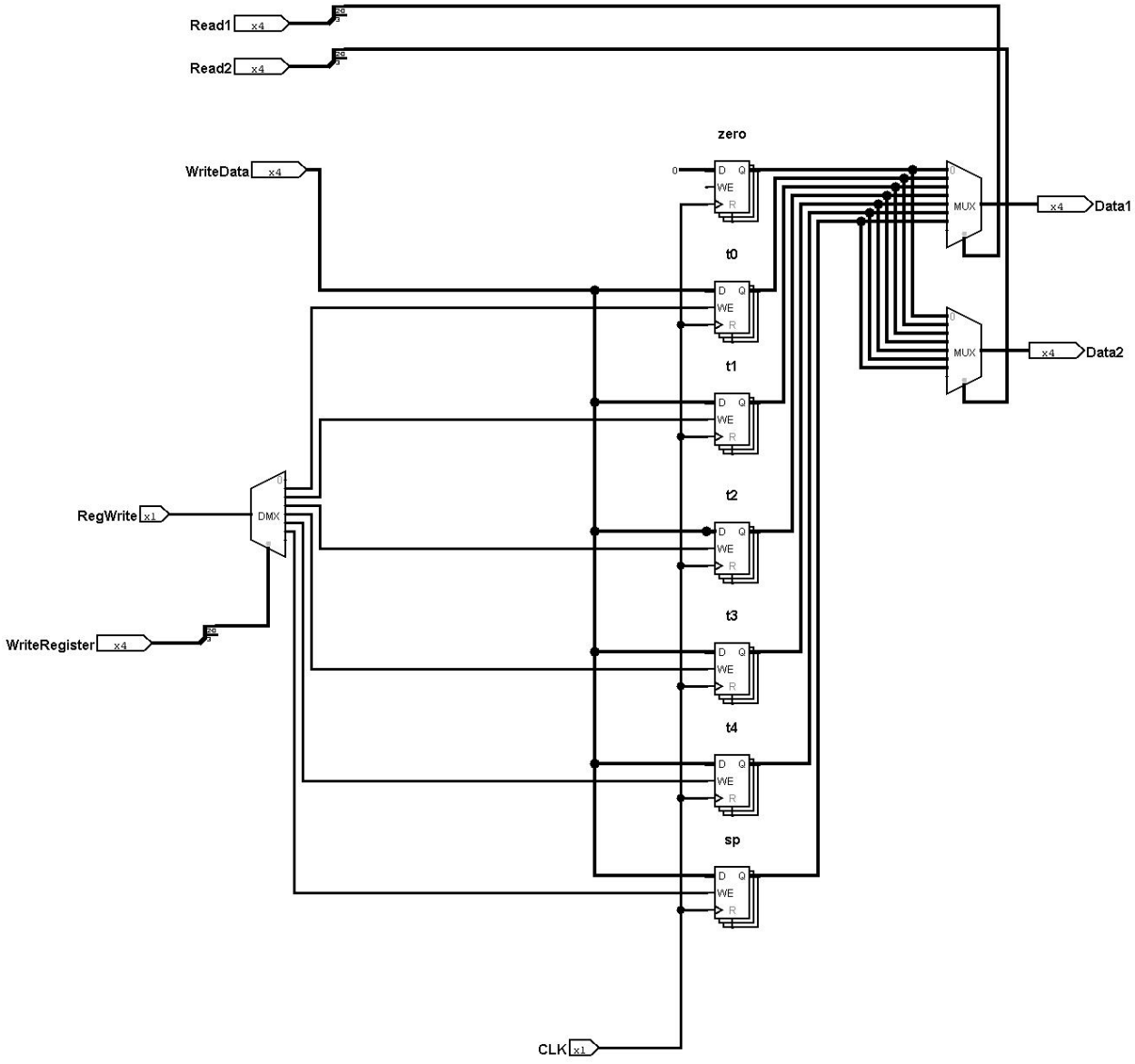
# 3. Circuit Diagram



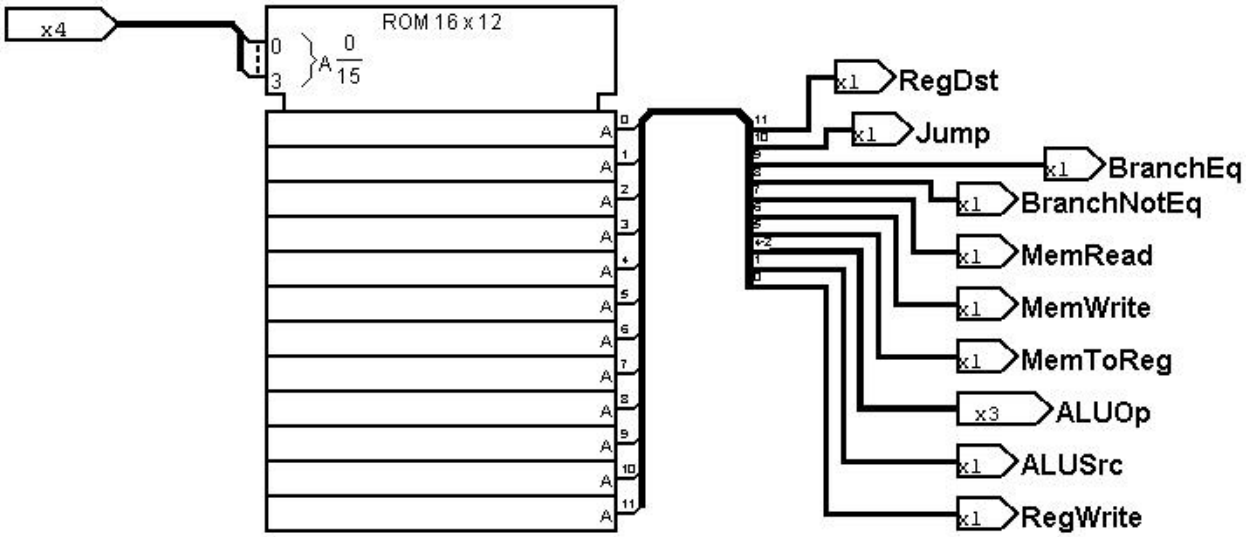

**ALU:**



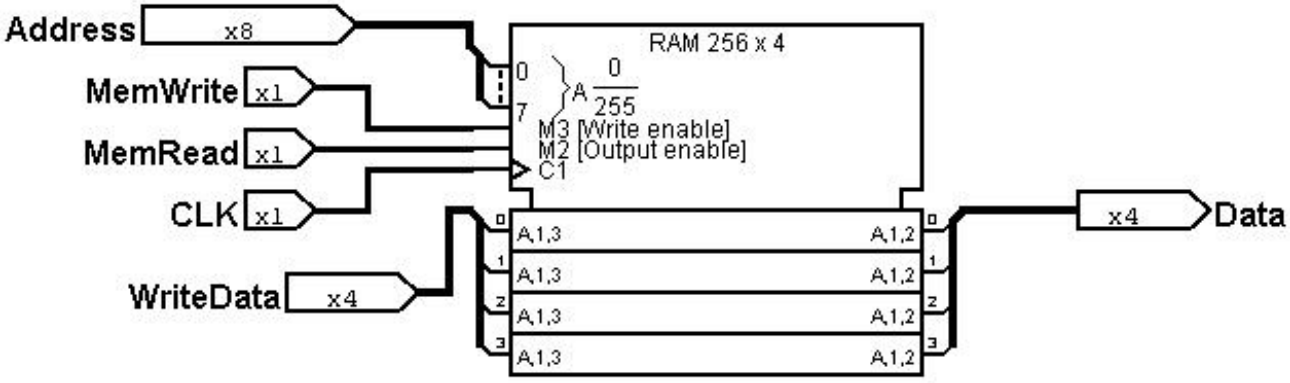
## Register File:



**Control Unit:**




**Memory File:**



# 4. Program Creation and Execution

- Write the assembly code in a [file_name].txt file
- Compile mips2hex.cpp: **`g++ -std=c++20 mips2hex.cpp`**
- Run the command: **`./a.exe [file_name]`**
- A hex.txt file should automatically be created in the current directory.
- Flash the hex code in the instructions.txt into the Instruction EEPROM
- The processor is now ready to execute.
- Step through the instructions one at a time using the Clock Pulse button.
- Instruction, Control, ALU, Register and Memory contents are displayed with LEDS

# 5. Special Features

The microprocessor supports stack operations using a dedicated stack pointer $sp. Since the 4-bit ALU can address 16 memory addresses, we start inserting into stack from address 1111 (15) and continue backwards. Each stack operation is replaced by two instructions by the assembler. For example:

| | | | |
|---|---|---|---|
| push $t1 | = | sw $t1, 0($sp) | subi $sp, $sp, 1 |
| pop $t1 | = | lw $t1, 1($sp) | addi $sp, $sp, 1 |

# 6. IC User with Count

| IC | Type | Count |
|---|---|---|
| ATMEGA32 | ROM/RAM/ALU emulation | 4 |
| 74157 | Quad 2x1 MUX | 7 |
| 7483 | 4 Bit Adder | 5 |
| 7432 | Quad OR Gate | 1 |
| 7404 | Quad NOT Gate | 4 |
| 7408 | Quad AND Gate | 1 |
| 74138 | 3Bit Decoder | 1 |
| 74151 | 3to8 MUX | 8 |
| 7474 | D Flip Flop | 12 |

# 7. Discussion:

- We noticed that to achieve 1 CPI for all instructions, the memory needs to be read at the rising edge of the clock pulse and the registers need to be written to at the falling edge.

- As per problem specification, the address bus is 8 bits long. A total of 256 instructions can be loaded at once and the Memory File has 256×4 = 1024 bits total capacity. However, since the ALU output is 4 bits, which is zero extended to 8 bits, only the first 16 addresses can be accessed.

- For *serial left shift* [sll] and *serial right shift* [srl] operations, if the shift amount exceeds 3, we have taken to be *modulo 4*. For example, **sll $t0, $t1, 7** is executed as **sll $t0, $t1, 3** since 7%4=3. This was done because in shifter IC chips, the shift amount cannot exceed the operand bit size.

- For branching operations [beq] and [bneq], the offset is sign extended. This allows branching 8 instructions backwards and 7 instructions forward.

- Due to the market scarcity of 4-bit ALU **IC 74181** chip, it had to be simulated using an **ATMEGA32**. The **ATMEGA32** also performs the shift operations which are not natively supported in **IC 74181**.

- Due to the market scarcity of parallel **EEPROM**, the Control Unit, Instruction File and Memory File had to be simulated using **ATMEGA32**. The clock cycle of the microcontroller is 1MHz and hence fast enough that operations appear instantaneous as in the combinatorial circuits it replaces.

- The two additional Adders used for PC increment and branching were implemented using only **IC 7483.**

- The overall design philosophy was to create the processor in such a way that the **ATMEGA32** could be easily replaced with their corresponding IC chips, hence staying true to the intention of this assignment: *building a processor from the ground up.*