# **CSE 306**

# **Computer Architecture Sessional**

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

**Submitted By:**

**Group No:** 01

**Sub-Section:** B2

**Group Members:**

1. Sk Ruhul Azgor (ID: 1805091)
2. Nazmul Islam Ananta (ID: 1805093)
3. Yasir Khandakar (ID: 1805105)
4. Sk Sabit Bin Mosaddeq (ID: 1805106)
5. Redwanul Karim (ID: 1805111)

**Level:** 3 **Term:** 1

Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology (BUET)

**Date of submission:** 17 August 2022

**Problem Specification:**

In this assignment, we had to design, simulate (in S/W), and implement (in H/W) a modified and reduced version of the MIPS instruction set

**Introduction:**

MIPS (Microprocessor without Interlocked Pipelined Stages) is a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Computer Systems. Each instruction takes 1 clock cycle to be executed so clock cycle is the time to execute the longest instruction in the MIPS instruction set. The main components of the processor are as follows: instruction memory, data memory, register file, ALU and a control unit.

We will construct the data path and control unit of MIPS Instruction set:

■ The memory-reference instructions load word (lw) and store word (sw).

■ The arithmetic-logical instructions add, sub, addi, subi, AND, OR, NOR, ANDI, ORI, sll and srl.

■ The instructions branch equal (beq), branch not equal (bneq) and jump (j)

**Instruction Set:**

Our MIPS instruction is 16 bits long and we have 4 formats.

* **R Type**

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Src Reg1 | Src Reg2 | Dst Reg |

4 bits 4 bits 4 bits 4 bits

* **I Type**

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Src Reg1 | Src Reg2/Dest Reg | Address / Immediate |

4 bits 4 bits 4 bits 4 bits

* **J Type**

|  |  |  |
| --- | --- | --- |
| Opcode | Target Jump Address | 0 |

4 bits 8 bits 4 bits

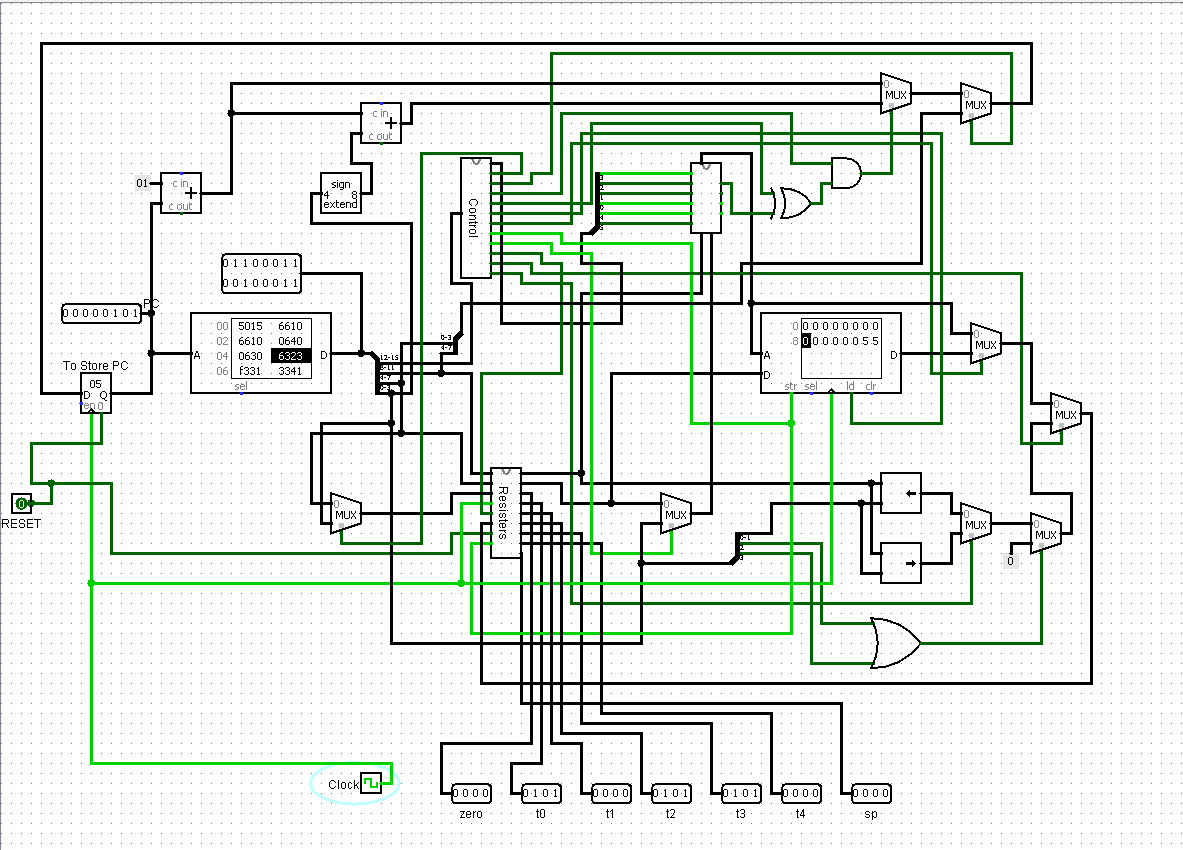
**S Type**

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Src Reg1 | Dest Reg | Shamt |

Control ROM Map as value of 4 bits consists of low bit to high--> ALU OP Code (6), RegDst(1),Jump(1),Branch(1),BNE(1),MemRead(1),MemToReg(1),MemWrite(1),AluSrc(1),RegWrite(1),ShiftControl(1), ShiftLeftOrRight(1). So, in total 17 bits.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Instruction ID | MIPS Opcode(binary) | Control ROM Map  (hex) | Operation | Type | Category |
| L | 0000 | 6c19 | lw | I | Memory |
| O | 0001 | 306 | bneq | I | Control - conditional |
| D | 0010 | 6006 | subi | I | Arithmetic |
| N | 0011 | 106 | beq | I | Control - conditional |
| P | 0100 | 80 | j | J | Control - unconditional |
| B | 0101 | 6019 | addi | I | Arithmetic |
| M | 0110 | 3019 | sw | I | Memory |
| G | 0111 | 406e | or | R | Logic |
| A | 1000 | 4059 | add | R | Arithmetic |
| E | 1001 | 406b | and | R | Logic |
| H | 1010 | 602e | ori | I | Logic |
| K | 1011 | 4061 | nor | R | Logic |
| F | 1100 | 602b | andi | I | Logic |
| C | 1101 | 4046 | sub | R | Arithmetic |
| I | 1110 | c000 | sll | S | Shift |
| J | 1111 | 1c000 | srl | S | Shift |

**Block Diagram:**

****

**Figure 1: Block Diagram**

The above figure is MIPS Processor which works as follow: passing opcode from the instruction memory to the control unit and data need to read from resister. Then the ALU is operated as instructed, several control unit control data path. Then store / write data to the specific register/ memory.

**Diagram

Description automatically generated**

**Figure 2: Instruction Memory with PC**

When the clock is high, MIPS instructions are loaded from Instruction Memory according to the pc value and pass the instructions to the control unit and register file.

**Diagram, schematic

Description automatically generated**

**Figure 3: Register File**

Register files consist of all the registers as well as sp resister. From this file, we can read and update the resister file according to the instruction. Data can be written at rising edge.

**Diagram

Description automatically generated**

**Figure 4: Data Memory**

Data memory stores the data from the start and holds the stack from last.

**Diagram

Description automatically generated**

**Figure 5: Control Unit**

Control unit controls the ALU opcode and data path according to the instruction.

**How to write and execute a program in this machine:**

We write a program which takes the MIPS code and convert it to a file which contains the hexadecimal value of the instruction machine code. Then we need to upload it to the instruction ROM. For simulation at Logisim, we will upload the file at ROM. The program will start as soon as we give clock to the processor. By the time the report is written, we didn’t implement the hardware part, so can’t tell you exactly what need to be done.\

The file looks like:

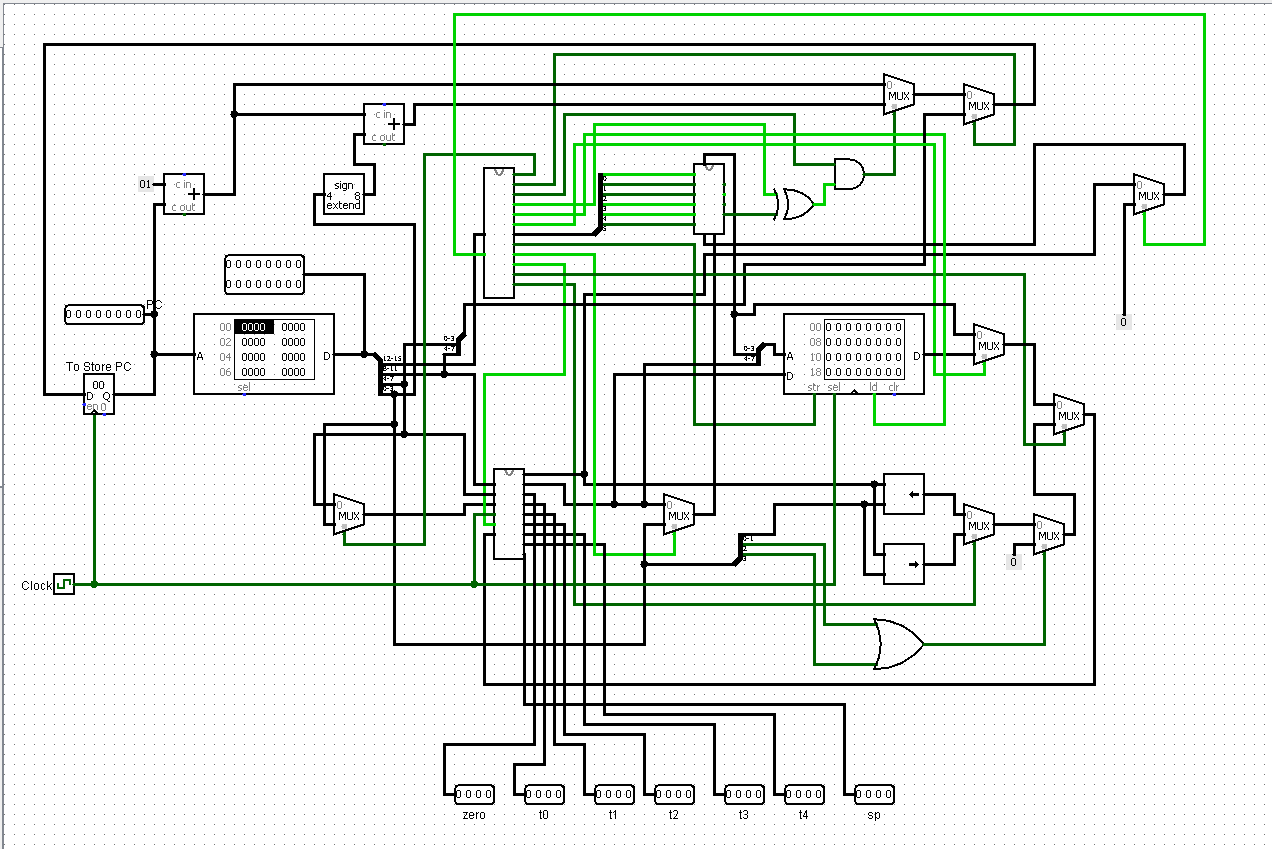
Graphical user interface, text, application, chat or text message

Description automatically generated

**Special Features:**

**Segment offset:**

As in this assignment, we implement 4-bit processor. So, the address bit restricted to on 4 bits. Which means, we only can use 16B data memory. It doesn’t sound great. So, we try to increase memory by dividing it with segments which is a design of intel 8088 type processor. But as it doesn’t match with MIPS architecture, so we didn’t final the design. With this segment offset, we can add 256B data memory.



**Figure: Design to use 8-bit memory**

**Implementation of stack:**

We take a register sp (stack pointer) to store the position of stack-pointer from the topmost address(0xF). We implement stack at one instruction for push and pop. As it is a lengthy process which cause longer clock cycle.

addi $sp, zero,15

Every type of push and pop operations has been converted to a memory operation.

1. When a number is pushed, we convert it to a store word instruction which offset is 0. Internally it will decrease the stack pointer by one something like how pc is increased automatically. Also, sp will decrease first so for the first push it will become 0xf from 0x0. For example,

push $t0 will converted to sw $t0,0($sp)

1. In case of popping, the number stored in the address is loaded and the value of stack-pointer is increased by 1.

pop $t0 will converted to lw $t0,0($sp)

Also, sp will be used only for memory operation. We won’t allow any kind of resister operation….

**IC used with count:**

|  |  |
| --- | --- |
| IC | Count |
| 7408 (AND) | 4 |
| 7404 (NOT) | 1 |
| 7432 (OR) | 1 |
| 7480 (FULL ADDER) | 7 |
| 8 Bit Shifter | 2 |
| 74151 (8 input multiplexer) | 3 |
| 4 x 8 ROM | 1 |
| 8 x 20 ROM | 1 |
| 8 x 8 RAM | 1 |
| 74157 (Quad 2-to-1 multiplexer) | 9 |
| 74154 (4X16 decoder) | 1 |
| 74173 (4 bit D type register) | 18 |

**Discussion:**

In this assignment, we designed an 8-bit processor that implements the MIPS instruction set. For this purpose, we used basic gates (AND, OR, NOT, NOR), universal gates (XOR, XNOR), some other necessary gates (MUX, Shifter, Bit Finder, Adder, Subtractor, Bit Extender), RAM, ROM and register. The Processor takes a 20-bit binary number (Instruction) and the circuit reads/stores the register values and memory values. We used the known technique of designing the processor for MIPS instruction set. Here all types of instructions are executed in a single clock pulse other than pop and push commands. Moreover, Control ROM made our mapping quite easier, otherwise it would have been tedious by introducing combinational logic or elsehow. We designed an assembler in CPP and loaded the produced code in another ROM (Instruction Memory). We could not use the single memory (i.e., using Data Memory as Instruction Memory) because we cannot include 2 different addresses (or 2 different data read at a time) in a single memory unit while working in a single clock cycle method.

We followed the provided circuit diagram and implemented the circuit. While designing, we put emphasis on simplifying the circuit. We asserted our design by testing various inputs and matching the corresponding outputs. However, testing the outputs, we successfully finished our simulation.