**Department of Electrical and Computer Engineering**
**North South University**


# Computer Organization & Design

# CSE332

# PROJECT REPORT

**Name: Abrar Raiyan**

**ID: 1721710042**

**Section: 03**

**Faculty:**

**MS. TANJILA FARAH**

**ECE Department**

# Introduction:

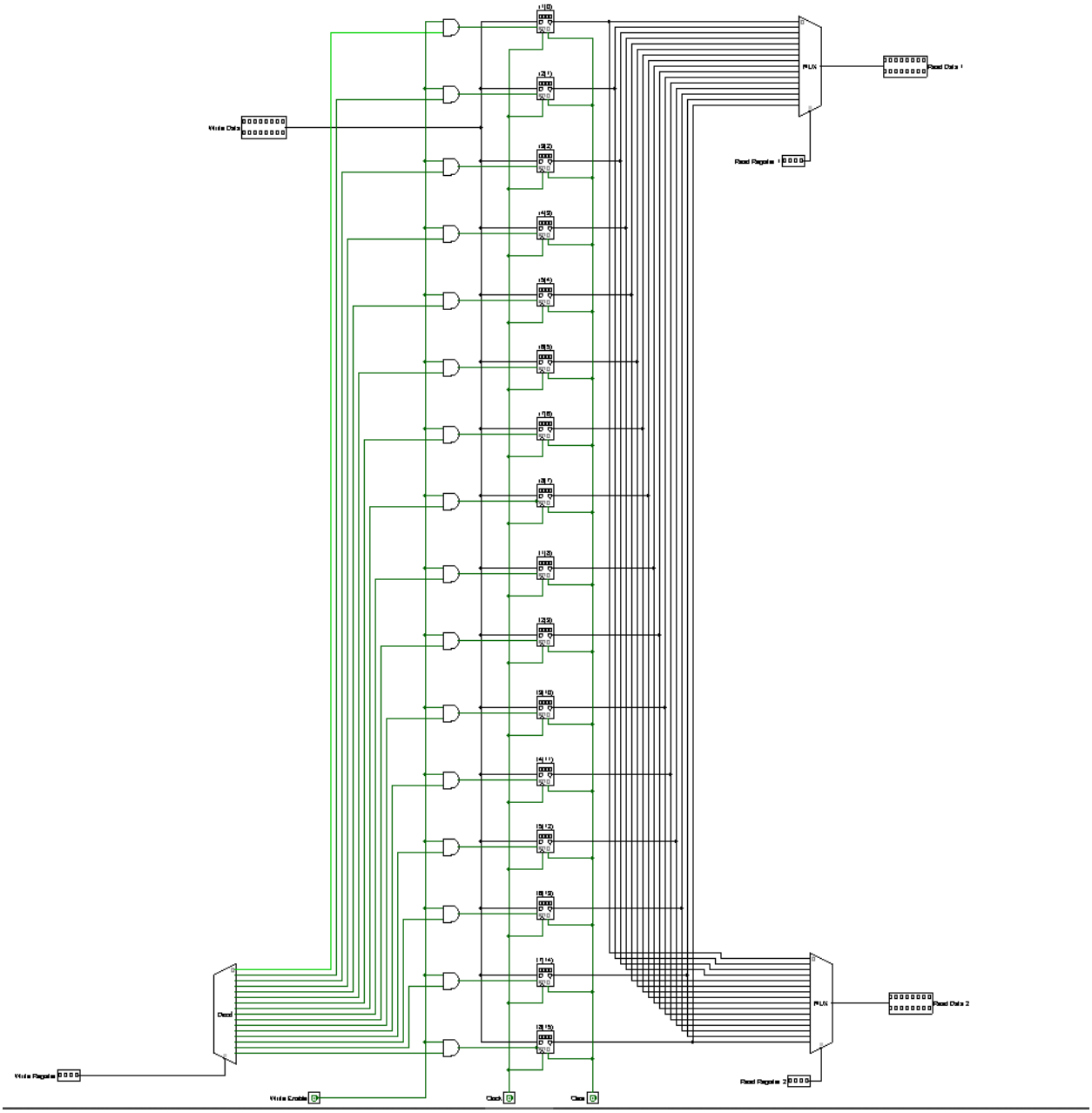
ISA decides how to represent an instruction of a processor in Instruction memory, how the instruction is oriented, what is the length of an instruction, in which way the instruction should be decoded so that the CPU gets appropriate data. We already have our ISA. The objective is to design our processor based on the ISA. In this section, we have designed a 16-bit ALU, 16-bit Register File & a Control Unit. Then connected all the component with Program counter, Instruction Memory, Data Memory etc. so that our processor works. Our Goal is to execute the 10 instructions described in ISA successfully in our processor.

# Components:

## Register File:

In processor, ALU gets data from a Register Bank / Register File. As we have designed our ISA, our register file should have 16 registers. And the design should be easy enough to choose rs, rt, rd values. The circuit below is our register file design. Note: each register is able to store 16-bit data.

## Register File Design:

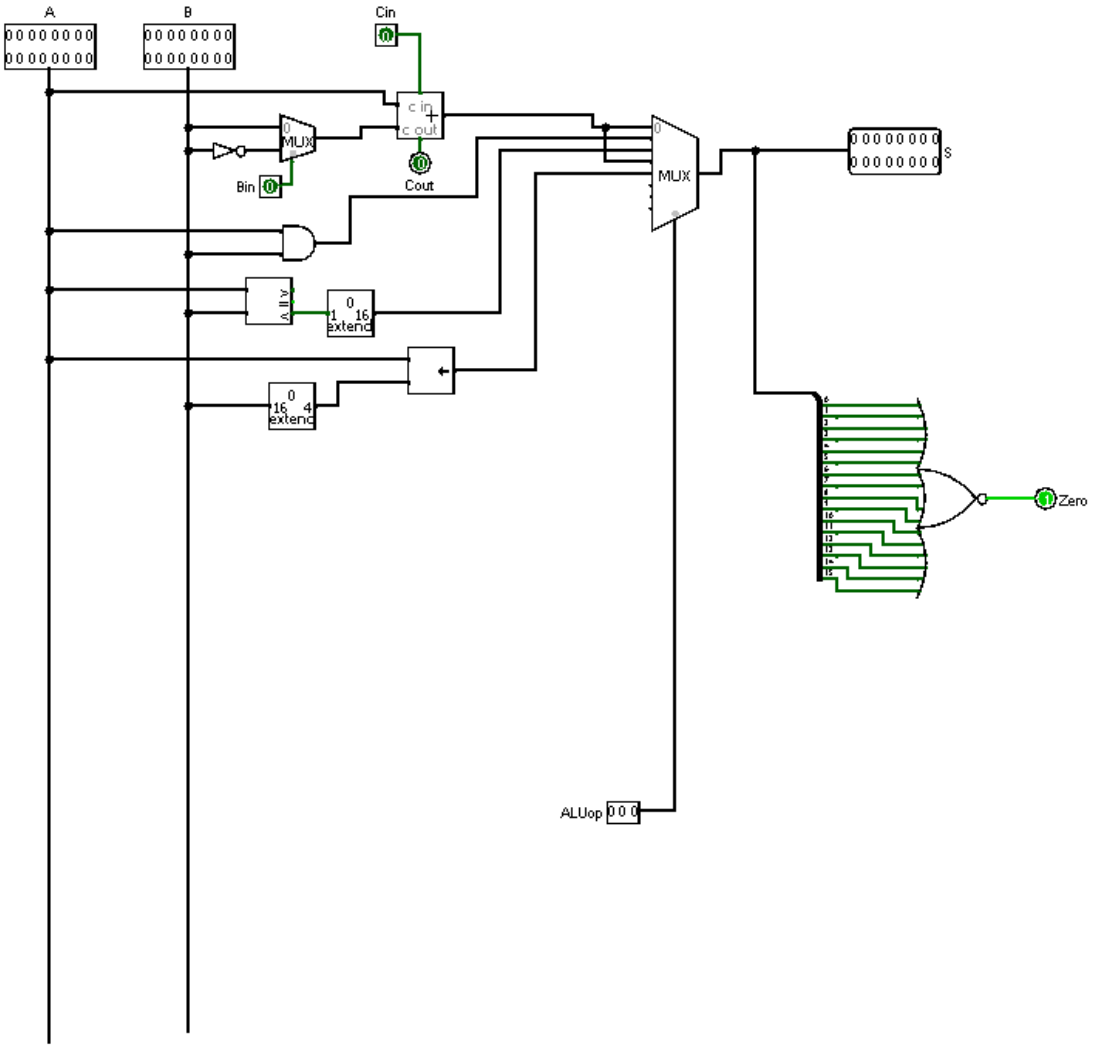


## Arithmetic & Logic Unit (ALU):

ALU stands for “ARITHMETIC AND LOGIC UNIT” which does a bunch of arithmetic and logical operations. It is a very important hardware component of central processing unit of a computer. ALU is the main hardware part where the operations get executed. For our CPU, we will design an ALU which will be able to do Arithmetic Op – (Addition, Subtraction), Logical

Op– (and, or, nor, xor). To do add operation, we used carry look ahead adder. So that our adder won't have any propagation delay. As our ALU will do operation with 16-bit data, logically we should build a 16-bit carry look ahead adder.

**16-bit ALU Design:**




---

**Program Counter:**

A program counter (PC) is a CPU register in the computer processor which has the address of the next instruction to be executed from memory. It is a digital counter needed for faster execution of tasks as well as for tracking the current execution point.

A program counter is also known as an instruction counter, instruction pointer, instruction address register or sequence control register.

All instructions as well as data in memory have a specific address. As each instruction is processed, the software application responsible updates the program counter with the upcoming instructions' address which needs to be fetched. The program counter in turn passes this information to the memory address register as part of the execution cycle/standard fetch. The program counter increases the stored value by one as the next instruction is fetched. If the computer is reset or restarts, the program counter usually reverts to the value of zero.

**Ram:**

RAM (Random Access Memory) is a part of computer's Main Memory which is directly accessible by CPU. RAM is used to Read and Write data into it which is accessed by CPU randomly. RAM is volatile in nature, it means if the power goes off, the stored information is lost. RAM is used to store the data that is currently processed by the CPU. Most of the programs and data that are modifiable are stored in RAM.

Integrated RAM chips are available in two form:

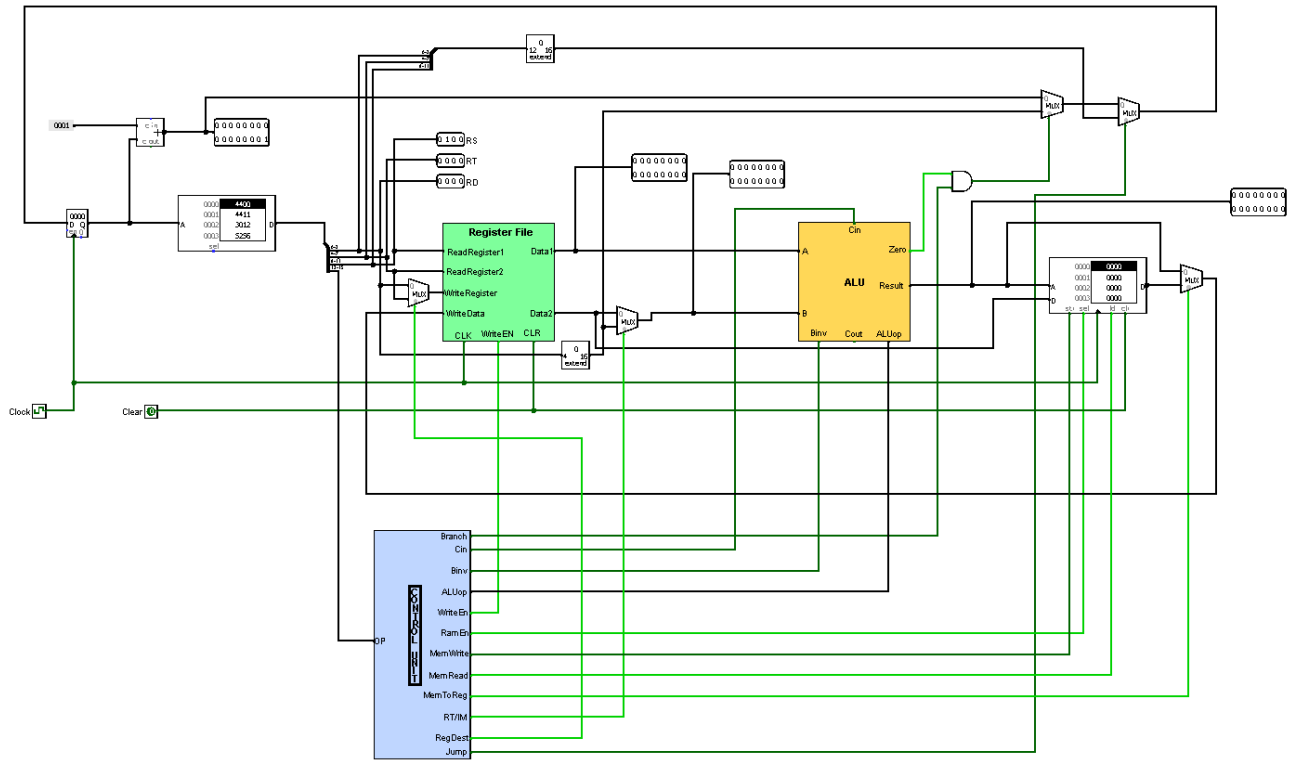
1. SRAM (Static RAM)
2. DRAM (Dynamic RAM)

**Rom:**

A ROM (read-only memory) is a memory unit that performs the read operation only; it does not have a write capability. This implies that the binary information stored in a ROM is made permanent during the hardware production of the unit and cannot be altered by writing different words into it.

Whereas a RAM is a general-purpose device whose contents can be altered during the computational process, a ROM is restricted to reading words that are permanently stored within the unit. The binary information to be stored, specified by the designer, is then embedded in the unit to form the required interconnection pattern. ROMs come with special internal electronic fuses that can be programmed for a specific configuration. Once the pattern is established, it stays within the unit even when power is turned off and on again.

**DATAPATH:**




# Control Unit:

Control unit is the circuit in processor which controls the flow of data and operation of ALU. Initially an instruction is loaded in instruction memory. According to Opcode of that instruction, control unit gets understood what the instruction wants to execute. Then it flows the data in such a path so that ALU gets necessary data to execute.

**Control unit design:** As our designed ISA, our CPU should be able to do 10 operations. The table below represents a demo of 10 operations, binary form of the operations following our ISA design & hex form of the operation.

**Instructions Demo:**

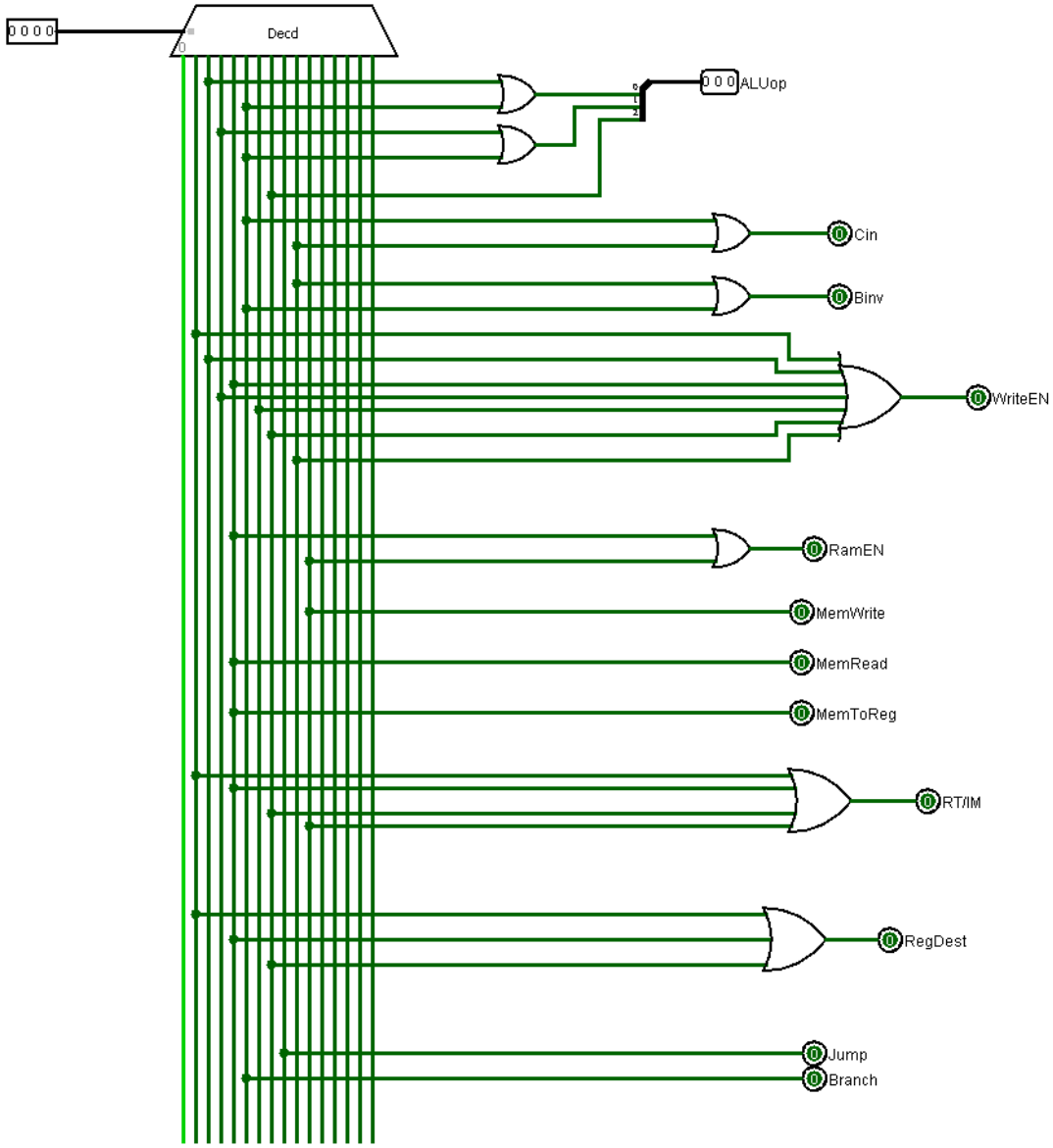
| Assembly CODE | Binary Value | Hex Value |
|---|---|---|
| addi $r1 $r2 6 | 0001 0000 0001 0110 | 1016 |
| and $r1 $r2 $r3 | 0010 0000 0001 0010 | 2012 |
| slt $r1 $r2 $r3 | 0011 0000 0001 0010 | 3012 |
| lw $r1 $r2 7 | 0100 0000 0001 0111 | 4017 |
| beq $r3 $r4 5 | 0101 0010 0011 0101 | 5235 |
| add $t1 $t2 $t3 | 0110 1000 1001 1010 | 689A |

| | | |
|---|---|---|
| sll $t1 $t2 4 | 0111 1000 1001 0100 | 7894 |
| jmp 4 | 1000 0000 0000 0100 | 8004 |
| sub $t1 $t2 $t3 | 1001 1000 1001 1010 | 989A |
| sw $t3 $t2 0 | 1010 1010 1001 0000 | AA90 |

**Control Table:**

| **op** | **Instructions** | **ALUop** | **Cin** | **Binv** | **WriteEN** | **RamEN** | **MemWrite** | **MemRead** | **MemTOReg** | **RT/IM** | **RegDest** | **Jump** | **Branch** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | nop | xxx | x | x | x | x | x | x | x | x | x | x | x |
| 0001 | addi | 000 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0010 | and | 001 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0011 | slt | 010 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0100 | lw | 000 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0101 | beq | 011 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0110 | add | 000 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0111 | sll | 100 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1000 | jmp | xxx | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1001 | sub | 000 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1010 | sw | 000 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |

## Control Unit Circuit:



# Assembler documentation:

Our task was to design an assembler which will convert the assembly code to machine language.

Our main goal was to generate a machine code from a file containing assembly language. The assembler reads a program written in an assembly language, then translate it into binary code and generates output file containing machine code.

In the input file the user has to give some instructions to convert into machine codes. The system will convert valid MIPS instructions into machine language and generate those codes into output file.

The input file is located in a folder named “inputs”. User will write down the MIPS code in this file.

## Register List

We have selected registers from $r1-$r8 and $t1-$t8 for general purpose. We assigned 4 bits for each of the register as we know in the instruction field in our ISA containing the register rs, rt and rd contains 4 bits each.

| Conventional Name | Register Number | Binary Value |
|---|---|---|
| $r1 | 0 | 0000 |
| $r2 | 1 | 0001 |
| $r3 | 2 | 0010 |
| $r4 | 3 | 0011 |
| $r5 | 4 | 0100 |
| $r6 | 5 | 0101 |
| $r7 | 6 | 0110 |
| $r8 | 7 | 0111 |
| $t1 | 8 | 1000 |
| $t2 | 9 | 1001 |
| $t3 | 10 | 1010 |
| $t4 | 11 | 1011 |
| $t5 | 12 | 1100 |
| $t6 | 13 | 1101 |
| $t7 | 14 | 1110 |

| | | |
|---|---|---|
| $t8 | 15 | 1111 |

**Op-Code List:** We have selected following op codes and assigned op-code binary values (4 bits) for each of the op codes.

| *Op-Code name* | *Type* | *Op-Code binary* |
|---|---|---|
| nop | | 0000 |
| addi | I-type | 0001 |
| and | R-type | 0010 |
| slt | R-type | 0011 |
| lw | I-type | 0100 |
| beq | I-type | 0101 |
| and | R-type | 0110 |
| sll | I-type | 0111 |
| jmp | J-type | 1000 |
| sub | R-type | 1001 |
| sw | I-type | 1010 |

***Instruction Description***

**add:** It adds two registers and stores the result in destination register.

- Operation: $r3 = $r1 + $r2
- Syntax: add $r1 $r2 $r3

**sub:** It subtracts two registers and stores the result in destination register.

- Operation: $r3 = $r1 - $r3
- Syntax: sub $r1 $r2 $r3

**addi:** It adds a value from register with an integer value and stores the result in destination register.

- Operation: $r2 = $r1 + offset
- Syntax: addi $r1 $r2 offset

**sll:** It shifts bits to the left and fill the empty bits with zeros. The shift amount is depended on the offset value.

- Operation: $r2= $r1 << offset
- Syntax**:** sll $r1 $r2 offset

**and:** It AND's two register values and stores the result in destination register. Basically, it sets some bits to 0.

- Operation: $r3 = $r1 && $r2
- Syntax: and $r1 $r2 $r3

**lw:** It loads required value from the memory and write it back into the register.

- Operation: $r2 = MEM[$r1 + offset]
- Syntax: lw $r1 $r2 offset

**sw:** It stores specific value from register to memory.

- Operation: MEM[$r1 + offset] = $r2
- Syntax: sw $r1 r2 offset

**beq:** It checks whether the values of two register s are same or not. If it's same it performs the operation located in the address at offset value.

- Operation: if ($r1==$r2) jump to offset

else goto next line

- Syntax: beq $r1 $r2 offset

**slt:** If $r1 is less than $r2, $r3 is set to one. It gets zero otherwise.

- Operation: if $r1< $r2

$r3 = 1

else $r3 = 0

- Syntax: slt $r1 $r2 $r3

## Conclusion:

The processor is able to successfully execute the 10 operations from ISA design. To execute any instruction, the syntax and the orientation of different type of instructions explained in ISA must be followed. Otherwise, the CPU won't work properly.