North South University

Department of Electrical and Computer Engineering (ECE)

CSE 332
Project documentation

Project Title: Designing a 12-bit Custom RISC-V Microprocessor

Submitted to
Dr. Mainul Hossain

Semester: Summer 2022
Section: 6
Group number: 8

Members:

1	2121340642	Fatima Ibrahim
2	2013624042	Farzana Israt Tutul
3	2021162642	Sagor Mahmud Opu
4	2031589642	Abu Bakar Siddik Minhaz

Objective:

Our main goal was to design a new 12-bit RISC type of CPU.

How it works:

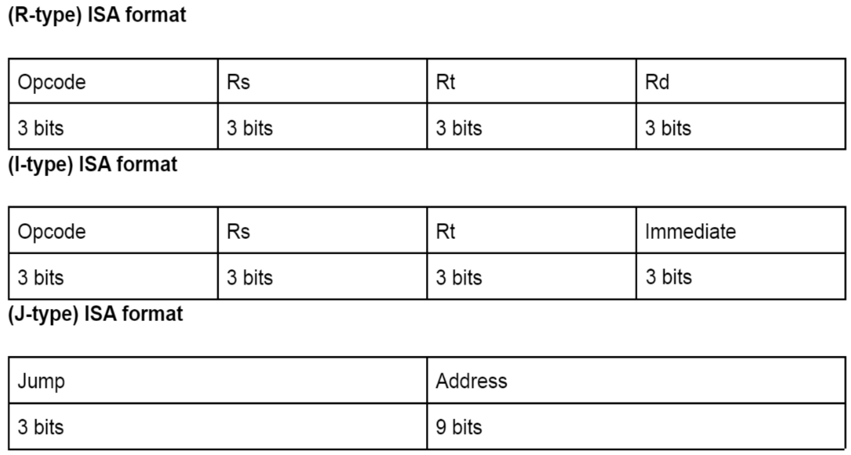
In this project, we wrote an assembler for our ISA. The assembler reads a program written using assembly language in a text file, then translates it into binary code and generates an output file(.txt) containing machine code. The generated output files will later be useful to run a program while we run our instructions in logisim using machine language

Input File:

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

Type of Instructions:

R-Type and I-Type instruction is used here. As we know a target address, immediate value, or branch displacement is not required to use an R-type coding format given in the MIPS R-Type Instruction set. The 3 fields of the format contain the opcode and specification of the three registers. Whereas the I – Type format can specify two registers for source/destination and an immediate value. The Jump Format has 2 fields: Opcode and Address.



(R-type) ISA format

Opcode	Rs	Rt	Rd
3 bits	3 bits	3 bits	3 bits

(I-type) ISA format

Opcode	Rs	Rt	Immediate
3 bits	3 bits	3 bits	3 bits

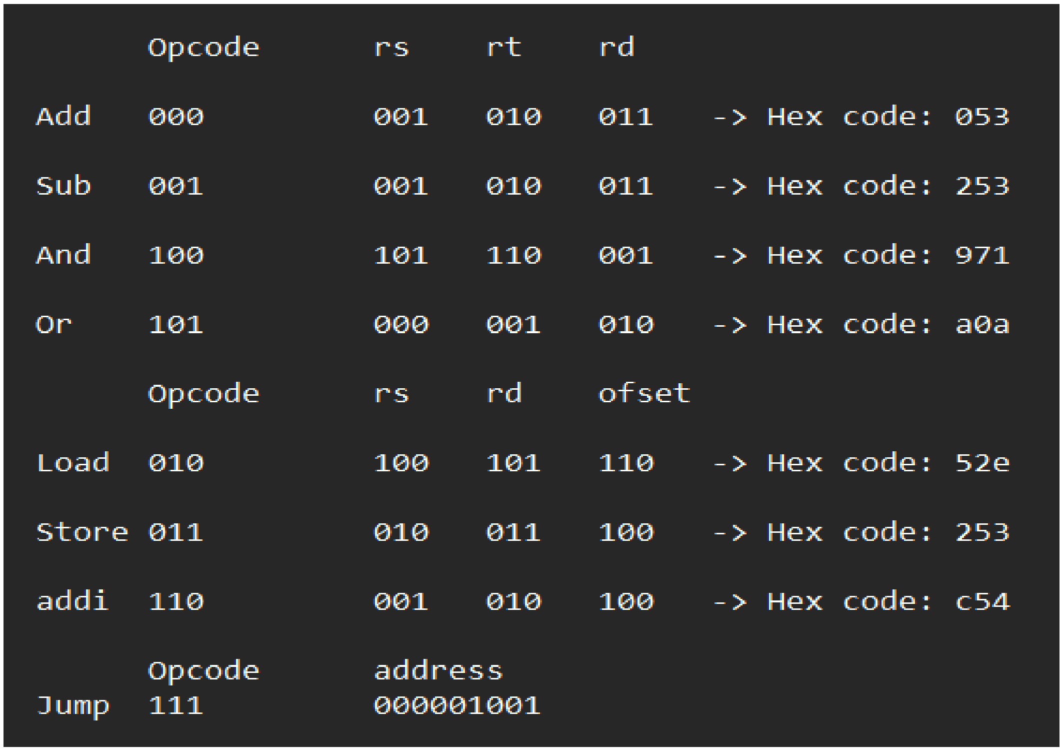
(J-type) ISA format

Jump	Address
3 bits	9 bits

Instruction Description and Operands:

The instructions we used have 3 operands and 8 operations/instructions. The table for the opcodes and example of the instructions are given below:

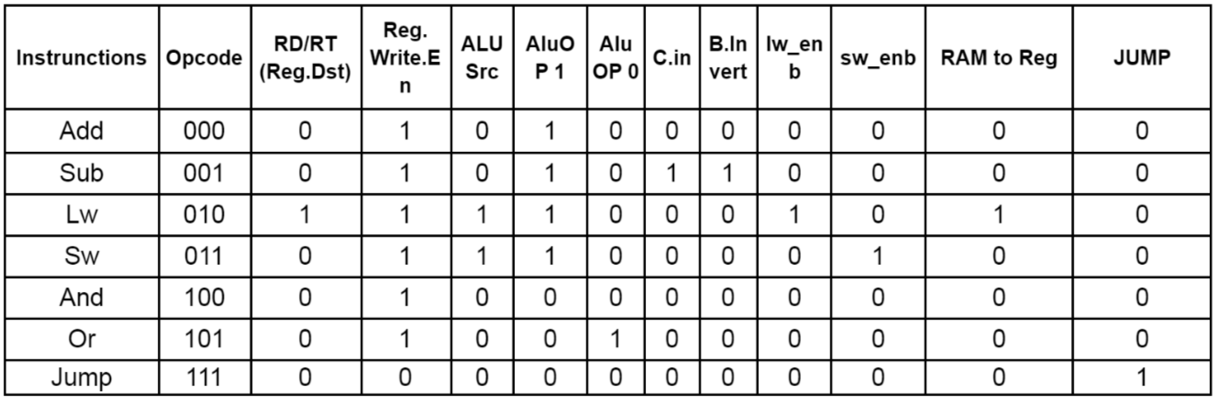
Number	Op-Code	Type
0	000	add
1	001	sub
2	010	lw
3	011	store
4	100	and
5	101	or
6	110	addi
7	111	jump



```
      Opcode      rs    rt    rd

Add   000         001   010   011   -> Hex code: 053

Sub   001         001   010   011   -> Hex code: 253

And   100         101   110   001   -> Hex code: 971

Or    101         000   001   010   -> Hex code: a0a

      Opcode      rs    rd    ofset

Load  010         100   101   110   -> Hex code: 52e

Store 011         010   011   100   -> Hex code: 253

addi  110         001   010   100   -> Hex code: c54

      Opcode      address
Jump  111         000001001
```

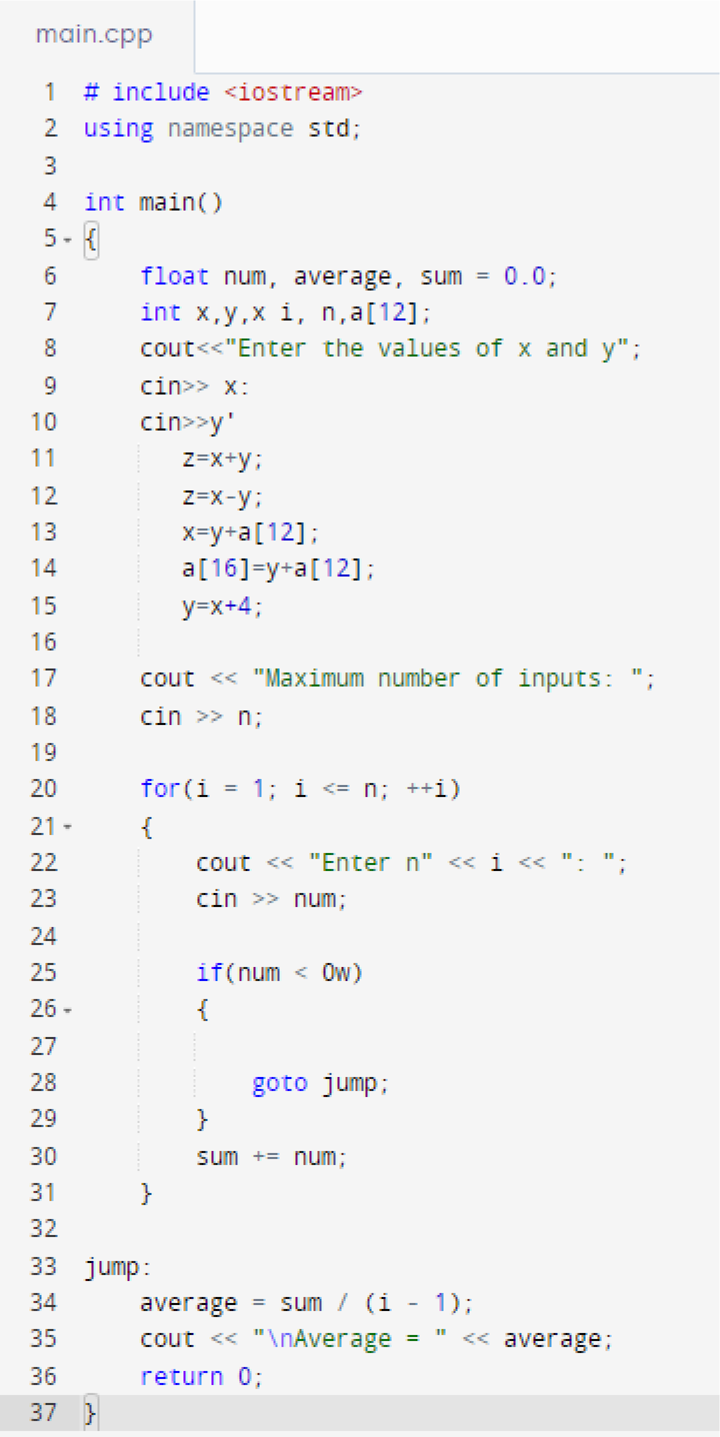
Control Unit Table:

Here is the table for the control unit:



Instrunctions	Opcode	RD/RT (Reg.Dst)	Reg. Write.En	ALU Src	AluOP 1	Alu OP 0	C.in	B.Invert	lw_enb	sw_enb	RAM to Reg	JUMP
Add	000	0	1	0	1	0	0	0	0	0	0	0
Sub	001	0	1	0	1	0	1	1	0	0	0	0
Lw	010	1	1	1	1	0	0	0	1	0	1	0
Sw	011	0	1	1	1	0	0	0	0	1	0	0
And	100	0	1	0	0	0	0	0	0	0	0	0
Or	101	0	1	0	0	1	0	0	0	0	0	0
Jump	111	0	0	0	0	0	0	0	0	0	0	1

High language program:



```cpp
# include <iostream>
using namespace std;

int main()
{
    float num, average, sum = 0.0;
    int x,y,x i, n,a[12];
    cout<<"Enter the values of x and y";
    cin>> x:
    cin>>y'
       z=x+y;
       z=x-y;
       x=y+a[12];
       a[16]=y+a[12];
       y=x+4;

    cout << "Maximum number of inputs: ";
    cin >> n;

    for(i = 1; i <= n; ++i)
    {
        cout << "Enter n" << i << ": ";
        cin >> num;

        if(num < 0w)
        {

            goto jump;
        }
        sum += num;
    }

jump:
    average = sum / (i - 1);
    cout << "\nAverage = " << average;
    return 0;
}
```

Assembly language:

```
Add   $R1,  $R2,  $R3          #R3 = R1+R2

Sub   $R1,  $R2,  $R3          #R3 = R1-R2

LW    $R5,  6($R4)

SW    $R3,  4($R2)

And   $R5,  $R6,  $R1

Or    $R0,  $R1,  $R2

Addi  $R1,  $R2,  $R4

Jump  $R9
```