metin, yazı tipi, daire, logo içeren bir resim

Açıklama otomatik olarak oluşturuldu

**CSE3215**

**DIGITAL LOGIC DESIGN**

**TERM PROJECT**

150120035 Ömer Deligöz

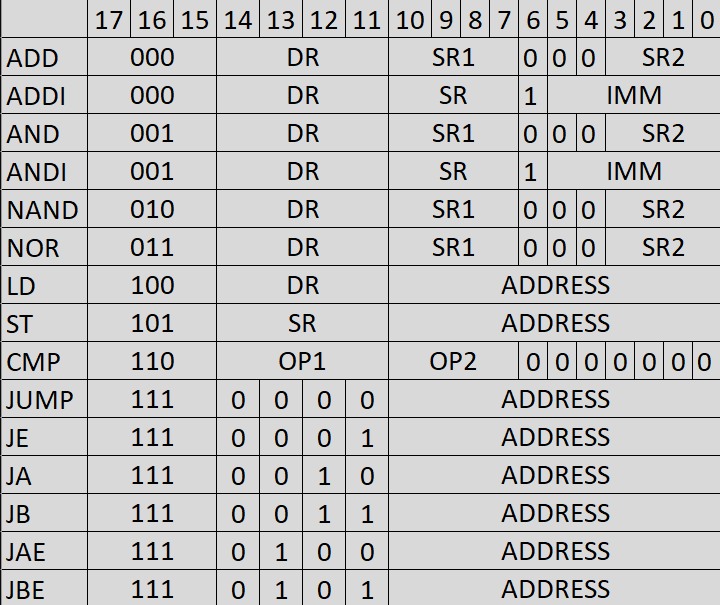
150120005 Ayşe Gülsüm Eren

150121025 Ahmet Abdullah Gültekin

**Description**

This project aims to learn how to design a CPU in Logisim and code in Verilog. First, we designed our instruction set architecture which is a fundamental part of this project. Then we created an assembler for converting this ISA to binary. We decided to requirements for creating a circuit which satisfies our expectations. Then we created all the needed components in Logisim. We wrote the Verilog code of our circuit. We had a CPU which has a control unit, arithmetic logic unit, program counter, comparator, instruction memory, data memory, register file and jump system.

**Instruction Set Architecture (ISA)**

****

**Register**

This is the implementation of 18 bit register with enable and reset functions.

A diagram of a computer circuit

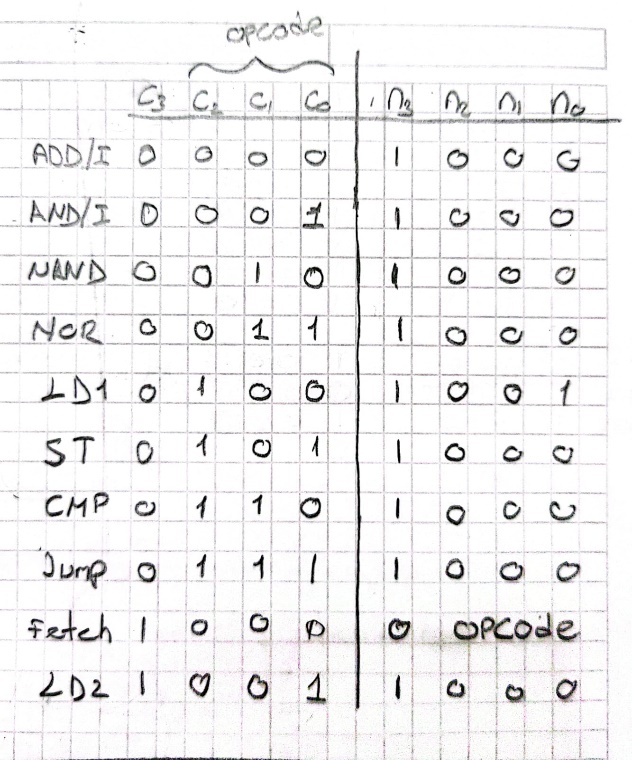
Description automatically generated

**Finite State Machine (FSM)**

This is our FSM diagram. There is a fetch state that takes the instruction to be processed from instruction memory. After the fetch state, moves to the relevant state according to the opcode in the instruction. After the instruction is completed, it returns to the fetch state to get the next instruction.

A screenshot of a computer

Description automatically generated

And this is the truth table of the FSM.

A diagram of a circuit

Description automatically generatedInternal Structure of FSM:

This is the implementation of states. The state coming from FSM is separated by the decoder and the necessary signals are sent for each state.

A diagram of a circuit

Description automatically generated

**Fetch**

FetchInstruction = 1

**ADD/I**

IMM = Instruction [5:0]

IR [6] = Instruction [6]

IR [16:15] = Instruction [16:15]

WriteEnable = 1

ReadEnable1 = 1

PcWrite = 1

If (Instruction [6] = 1)

ReadEnable2 = 1

**AND/I**

IMM = Instruction [5:0]

IR [6] = Instruction [6]

IR [16:15] = Instruction [16:15]

WriteEnable = 1

ReadEnable1 = 1

PcWrite = 1

If (Instruction [6] == 1)

ReadEnable2 = 1

**NAND**

IR [16:15] = Instruction [16:15]

WriteEnable = 1

ReadEnable1 = 1

PcWrite = 1

ReadEnable2 = 1

**NOR**

IR [16:15] = Instruction [16:15]

WriteEnable = 1

ReadEnable1 = 1

PcWrite = 1

ReadEnable2 = 1

**LD1**

ADDRESS = Instruction [10:0]

MemRead = 1

**LD2**

ADDRESS = Instruction [10:0]

WriteEnable = 1

MemRead = 1

PcWrite = 1

**ST**

ADDRESS = Instruction [10:0]

ReadEnable1 = 1

MemWrite = 1

PcWrite = 1

**CMP**

CMP Signal = 1

ReadEnable1 = 1

PcWrite = 1

ReadEnable2 = 1

**JUMP**

ADDRESS = IR [10:0]

Jump Signal = 1

IR [13:11] = Instruction [13:11]

PcWrite = 1

**Verilog**

This is the Verilog implementation of main part of our CPU. metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, yazı tipi, doküman, belge içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Conclusion**

We made the necessary tests, and our design works truly at the end of this process.