Project Report

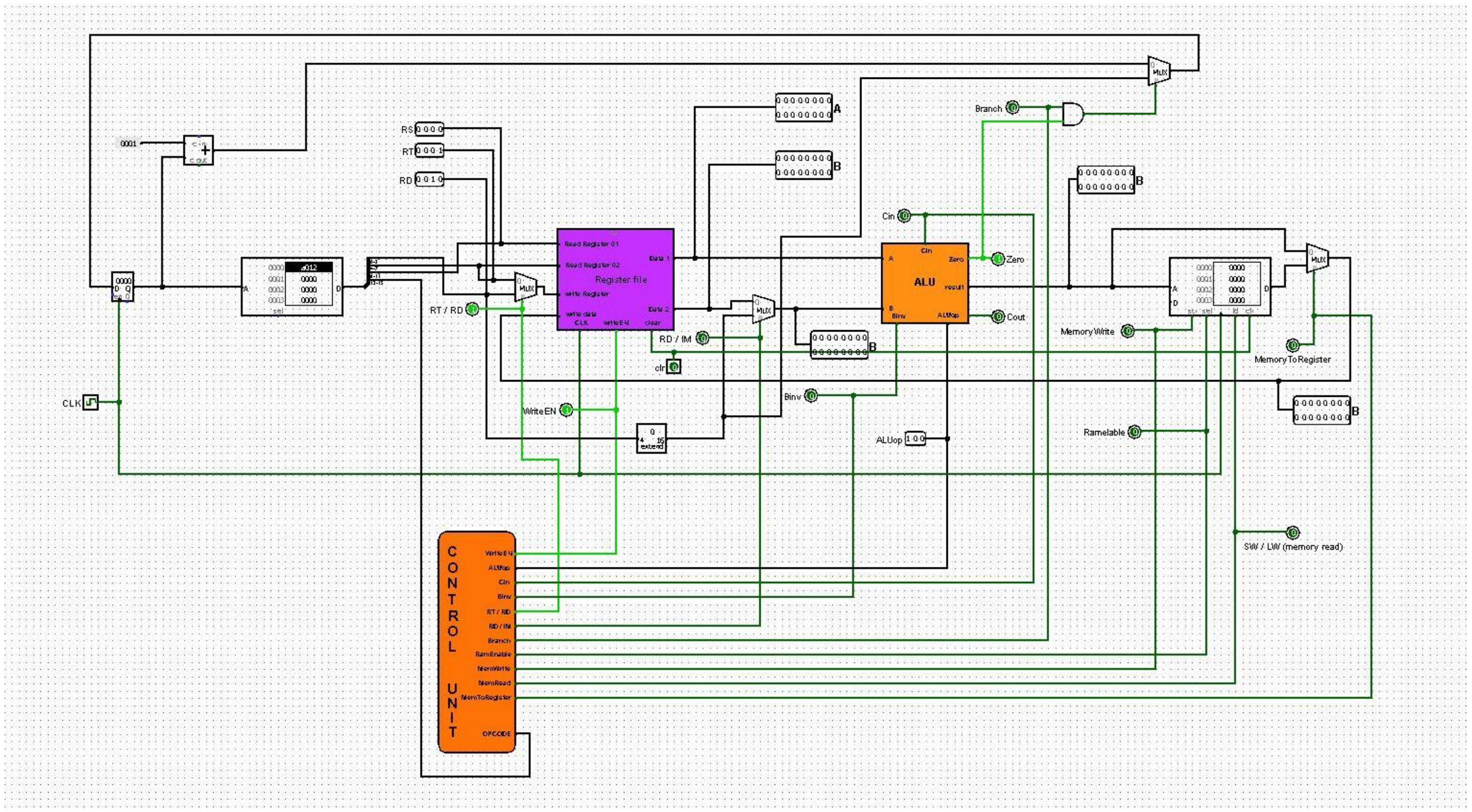
Name: Mehdi Hassan fahim
ID: 1921539042
Course: CSE332
Section: 04

Introduction: This project is about 16-bit Single Cycle Processor. This project based on MIPS architecture. Which architecture have three types such as R-type, I-type, J-type. By using this format I build a 16 bit ISA. Which can do lots of operation such as arithmetic operation, data transfer and logical operation. In this Architecture we have control unit, ALU, Register file, ROM and RAM.

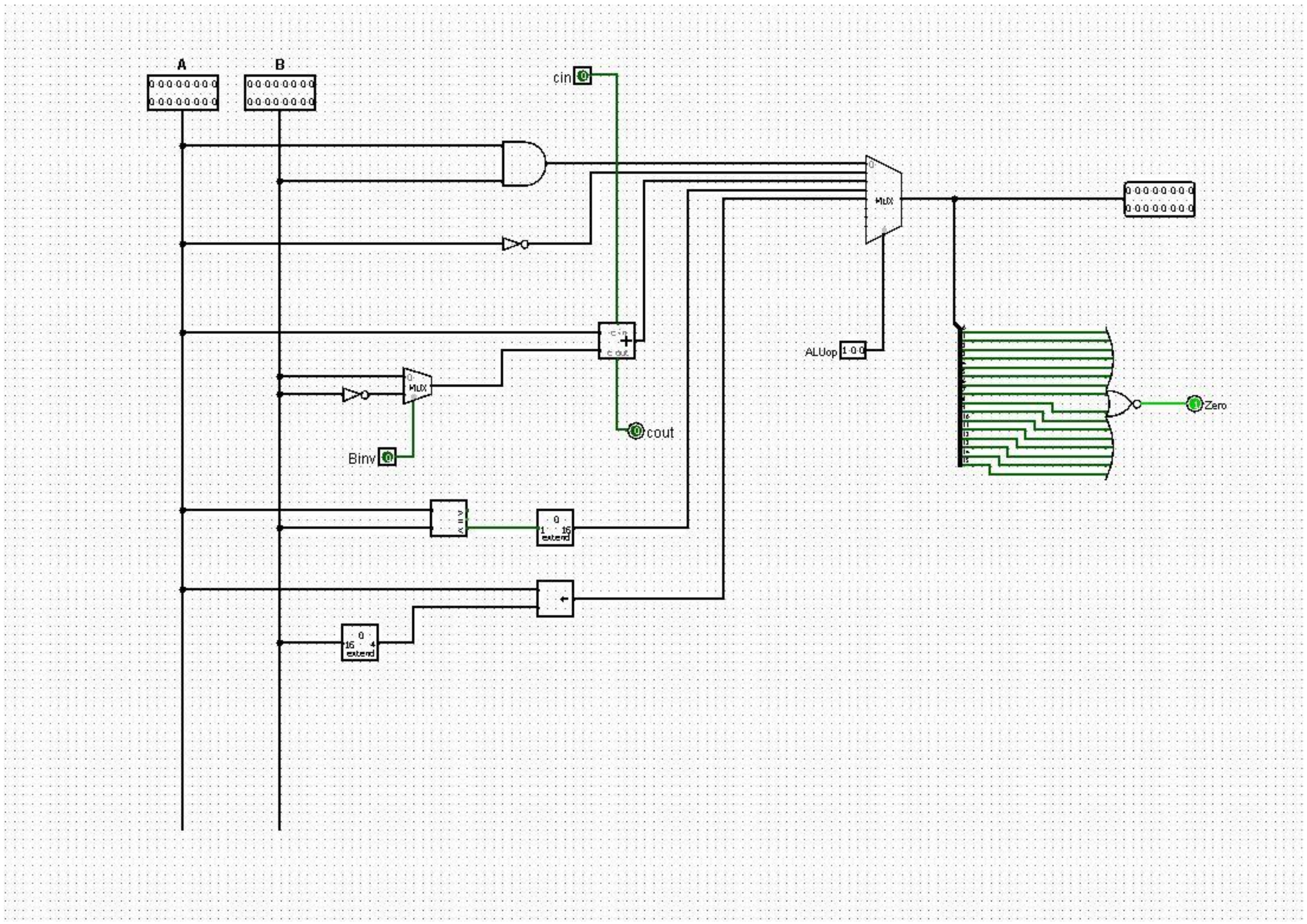
Components:

1. **ROM:** ROM is just a short form of read-only memory. It refers to computer memory chips that contain computer permanent or semi-permanent data. ROM is non-volatile as opposed to ROM; Even after you turn off your computer, ROM content will remain.
2. **PC:** The program counter (PC) is a register that manages the memory address of the instruction and is subsequently executed.
3. **ALU:** An addition is a digital circuit that performs the addition of numbers. Adders are used in arithmetical logic units or ALUs on many computers and other types of processors. They are also used in other parts of the processor, where they are used to calculate addresses and similar activities.
4. **Register File:** A register file is an array of processor registers in a central processing unit (CPU). Modern integrated circuit-based register files are usually applied via fast static RAM with multiple ports.
5. **RAM:** RAM (random access memory) is the hardware of a computing device that stores the operating system, application programs, and current usage data so that they can be quickly accessed by the device's processor. RAM RAM is the main memory of a computer.

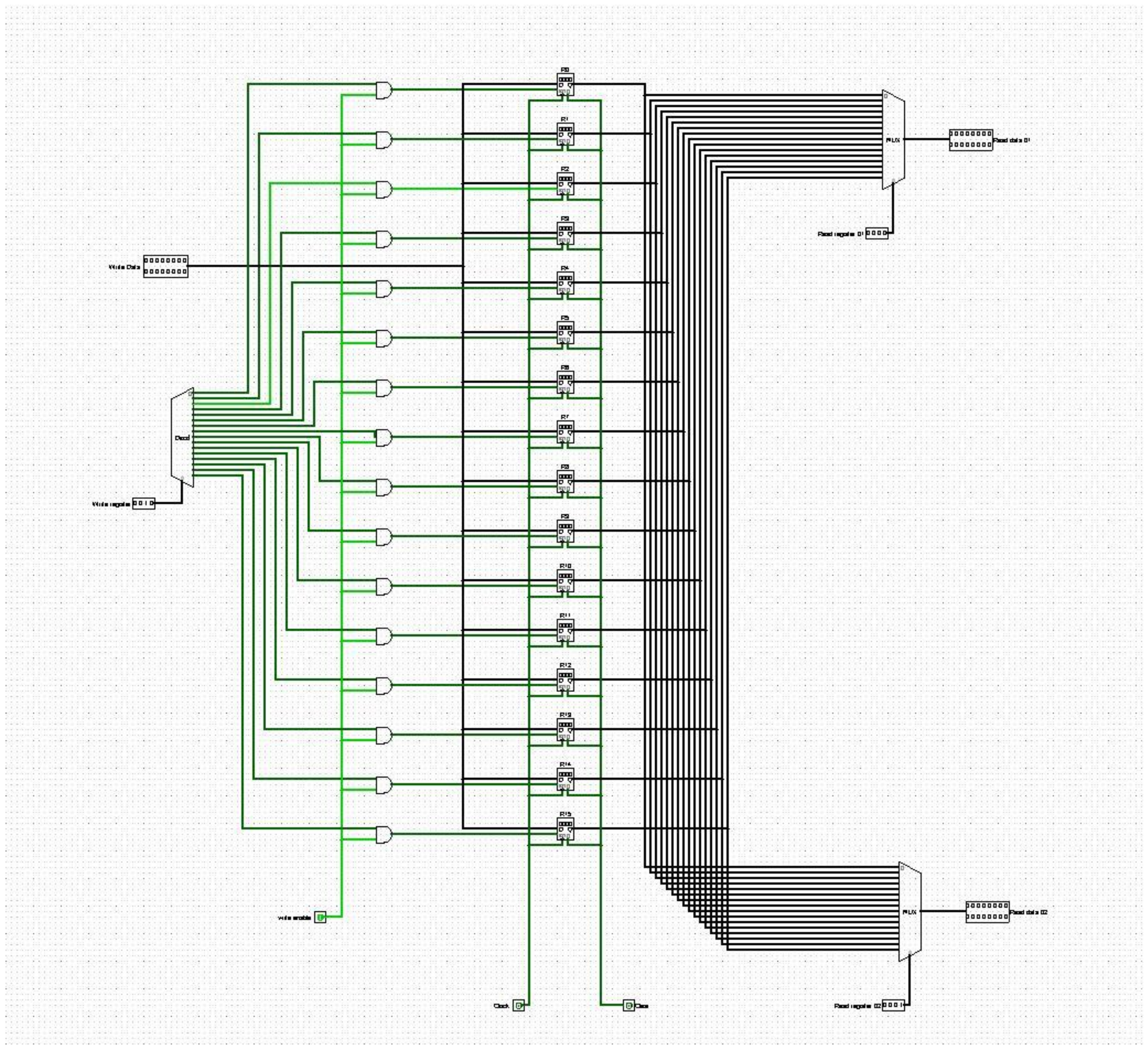
DataPath & components:



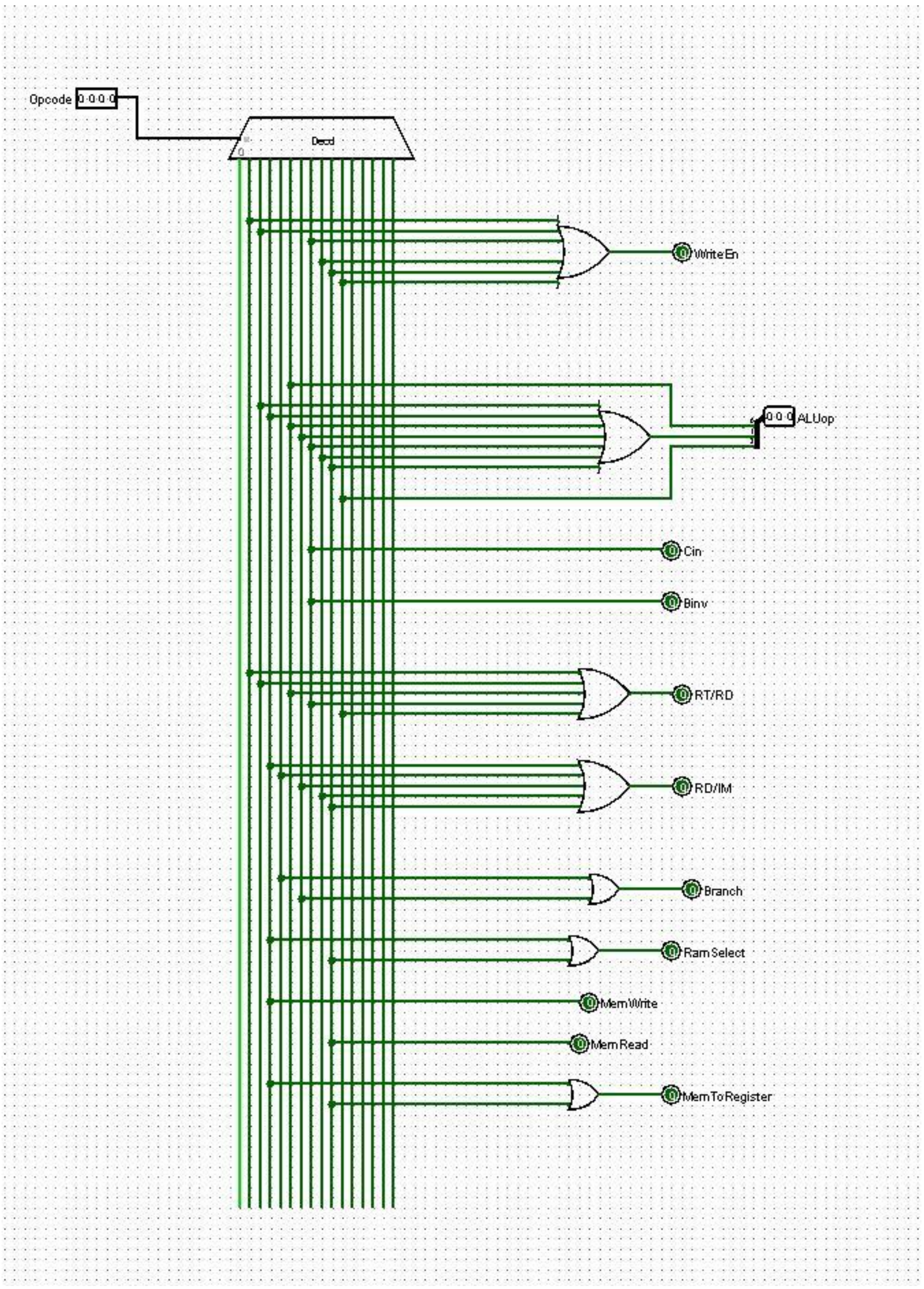
DataPath



ALU



RegisterFile



Control Unit

Control Unit:

WriteEN : If this one is active it will write the data in register.
ALUop: If this one is active it will decide that which arithmetical operation will work.
Cin: when we do subtraction it will active.
Binv: At the moment of subtraction we active this one.
RT/RD: If RT its value is 0 and RD its value 1. If RD is active it's going to be R type.
RD/IM: If RD its value is 0 and IM its value 1. If IM is active its going to be I type.
Branch: When we do beq and jmp this one is active.
RamEnable: when we do load word and store word.
MemWrite: when we do store word this will active.
MemRead: when we do load word this will active.
MemToRegister: when we do load word and store word.

Assembeler:

```
def convertBinToHex(bin):
    hex =" "
    if bin == "0000":
        hex = "0"
    elif bin == "0001":
        hex = "1"
    elif bin == "0010":
        hex = "2"
    elif bin == "0011":
        hex = "3"
    elif bin == "0100":
        hex = "4"
    elif bin == "0101":
        hex = "5"
    elif bin == "0110":
        hex = "6"
    elif bin == "0111":
        hex = "7"
    elif bin == "1000":
        hex = "8"
    elif bin == "1001":
        hex = "9"
    elif bin == "1010":
        hex = "A"
    elif bin == "1011":
        hex = "B"
    elif bin == "1100":
        hex = "C"
    elif bin == "1101":
        hex = "D"
    elif bin == "1110":
        hex = "E"
    elif bin == "1111":
        hex = "F"
    return hex

# AND   R3    R4     R5
# 0001  0011  0100   0101
# 1     3      4      5
# 1345

def checkInstruction(inst):
    convertInstruction = " "
    if inst == "nop":
        convertInstruction = "0000"
    elif  inst == "and":
        convertInstruction = "0001"
    elif inst == "add":
        convertInstruction = "0010"
    elif inst == "sw":
        convertInstruction = "0011"
    elif inst == "jmp":
        convertInstruction = "0100"
    elif inst == "slt":
        convertInstruction = "0101"
    elif inst == "beq":
        convertInstruction = "0110"
    elif inst == "sub":
        convertInstruction = "0111"
    elif inst == "addi":
        convertInstruction = "1000"
    elif inst == "lw":
        convertInstruction = "1001"
    elif inst == "sll":
        convertInstruction = "1010"
    else:
        convertInstruction = "Invalid instrcutions"
    return convertInstruction


def checkRegister(reg):

    convertReg = ""
    if  reg == "R0":
        convertReg ="0000"
    elif reg == "R1":
        convertReg ="0001"
    elif reg == "R2":
        convertReg ="0010"
    elif reg == "R3":
        convertReg ="0011"
    elif reg == "R4":
        convertReg ="0100"
    elif reg == "R5":
        convertReg ="0101"
    elif reg == "R6":
        convertReg ="0110"
    elif reg == "R7":
        convertReg ="0111"
    elif reg == "R8":
        convertReg ="1000"
    elif reg == "R9":
        convertReg ="1001"
    elif reg == "R10":
        convertReg ="1010"
    elif reg == "R11":
        convertReg ="1011"
    elif reg == "R12":
        convertReg ="1100"
    elif reg == "R13":
        convertReg ="1101"
    elif reg == "R14":
        convertReg ="1110"
    elif reg == "R15":
        convertReg ="1111"
    else:
        convertReg =="Invalid Register"

    return convertReg


def decimalToBinary(num):

    if(num<0):
        num =  16 + num

    ext = ""
    result = ""

    while(num>0):
        if num % 2 == 0:
            result = "0" + result
        else:
            result = "1" + result
        #result = (num%2 == 0 ? "0" : "1") + result
        num = num//2

    for i in range(4-len(result)):
        ext = "0" + ext

    result = ext + result


    return result

#a[1,6,7,8]
#for(i=0, i<4, i++ )
```

```
#    {
#        a[i];
#    }

#a = ['apple', 'ball', 'cat', 'dog']
#for i in a:
#    i

readf = open("inputs","r")
writef = open("outputs","w")
writef.write("v2.0 raw\n")

# A quick brown fox jumped over a lazy dog

#print(f.readline())
for i in readf:
    splitted = i.split()

    if(splitted[0] == "nop" or splitted[0] == "and" or splitted[0] == "add" or splitted[0] == "slt" or splitted[0] == "sub" or splitted[0]
== "sll"):
        conv_inst = convertBinToHex(checkInstruction(splitted[0]))
        conv_rs = convertBinToHex(checkRegister(splitted[1]))
        conv_rt = convertBinToHex(checkRegister(splitted[2]))
        conv_rd = convertBinToHex(checkRegister(splitted[3]))

        out = conv_inst + conv_rs + conv_rt + conv_rd
        print(out)
        writef.write(out+"\n")

    elif(splitted[0] == "sw" or splitted[0] == "beq" or splitted[0] == "addi" or splitted[0] == "lw"):
        conv_inst = convertBinToHex(checkInstruction(splitted[0]))
        conv_rs = convertBinToHex(checkRegister(splitted[1]))
        conv_rt = convertBinToHex(checkRegister(splitted[2]))
        conv_im = convertBinToHex(decimalToBinary(int(splitted[3])))

        out = conv_inst + conv_rs + conv_rt + conv_im
        print(out)
        writef.write(out+"\n")

    elif(splitted[0] == "jmp"):
        conv_inst = convertBinToHex(checkInstruction(splitted[0]))
        #conv_target = convertBinToHex(decimalToBinary(int(splitted[1])))
        hexval = hex(int(splitted[1]))
        exF2 = hexval[2:]
        ext = ""
        for i in range(3 - len(exF2)):
            ext = "0" + ext

        conv_target = ext + exF2

        out = conv_inst + conv_target
        print(out)
        writef.write(out+"\n")
```