Name: MD. Mehedi Hasan Bhuiyan Nipu

ID: 1911870642

Section :4

Course Code: CSE 332

Project name: 16-bit Single Cycle Processor

Submitted to: MS. TANJILA FARAH

Introduction

In this project we will implement a 16-bit single cycle Processor by the help with MIPS architecture. In this project we will Use 3 type of architecture. R-type, I-type, J type. We build a complete Datapath for this project. In this Datapath we will be able to do many operations such as arithmetic operation, data transfer and logical operation. By designing this whole Datapath we had to design Rom, Ram, control unit, Alu, Register file.

Components Definition:

Alu: An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. The control unit tells the ALU what operation to perform on that data and the ALU stores the result in an output register.

Register file: A register file is a means of memory storage within central processing unit. The register files contain bits of data and mapping locations. These locations specify certain addresses that are input components of a register file.

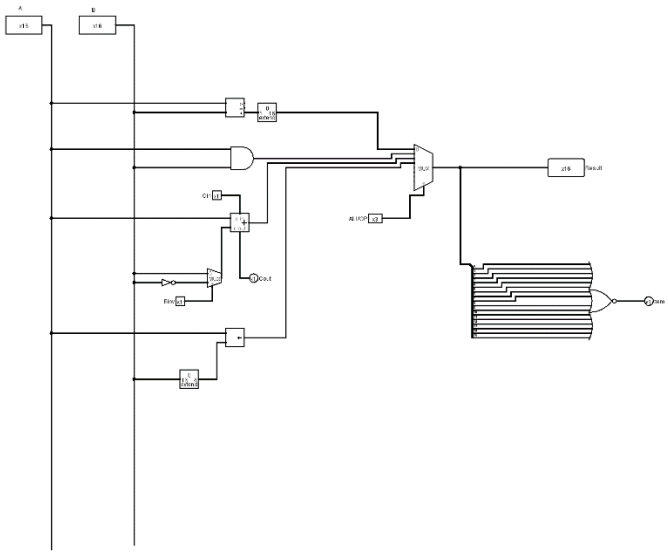
Rom: Read-only memory (ROM) is a type of non-volatile memory used in computers and other electronic devices.

Ram: Random Access Memory (RAM) provides space for your computer to read and write data to be accessed by the CPU (central processing unit). ... RAM is volatile, so data stored in RAM stays there only as long as your computer is running. As soon as you turn the computer off, the data stored in RAM disappears

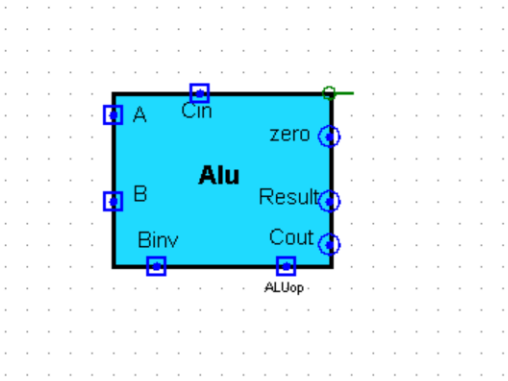
Program Counter: The program counter (PC) is a register that controls the instruction's memory address before it is performed.

Components Snap

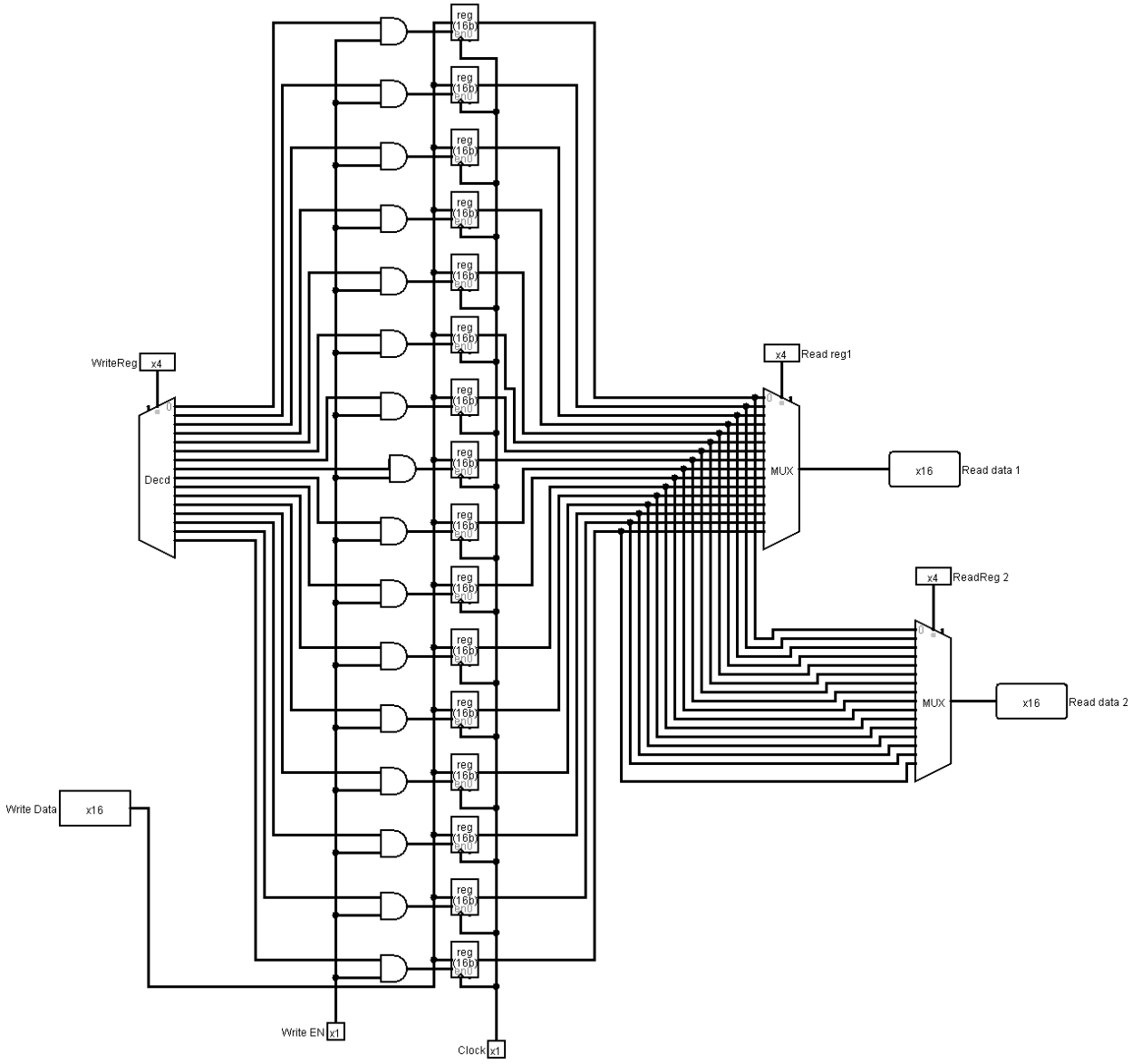
Alu main :



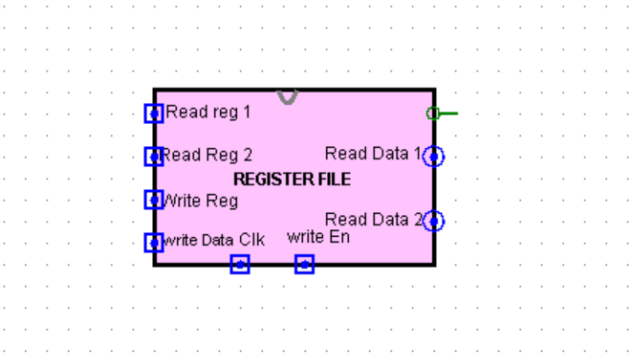
Alu Sub:



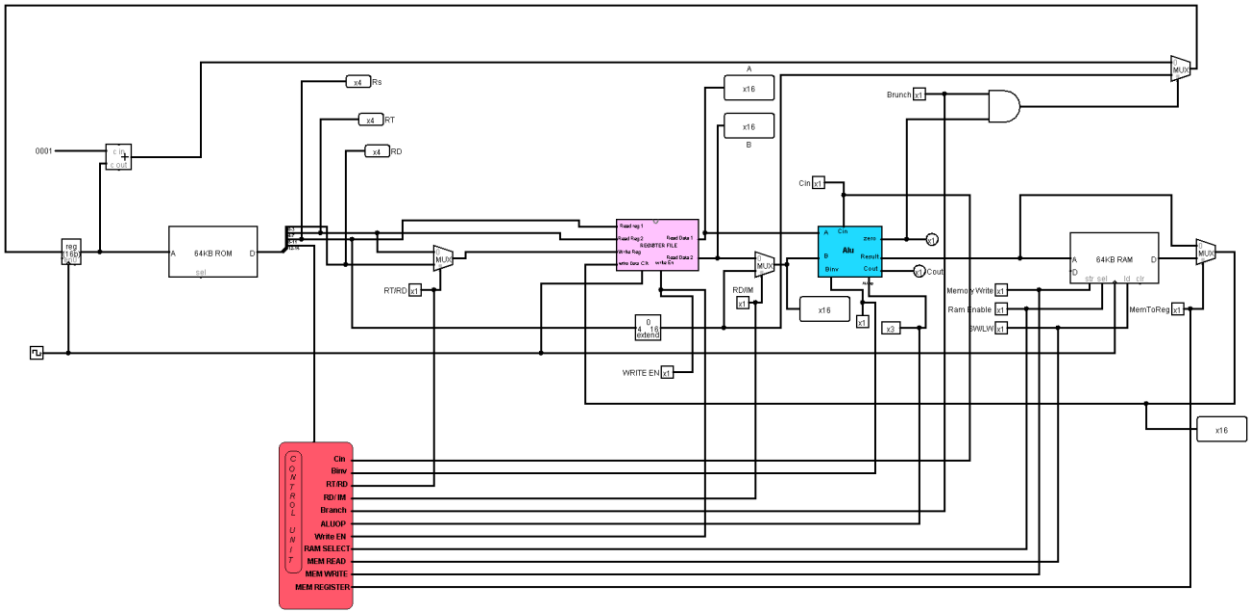
Register



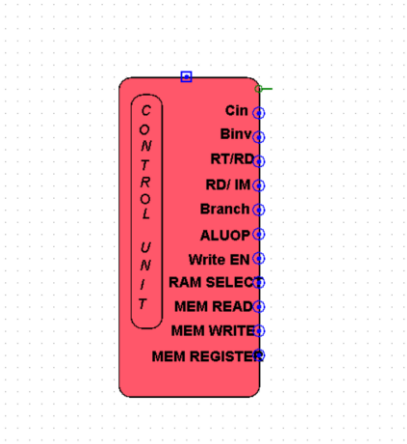
Register file Sub:



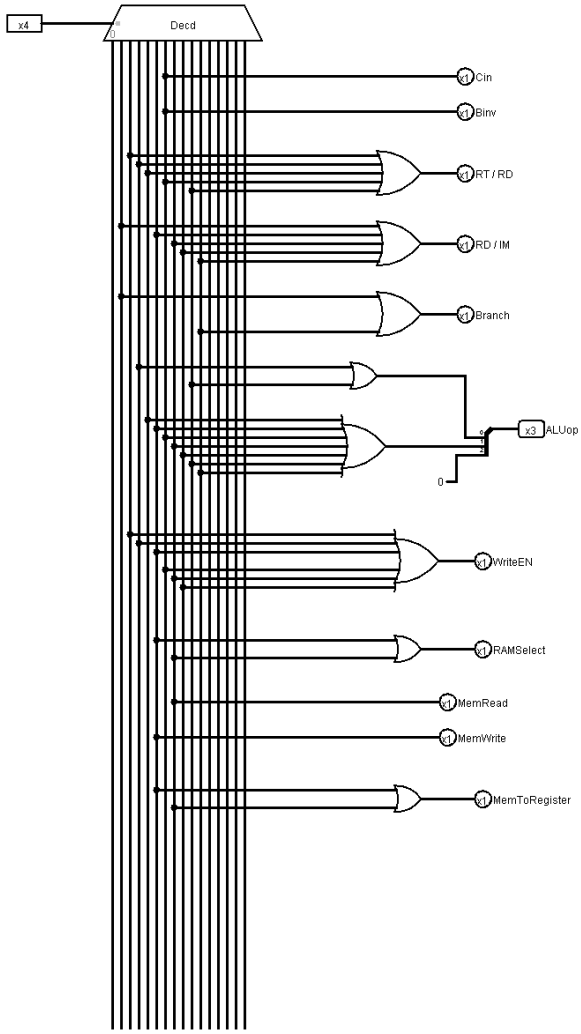
Datapath:


0001
64kB ROM
Rs
RT
RD
MUX
RT/RD
WRITE EN
RD/IM
Alu
Cin
Cout
Brunch
64kB RAM
Memory Write
Ram Enable
MemToReg
CONTROL UNIT
Cin
Binv
RT/RD
RD/ IM
Branch
ALUOP
Write EN
RAM SELECT
MEM READ
MEM WRITE
MEM REGISTER

Control Unit Sub :



Control Unit :



Control Unit Described:

Alu Op: By this we can decide that which arithmetical operation will work.

Binv: For sub we will active this one.

Cin: When we do sub it will active.

Branch: If we want beq & jmp it will active.

Mem write: For store word
Mem read: For load word
Mem To Register: when we do load word and store word.
RT/RD: We can decide by seeing their value what type of instruction it will be. (RT=0 &RD=1 and RD is active then it will 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.

Write Enable: It will write data in register if it's active.

Assembler Documentation: In the assembler I did use python to implement the program .Here In the first thing I did to write the code in **Check instruction** according to my instruction serial that I was given by . after that in the **checkRegister** I have to put the register name I want to use so that the assembler can convert it. So I Used **rg** as register name(convert).

```
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 == "jmp":
        convertInstruction = "0001"
    elif inst == "slt":
        convertInstruction = "0010"
    elif inst == "and":
```

```
        convertInstruction = "0011"
    elif inst == "add":
        convertInstruction = "0100"
    elif inst == "sw":
        convertInstruction = "0101"
    elif inst == "sub":
        convertInstruction = "0110"
    elif inst == "lw":
        convertInstruction = "0111"
    elif inst == "addi":
        convertInstruction = "1000"
    elif inst == "sll":
        convertInstruction = "1001"
    elif inst == "beq":
        convertInstruction = "1010"

    else:
        convertInstruction = "Invalid instrcutions"
    return convertInstruction



def checkRegister(reg):

    convertReg = ""
    if  reg == "rg0":
        convertReg ="0000"
    elif reg == "rg1":
        convertReg ="0001"
    elif reg == "rg2":
        convertReg ="0010"
    elif reg == "rg3":
        convertReg ="0011"
    elif reg == "rg4":
        convertReg ="0100"
    elif reg == "rg5":
```

```
        convertReg ="0101"
    elif reg == "rg6":
        convertReg ="0110"
    elif reg == "rg7":
        convertReg ="0111"
    elif reg == "rg8":
        convertReg ="1000"
    elif reg == "rg9":
        convertReg ="1001"
    elif reg == "rg10":
        convertReg ="1010"
    elif reg == "rg11":
        convertReg ="1011"
    elif reg == "rg12":
        convertReg ="1100"
    elif reg == "rg13":
        convertReg ="1101"
    elif reg == "rg14":
        convertReg ="1110"
    elif reg == "rg15":
        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] == "slt" or splitted[0] == "and" or splitted[0] == "add" 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] == "lw" or splitted[0] == "addi" or splitted[0] == "beq"):
    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")
```

Discussion: In this project we did some simple operation like add , sub, jmp, slt , sw, e.t.c.They did particular thing individual. And also have different type of op code to set . We can load data , add data, sub data, shift left and many more. Also we did write assembly language to generate the value for them. After that we set the value at our datapath and check if it's working or not .

For example :If I want to implement add operation I can do it by two ways 1st : If I want to do it manually I have to set the value of my opcode at rom . because add operation is a R type operation so we know that RS RT & RD method will work . So in rom I have to put 1st op code in my 1st value then RS & RT will have the value I want to add . and RD will store the value after add (RS+RT)= RD.

And secondly I can do the same operation with assembler. If I write assembly language like **add rg0 rg1**

(rg2=rg0+rg1) then we will have a output . then we have to simply load our output in datapath rom. Then we will put value in our register. Reg 2 will have the value after addition of rg0, rg1.

Thank you