**RISC 5 (32 BIT) ASSEMBLER ON PYTHON3**

**Google colab link**

[**https://colab.research.google.com/drive/161R08fIS8p-c7pbGmbx52FdygY3qQxFF?usp=sharing**](https://colab.research.google.com/drive/161R08fIS8p-c7pbGmbx52FdygY3qQxFF?usp=sharing)

**Testbench(CHECKED AND VERIFIED)**

**1. jal x1,681976**

Details of your pnemonics are as follows---

opcode-- 1101111

destination address-- 00001

immediate\_bitcode-- 010100110011111111000

BITCODE value-- 0111111110001010011000001110111132 hexadecimal Code--- 0x 7F8A60EF

**2.** **lui x21,577007**

Details of your pnemonics are as follows---

opcode-- 0110111

destination address-- 10101

immediate\_bitcode-- 10001100110111101111

BITCODE value-- 10001100110111101111101010110111

32 B hexadecimal Code--- 0x 8CDEFAB7

**3. bne x24,x5,-2908**

Details of your pnemonics are as follows---

Instruction Type-- SB

['BNE', 'X24', 'X5', '-2908']

opcode-- 1100011

source1 code-- 11000

func3 code-- 001

source2 code-- 00101

immediate\_bitcode-- 1010010100100

BITCODE value-- 11001010010111000001001001100011

32 B hexadecimal Code--- 0x CA5C1263

**4.SH X20,23(X5)**

Details of your pnemonics are as follows---

['SH', 'X20', '23', 'X5']

opcode-- 0100011

source1 address-- 00101

source2 address-- 10100

func3 code-- 001

immediate\_bitcode-- 000000010111

BITCODE value-- 00000001010000101001101110100011

32 B hexadecimal Code--- 0x 01429BA3

**5.LW X28,-36(X20)**

Details of your pnemonics are as follows---

['LW', 'X28', '-36', 'X20']

opcode-- 0000011

destination address-- 11100

func3 code-- 010

source address-- 10100

immediate\_bitcode-- 111111011100

BITCODE value-- 11111101110010100010111000000011

32 B hexadecimal Code--- 0x FDCA2E03

**6.SRAI X6,X7,29**

Details of your pnemonics are as follows---

Instruction Type-- I

['SRAI', 'X6', 'X7', '29']

opcode-- 0010011

destination code-- 00110

func3 code-- 101

source1 code-- 00111

immediate\_bitcode-- 000000011101

BITCODE value-- 01000001110100111101001100010011

32 B hexadecimal Code--- 0x 41D3D313

**7.LW X7,-6(X19)**

Details of your pnemonics are as follows---

['LW', 'X7', '-6', 'X19']

opcode-- 0000011

destination address-- 00111

func3 code-- 010

source address-- 10011

immediate\_bitcode-- 111111111010

BITCODE value—11111111101010011010001110000011

32 B hexadecimal Code--- 0x FFA9A383

**8.ADDI X8,X9,12**

Details of your pnemonics are as follows---

Instruction Type-- I

['ADDI', 'X8', 'X9', '12']

opcode-- 0010011

destination code-- 01000

func3 code-- 000

source1 code-- 01001

immediate\_bitcode-- 000000001100

BITCODE value-- 00000000110001001000010000010011

32 B hexadecimal Code--- 0x 00C48413

**9. ADD X18,X19,X20**

Details of your pnemonics are as follows---

Instruction Type-- R

['ADD', 'X18', 'X19', 'X20']

opcode-- 0110011

destination code-- 10010

func3 code-- 000

func7 code-- 0000000

source1 code-- 10011

source2 code-- 10100

BITCODE VALUE -- 00000001010010011000100100110011

32 B hexadecimal Code--- 0x 01498933

**10.SUB X5,X6,X7**

Details of your pnemonics are as follows---

Instruction Type-- R

['SUB', 'X5', 'X6', 'X7']

opcode-- 0110011

destination code-- 00101

func3 code-- 000

func7 code-- 0100000

source1 code-- 00110

source2 code-- 00111

BITCODE VALUE -- 01000000011100110000001010110011

32 B hexadecimal Code--- 0x 407302B3

**Code of Implementation**

 encoder\_dic={

#OPCODE DICTIONARY

# R TYPE INSTRUCTION

"ADD":["R","0110011","000","0000000"],

"SUB":["R","0110011","000","0100000"],

"SLL":["R","0110011","001","0000000"],

"XOR":["R","0110011","100","0000000"],

"SRL":["R","0110011","101","0000000"],

"SRA":["R","0110011","101","0010000"],

"OR":["R","0110011","110","0000000"],

"AND":["R","0110011","111","0000000"],

# I TYPE INSTRUCTION

#type1

"LB":["I","0000011","000","NA"],

"LH":["I","0000011","001","NA"],

"LW":["I","0000011","010","NA"],

"LBU":["I","0000011","100","NA"],

"LHU":["I","0000011","101","NA"],

#type2

"ADDI":["I","0010011","000","NA"],

"SLLI":["I","0010011","001","0000000"],

"XORI":["I","0010011","100","NA"],

"SRAI":["I","0010011","101","0100000"],

"SLLI":["I","0010011","001","0000000"],

"ORI":["I","0010011","110","NA"],

"ANDI":["I","0010011","111","NA"],

#type3

"JALR":["I","1100111","000","NA"],

#S TYPE INSTRUCTION

"SB":["S","0100011","000","NA"],

"SH":["S","0100011","001","NA"],

"SW":["S","0100011","010","NA"],

#SB TYPE INSTRUCTION

"BEQ":["SB","1100011","000","NA"],

"BNE":["SB","1100011","001","NA"], #CONFIRM WITH SIR 1100111 OR 1100011

"BLT":["SB","1100011","100","NA"],

"BGE":["SB","1100011","101","NA"],

"BLTU":["SB","1100011","110","NA"],

"BGEU":["SB","1100011","111","NA"],

#U TYPE INSTRUCTION

"LUI":["U","0110111","NA","NA"],

#UJ TYPE INSTRUCTION

"JAL":["UJ","1101111","NA","NA"],

}

register\_dic={"X0":"00000", #REGISTER DICTIONARY

"X1":"00001",

"X2":"00010",

"X3":"00011",

"X4":"00100",

"X5":"00101",

"X6":"00110",

"X7":"00111",

"X8":"01000",

"X9":"01001",

"X10":"01010",

"X11":"01011",

"X12":"01100",

"X13":"01101",

"X14":"01110",

"X15":"01111",

"X16":"10000",

"X17":"10001",

"X18":"10010",

"X19":"10011",

"X20":"10100",

"X21":"10101",

"X22":"10110",

"X23":"10111",

"X24":"11000",

"X25":"11001",

"X26":"11010",

"X27":"11011",

"X28":"11100",

"X29":"11101",

"X30":"11110",

"X31":"11111",

}

def two\_complement(immediate\_bitcode): # 2'S COMPLEMENT FUNCTIONS

immediate\_bitcode\_list=list(immediate\_bitcode)[::-1]

for i in range(0,len(immediate\_bitcode\_list)):

if immediate\_bitcode\_list[i]=='1':

for j in range(i+1,len(immediate\_bitcode\_list)):

if immediate\_bitcode\_list[j]=='0':

immediate\_bitcode\_list[j]='1'

else:

immediate\_bitcode\_list[j]='0'

break

immediate\_bitcode="".join(immediate\_bitcode\_list[::-1])

return immediate\_bitcode

def zero\_padding(val,size): # PADDING FUNCTION

bit\_size=len(val)

for i in range(size-bit\_size):

val="0"+val

return val

def binary\_hex\_32\_bit(binarycode): #BINARY TO HEXADECIMAL CONVERTOR FUNCTION

hex=""

for i in range(0,31,4):

code=binarycode[i:i+4]

int\_val=0

for j in range(4):

int\_val+=(2\*\*(3-j))\*int(code[j])

if int\_val>9:

if int\_val==10:

hex\_code='A'

if int\_val==11:

hex\_code='B'

if int\_val==12:

hex\_code='C'

if int\_val==13:

hex\_code='D'

if int\_val==14:

hex\_code='E'

if int\_val==15:

hex\_code='F'

else:

hex\_code=str(int\_val)

hex+=hex\_code

return hex

def risc5assembler(code\_arr):

  #R TYPE INSTRUCTION

  if encoder\_dic[code\_arr[0]][0]=="R":

    print("Instruction Type--",encoder\_dic[code\_arr[0]][0])

    print(code\_arr)

    print("opcode--",encoder\_dic[code\_arr[0]][1])

    print("destination code--",register\_dic[code\_arr[1]])

    print("func3 code--",encoder\_dic[code\_arr[0]][2])

    print("func7 code--",encoder\_dic[code\_arr[0]][3])

    print("source1 code--",register\_dic[code\_arr[2]])

    print("source2 code--",register\_dic[code\_arr[3]])

    bitcode=encoder\_dic[code\_arr[0]][3]+register\_dic[code\_arr[3]]+register\_dic[code\_arr[2]]+encoder\_dic[code\_arr[0]][2]+register\_dic[code\_arr[1]]+encoder\_dic[code\_arr[0]][1]

    print("BITCODE VALUE --",bitcode)

    hex=binary\_hex\_32\_bit(bitcode)

    print("32 B hexadecimal Code---","0x",hex)

    #I TYPE INSTRUCTION

  #TYPE 1 INSTRUCTION

  if encoder\_dic[code\_arr[0]][0]=="I" and encoder\_dic[code\_arr[0]][1]=="0000011":

    last\_arr=code\_arr[-1].split('(')

    code\_arr=code\_arr[:-1]+[last\_arr[0],last\_arr[-1][0:-1]]

    print(code\_arr)

    opcode=encoder\_dic[code\_arr[0]][1]

    des\_address=register\_dic[code\_arr[1]]

    func3=encoder\_dic[code\_arr[0]][2]

    s\_address=register\_dic[code\_arr[-1]]

    print("opcode--",opcode)

    print("destination address--",des\_address)

    print("func3 code--",func3)

    print("source address--",s\_address)

    immediate=int(code\_arr[-2]) #uses 2's complement system

    immediate\_bitcode=zero\_padding(bin(abs(immediate))[2:],12) #  positive number's 2's complement calculation

    # negative number's 2's complement calculation

    if immediate<0:

      immediate\_bitcode=two\_complement(immediate\_bitcode)

    print("immediate\_bitcode--",immediate\_bitcode)

    bitcode=immediate\_bitcode+s\_address+func3+des\_address+opcode

    print("BITCODE value--",bitcode)

    hex=binary\_hex\_32\_bit(bitcode)

    print("32 B hexadecimal Code---","0x",hex)

  #TYPE 2 INSTRUCTION

  if encoder\_dic[code\_arr[0]][0]=="I" and encoder\_dic[code\_arr[0]][1]=="0010011":

    print("Instruction Type--",encoder\_dic[code\_arr[0]][0])

    print(code\_arr)

    opcode=encoder\_dic[code\_arr[0]][1]

    des\_address=register\_dic[code\_arr[1]]

    func3=encoder\_dic[code\_arr[0]][2]

    func7=encoder\_dic[code\_arr[0]][3]

    s\_address=register\_dic[code\_arr[-2]]

    print("opcode--",opcode)

    print("destination code--",des\_address)

    print("func3 code--",func3)

    print("source1 code--",s\_address)

    immediate=int(code\_arr[-1]) #uses 2's complement system

    immediate\_bitcode=zero\_padding(bin(abs(immediate))[2:],12) #  positive number's 2's complement calculation

    # negative number's 2's complement calculation

    if immediate<0:

      immediate\_bitcode=two\_complement(immediate\_bitcode)

    print("immediate\_bitcode--",immediate\_bitcode)

    if(code\_arr[0]=="SRAI"):

              immediate\_bitcode=immediate\_bitcode[0]+"1"+immediate\_bitcode[2:] #confirm with sir

    bitcode=immediate\_bitcode+s\_address+func3+des\_address+opcode

    print("BITCODE value--",bitcode)

    hex=binary\_hex\_32\_bit(bitcode)

    print("32 B hexadecimal Code---","0x",hex)

  #TYPE 3 INSTRUCTION

  if encoder\_dic[code\_arr[0]][0]=="I" and encoder\_dic[code\_arr[0]][1]=="1100111":

    last\_arr=code\_arr[-1].split('(')

    code\_arr=code\_arr[:-1]+[last\_arr[0],last\_arr[-1][0:-1]]

    print(code\_arr)

    opcode=encoder\_dic[code\_arr[0]][1]

    des\_address=register\_dic[code\_arr[1]]

    func3=encoder\_dic[code\_arr[0]][2]

    s\_address=register\_dic[code\_arr[3]]

    print("opcode--",opcode)

    print("destination address--",des\_address)

    print("func3 code--",func3)

    print("source address--",s\_address)

    immediate=int(code\_arr[-2]) #uses 2's complement system

    immediate\_bitcode=immediate\_bitcode=zero\_padding(bin(abs(immediate))[2:],12) #  positive number's 2's complement calculation

    # negative number's 2's complement calculation

    if immediate<0:

      immediate\_bitcode=two\_complement(immediate\_bitcode)

    print("immediate\_bitcode--",immediate\_bitcode)

    bitcode=immediate\_bitcode+s\_address+func3+des\_address+opcode

    print("BITCODE value--",bitcode)

    hex=binary\_hex\_32\_bit(bitcode)

    print("32 B hexadecimal Code---","0x",hex)

  # U tYPE INSTRUCTION

  if encoder\_dic[code\_arr[0]][0]=="U":

    print("opcode--",encoder\_dic[code\_arr[0]][1])

    print("destination address--",register\_dic[code\_arr[1]])

    immediate=int(code\_arr[-1]) #uses 2's complement system

    immediate\_bitcode=zero\_padding(bin(abs(immediate))[2:],20) # positive number's 2's complement calculation

    # negative number's 2's complement calculation

    if immediate<0:

      immediate\_bitcode=two\_complement(immediate\_bitcode)

    print("immediate\_bitcode--",immediate\_bitcode)

    bitcode=immediate\_bitcode+register\_dic[code\_arr[1]]+encoder\_dic[code\_arr[0]][1]

    print("BITCODE value--",bitcode)

    hex=binary\_hex\_32\_bit(bitcode)

    print("32 B hexadecimal Code---","0x",hex)

  # UJ TYPE INSTRUCTION

  if encoder\_dic[code\_arr[0]][0]=="UJ":

        print("opcode--",encoder\_dic[code\_arr[0]][1])

        print("destination address--",register\_dic[code\_arr[1]])

        immediate=int(code\_arr[2]) #uses 2's complement system

        immediate\_bitcode=zero\_padding(bin(abs(immediate))[2:],21) # positive number's 2's complement calculation

        # negative number's 2's complement calculation

        if immediate<0:

          immediate\_bitcode=two\_complement(immediate\_bitcode)

        print("immediate\_bitcode--",immediate\_bitcode)

        bitcode=immediate\_bitcode[0]+immediate\_bitcode[10:20]+immediate\_bitcode[9]+immediate\_bitcode[1:9]+register\_dic[code\_arr[1]]+encoder\_dic[code\_arr[0]][1]

        print("BITCODE value--",bitcode)

        hex=binary\_hex\_32\_bit(bitcode)

        print("32 B hexadecimal Code---","0x",hex)

  # S TYPE INSTRUCTION

  if encoder\_dic[code\_arr[0]][0]=="S":

        last\_arr=code\_arr[-1].split('(')

        code\_arr=code\_arr[:-1]+[last\_arr[0],last\_arr[-1][0:-1]]

        print(code\_arr)

        opcode=encoder\_dic[code\_arr[0]][1]

        sou1=register\_dic[code\_arr[3]]

        sou2=register\_dic[code\_arr[1]]

        func3=encoder\_dic[code\_arr[0]][2]

        print("opcode--",opcode)

        print("source1 address--",sou1)

        print("source2 address--",sou2)

        print("func3 code--",func3)

        immediate=int(code\_arr[-2]) #uses 2's complement system

        immediate\_bitcode=zero\_padding(bin(abs(immediate))[2:],12) # positive number's 2's complement calculation

        # negative number's 2's complement calculation

        if immediate<0:

          immediate\_bitcode=two\_complement(immediate\_bitcode)

        print("immediate\_bitcode--",immediate\_bitcode)

        bitcode=immediate\_bitcode[0:7]+sou2+sou1+func3+immediate\_bitcode[7:12]+opcode

        print("BITCODE value--",bitcode)

        hex=binary\_hex\_32\_bit(bitcode)

        print("32 B hexadecimal Code---","0x",hex)

  #SB TYPE INSTRUCTION

  if encoder\_dic[code\_arr[0]][0]=="SB":

    print("Instruction Type--",encoder\_dic[code\_arr[0]][0])

    print(code\_arr)

    opcode=encoder\_dic[code\_arr[0]][1] #need to confirm opcode

    sou1\_address=register\_dic[code\_arr[1]]

    func3=encoder\_dic[code\_arr[0]][2]

    func7=encoder\_dic[code\_arr[0]][3]

    sou2\_address=register\_dic[code\_arr[-2]]

    print("opcode--",opcode)

    print("source1 code--",sou1\_address)

    print("func3 code--",func3)

    print("source2 code--",sou2\_address)

    immediate=int(code\_arr[-1]) #uses 2's complement system

    immediate\_bitcode=zero\_padding(bin(abs(immediate))[2:],13) #  positive number's 2's complement calculation

    # negative number's 2's complement calculation

    if immediate<0:

      immediate\_bitcode=two\_complement(immediate\_bitcode)

    print("immediate\_bitcode--",immediate\_bitcode)

    bitcode=immediate\_bitcode[0]+immediate\_bitcode[2:8]+sou2\_address+sou1\_address+func3+immediate\_bitcode[8:12]+immediate\_bitcode[1]+opcode

    print("BITCODE value--",bitcode)

    hex=binary\_hex\_32\_bit(bitcode)

    print("32 B hexadecimal Code---","0x",hex)

code=input("PLEASE ENTER ASSEMBLY CODE PNEMONICS-- \n")

code=code.upper();

code\_arr=code.split(',')

code\_arr=code\_arr[0].split(' ')+code\_arr[1:]

print("Details of your pnemonics are as follows---")

risc5assembler(code\_arr)

**Shubham kumar jha**

**2411ee22**