MIDDLE EAST TECHNICAL UNIVERSITY

NORTHERN CYPRUS CAMPUS

CNG 331 - Computer Organization

Term Project: Assembler Design

BMassembler

Name Surname: Berkay Aktunç

Student ID: 1862499

Name Surname: Muhammed Didin

Student ID: 2243384

Introduction to problem:

When we approach the physical level in hardware, we need binary codes, but if we get closer we need hexadecimal codes. Our main goal here is to write an assembler that will convert human-understandable instructions into binary and hexadecimal codes that the computer can understand.

Assembler design choices, which includes comments on which language and compiler (if any) you used to implement and test your assembler (and why).

We used version 3.8 of Python language. Because Python is a scripting and modern language which supports additional features such as powerful debug engine, data structures that will work for us, and variety of built-in functions. As a compiler, we used Visual Studio Code.

We wrote the code by dividing it into functions that will be useful later. In this way, while avoiding the trouble of typing the same code twice, it became easier for us to add new sections. In addition to these, we handled possible errors by checking the user inputs in the menu and main sections.

Conclusion

We achieved our goal, our program works well converting codes and calculating. In this assignment; we learnt how MIPS works and as computer engineer candidates, we took a closer look at the hardware part.

```
#--------------------------------------------------------------------------------
---------------------------------
#                          Predefined Items Starts

rTypes = ["add","move","slt","jr","sll"]
iTypes = ["addi","slti","lw","sw","beq","bne"]
jTypes = ["j","jal"]

labels = {}
function = {'jr':'8', 'jal':'9', 'add':'100000', 'slt':'2a', 'sll':'00', 'srl'
:'02'}
opcodes = {'beq':'4', 'bne':'5', 'jr':'0', 'jal':'0', 'j':'0', 'add':'0', 'slt
':'0',
            'sll':'0', 'srl':'0', 'lw':'23','sw':'2b','addi':'8','slti':'a','m
ove':'0'
        }


register = {'zero':'00000','at':'00001','v0':'00010','v1':'00011','a0':'00100'
,'a1':'00101','a2':'00110','a3':'00111',
            't0':'01000','t1':'01001','t2':'01010','t3':'01011','t4':'01100','
t5':'01101','t6':'01110','t7':'01111',
            's0':'10000','s1':'10001','s2':'10010','s3':'10011','s4':'10100','
s5':'10101','s6':'10110','s7':'10111',
            't8':'11000','t9':'11001','k0':'11010','k1':'11011','gp':'11100','
sp':'11101','fp':'11110','ra':'11111'
            }

#                          Predefined Items Ends
#--------------------------------------------------------------------------------
---------------------------------
#                          Math Convert Parts Starts

def hexToDec(s):
    return int(s, 16)


def binToHex(inputt,places):
    ret = hex(int(inputt, 2))[2:].zfill(places).replace('0x','')
    return ret


def hexToBin(inputt,bits):
    ret = bin(int(inputt, 16))[2:].zfill(bits)
    return ret


def decToBin(inputt,bits):
    inputt = int(inputt)
```

```
    if inputt < 0:
        return decToTwosComplment(inputt,bits)
    else:
        return bin(int(str(inputt),10))[2:].zfill(bits)


def decToTwosComplment(inputt, bits):
    intIn = int(inputt)
    if intIn>=0:
        val = decToBin(intIn, bits)
    else:
        msb = -2**bits
        rest = msb-intIn
        val = decToBin(rest,bits-1)
        val = val.replace('b','')
    return val

#                       Math Convert Parts Ends
#-------------------------------------------------------------------------------
---------------------------------
#-------------------------------------------------------------------------------
---------------------------------
#                       Type Individual Funcs Starts

def addiType(line):
    label = isLabel(line)

    op = hexToBin(opcodes[line[label]],6)
    rs = register[line[label+2]].zfill(5)
    rt = register[line[label+1]].zfill(5)
    number = decToBin(line[label+3],16)
    print(op,rs,rt,number)

    binary = op + rs + rt + number
    hexcode = binToHex(binary,8)

    return hexcode


def sltiType(line):
    label = isLabel(line)

    op = hexToBin(opcodes[line[label]],6)
    rs = register[line[label+2]]
    rt = register[line[label+1]]
    number = decToBin(line[label+3],16)

    binary = op + rs + rt + number
    hexcode = binToHex(binary,8)
```

```
    return hexcode


def lwType(line):
    label = isLabel(line)

    op = hexToBin(opcodes[line[label]],6)
    rt = register[line[label+1]]
    rs = register[line[label+3]]
    number = decToBin(line[label+2],16)

    binary = op + rs + rt + number
    hexcode = binToHex(binary,8)

    return hexcode


def swType(line):
    label = isLabel(line)

    op = hexToBin(opcodes[line[label]],6)
    rs = register[line[label+3]]
    rt = register[line[label+1]]
    number = decToBin(line[label+2],16)

    binary = op + rs + rt + number
    hexcode = binToHex(binary,8)

    return hexcode


def beqType(line):
    label = isLabel(line)

    op = hexToBin(opcodes[line[label]],6)
    rs = register[line[label+2]].zfill(5)
    rt = register[line[label+1]].zfill(5)

    address = hexToBin( labels[ line[label+3]+":"][-4:], 16)

    binary = op + rs + rt + address
    hexcode = binToHex(binary,8)

    return hexcode


def bneType(line):
    label = isLabel(line)
```

```
    op = hexToBin(opcodes[line[label]],6)
    rs = register[line[label+2]].zfill(5)
    rt = register[line[label+1]].zfill(5)
    address = hexToBin( labels[ line[label+3]+":"][-4:], 16)

    binary = op + rs + rt + address
    hexcode = binToHex(binary,8)

    return hexcode


def jumpType(line):
    label = isLabel(line)
    op = opcodes[line[label]]
    temp = "0x" + labels[line[label+1]+":"][-4:]
    number = hexToBin(temp,4)
    hexcode = binToHex(op+number,8)
    return hexcode


def jalType(line):
    label = isLabel(line)
    op = opcodes[line[label]]
    temp = "0x" + labels[line[label+1]+":"][-4:]
    number = hexToBin(temp,4)
    hexcode = binToHex(op+number,8)
    return hexcode


def RaddType(line):
    label = isLabel(line)

    op = opcodes[line[label]].zfill(6)
    rs = register[line[label+2]]
    rt = register[line[label+3]]
    rd = register[line[label+1]]
    shamt = '00000'
    funct = function[line[label]]

    binary = op + rs + rt + rd + shamt + funct
    hexcode = binToHex(binary,8)

    return hexcode


def RsltType(line):
    label = isLabel(line)
```

```
    op = opcodes[line[label]].zfill(6)
    rs = register[line[label+2]]
    rt = register[line[label+3]]
    rd = register[line[label+1]]
    funct = hexToBin(function[line[label]],11)

    binary = op + rs + rt + rd + funct
    hexcode = binToHex(binary,8)

    return hexcode


def RjrType(line):
    label = isLabel(line)

    op = opcodes[line[label]].zfill(6)
    rs = register[line[label+1]]
    funct = hexToBin(function[line[label]],20)

    binary = op + rs + funct
    hexcode = binToHex(binary,8)

    return hexcode


def RsllType(line):
    label = isLabel(line)

    op = opcodes[line[label]].zfill(6)
    rt = register[line[label+2]]
    rd = register[line[label+1]]
    shamt = decToBin(line[label+3],5)
    funct = hexToBin(function[line[label]],6)

    binary = op + rt + rd + shamt + funct
    hexcode = binToHex(binary,8)

    return hexcode


def RsrlType(line):
    label = isLabel(line)

    op = opcodes[line[label]].zfill(6)
    rt = register[line[label+2]]
    rd = register[line[label+1]]
    shamt = decToBin(line[label+3],5)
    funct = hexToBin(function[line[label]],6)
```

```
    binary = op + rt + rd + shamt + funct

    hexcode = binToHex(binary,8)

    return hexcode


def RmoveType(line):

    label = isLabel(line)

    op = opcodes[line[0]].zfill(6)
    rs = register[line[label+2]]
    rd = register[line[label+1]]
    shamt = '00000'
    funct = function['add']

    binary = op + rs + rd + shamt + funct
    hexcode = binToHex(binary,8)

    return hexcode

#                         Type Individual Funcs Ends
#---------------------------------------------------------------------------------
----------------------------------
#---------------------------------------------------------------------------------
----------------------------------
#                         Type Flow Decider Starts

def convertJType(line):
    label = isLabel(line)

    if line[label] == 'j':
        return jumpType(line)
    elif line[label] == "jal":
        return jalType(line)
    else:
        return "Unknown op!"


def convertIType(line):
    label = isLabel(line)

    if line[label] == 'addi':
        return addiType(line)
    elif line[label] == 'slti':
        return sltiType(line)
    elif line[label] == 'lw':
        return lwType(line)
```

```
    elif line[label] == 'sw':
        return swType(line)
    elif line[label] == 'beq':
        return beqType(line)
    elif line[label] == 'bne':
        return bneType(line)
    else:
        return "Unknown op!"


def convertRType(line):
    label = isLabel(line)

    if line[label] == "add":
        return RaddType(line)
    elif line[label] == "move":
        return RmoveType(line)
    elif line[label] == "slt":
        return RsltType(line)
    elif line[label] == "jr":
        return RjrType(line)
    elif line[label] == "sll":
        return RsllType(line)
    elif line[label] == "srl":
        return RsrlType(line)
    else:
        return "Unknown op!"


def calculateHex(userInput):
    typee = getType(userInput)
    if typee == 'R':
        return "0x" + str(convertRType(userInput))
    elif typee == 'I':
        return "0x" + str(convertIType(userInput))
    elif typee == 'J':
        return "0x" + str(convertJType(userInput))
    else:
        return "Unknown op!"


#                         Type Flow Decider Ends
#--------------------------------------------------------------------------
---------------------------------
#--------------------------------------------------------------------------
---------------------------------
#                         Helper Funcs Starts

def isLabel(line):
```

```
    if( line[0][-1] == ':'):
        return 1
    return 0


def getType(line):
    label = isLabel(line)

    if line[label] in rTypes:   return 'R'
    elif line[label] in iTypes: return 'I'
    elif line[label] in jTypes: return 'J'


def lineParser(line):
    parsed = line.replace(':',': ')
    parsed = parsed.replace(',','')
    parsed = parsed.replace('(',' ')
    parsed = parsed.replace(')',' ')
    parsed = parsed.replace('$','')
    parsed = parsed.lower()
    parsed = parsed.split()

    return parsed


def fillFile(line):
    with open('output.obj','a') as file:
        file.write( str(line) + "\n")


def fillLabels(key,value):
    if key not in labels:
        labels.update({key:value})


def incrementPC(pc, line):
    label = isLabel(line)

    if label == 0:
        temp = hexToDec(pc[2:])
        temp = temp + 4
        temp = decToBin(temp,8)
        temp = binToHex(temp,8)

        return "0x" + temp
    elif line[0] in labels:
        return labels[line[0]]
    elif line[0] not in labels:
        fillLabels(line[0],pc)
```

```
        temp = hexToDec(pc[2:])
        temp = temp + 4
        temp = decToBin(temp,8)
        temp = binToHex(temp,8)

        return "0x" + temp


def firstTour(filename):
    code = ""

    pc = '0x80001000'
    with open( filename,'r') as file:
        allLines = file.readlines()
        for line in allLines:

            if line[-2] == ' ':
                code = lineParser(line[0:-2])
            elif line[-1] == '\n':
                code = lineParser(line[0:-1])
            else:
                code = lineParser(line)

            if isLabel(line):
                fillLabels(code,pc)
            pc = incrementPC(pc,code)




#                        Helper Funcs Ends
#---------------------------------------------------------------------------
---------------------------------
#---------------------------------------------------------------------------
---------------------------------
#                        Main Flow Decider Stars
"""
The batch mode reads a source file with extension .src, assembles to hexadecim
al,
and outputs the result to an object code file with extension .obj.
"""
def batchMode():
    code = ""
    filename = input("Please enter filename: ")
    while(filename.endswith(".src") == 0):
        print("Please enter a .src file!")
        filename = input("Please enter filename: ")
```

```
    try:
        firstTour(filename)
        with open( filename,'r') as file:
            allLines = file.readlines()
            pc = '0x80001000'

            for line in allLines:

                if line[-2] == ' ':
                    code = lineParser(line[0:-2])
                elif line[-1] == '\n':
                    code = lineParser(line[0:-1])
                else:
                    code = lineParser(line)

                codeToPrint = calculateHex(code)

                fillFile(codeToPrint)

                pc = incrementPC(pc,code)
    except FileNotFoundError:
        print("File NOT Found!\nReturning to main menu\n\n\n")
        return
    except:
        print("Anything else happened!")


"""
The interactive mode reads an instruction from command line,
assembles it to hexadecimal (converting from pseudo-instruction as necessary)
outputs the result to the screen.
The batch mode reads a source file with extension .src,
assembles to hexadecimal
outputs the result to an object code file with extension .obj.
"""
def interactiveMode():
    declineList = ['move','jr','jal','bne','beq']

    userInput = input("Enter the instruction or alternatively\nenter 0 to retu
rn to main menu: ")
    # print("you entered", userInput)
    if(userInput == 0):
        return
    else:
        userInput = lineParser(userInput)
        if isLabel(userInput) == 1 or userInput[0] in declineList:
            print("Cannot convert the input!")
        else:
            print( calculateHex(userInput) )
```

```
#                          Main Flow Decider Ends
#-------------------------------------------------------------------------------
---------------------------------
#-------------------------------------------------------------------------------
---------------------------------
#                          Main of the App Here

def menu():
    print("1) Interactive Mode")
    print("2) Batche Mode")
    print("3) Exit")
    selection = (input("Enter your selection: "))
    return selection


def main():
    selection = 1
    while(selection != '3'):
        selection =  menu()
        if selection == '1':
            interactiveMode()
        elif selection == '2':
            batchMode()
            selection =  menu()
        elif selection == '3':
            print("Good Bye!")
        else:
            print("Wrong input!")


if __name__ == "__main__":
    main()
```

input.src

```
swap:sll $t1, $a1, 2
lw $t0, 0($t1)
lw $t2, 4($t1)
sw $t2, 0($t1)
sw $t0, 4($t1)
sort:addi $sp, $sp, -20
sw $ra, 16($sp)
sw $s3, 12($sp)
sw $s2, 8($sp)
sw $s1, 4($sp)
sw $s0, 0($sp)
move $s2, $a0
move $s3, $a1
move $s0, $zero
for1tst:slt $t0, $s0, $s3
beq $t0, $zero, exit1
addi $s1, $s0, -1
for2tst:slti $t0, $s1, 0
bne $t0, $zero, exit2
sll $t1, $s1, 2
add $t2, $s2, $t1
lw $t3, 0($t2)
lw $t4, 4($t2)
slt $t0, $t4, $t3
beq $t0, $zero, exit2
move $a0, $s2
move $a1, $s1
jal sort
addi $s1, $s1, -1
j for2tst
exit2:addi $s0, $s0, 1
j for1tst
exit1:lw $s0, 0($sp)
lw $s1, 4($sp)
lw $s2, 8($sp)
lw $s3, 12($sp)
lw $ra, 16($sp)
addi $sp, $sp, 20
jr $ra
```

output.obj

```
0x00054880
0x8d280000
0x8d2a0004
0xad2a0000
0xad280004
0x11deffec
0xafbf0010
0xafb3000c
0xafb20008
0xafb10004
0xafb00000
0x00049020
0x00059820
0x00008020
0x0213402a
0x10081080
0x1108ffff
0x2a280000
0x14081078
0x00114880
0x02495020
0x8d4b0000
0x8d4c0004
0x018b402a
0x10081078
0x00122020
0x00112820
0x00001014
0x1118ffff
0x00001044
0x22100001
0x00001038
0x8fb00000
0x8fb10004
0x8fb20008
0x8fb3000c
0x8fbf0010
0x23bd0014
0x01f00008
```

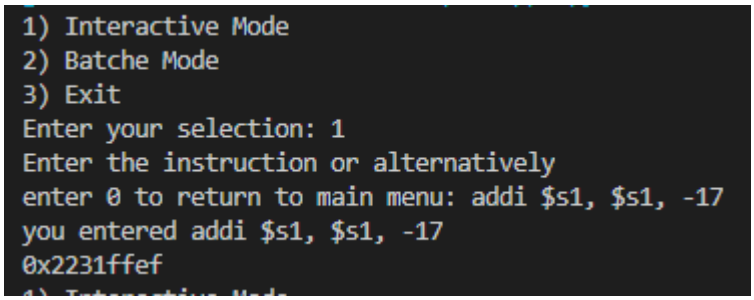

Figure 1: addi $s1, $s1, -17 in interactive mode