from array import \*  
import re  
  
# register constants and their respective binary codes  
Zero = "00"  
s0 = "01"  
s1 = "10"  
s2 = "11"  
  
  
# register constants and their respective hex codes  
Zerohex = "0"  
s0hex = "1"  
s1hex = "2"  
s2hex = "3"  
  
  
# opcode constants and their respective binary codes  
  
Add = "0000"  
Sub = "0001"  
Addi = "0010"  
And = " 0011"  
OR= "0100"  
Nor ="0101"  
beq = "0110"  
slti= "1000"  
Jump = "1001"  
lw ="1010"  
sw ="1011"  
#In = "1100 "  
#Out = "1101 "  
  
  
# opcode constants and their respective hexidecimal codes  
Addhex = "O"  
Subhex = "1"  
Addihex = "2"  
Andhex = " 3"  
ORhex= "4"  
#Norhex ="5"  
beqhex = "6"  
sltihex= "8"  
Jumphex = "9"  
lwhex ="a"  
swhex ="b"  
#Inhex = "c "  
#Outhex = "d"  
  
  
# write to output txt file  
def write\_to\_output(string):  
 f = open("output.txt", "a")  
 f.write(string + "\n")  
 f.close()  
  
  
# write to outputhex txt file  
def write\_to\_outputhex(string):  
 g = open("outputhex.txt", "a")  
 g.write(string + "\n")  
 g.close()  
  
  
# error notification  
def error\_register():  
 write\_to\_output("The registers trying to be used don't exist")  
 write\_to\_outputhex("The registers trying to be used don't exist")  
  
  
def error\_wordsperline():  
 write\_to\_output("This line has too many or too less words")  
 write\_to\_outputhex("This line has too many or too less words")  
  
  
def error\_ins():  
 write\_to\_output("The instruction in this line doesn't exist ")  
 write\_to\_outputhex("The instruction in this line doesn't exist in our design")  
  
  
def error\_num\_outofrange():  
 write\_to\_output("The immediate number supplied is out of range")  
 write\_to\_outputhex("The immediate number supplied is out of range")  
  
  
# Takes string, recognizes which opcode it is & assigns it's binary code  
def assign\_Op(op):  
 if (op == "add"):  
 return Add  
 elif (op == "sub"):  
 return Sub  
 elif (op == "addi"):  
 return Addi  
 elif (op == "and"):  
 return And  
 elif (op == "or"):  
 return OR  
 elif (op == "beq"):  
 return beq  
 elif (op == "slti"):  
 return slti  
 elif (op == "lw"):  
 return lw  
 elif (op == "sw"):  
 return sw  
 elif (op == "j"):  
 return Jump  
 else:  
 error\_ins()  
  
  
# Takes string, recognizes which opcode it is & assigns it's hex code  
def assign\_hexOp(op):  
 if (op == "add"):  
 return Addhex  
 elif (op == "sub"):  
 return Subhex  
 elif (op == "addi"):  
 return Addihex  
 elif (op == "and"):  
 return Andhex  
 elif (op == "or"):  
 return ORhex  
 elif (op == "beq"):  
 return beqhex  
 elif (op == "lw"):  
 return lwhex  
 elif (op == "sw"):  
 return swhex  
 elif (op == "j"):  
 return Jumphex  
 else:  
 error\_ins()  
  
  
# Takes string, recognizes which register it is & assigns it's binary code  
def assign\_reg(Reg):  
 if (Reg == "$Zero"):  
 return Zero  
 elif (Reg == "$s0"):  
 return s0  
 elif (Reg == "$s1"):  
 return s1  
 elif (Reg == "$s2"):  
 return s2  
 else:  
 error\_register()  
 return False  
  
  
# Takes string, recognizes which register it is & assigns it's hex code  
def assign\_reghex(Reg):  
 if (Reg == "$Zero"):  
 return Zerohex  
 elif (Reg == "$s0"):  
 return s0hex  
 elif (Reg == "$s1"):  
 return s1hex  
 elif (Reg == "$s2"):  
 return s2hex  
 else:  
 error\_register()  
 return False  
  
  
  
  
# converts decimal string to a binary in 2 bit for Itype within range and format  
def binary\_2bit(op\_check, strformat\_num):  
 num = int(strformat\_num)  
  
 if (op\_check == "beq" and num > 0 and num < 4):  
 r = '{0:02b}'.format(int(num))  
 return r  
  
 else:  
 error\_num\_outofrange()  
 return False  
  
  
# converts decimal string to a binary in 6 bit for j type within range and format  
def binary\_6bit(strformat\_num):  
 num = int(strformat\_num)  
 if (num > 0 and num < 64):  
 r = '{0:06b}'.format(int(num))  
 return r  
  
 else:  
 error\_num\_outofrange()  
 return 0  
  
  
def Rtype(op\_check, rs\_check, rt\_check, rd\_check):  
 op = assign\_Op(op\_check)  
 rs = assign\_reg(rs\_check)  
 rt = assign\_reg(rt\_check)  
 rd = assign\_reg(rd\_check)  
  
 if (rs != False and rt != False and rd != False):  
 binary\_str = op + rs + rt + rd  
  
 # writes machine code as op, rs, rt, rd  
 write\_to\_output(binary\_str)  
  
 ophex = assign\_hexOp(op\_check)  
 rshex = assign\_reghex(rs\_check)  
 rthex = assign\_reghex(rt\_check)  
 rdhex = assign\_reghex(rd\_check)  
 if (rshex != False and rthex != False and rdhex != False):  
 hex\_str = ophex + rshex + rthex + rdhex  
 write\_to\_outputhex(hex\_str)  
  
  
def Itype(op\_check, rt\_check, rs\_check, imm\_check):  
 op = assign\_Op(op\_check)  
 rt = assign\_reg(rt\_check)  
 rs = assign\_reg(rs\_check)  
 imm = binary\_2bit(op\_check, imm\_check)  
  
 if (imm > False and rs != False and rt != False):  
 binary\_str = op + rs + rt + imm  
  
 # writes machine code as op, rs, rt, imm  
 write\_to\_output(binary\_str)  
  
 ophex = assign\_hexOp(op\_check)  
 rshex = assign\_reghex(rs\_check)  
 rthex = assign\_reghex(rt\_check)  
 if (imm > False and rshex != False and rthex != False):  
 immhex = '{:x}'.format(int(imm, 2))  
 hex\_str = ophex + rshex + rthex + immhex  
 write\_to\_outputhex(hex\_str)  
  
  
def Jtype(op\_check, imm\_check):  
 op = assign\_Op(op\_check)  
 imm = binary\_2bit(imm\_check)  
  
 if (imm > False):  
 binary\_str = op + imm  
 # writes machine code as op, imm  
 write\_to\_output(binary\_str)  
  
 ophex = assign\_hexOp(op\_check)  
 if (imm > False):  
 immhex = '{:x}'.format(int(imm, 2))  
 hex\_str = ophex + immhex + "00"  
 write\_to\_outputhex(hex\_str)  
  
  
  
def wtype(op\_check, r1st\_check, imm\_check, r2nd\_check):  
 patn = re.sub(r"[\([{})\]]", "", r2nd\_check)  
  
 op = assign\_Op(op\_check)  
 rt = assign\_reg(r1st\_check)  
 imm = binary\_2bit(op\_check, imm\_check)  
 rs = assign\_reg(patn)  
  
 if (imm > False and rt != False and rs != False):  
 binary\_str = op + rs + rt + imm  
  
 # writes machine code as op, rt, rs, imm  
 write\_to\_output(binary\_str)  
  
 ophex = assign\_hexOp(op\_check)  
 rthex = assign\_reghex(r1st\_check)  
 rshex = assign\_reghex(patn)  
 if (imm > False and rthex != False and rshex != False):  
 immhex = '{:x}'.format(int(imm, 2))  
 hex\_str = ophex + rshex + rthex + immhex  
 write\_to\_outputhex(hex\_str)  
  
  
def Main():  
 f = open("input.txt", "r")  
 line\_list = f.readlines()  
  
 for i in range(0, len(line\_list)):  
 word\_list = line\_list[i].split()  
  
 if (len(word\_list) == 4):  
  
 if (word\_list[0] == "add" or word\_list[0] == "sub"):  
 Rtype(word\_list[0], word\_list[1], word\_list[2], word\_list[3])  
  
 elif (word\_list[0] == "addi" or word\_list[0] == "subi" or word\_list[0] == "beq"):  
 Itype(word\_list[0], word\_list[1], word\_list[2], word\_list[3])  
  
 elif (word\_list[0] == "lw" or word\_list[0] == "sw"):  
 wtype(word\_list[0], word\_list[1], word\_list[2], word\_list[3])  
  
 else:  
 error\_ins()  
 continue  
  
  
 elif (len(word\_list) == 2):  
  
 if (word\_list[0] == "j"):  
 Jtype(word\_list[0], word\_list[1])  
  
  
  
 else:  
 error\_ins()  
 continue  
  
  
 elif (len(word\_list) < 2 or len(word\_list) == 3 or len(word\_list) > 4):  
 error\_wordsperline()  
 continue  
  
 f.close()