实验五

```
import queue
import threading
import time

class Cpu:
    def __init__(self):
        self.memory = ["0"] * 1000   # 内存
        self.queue = queue.Queue()  # 指令队列
        self.stack = [] * 100       # 堆栈
        self.address_bus = {}       # 地址总线
        self.data_bus = {}          # 数据总线
        self.control_bus = {}       # 控制总线
        self.syn = 1                # 同步eu biu
        # 专用寄存器
        self.special_registers = {
            "DS": 1,  # 数据段 0 415  地址为DS*16 + AX  (AX <= 255)
            "CS": 11,   # 代码段  416 开始到 1000
            "SS": 0,   # 堆栈段 单独设置
            "ES": 0,   # 附加段
            "IP": 0    # 指令寄存器
        }
        # 通用寄存器
        self.general_registers = {
            "AX": 0,
            "AH": 0,
            "AL": 0,
            "BX": 0,
            "BH": 0,
            "BL": 0,
            "CX": 0,
            "CH": 0,
            "CL": 0,
            "DX": 0,
            "DH": 0,
            "DL": 0,
            "SP": 0,  # 堆栈指针
            "BP": 0,  # 存取堆栈指针
            "DI": 0,  # 目的变址寄存器
            "SI": 0   # 源变址寄存器
        }
        # 标志寄存器
        self.flags = {
            "CF": 0,  # 进位标志位
            "PF": 0,  # 奇偶标志位
            "AF": 0,  # 辅助进位标志位
            "ZF": 0,  # 零标志位
            "SF": 0,  # 符号标志位
            "OF": 0,  # 溢出标志位
            "IF": 1   # 中断标志位
        }
        # 指令集
        self.instructions = {
            'MOV'  : self.mov,
            'PUSH' : self.push,
            'POP'  : self.pop,
            'XCHG' : self.xchg,
            'ADD'  : self.add,
            'SUB'  : self.sub,
            'ADC'  : self.adc,
            'SBB'  : self.sbb,
            'INC'  : self.inc,
            'DEC'  : self.dec,
            'MUL'  : self.mul,
            'IMUL' : self.imul,
            'DIV'  : self.div,
            'IDIV' : self.idiv,
            'AND'  : self.And,
            'OR'   : self.Or,
            'XOR'  : self.Xor,
            'NOT'  : self.Not,
            'TEST' : self.Test,
            'MOVSB': self.movsb,
            'MOVSW': self.movsw,
            'CMPSB': self.cmpsb,
            'CMPSW': self.cmpsw,
            'SCASB': self.scasb,
            'SCASW': self.scasw,
            'LODSB': self.lodsb,
            'LODSW': self.lodsw,
            'STOSB': self.stosb,
            'STOSW': self.stosw,
            'NOP'  : self.nop,
            'CLC'  : self.clc,
            'STC'  : self.stc,
            'CMC'  : self.cmc,
            'CLCD' : self.cld,
            'STD'  : self.std,
            'CLI'  : self.cli,
            'STI'  : self.sti,
            'HLT'  : self.hlt,
            # 'JMP'  : self.jmp,
            # 'CALL' : self.call,
            # 'RET'  : self.ret,
            'INT'  : self.Int,
            # 'IRET' : self.iret,
            # 'LOOP' : self.Loop,
            # 'LOOPZ': self.Loopz,
            # 'LOOPNZ': self.Loopnz,
        }

    # 获取指令
    def fetch(self):
        if len(self.memory) > self.special_registers["IP"] and
self.memory[self.special_registers["IP"]] != '0' and self.syn == 1:
            component = self.memory[self.special_registers["IP"]]
            print(f"正在从内存地址 {self.special_registers['IP']} 获取指令: {component}")
            parts = component.split(' ')
            op = parts[0]
            reg = parts[1] if len(parts) >= 2 else None
            val = parts[2] if len(parts) == 3 else None
            if op == 'JMP':
                self.jmp(reg)
            elif op == 'CALL':
                self.call(reg)
            elif op == 'RET':
                self.ret()
            elif op == 'IRET':
                self.iret()
            elif op == 'LOOP':
                self.Loop(reg)
            elif op == 'LOOPZ':
                self.Loopz(reg)
            elif op == 'LOOPNZ':
                self.Loopnz(reg)
            elif op == 'HTL':
                self.syn = 0
            else:
                self.special_registers["IP"] += 1
            return component
        else:
            exit(0)

    # 解码指令
    def decodes(self, part):
        if self.queue.qsize() != None:
            t_d1 = time.process_time()
            parts = part.split(' ')
            op = parts[0]
            reg = parts[1] if len(parts) >= 2 else None
            val = parts[2] if len(parts) == 3 else None
            t_d2 = time.process_time()
            if op in ['INC', 'DEC']:
                print(f"正在解码指令: {part} 为 操作码 {op}, 寄存器 AX  所用时间为{t_d2 - t_d1}")
            elif op in ['MUL', 'DIV']:
                print(f"正在解码指令: {part} 为 操作码 {op}, 寄存器 AX  所用时间为{t_d2 - t_d1}")
            else:
                print(f"正在解码指令: {part} 为 操作码 {op}, 寄存器 {reg}, 值 {val} 所用时间为{t_d2 - t_d1}")
            return op, reg, val

    # 执行指令
    def execute(self, operation, register, value):
        if operation in self.instructions:
            t_e1 = time.process_time()
            if operation == 'HLT':
                self.instructions[operation]()
            elif value is not None:
                if value.isdigit():
                    value = int(value)
                self.instructions[operation](register, value)
                print(f"寄存器 {register} 的新值为
{self.general_registers[register]}")
            elif register is not None:
                self.instructions[operation](register)
            else:
                self.instructions[operation]()
            t_e2 = time.process_time()
            print(f"正在执行指令 {operation}...所用时间为{t_e2 - t_e1}")

            self.print_registers()
            self.update_buses(operation, register, value)
            self.syn = 1

    # 地址解析
    def address_resolution(self, value):
        # 立即数寻址 100
        value = str(value)
        global address
        if value.isdigit():
            return int(value)
        # 寄存器寻址 AX
        elif value in self.general_registers:
            return self.general_registers[value]
        else:
            parts = value.split(':')
            if len(parts) == 2:
                sr = parts[0]
                g = parts[1]
                gr = g[1:len(g) - 1]
                # DS:[BX] 寄存器间接寻址
                if gr in self.general_registers:
                    address = self.special_registers[sr] * 16 +
self.general_registers[gr]
                    print(address)
                    return int(self.memory[address])
                # DS:[100] 直接寻址
                elif gr.isdigit():
                    address = int(self.special_registers[sr]) * 16 + int(gr)
                    return int(self.memory[address])
            elif len(parts) == 1:
                g = parts[0]
                gr = g[1:len(g) - 1]
                string = gr.split('+')
                if len(string) == 2:
                    s1 = string[0]
                    s2 = string[1]
                    if s2.isdigit():
                        # [SI+CNT] 相对寻址
                        if s1 == 'BP':
                            address = self.special_registers['ss'] * 16 +
self.general_registers['BP'] + int(s2)
                        elif s1 == 'BX':
                            address = self.special_registers['DS'] * 16 +
self.general_registers['BX'] + int(s2)
```

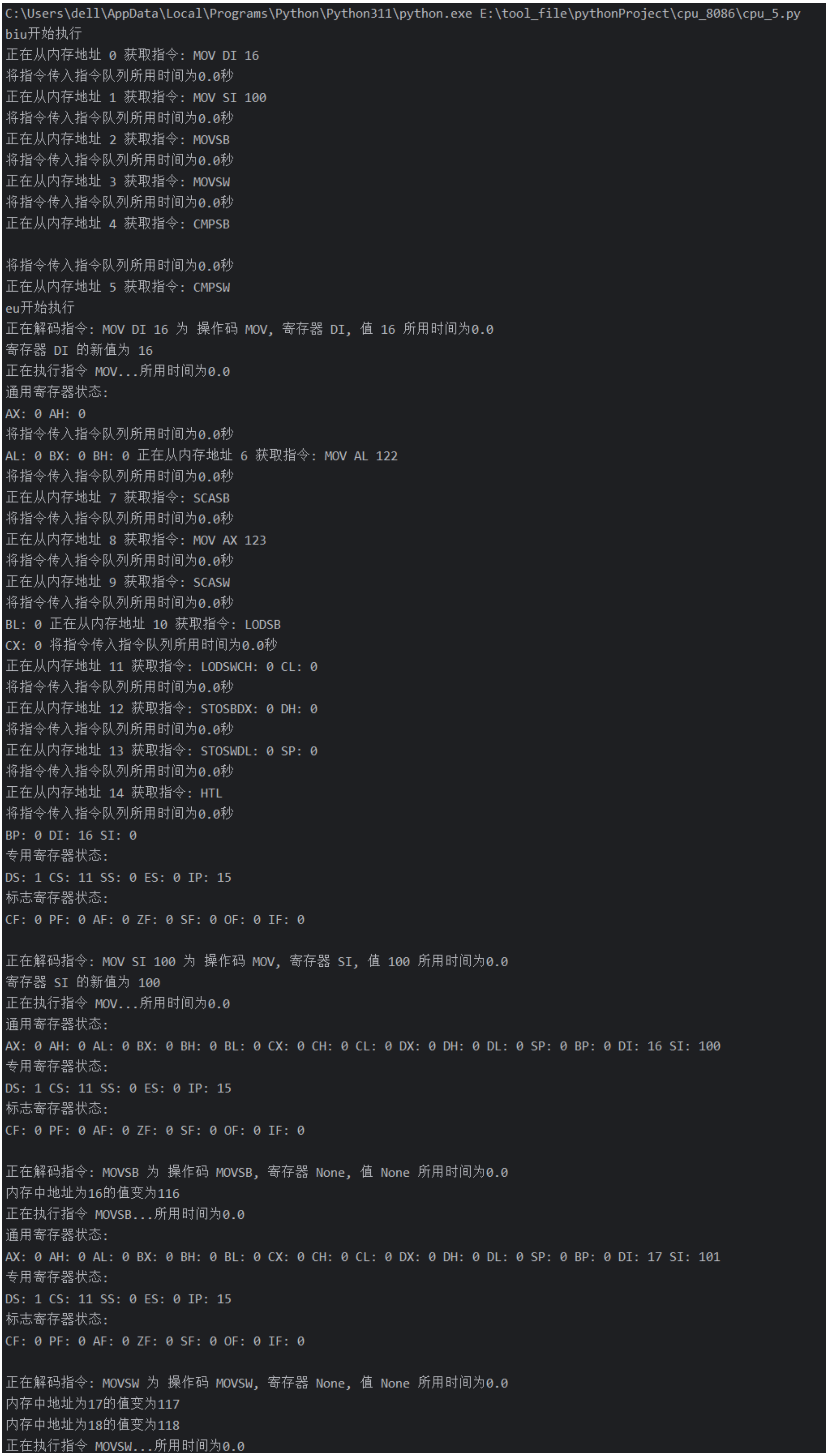
```
                        elif s1 == 'SI':
                            address = self.special_registers['DS'] * 16 +
self.general_registers['SI'] + int(s2)
                        elif s1 == 'DI':
                            address = self.special_registers['DS'] * 16 +
self.general_registers['DI'] + int(s2)
                        return int(self.memory[address])
                    else:
                        # [BX+SI] 基址变址寻址
                        if s1 == 'BX':
                            address = self.special_registers['DS'] * 16 +
self.general_registers['BX'] + self.general_registers[s2]
                        elif s1 == 'BP':
                            address = self.special_registers['SS'] * 16 +
self.general_registers['BP'] + self.general_registers[s2]
                        return int(self.memory[address])
                elif len(string) == 1:
                    # [BX][SI]  基址变址寻址
                    string = string[0]
                    s1 = string[1:3]
                    s2 = string[5:7]
                    if s1 == 'BX':
                        address = self.special_registers['DS'] * 16 +
self.general_registers['BX'] + self.general_registers[s2]
                    elif s1 == 'BP':
                        address = self.special_registers['SS'] * 16 +
self.general_registers['BP'] + self.general_registers[s2]
                    return int(self.memory[address])
                # [BX] 间接寻址
                parts = parts[0]
                s1 = parts[1:len(parts)-1]
                if s1 == 'BP':
                    address = self.special_registers['SS'] * 16 +
int(self.general_registers[s1])
                else:
                    address = self.special_registers['DS'] * 16 +
int(self.general_registers[s1])
                return int(self.memory[address])


    # 改变标志寄存器
    def change_flags(self, a, b, c, op, hl):
        if bin(c).replace('0b', '').count('1') % 2 == 0:
            self.flags['PF'] = 1
        else:
            self.flags['PF'] = 0
        if  c == 0 :
            self.flags['ZF'] = 1
        else:
            self.flags['ZF'] = 0
        if c < 0:
            self.flags['SF'] = 1
        else:
            self.flags['SF'] = 0
        if c>255 :
            self.flags['OF'] = 1
        else:
            self.flags['OF'] = 0
        if op == '+' :
            sa = a & 3
            sb = b & 3
            sc = sa+sb
            if sc & 4 == 1:
                self.flags['AF'] = 1
            else:
                self.flags['AF'] = 0
            ta = a & 127
            tb = b & 127
            tc = ta+tb
            if tc & 128 == 1:
                self.flags['CF'] = 1
            else:
                self.flags['CF'] = 0
        elif op == '-':
            sa = a & 3
            sb = b & 3
            if sa-sb < 0:
                self.flags['AF'] = 1
            else:
                self.flags['AF'] = 0
            if c < 0:
                self.flags['CF'] = 1
            else:
                self.flags['CF'] = 0

        elif op == '*' :
            if hl == 1:
                if c > 255:
                    self.flags['AF'] = 1
                    self.flags['CF'] = 1
                else:
                    self.flags['AF'] = 0
                    self.flags['CF'] = 0
            elif hl == 2:
                if self.general_registers['DX'] > 0:
                    self.flags['AF'] = 1
                    self.flags['CF'] = 1
                else:
                    self.flags['AF'] = 0
                    self.flags['CF'] = 0

    # 调整通用寄存器
    def adjust_register(self):
        self.general_registers['AH'] = self.general_registers['AX'] // 16
        self.general_registers['AL'] = self.general_registers['AX'] % 16
        self.general_registers['BH'] = self.general_registers['BX'] // 16
        self.general_registers['BL'] = self.general_registers['BX'] % 16
        self.general_registers['CH'] = self.general_registers['CX'] // 16
        self.general_registers['CL'] = self.general_registers['CX'] % 16
        self.general_registers['DH'] = self.general_registers['DX'] // 16
        self.general_registers['DL'] = self.general_registers['DX'] % 16
    # 转移指令
    # 将值移动到指定寄存器
    def mov(self, register1, value):
        if register1 in self.general_registers:
            self.general_registers[register1] = self.address_resolution(value)
        elif register1 in self.special_registers:
            self.special_registers[register1] = self.address_resolution(value)

    # 入栈
    def push(self, register1):
        if register1.isdigit():
            self.stack.append(register1)
        elif register1 in self.general_registers:
            self.stack.append(self.general_registers[register1])

    # 出栈
    def pop(self, register1):
        self.general_registers[register1] = self.stack.pop()

    # 交换值
    def xchg(self, register1, value):
        self.general_registers[register1], self.general_registers[value] = self.general_registers[value], self.general_registers[register1]

    # 算数运算指令
    # 加法
    def add(self, register1, value):
        self.general_registers[register1] += self.address_resolution(value)
        # 给指定寄存器中的值加上一个数
        a = self.general_registers[register1]
        b = self.address_resolution(value)
        c = a + b
        self.change_flags(a, b, c, '+', 1)

    # 带进位加法
    def adc(self, register1, value):
        self.general_registers[register1] += self.flags['CF']
        self.general_registers[register1] += self.address_resolution(value)
        a = self.general_registers[register1] + 1
        b = self.address_resolution(value)
        c = a + b
        self.change_flags(a, b, c, '+', 1)

    # 减法
    def sub(self, register1, value):
        self.general_registers[register1] -= self.address_resolution(value)
        a = self.general_registers[register1]
        b = self.address_resolution(value)
        c = a - b
        self.change_flags(a, b, c, '-', 1)

    # 带借位减法
    def sbb(self, register1, value):
        self.general_registers[register1] -= self.flags['CF']
        self.general_registers[register1] -= self.address_resolution(value)
        a = self.general_registers[register1] - 1
        b = self.address_resolution(value)
        c = a - b
        self.change_flags(a, b, c, '-', 1)

    # 乘法
    def mul(self, register1):
        if register1 in ['AL', 'AH', 'BL', 'BH', 'CL', 'CH', 'DL', 'DH']:
            a = self.general_registers[register1]
            b = self.general_registers['AL']
            c = a * b
            self.change_flags(a, b, c, '*', 1)
            self.general_registers['AX'] = self.general_registers[register1] * self.general_registers['AL']
            self.general_registers['AH'] = self.general_registers['AX'] // 16
            self.general_registers['AL'] = self.general_registers['AX'] % 16
        elif register1 in ['AX', 'BX', 'CX', 'DX']:
            a = self.general_registers[register1]
            b = self.general_registers['AX']
            c = a * b
            self.change_flags(a,b,c,'*',2)
            self.general_registers['DX'] = self.general_registers[register1] * self.general_registers['AX'] // 256
            self.general_registers['DH'] = self.general_registers['DX'] // 16
            self.general_registers['DL'] = self.general_registers['DX'] % 16
            self.general_registers['AX'] = self.general_registers[register1] * self.general_registers['AX'] % 256
            self.general_registers['AH'] = self.general_registers['AX'] // 16
            self.general_registers['AL'] = self.general_registers['AX'] % 16
        elif register1.isdigit():
            a = int(register1)
            b = self.general_registers['AX']
            c = a * b
            self.change_flags(a, b ,c, '*', 2)
            self.general_registers['DX'] = a * self.general_registers['AX'] // 256
            self.general_registers['DH'] = self.general_registers['DX'] // 16
            self.general_registers['DL'] = self.general_registers['DX'] % 16
            self.general_registers['AX'] = a * self.general_registers['AX'] % 256
            self.general_registers['AH'] = self.general_registers['AX'] // 16
            self.general_registers['AL'] = self.general_registers['AX'] % 16

    # 带符号乘法
    def imul(self, register1):
        if register1 in ['AL', 'AH', 'BL', 'BH', 'CL', 'CH', 'DL', 'DH']:
            a = self.general_registers[register1]
            b = self.general_registers['AL']
            c = a * b
            self.change_flags(a, b, c, '*', 1)
            self.general_registers['AX'] = self.general_registers[register1] * self.general_registers['AL']
            self.general_registers['AH'] = self.general_registers['AX'] // 16
            self.general_registers['AL'] = self.general_registers['AX'] % 16
        if register1 in ['AX', 'BX', 'CX', 'DX']:
            a = self.general_registers[register1]
            b = self.general_registers['AX']
            c = a * b
            self.change_flags(a, b, c, '*', 2)
            self.general_registers['DX'] = self.general_registers[register1] *
self.general_registers['AX'] // 256
            self.general_registers['DH'] = self.general_registers['DX'] // 16
            self.general_registers['DL'] = self.general_registers['DX'] % 16
            self.general_registers['AX'] = self.general_registers[register1] *
self.general_registers['AX'] % 256
            self.general_registers['AH'] = self.general_registers['AX'] // 16
            self.general_registers['AL'] = self.general_registers['AX'] % 16

    # 除法
    def div(self, register1):
        if register1 in ['AL', 'AH', 'BL', 'BH', 'CL', 'CH', 'DL', 'DH']:
            self.general_registers['AL'] = self.general_registers['AX'] //
self.general_registers[register1]
            self.general_registers['AH'] = self.general_registers['AX'] %
self.general_registers[register1]
            self.general_registers['AX'] = self.general_registers['AH'] * 16 +
self.general_registers['AL']
        elif register1 in ['AX', 'BX', 'CX', 'DX']:
            self.general_registers['DX'] = (self.general_registers['DX'] * 256 +
self.general_registers['AL']) // self.general_registers[register1] // 256
            self.general_registers['DH'] = self.general_registers['DX'] // 16
            self.general_registers['DL'] = self.general_registers['DX'] % 16
            self.general_registers['AX'] = (self.general_registers['DX'] * 256 +
self.general_registers['AL']) % self.general_registers[register1] % 256
            self.general_registers['AH'] = self.general_registers['AX'] // 16
            self.general_registers['AL'] = self.general_registers['AX'] % 16
        elif register1.isdigit():
            self.general_registers['AL'] = self.general_registers['AX'] //
int(register1)
            self.general_registers['AH'] = self.general_registers['AX'] %
int(register1)
            self.general_registers['AX'] = self.general_registers['AH'] * 16 +
self.general_registers['AL']

    # 带符号除法
    def idiv(self, register1, value):
        if register1 in ['AL', 'AH', 'BL', 'BH', 'CL', 'CH', 'DL', 'DH']:
            self.general_registers['AL'] = self.general_registers['AX'] //
self.general_registers[register1]
            self.general_registers['AH'] = self.general_registers['AX'] %
self.general_registers[register1]
            self.general_registers['AX'] = self.general_registers['AL'] * 16 +
self.general_registers['AH']
        if register1 in ['AX', 'BX', 'CX', 'DX']:
            self.general_registers['DX'] = (self.general_registers['DX'] * 256 +
self.general_registers['AL']) // self.general_registers[register1] // 256
            self.general_registers['DH'] = self.general_registers['DX'] // 16
            self.general_registers['DL'] = self.general_registers['DX'] % 16
            self.general_registers['AX'] = (self.general_registers['DX'] * 256 +
self.general_registers['AL']) % self.general_registers[register1] % 256
            self.general_registers['AH'] = self.general_registers['AX'] // 16
            self.general_registers['AL'] = self.general_registers['AX'] % 16
    # 自增
    def inc(self):
```

```python
        a = self.general_registers['AX']
        self.general_registers['AX'] += 1
        self.general_registers['AH'] = self.general_registers['AX'] // 16
        self.general_registers['AL'] = self.general_registers['AX'] % 16
        if a > 0 and self.general_registers['AX'] > 255:
            self.flags['OF'] = 1
        else:
            self.flags['OF'] = 0
    # 自减
    def dec(self):
        a = self.general_registers['AX']
        self.general_registers['AX'] -= 1
        self.general_registers['AH'] = self.general_registers['AX'] // 16
        self.general_registers['AL'] = self.general_registers['AX'] % 16
        if a < 0 and self.general_registers['AX'] < -255:
            self.flags['OF'] = 1
        else:
            self.flags['OF'] = 0

    # 逻辑运算指令
    # 与
    def And(self, register1, value):
        if str(value).isdigit():
            self.general_registers[register1] &= int(value)
        else:
            self.general_registers[register1] &= self.address_resolution(value)
        c = self.general_registers[register1]
        if bin(c).replace('0b','').count('1') % 2 == 0:
            self.flags['PF'] = 1
        else:
            self.flags['PF'] = 0
        self.flags['CF'] = 0
        self.flags['OF'] = 0

    # 或
    def Or(self, register1, value):
        if str(value).isdigit():
            self.general_registers[register1] |= int(value)
        else:
            self.general_registers[register1] |= self.address_resolution(value)
        c = self.general_registers[register1]
        if bin(c).replace('0b','').count('1') % 2 == 0:
            self.flags['PF'] = 1
        else:
            self.flags['PF'] = 0
        self.flags['CF'] = 0
        self.flags['OF'] = 0

    # 异或
    def Xor(self, register1, value):
        if str(value).isdigit():
            self.general_registers[register1] ^= int(value)
        else:
            self.general_registers[register1] ^= self.address_resolution(value)
        c = self.general_registers[register1]
        if bin(c).replace('0b', '').count('1') % 2 == 0:
            self.flags['PF'] = 1
        else:
            self.flags['PF'] = 0
        self.flags['CF'] = 0
        self.flags['OF'] = 0
    # 测试指令
    def Test(self, register1, value):
        if str(value).isdigit():
            c = self.general_registers[register1] & int(value)
        else:
            c =self.general_registers[register1] & self.address_resolution(value)
        if bin(c).replace('0b','').count('1') % 2 == 0:
            self.flags['PF'] = 1
        else:
            self.flags['PF'] = 0
        self.flags['CF'] = 0
        self.flags['OF'] = 0

    # 取反
    def Not(self, register1):
        self.general_registers[register1] = ~self.general_registers[register1]

    # 字符串指令
    # DSI -> ESI
    def movsb(self):
        str1 = self.special_registers["DS"] * 16 + self.general_registers["SI"]
        str2 = self.special_registers["ES"] * 16 + self.general_registers["DI"]
        self.memory[str2] = self.memory[str1]
        print(f"内存中地址为{str2}的值变为{self.memory[str1]}")
        self.general_registers['SI'] += 1
        self.general_registers['DI'] += 1
    # DSI -> ESI (两位)
    def movsw(self):
        str1 = self.special_registers["DS"] * 16 + self.general_registers["SI"]
        str2 = self.special_registers["ES"] * 16 + self.general_registers["DI"]
        self.memory[str2],self.memory[str2 + 1] =
self.memory[str1],self.memory[str1 + 1]
        print(f"内存中地址为{str2}的值变为{self.memory[str1]}")
        print(f"内存中地址为{str2+1}的值变为{self.memory[str1+1]}")
        self.general_registers['SI'] += 2
        self.general_registers['DI'] += 2

    # 比较 ESI 和 DSI 改变标志位
    def cmpsb(self):
        str1 = self.special_registers["DS"] * 16 + self.general_registers["SI"]
        str2 = self.special_registers["ES"] * 16 + self.general_registers["DI"]
        if self.memory[str2] == self.memory[str1] :
            print(f"内存中地址为{str2}的值和内存中地址为{str1}的值相等, ZF变为1")
            self.flags['ZF'] = 1
        else:
            print(f"内存中地址为{str2}的值和内存中地址为{str1}的值不相等, ZF变为0")
            self.flags['ZF'] = 0
        self.general_registers['SI'] += 1
        self.general_registers['DI'] += 1

    # 比较 ESI 和 DSI 改变标志位（两位）
    def cmpsw(self):
        str1 = self.special_registers["DS"] * 16 + self.general_registers["SI"]
        str2 = self.special_registers["ES"] * 16 + self.general_registers["DI"]
        if self.memory[str2] == self.memory[str1] and self.memory[str2 + 1] ==
self.memory[str1 + 1] :
            print(f"内存中地址为{str2}的值和内存中地址为{str1}的值相等，并且内存中地址为
{str2+1}的值和内存中地址为{str1+1}的值也相等，ZF变为1")
            self.flags['ZF'] = 1
        else:
            self.flags['ZF'] = 0
        self.general_registers['SI'] += 1
        self.general_registers['DI'] += 1

    # 比较 ESI 和 AL 改变标志位
    def scasb(self):
        str1 = self.general_registers["AL"]
        str2 = self.special_registers["ES"] * 16 + self.general_registers["DI"]
        if self.memory[str2] == str1:
            print(f"内存中地址为{str2}的值和寄存器AL的值相等，ZF变为1")
            self.flags['ZF'] = 1
        else:
            self.flags['ZF'] = 0
            print(f"内存中地址为{str2}的值和寄存器AL的值不相等，ZF变为0")
        self.general_registers['DI'] += 1

    # 比较 ESI 和 AX 改变标志位
    def scasw(self):
        str1 = self.general_registers["AX"]
        str2 = self.special_registers["ES"] * 16 + self.general_registers["DI"]
        if self.memory[str2] == str1:
            print(f"内存中地址为{str2}的值和寄存器AX的值相等，ZF变为1")
            self.flags['ZF'] = 1
        else:
            print(f"内存中地址为{str2}的值和寄存器AX的值不相等，ZF变为0")
            self.flags['ZF'] = 0
        self.general_registers['DI'] += 1

    # 将地址值存储到AL中
    def lodsb(self):
        str = self.special_registers["ES"] * 16 + self.general_registers["DI"]
        print(f"将寄存器AL的值变为内存中地址为{str}的值")
        self.general_registers["AL"] = int(self.memory[str])
        self.general_registers['DI'] += 1
    # 将地址值存储到AX中
    def lodsw(self):
        str = self.special_registers["ES"] * 16 + self.general_registers["DI"]
        print(f"将寄存器AX的值变为内存中地址为{str}的值")
        self.general_registers["AX"] = int(self.memory[str])
        self.general_registers['DI'] += 1
    # 将AL存储到ESI中
    def stosb(self):
        str1 = self.general_registers["AL"]
        str2 = self.special_registers["ES"] * 16 + self.general_registers["DI"]
        print(f"将内存中地址为{str2}的值变为寄存器AL的值{str1}")
        self.memory[str2] = str1
        self.general_registers['DI'] += 1
    # 将AX存储到ESI中
    def stosw(self):
        str1 = self.general_registers["AX"]
        str2 = self.special_registers["ES"] * 16 + self.general_registers["DI"]
        print(f"将内存中地址为{str2}的值变为寄存器AX的值{str1}")
        self.memory[str2] = str1
        self.general_registers['DI'] += 1
    def nop(self):
        print("执行了NOP指令")
    def clc(self):
        self.flags['CF'] = 0
    def stc(self):
        self.flags['CF'] = 0
    def cmc(self):
        self.flags['CF'] = 0 if self.flags['CF'] == 1 else 1
    def cld(self):
        self.flags['DF'] = 0
    def std(self):
        self.flags['DF'] = 1
    def cli(self):
        self.flags['IF'] = 0
    def sti(self):
        self.flags['IF'] = 1
    def jmp(self,value):
        ip = self.special_registers['IP']
        if value.isdigit():
            self.special_registers['IP'] = int(value)
            print(f"程序IP由{ip}跳到{int(value)}")
        else:
            self.flags['IP'] = self.general_registers[value]
            print(f"程序IP由{ip}跳到{self.general_registers[value]}")
    def call(self,value):
        ip = self.special_registers['IP'] + 1
        print(ip)
        self.stack.append(int(ip))
        if value.isdigit():
            self.special_registers['IP'] = int(value)
            print(f"程序IP由{ip}跳到{int(value)}")
        else:
            self.flags['IP'] = self.general_registers[value]
            print(f"程序IP由{ip}跳到{self.general_registers[value]}")
    def ret(self):
        ip = self.stack.pop()
        self.special_registers['IP'] = ip
        print(f"程序IP跳回{self.special_registers['IP']}")
    def Int(self,value):
        if self.flags['IF'] != 1:
            print("中断程序未开启")
        else:
            str = value.replace('h','')
            if int(str) == 21:
                if self.general_registers['AH'] == 2:
                    print(f"DL的值为{self.general_registers['DL']}")

    def iret(self):
        #无中断返回
        print("执行了IRET指令")
        self.special_registers["IP"] += 1
    def Loop(self,value):
        ip = self.special_registers['IP']
        self.general_registers['CX'] -= 1
        if self.general_registers['CX'] > 0:
            if value.isdigit():
                self.special_registers['IP'] = int(value)
                print(f"程序IP由{ip}跳到{int(value)}")
            else:
                self.flags['IP'] = self.general_registers[value]
                print(f"程序IP由{ip}跳到{self.general_registers[value]}")
        else:
            self.special_registers['IP'] += 1

    def Loopz(self,value):
        ip = self.special_registers['IP']
        self.general_registers['CX'] -= 1
        if self.general_registers['CX'] != 0 and self.flags['ZF'] == 1:
            if value.isdigit():
                self.special_registers['IP'] = int(value)
                print(f"程序IP由{ip}跳到{int(value)}")
            else:
                self.flags['IP'] = self.general_registers[value]
                print(f"程序IP由{ip}跳到{self.general_registers[value]}")
        else:
            self.special_registers['IP'] += 1
    def Loopnz(self,value):
        ip = self.special_registers['IP']
        self.general_registers['CX'] -= 1
        if self.general_registers['CX'] != 0 and self.flags['ZF'] == 0:
            if value.isdigit():
                self.special_registers['IP'] = int(value)
                print(f"程序IP由{ip}跳到{int(value)}")
            else:
                self.flags['IP'] = self.general_registers[value]
                print(f"程序IP由{ip}跳到{self.general_registers[value]}")
        else:
            self.special_registers['IP'] += 1

    # 停机指令
    def hlt(self):
        # 停止执行
        print("停止执行")
        exit(0)

    def print_registers(self):

        self.adjust_register()
        # 输出所有寄存器的状态
        print("通用寄存器状态:")
        for reg, val in self.general_registers.items():
            print(f"{reg}: {val} ", end='')
        print("\n专用寄存器状态:")
        for reg, val in self.special_registers.items():
            print(f"{reg}: {val} ", end='')
        print("\n标志寄存器状态:")
        for reg, val in self.flags.items():
            print(f"{reg}: {val} ", end='')
        print('\n')

    # 更新总线状态
    def update_buses(self, operation, register, value):
        if operation == 'MOV':
            self.address_bus['source'] = f"{register}"
            self.address_bus['destination'] = f"{register}"
            self.data_bus['data'] = self.address_resolution(value)
            self.control_bus['read'] = True
            self.control_bus['write'] = False
        elif operation == 'PUSH':
            self.address_bus['source'] = f"{register}"
            self.address_bus['destination'] = f"{register}"
            self.data_bus['data'] = self.general_registers[register]
            self.control_bus['read'] = False
            self.control_bus['write'] = True
        elif operation == 'POP':
            self.address_bus['source'] = f"{register}"
            self.address_bus['destination'] = f"{register}"
            self.data_bus['data'] = self.general_registers[register]
            self.control_bus['read'] = True
            self.control_bus['write'] = False
        elif operation == ['XCHG', 'TEST', 'HTL']:
            self.address_bus['source'] = ''
            self.address_bus['destination'] = ''
            self.data_bus['data'] = 0
            self.control_bus['read'] = False
            self.control_bus['write'] = False
        elif operation in ['ADD', 'ADC', 'SUB', 'SBB']:
            self.address_bus['source'] = f"{register}"
            self.address_bus['destination'] = f"{register}"
            self.data_bus['data'] = self.address_resolution(value)
            self.control_bus['read'] = True
            self.control_bus['write'] = True
        elif operation == ['INC','DEC']:
            self.address_bus['source'] = f"{register}"
            self.address_bus['destination'] = f"{register}"
            self.data_bus['data'] = 1
            self.control_bus['read'] = False
            self.control_bus['write'] = False
        elif operation == ['MUL', 'IMUL']:
            self.address_bus['source'] = f"{register} AL"
            self.address_bus['destination'] = f"AX"
            self.data_bus['data'] = self.general_registers['AX']
            self.control_bus['read'] = True
            self.control_bus['write'] = True
        elif operation == ['DIV', 'IDIV']:
            self.address_bus['source'] = f"{register}"
            self.address_bus['destination'] = f"AH AL"
            self.data_bus['data'] = str(self.general_registers['AH']) + ' ' +
str(self.general_registers['AL'])
            self.control_bus['read'] = True
            self.control_bus['write'] = True
        elif operation == ['AND', 'OR', 'XOR']:
            self.address_bus['source'] = f"{register}"
            self.address_bus['destination'] = f"{register}"
            self.data_bus['data'] = str(self.general_registers[register]) + ' ' + str(self.address_resolution(value))
            self.control_bus['read'] = False
            self.control_bus['write'] = False
        elif operation == 'NOT':
            self.address_bus['source'] = f"{register}"
            self.address_bus['destination'] = f"{register}"
            self.data_bus['data'] = self.general_registers[register]
            self.control_bus['read'] = False
            self.control_bus['write'] = True

        # print("总线状态:")
        # print("地址总线:")
        # for key, val in self.address_bus.items():
        #     print(f"{key}: {val}")
        # print("数据总线:")
        # for key, val in self.data_bus.items():
        #     print(f"{key}: {val}")
        # print("控制总线:")
        # for key, val in self.control_bus.items():
        #     print(f"{key}: {val}")
        # print('\n')

    # 运行biu
    def biu_run(self):
        print("biu开始执行")
        while True:
            t1 = time.process_time()
            com = self.fetch()
            self.queue.put(com)
            t2 = time.process_time()
            print(f"将指令传入指令队列所用时间为{t2 - t1}秒")
    # 运行eu
    def eu_run(self):

        print()
        print("\neu开始执行")
        while True:
            op, reg, val = self.decodes(self.queue.get())
            self.execute(op, reg, val)
            if op == 'HLT':
                break

if __name__ == "__main__":
    cpu = Cpu()
    cpu.stack.append(2)
    cpu.memory = ['0'] * 1000
    cpu.memory[16:25] = ['16','17','18','119','120','121',"22","23","24","25"]
    cpu.memory[116:121] = ['116','117','118','119','120','121']

    cpu.memory[0:5] = ["MOV CX 2","MOV AX 1","ADD AX AX","LOOP 2","HTL"]
    eu = threading.Thread(target=cpu.eu_run)
    biu = threading.Thread(target=cpu.biu_run)
    biu.start()
    eu.start()
    biu.join()
    eu.join()
```

```
设计思路
1.定义结构体Cpu
    属性:
            内存，指令队列，堆栈，三条总线，通用寄存器，标志寄存器，专用寄存器，指令集
    函数:
            fetch() 获取指令
            decode() 解码指令
            execute() 执行指令
            address_resolution() 地址解析
            change_flags() 改变标志寄存器
            adjust_register()  调整通用寄存器
            print_registers() 输出所有寄存器状态
            update_buses()  输出所有总线状态
            指令函数 :
                    mov(),puch(),pop(),xchg(),

add(),adc(),sub(),sbb(),mul(),imul(),div(),idiv(),inc(),dec(),
                    and(),or(),xor(),test(),not(),
                    nop(),hlt(),clc(),stc(),cmc(),cld(),std(),cli(),sti()

movsb(),mmovsw(),cmpsb(),cmpsw(),scasb(),scasw(),lodsb(),lodsw(),stosb(),stosw()
                    jmp(),call(),ret(),iret(),loop(),loopz(),loopnz()

2.总线接口单元(BIU)
    定义函数biu_run():
            调用fetch()函数从内存中读取指令到指令队列
            输出指令所用时间
3.执行单元（EU）
    定义函数eu_run():
            从指令队列中获取指令，解码指令decode()，执行指令execute(),打印cpu状态
print_registers()，update_buses()
            输出执行每条任务所用时间
4.主函数
    创建两个线程:
            eu = threading.Thread(target=cpu.eu_run)
            biu = threading.Thread(target=cpu.biu_run)
    同时执行，模拟BIU，EU同时工作
```

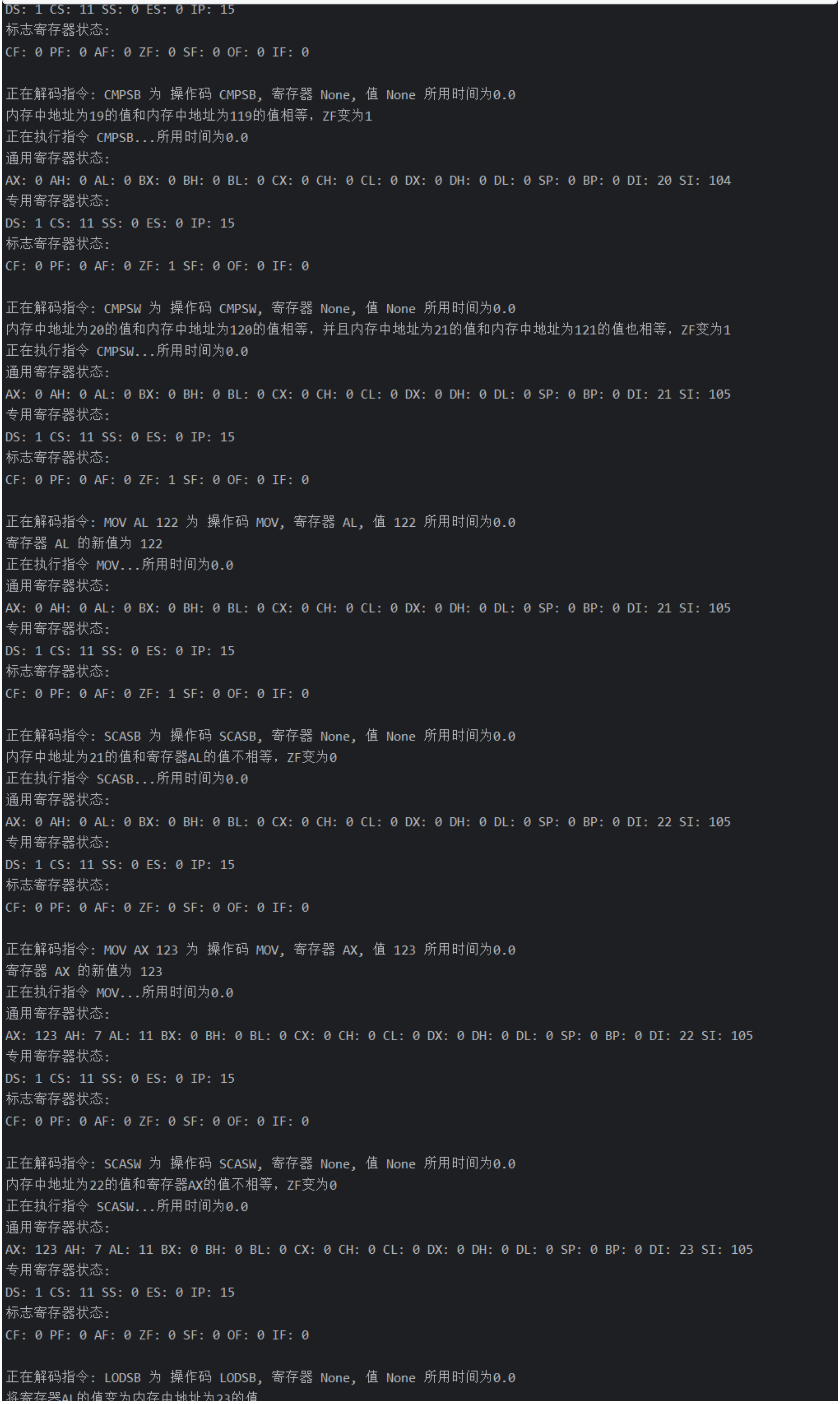
```
运行结果
```

```
字符串函数
指令
cpu.memory = ['0'] * 1000
cpu.memory[16:25] = ['16','17','18','119','120','121',"22","23","24","25"]
cpu.memory[116:121] = ['116','117','118','119','120','121']
cpu.memory[0:15] =
["MOV DI 16",
"MOV SI 100",
"MOVSB","MOVSW",
"CMPSB","CMPSW",
"MOV AL 122","SCASB",
"MOV AX 123","SCASW",
"LODSB","LODSW",
"STOSB","STOSW",
"HTL"]
```

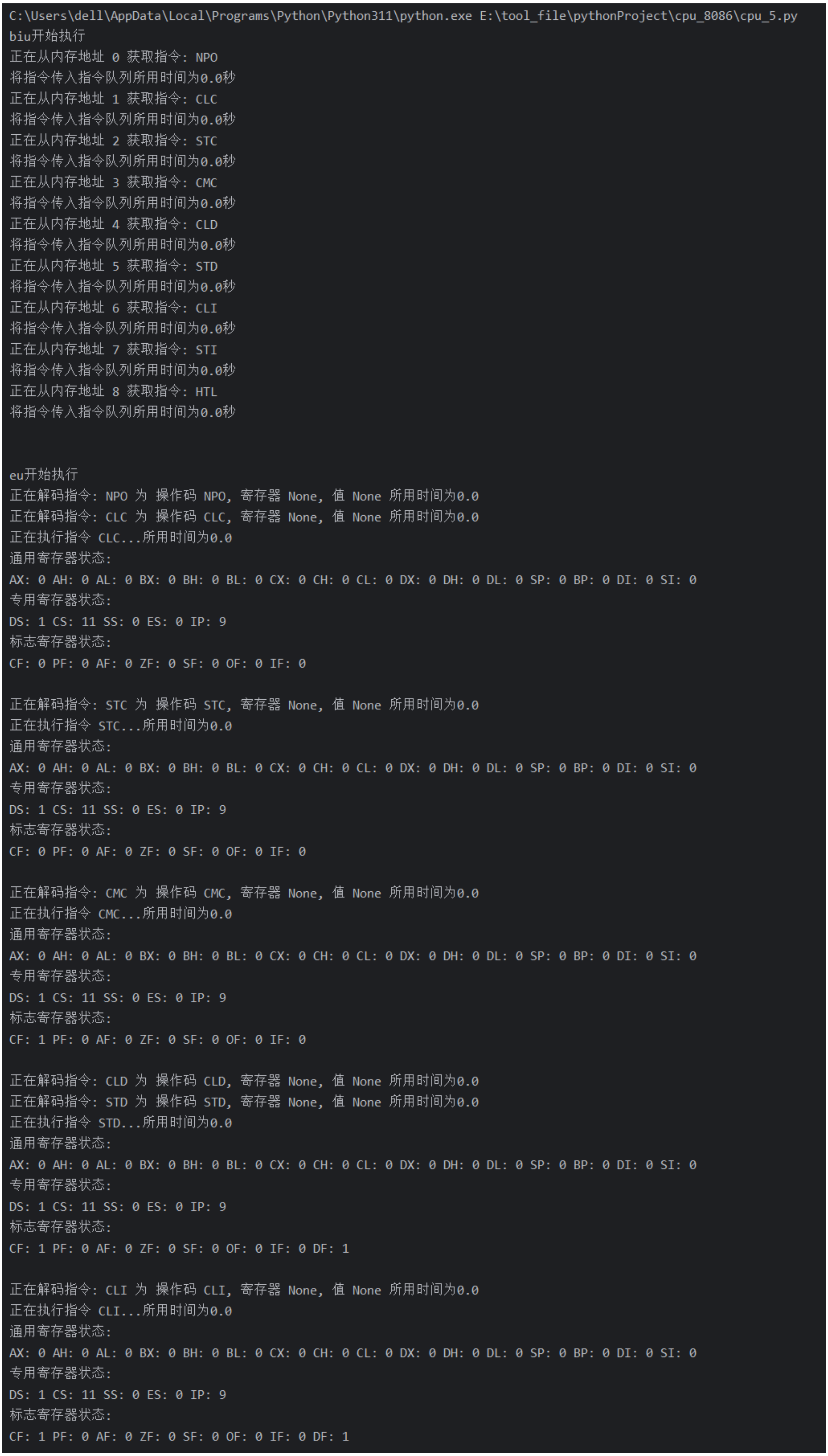


```
C:\Users\dell\AppData\Local\Programs\Python\Python311\python.exe E:\tool_file\pythonProject\cpu_8086\cpu_5.py
biu开始执行
正在从内存地址 0 获取指令: MOV DI 16
将指令传入指令队列所用时间为0.0秒
正在从内存地址 1 获取指令: MOV SI 100
将指令传入指令队列所用时间为0.0秒
正在从内存地址 2 获取指令: MOVSB
将指令传入指令队列所用时间为0.0秒
正在从内存地址 3 获取指令: MOVSW
将指令传入指令队列所用时间为0.0秒
正在从内存地址 4 获取指令: CMPSB

将指令传入指令队列所用时间为0.0秒
正在从内存地址 5 获取指令: CMPSW
eu开始执行
正在解码指令: MOV DI 16 为 操作码 MOV, 寄存器 DI, 值 16 所用时间为0.0
寄存器 DI 的新值为 16
正在执行指令 MOV...所用时间为0.0
通用寄存器状态:
AX: 0 AH: 0
将指令传入指令队列所用时间为0.0秒
AL: 0 BX: 0 BH: 0 正在从内存地址 6 获取指令: MOV AL 122
将指令传入指令队列所用时间为0.0秒
正在从内存地址 7 获取指令: SCASB
将指令传入指令队列所用时间为0.0秒
正在从内存地址 8 获取指令: MOV AX 123
将指令传入指令队列所用时间为0.0秒
正在从内存地址 9 获取指令: SCASW
将指令传入指令队列所用时间为0.0秒
BL: 0 正在从内存地址 10 获取指令: LODSB
CX: 0 将指令传入指令队列所用时间为0.0秒
正在从内存地址 11 获取指令: LODSWCH: 0 CL: 0
将指令传入指令队列所用时间为0.0秒
正在从内存地址 12 获取指令: STOSBDX: 0 DH: 0
将指令传入指令队列所用时间为0.0秒
正在从内存地址 13 获取指令: STOSWDL: 0 SP: 0
将指令传入指令队列所用时间为0.0秒
正在从内存地址 14 获取指令: HTL
将指令传入指令队列所用时间为0.0秒
BP: 0 DI: 16 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 15
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0

正在解码指令: MOV SI 100 为 操作码 MOV, 寄存器 SI, 值 100 所用时间为0.0
寄存器 SI 的新值为 100
正在执行指令 MOV...所用时间为0.0
通用寄存器状态:
AX: 0 AH: 0 AL: 0 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 16 SI: 100
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 15
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0

正在解码指令: MOVSB 为 操作码 MOVSB, 寄存器 None, 值 None 所用时间为0.0
内存中地址为16的值变为116
正在执行指令 MOVSB...所用时间为0.0
通用寄存器状态:
AX: 0 AH: 0 AL: 0 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 17 SI: 101
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 15
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0

正在解码指令: MOVSW 为 操作码 MOVSW, 寄存器 None, 值 None 所用时间为0.0
内存中地址为17的值变为117
内存中地址为18的值变为118
正在执行指令 MOVSW...所用时间为0.0
```

```
处理器控制指令
cpu.memory[0:9] = ["NPO","CLC","STC","CMC","CLD","STD","CLI","STI","HTL"]
```



```
DS: 1 CS: 11 SS: 0 ES: 0 IP: 15
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0

正在解码指令: CMPSB 为 操作码 CMPSB, 寄存器 None, 值 None 所用时间为0.0
内存中地址为19的值和内存中地址为119的值相等，ZF变为1
正在执行指令 CMPSB...所用时间为0.0
通用寄存器状态:
AX: 0 AH: 0 AL: 0 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 20 SI: 104
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 15
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 1 SF: 0 OF: 0 IF: 0

正在解码指令: CMPSW 为 操作码 CMPSW, 寄存器 None, 值 None 所用时间为0.0
内存中地址为20的值和内存中地址为120的值相等，并且内存中地址为21的值和内存中地址为121的值也相等，ZF变为1
正在执行指令 CMPSW...所用时间为0.0
通用寄存器状态:
AX: 0 AH: 0 AL: 0 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 21 SI: 105
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 15
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 1 SF: 0 OF: 0 IF: 0

正在解码指令: MOV AL 122 为 操作码 MOV, 寄存器 AL, 值 122 所用时间为0.0
寄存器 AL 的新值为 122
正在执行指令 MOV...所用时间为0.0
通用寄存器状态:
AX: 0 AH: 0 AL: 0 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 21 SI: 105
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 15
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 1 SF: 0 OF: 0 IF: 0

正在解码指令: SCASB 为 操作码 SCASB, 寄存器 None, 值 None 所用时间为0.0
内存中地址为21的值和寄存器AL的值不相等，ZF变为0
正在执行指令 SCASB...所用时间为0.0
通用寄存器状态:
AX: 0 AH: 0 AL: 0 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 22 SI: 105
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 15
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0

正在解码指令: MOV AX 123 为 操作码 MOV, 寄存器 AX, 值 123 所用时间为0.0
寄存器 AX 的新值为 123
正在执行指令 MOV...所用时间为0.0
通用寄存器状态:
AX: 123 AH: 7 AL: 11 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 22 SI: 105
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 15
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0

正在解码指令: SCASW 为 操作码 SCASW, 寄存器 None, 值 None 所用时间为0.0
内存中地址为22的值和寄存器AX的值不相等，ZF变为0
正在执行指令 SCASW...所用时间为0.0
通用寄存器状态:
AX: 123 AH: 7 AL: 11 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 23 SI: 105
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 15
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0

正在解码指令: LODSB 为 操作码 LODSB, 寄存器 None, 值 None 所用时间为0.0
将寄存器AL的值变为内存中地址为23的值
```

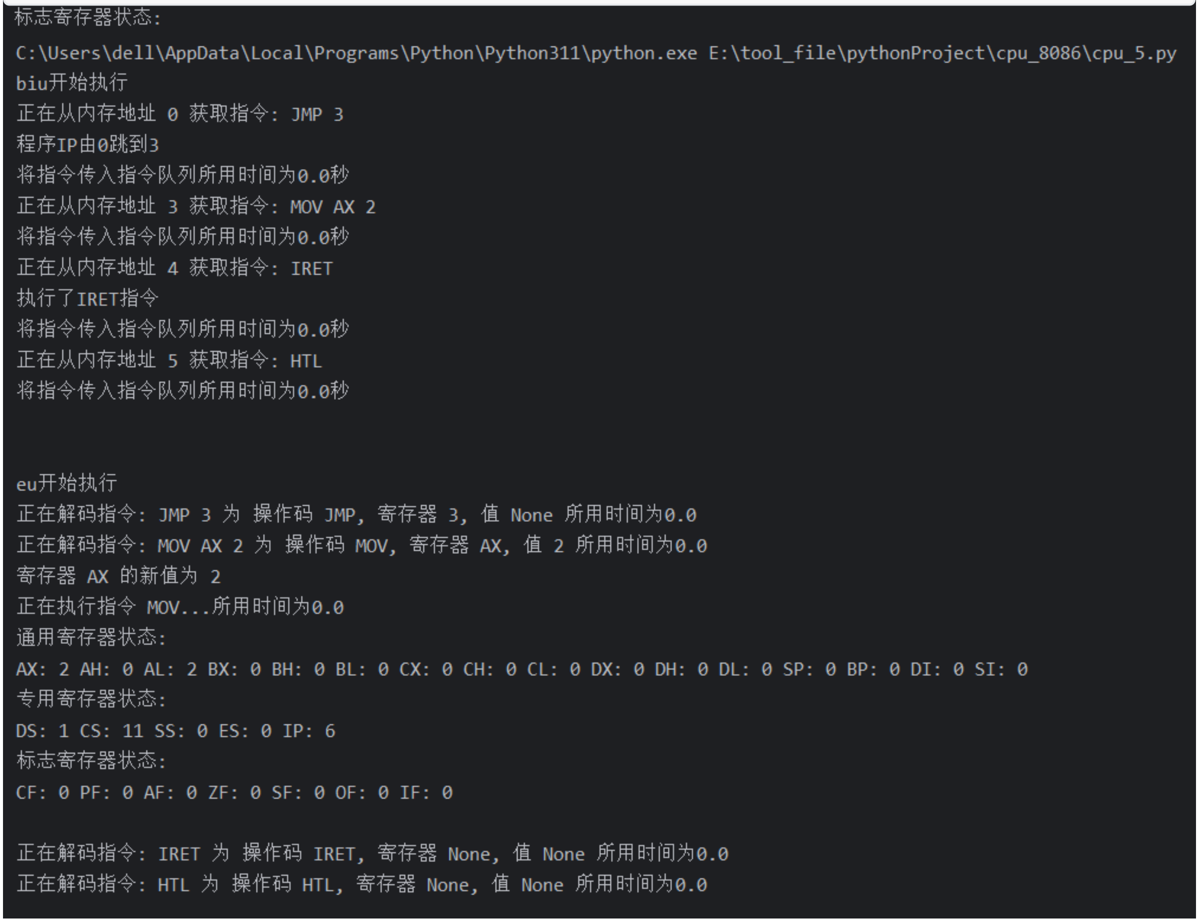


```
C:\Users\dell\AppData\Local\Programs\Python\Python311\python.exe E:\tool_file\pythonProject\cpu_8086\cpu_5.py
biu开始执行
正在从内存地址 0 获取指令: NPO
将指令传入指令队列所用时间为0.0秒
正在从内存地址 1 获取指令: CLC
将指令传入指令队列所用时间为0.0秒
正在从内存地址 2 获取指令: STC
将指令传入指令队列所用时间为0.0秒
正在从内存地址 3 获取指令: CMC
将指令传入指令队列所用时间为0.0秒
正在从内存地址 4 获取指令: CLD
将指令传入指令队列所用时间为0.0秒
正在从内存地址 5 获取指令: STD
将指令传入指令队列所用时间为0.0秒
正在从内存地址 6 获取指令: CLI
将指令传入指令队列所用时间为0.0秒
正在从内存地址 7 获取指令: STI
将指令传入指令队列所用时间为0.0秒
正在从内存地址 8 获取指令: HTL
将指令传入指令队列所用时间为0.0秒


eu开始执行
正在解码指令: NPO 为 操作码 NPO, 寄存器 None, 值 None 所用时间为0.0
正在解码指令: CLC 为 操作码 CLC, 寄存器 None, 值 None 所用时间为0.0
正在执行指令 CLC...所用时间为0.0
通用寄存器状态:
AX: 0 AH: 0 AL: 0 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 0 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 9
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0

正在解码指令: STC 为 操作码 STC, 寄存器 None, 值 None 所用时间为0.0
正在执行指令 STC...所用时间为0.0
通用寄存器状态:
AX: 0 AH: 0 AL: 0 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 0 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 9
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0

正在解码指令: CMC 为 操作码 CMC, 寄存器 None, 值 None 所用时间为0.0
正在执行指令 CMC...所用时间为0.0
通用寄存器状态:
AX: 0 AH: 0 AL: 0 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 0 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 9
标志寄存器状态:
CF: 1 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0

正在解码指令: CLD 为 操作码 CLD, 寄存器 None, 值 None 所用时间为0.0
正在解码指令: STD 为 操作码 STD, 寄存器 None, 值 None 所用时间为0.0
正在执行指令 STD...所用时间为0.0
通用寄存器状态:
AX: 0 AH: 0 AL: 0 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 0 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 9
标志寄存器状态:
CF: 1 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0 DF: 1

正在解码指令: CLI 为 操作码 CLI, 寄存器 None, 值 None 所用时间为0.0
正在执行指令 CLI...所用时间为0.0
通用寄存器状态:
AX: 0 AH: 0 AL: 0 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 0 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 9
标志寄存器状态:
CF: 1 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0 DF: 1
```

```
程序控制类指令
```

```
正在执行指令 STI...所用时间为0.0
通用寄存器状态:
```

```
jmp iret
["JMP 3", "", "", "MOV AX 2","IRET", "HTL"]
```

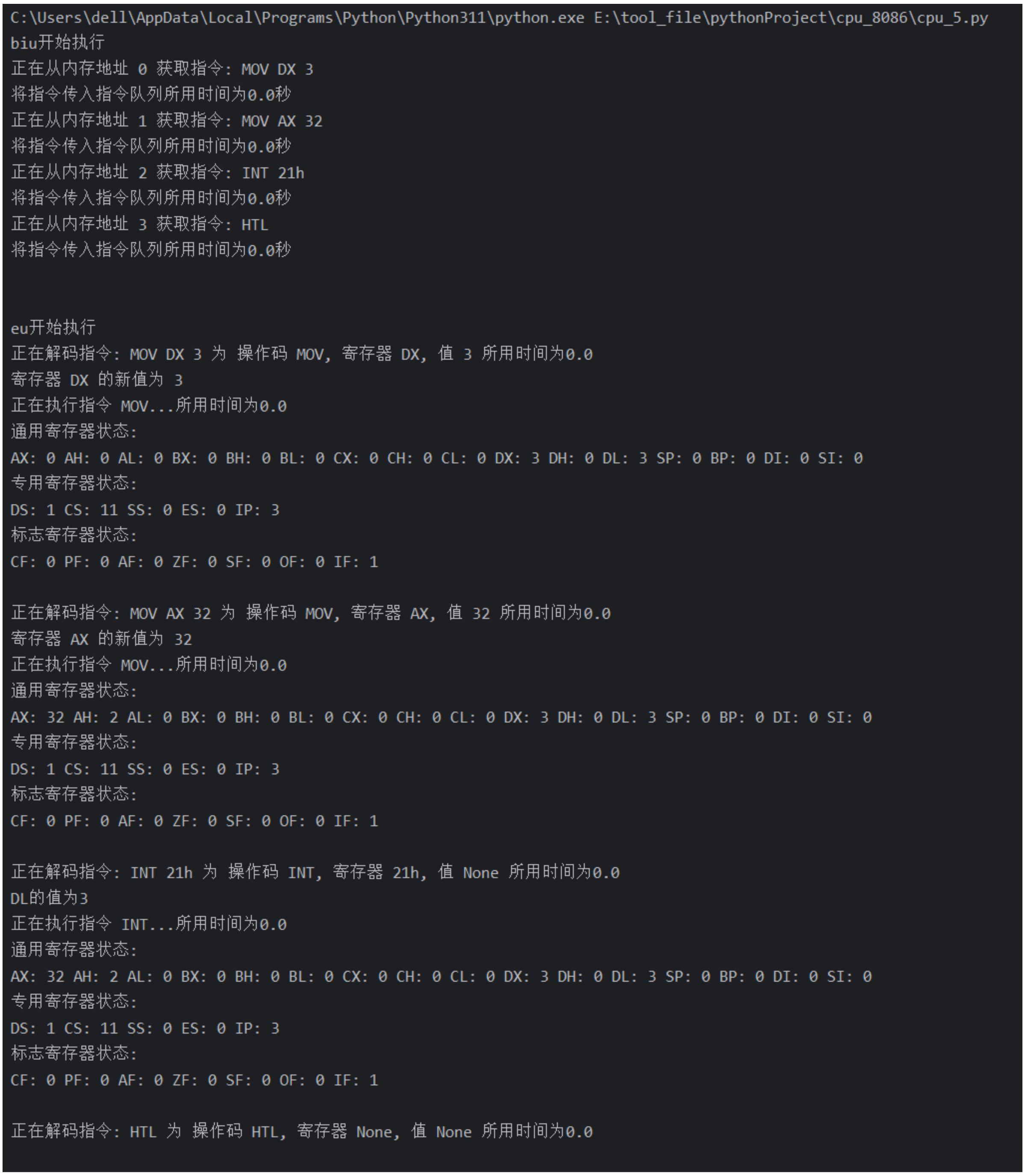


```
标志寄存器状态:
C:\Users\dell\AppData\Local\Programs\Python\Python311\python.exe E:\tool_file\pythonProject\cpu_8086\cpu_5.py
biu开始执行
正在从内存地址 0 获取指令: JMP 3
程序IP由0跳到3
将指令传入指令队列所用时间为0.0秒
正在从内存地址 3 获取指令: MOV AX 2
将指令传入指令队列所用时间为0.0秒
正在从内存地址 4 获取指令: IRET
执行了IRET指令
将指令传入指令队列所用时间为0.0秒
正在从内存地址 5 获取指令: HTL
将指令传入指令队列所用时间为0.0秒



eu开始执行
正在解码指令: JMP 3 为 操作码 JMP, 寄存器 3, 值 None 所用时间为0.0
正在解码指令: MOV AX 2 为 操作码 MOV, 寄存器 AX, 值 2 所用时间为0.0
寄存器 AX 的新值为 2
正在执行指令 MOV...所用时间为0.0
通用寄存器状态:
AX: 2 AH: 0 AL: 2 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 0 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 6
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0

正在解码指令: IRET 为 操作码 IRET, 寄存器 None, 值 None 所用时间为0.0
正在解码指令: HTL 为 操作码 HTL, 寄存器 None, 值 None 所用时间为0.0
```

```
call ret
cpu.memory[0:6] = ["CALL 3", "MOV BX 3", "HLT", "","MOV AX 2","RET"]
```

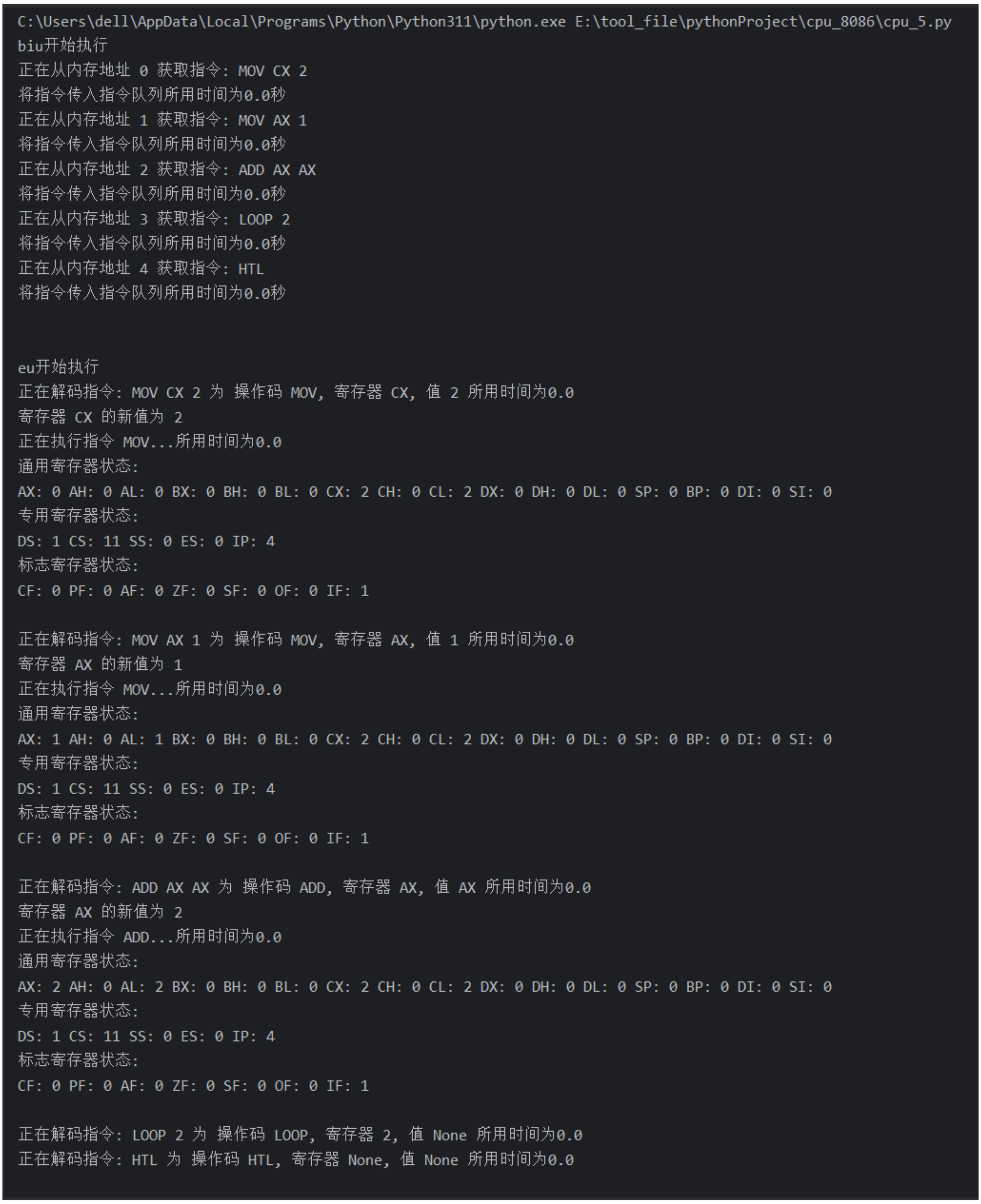
```
biu开始执行
正在从内存地址 0 获取指令: CALL 3
1
程序IP由1跳到3
将指令传入指令队列所用时间为0.0秒
正在从内存地址 3 获取指令:
将指令传入指令队列所用时间为0.0秒
正在从内存地址 4 获取指令: MOV AX 2
将指令传入指令队列所用时间为0.0秒

eu开始执行
正在解码指令: CALL 3 为 操作码 CALL, 寄存器 3, 值 None 所用时间为0.0
正在解码指令:  为 操作码 , 寄存器 None, 值 None 所用时间为0.0

正在从内存地址 5 获取指令: RET
程序IP跳回1
将指令传入指令队列所用时间为0.0秒
正在从内存地址 1 获取指令: MOV BX 3
将指令传入指令队列所用时间为0.0秒
正在从内存地址 2 获取指令: HLT
将指令传入指令队列所用时间为0.0秒
正在从内存地址 3 获取指令:
将指令传入指令队列所用时间为0.0秒
正在从内存地址 4 获取指令: MOV AX 2
将指令传入指令队列所用时间为0.0秒
正在从内存地址 5 获取指令: RET
程序IP跳回2正在解码指令: MOV AX 2 为 操作码 MOV, 寄存器 AX, 值 2 所用时间为0.0
将指令传入指令队列所用时间为0.0秒
正在从内存地址 2 获取指令: HLT
将指令传入指令队列所用时间为0.0秒
正在从内存地址 3 获取指令:
将指令传入指令队列所用时间为0.0秒
正在从内存地址 4 获取指令: MOV AX 2
将指令传入指令队列所用时间为0.0秒
正在从内存地址 5 获取指令: RET

寄存器 AX 的新值为 2
正在执行指令 MOV...所用时间为0.0
通用寄存器状态:
AX: 2 AH: 0 AL: 2 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 0 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 5
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0

正在解码指令: RET 为 操作码 RET, 寄存器 None, 值 None 所用时间为0.0
正在解码指令: MOV BX 3 为 操作码 MOV, 寄存器 BX, 值 3 所用时间为0.0
寄存器 BX 的新值为 3
正在执行指令 MOV...所用时间为0.0
通用寄存器状态:
AX: 2 AH: 0 AL: 2 BX: 3 BH: 0 BL: 3 CX: 0 CH: 0 CL: 0 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 0 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 5
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 0

正在解码指令: HLT 为 操作码 HLT, 寄存器 None, 值 None 所用时间为0.0
停止执行
```

```
int
["MOV DX 3","MOV AX 32","INT 21h","HTL"]
```



```
C:\Users\dell\AppData\Local\Programs\Python\Python311\python.exe E:\tool_file\pythonProject\cpu_8086\cpu_5.py
biu开始执行
正在从内存地址 0 获取指令: MOV DX 3
将指令传入指令队列所用时间为0.0秒
正在从内存地址 1 获取指令: MOV AX 32
将指令传入指令队列所用时间为0.0秒
正在从内存地址 2 获取指令: INT 21h
将指令传入指令队列所用时间为0.0秒
正在从内存地址 3 获取指令: HTL
将指令传入指令队列所用时间为0.0秒


eu开始执行
正在解码指令: MOV DX 3 为 操作码 MOV, 寄存器 DX, 值 3 所用时间为0.0
寄存器 DX 的新值为 3
正在执行指令 MOV...所用时间为0.0
通用寄存器状态:
AX: 0 AH: 0 AL: 0 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 3 DH: 0 DL: 3 SP: 0 BP: 0 DI: 0 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 3
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 1

正在解码指令: MOV AX 32 为 操作码 MOV, 寄存器 AX, 值 32 所用时间为0.0
寄存器 AX 的新值为 32
正在执行指令 MOV...所用时间为0.0
通用寄存器状态:
AX: 32 AH: 2 AL: 0 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 3 DH: 0 DL: 3 SP: 0 BP: 0 DI: 0 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 3
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 1

正在解码指令: INT 21h 为 操作码 INT, 寄存器 21h, 值 None 所用时间为0.0
DL的值为3
正在执行指令 INT...所用时间为0.0
通用寄存器状态:
AX: 32 AH: 2 AL: 0 BX: 0 BH: 0 BL: 0 CX: 0 CH: 0 CL: 0 DX: 3 DH: 0 DL: 3 SP: 0 BP: 0 DI: 0 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 3
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 1

正在解码指令: HTL 为 操作码 HTL, 寄存器 None, 值 None 所用时间为0.0
```

```
loop loopz loopnz 只测试一个 其他只要改变ZF的值即可
["MOV CX 2","MOV AX 1","ADD AX AX","LOOP 2","HTL"]
```



```
C:\Users\dell\AppData\Local\Programs\Python\Python311\python.exe E:\tool_file\pythonProject\cpu_8086\cpu_5.py
biu开始执行
正在从内存地址 0 获取指令: MOV CX 2
将指令传入指令队列所用时间为0.0秒
正在从内存地址 1 获取指令: MOV AX 1
将指令传入指令队列所用时间为0.0秒
正在从内存地址 2 获取指令: ADD AX AX
将指令传入指令队列所用时间为0.0秒
正在从内存地址 3 获取指令: LOOP 2
将指令传入指令队列所用时间为0.0秒
正在从内存地址 4 获取指令: HTL
将指令传入指令队列所用时间为0.0秒


eu开始执行
正在解码指令: MOV CX 2 为 操作码 MOV, 寄存器 CX, 值 2 所用时间为0.0
寄存器 CX 的新值为 2
正在执行指令 MOV...所用时间为0.0
通用寄存器状态:
AX: 0 AH: 0 AL: 0 BX: 0 BH: 0 BL: 0 CX: 2 CH: 0 CL: 2 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 0 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 4
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 1

正在解码指令: MOV AX 1 为 操作码 MOV, 寄存器 AX, 值 1 所用时间为0.0
寄存器 AX 的新值为 1
正在执行指令 MOV...所用时间为0.0
通用寄存器状态:
AX: 1 AH: 0 AL: 1 BX: 0 BH: 0 BL: 0 CX: 2 CH: 0 CL: 2 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 0 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 4
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 1

正在解码指令: ADD AX AX 为 操作码 ADD, 寄存器 AX, 值 AX 所用时间为0.0
寄存器 AX 的新值为 2
正在执行指令 ADD...所用时间为0.0
通用寄存器状态:
AX: 2 AH: 0 AL: 2 BX: 0 BH: 0 BL: 0 CX: 2 CH: 0 CL: 2 DX: 0 DH: 0 DL: 0 SP: 0 BP: 0 DI: 0 SI: 0
专用寄存器状态:
DS: 1 CS: 11 SS: 0 ES: 0 IP: 4
标志寄存器状态:
CF: 0 PF: 0 AF: 0 ZF: 0 SF: 0 OF: 0 IF: 1

正在解码指令: LOOP 2 为 操作码 LOOP, 寄存器 2, 值 None 所用时间为0.0
正在解码指令: HTL 为 操作码 HTL, 寄存器 None, 值 None 所用时间为0.0
```