python代码

```
import queue
import threading
import time

class Cpu:
    def __init__(self):
        self.memory = [""] * 256  # 内存
        self.queue = queue.Queue()  # 指令队列
        self.address_bus = {}  # 地址总线
        self.data_bus = {}  # 数据总线
        self.control_bus = {}  # 控制总线
        # 专用寄存器
        self.special_registers = {
            "CS": 0,
            "DS": 0,
            "SS": 0,
            "ES": 0,
            "IP": 0
        }
        # 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,
            "DP": 0,
            "DI": 0,
            "SI": 0
        }
        # 标志寄存器
        self.flags = {
            "CF": 0,
            "PF": 0,
            "AF": 0,
            "ZF": 0,
            "SF": 0,
            "OF": 0
```

```
        }
        # 指令集
        self.instructions = {
            'MOV': self.mov,
            'ADD': self.add,
            'HLT': self.hlt
        }

    def fetch(self):
        # 获取指令
        if len(self.memory) > self.special_registers["IP"]:
            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}")
            self.special_registers["IP"] += 1
            return component
        else:
            exit(0)

    def decodes(self, part):
        # 解码指令
        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()
            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

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

            self.print_registers()
            self.update_buses(operation, register, value)

    def mov(self, register1, value):
```

```
        # 将值移动到指定寄存器
        if value in self.general_registers:
            print(f"将 {value} 移动到寄存器 {register1}")
            self.general_registers[register1] = self.general_registers[value]
        else:
            print(f"将 {value} 移动到寄存器 {register1}")
            self.general_registers[register1] = value

    def add(self, register1, value):
        # 给指定寄存器中的值加上一个数
        a = self.general_registers[register1]
        if value in self.general_registers:
            b = self.general_registers[value]
            print(f"给寄存器 {register1} 加上 {value}")
            self.general_registers[register1] += self.general_registers[value]
        else:
            b = value
            print(f"给寄存器 {register1} 加上 {value}")
            self.general_registers[register1] += value
        c = a+b
        if c & 1 == 1:
            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"] = 0
        else:
            self.flags["SF"] = 1

    @staticmethod
    def hlt():
        # 停止执行
        print("停止执行")
        exit(0)

    def print_registers(self):
        # 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}")
        print("专用寄存器状态:")
        for reg, val in self.special_registers.items():
            print(f"{reg}: {val}")
        print("标志寄存器状态:")
        for reg, val in self.flags.items():
            print(f"{reg}: {val}")
        print('\n')
```

```
    def update_buses(self, operation, register, value):
        # 更新总线状态
        if operation == 'MOV':
            self.address_bus['source'] = f"{register}"
            self.address_bus['destination'] = f"memory[{self.special_registers['IP']}]"
            self.data_bus['data'] = value
            self.control_bus['read'] = True
            self.control_bus['write'] = False
        elif operation == 'ADD':
            self.address_bus['source'] = f"{register}"
            self.address_bus['destination'] = f"{register}"
            self.data_bus['data'] = value
            self.control_bus['read'] = False
            self.control_bus['write'] = True
        elif operation == 'HLT':
            self.address_bus['source'] = ""
            self.address_bus['destination'] = ""
            self.data_bus['data'] = 0
            self.control_bus['read'] = False
            self.control_bus['write'] = False

        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')


    def biu_run(self):
        # 运行biu
        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")


    def eu_run(self):
        # 运行eu
        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.memory = ["MOV AX 1", "ADD AX 2", "HLT"]
    #创建两个线程
    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，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

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(),add(),hlt()

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

汤太阳负责执行单元(EU)函数

赵横负责总线接口单元(BIU)函数

运行结果

```
/usr/bin/python3 /Users/tangtaiyang/DevelopTool_FileManage/PycharmProjects/cpu_build/第三次改进/cpu_3.py
biu开始执行
正在从内存地址 0 获取指令: MOV AX 1
将指令传入指令队列所用时间为1.4000000000000123e-05秒
正在从内存地址 1 获取指令: ADD AX 2
将指令传入指令队列所用时间为9.999999999996123e-06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5.999999999999062e-06秒


eu开始执行
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1.0999999999997123e-05
将 1 移动到寄存器 AX
正在执行指令 MOV...所用时间为1.2000000000005062e-05
寄存器 AX 的新值为 1
通用寄存器状态:
AX: 1
AH: 0
AL: 0
BX: 0
BH: 0
BL: 0
CX: 0
CH: 0
CL: 0
DX: 0
DH: 0
DL: 0
SP: 0
DP: 0
DI: 0
SI: 0
专用寄存器状态:
```

```
CS: 0
DS: 0
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


总线状态:
地址总线:
source: AX
destination: memory[3]
数据总线:
data: 1
控制总线:
read: True
write: False


正在解码指令: ADD AX 2 为 操作码 ADD, 寄存器 AX, 值 2 所用时间为0.0
给寄存器 AX 加上 2
正在执行指令 ADD...所用时间为4.000000000004e-06
寄存器 AX 的新值为 3
通用寄存器状态:
AX: 3
AH: 0
AL: 0
```

```
BX: 0
BH: 0
BL: 0
CX: 0
CH: 0
CL: 0
DX: 0
DH: 0
DL: 0
SP: 0
DP: 0
DI: 0
SI: 0
专用寄存器状态:
CS: 0
DS: 0
SS: 0
ES: 0
IP: 3
标志寄存器状态:
CF: 0
PF: 1
AF: 0
ZF: 0
SF: 0
OF: 0


总线状态:
地址总线:
source: AX
destination: AX
数据总线:
```

```
data: 2
控制总线:
read: False
write: True


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

进程已结束, 退出代码为 0
```