python代码

```
class Simple8086:
    def __init__(self):
        self.memory = [""] * 256  # 简化的内存
        self.registers = {
            'AX': 0,  # 累加器寄存器，通常用于算术运算
            'BX': 0,  # 基址寄存器，常用于数据寻址
            'CX': 0,  # 计数寄存器，通常用于循环计数
            'DX': 0,  # 数据寄存器，可以作为辅助寄存器使用
            'SP': 0,  # 堆栈指针
            'BP': 0,  # 基址指针
            'SI': 0,  # 源变址寄存器
            'DI': 0,  # 目的变址寄存器
            'IR': '0',  # 指令寄存器
            'PC': 0   # 程序计数器
        }  # 简化的寄存器
        self.address_bus = {}  # 地址总线
        self.data_bus = {}     # 数据总线
        self.control_bus = {}  # 控制总线
        self.instructions = {
            'MOV': self.mov,
            'ADD': self.add,
            'HLT': self.hlt
        }
    def fetch(self):
        # 从内存中获取指令
        instruction = self.memory[self.registers['PC']]
        print(f"正在从内存地址 {self.registers['PC']} 获取指令: {instruction}")
        self.registers['PC'] += 1
        return instruction

    def decode(self, instruction):
        # 解码指令
        parts = instruction.split(' ')
        op = parts[0]
        reg = parts[1] if len(parts) >= 2 else None
        val = parts[2] if len(parts) == 3 else None
        print(f"正在解码指令: {instruction} 为 操作码 {op}, 寄存器 {reg}, 值 {val}")
        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:
            print(f"正在执行指令 {operation}...")
            if operation == 'HLT':
                self.instructions[operation]()
            elif value is not None:
                if value not in self.registers:
                    value = int(value)
                self.instructions[operation](register, value)
            else:
                self.instructions[operation](register)
            if register:
```

```
                print(f"寄存器 {register} 的新值为 {self.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.registers:
            print(f"将 {value} 移动到寄存器 {register1}")
            self.registers[register1] = self.registers[value]
        else:
            print(f"将 {value} 移动到寄存器 {register1}")
            self.registers[register1] = value


    def add(self, register1,value):
        # 给指定寄存器中的值加上一个数
        if value in self.registers:
            print(f"给寄存器 {register1} 加上 {value}")
            self.registers[register1] += self.registers[value]
        else:
            print(f"给寄存器 {register1} 加上 {value}")
            self.registers[register1] += value


    def hlt(self):
        # 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("寄存器状态:")
        for reg, val in self.registers.items():
            print(f"{reg}: {val}")

    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.registers['PC']}]"
            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}")

    def run(self):
        # 开始运行
        print("开始执行...")
        while True:
            self.registers['IR'] = self.fetch()
            op, reg, val = self.decode(self.registers['IR'])
            self.execute(op, reg, val)
            print(f"当前寄存器状态: {self.registers}\n")
            if op == 'HLT':
                break

if __name__ == "__main__":
    # 初始化处理器并设置一些内存内容
    cpu = Simple8086()
    # 简化指令集: MOV AX, 1; ADD AX, 2; HLT
    cpu.memory = ["MOV AX 1","MOV BX 2", "ADD AX BX", "HLT"]
    cpu.run()
```

设计思路

先定义简单的结构体Simple8086
属性 memory 内存，registers 寄存器，address_bus 地址总线 data_bus 数据总线
control_bus 控制总线 instructions 指令

之后再定义结构体相关函数
fetch() 获取指令 decode() 解码指令 execute() 执行指令 mov() 转移指令 add() 相加指令
hlt() 停止指令 print_registers() 打印寄存器信息 updata_buses() 更新总线 run() 运行

测试结果

执行的指令为 "MOV AX 1","MOV BX 2", "ADD AX BX", "HLT"

```
开始执行...
正在从内存地址 0 获取指令: MOV AX 1
正在解码指令: MOV AX 1 为 操作码 MOV, 寄存器 AX, 值 1
正在执行指令 MOV...
将 1 移动到寄存器 AX
寄存器 AX 的新值为 1
寄存器状态:
AX: 1
BX: 0
CX: 0
DX: 0
SP: 0
BP: 0
SI: 0
DI: 0
IR: MOV AX 1
PC: 1
总线状态:
地址总线:
source: AX
destination: memory[1]
数据总线:
data: 1
控制总线:
read: True
write: False
当前寄存器状态: {'AX': 1, 'BX': 0, 'CX': 0, 'DX': 0, 'SP': 0, 'BP': 0, 'SI': 0, 'DI': 0,
 'IR': 'MOV AX 1', 'PC': 1}
```

```
正在从内存地址 1 获取指令: MOV BX 2
正在解码指令: MOV BX 2 为 操作码 MOV, 寄存器 BX, 值 2
正在执行指令 MOV...
将 2 移动到寄存器 BX
寄存器 BX 的新值为 2
寄存器状态:
AX: 1
BX: 2
CX: 0
DX: 0
SP: 0
BP: 0
SI: 0
DI: 0
IR: MOV BX 2
PC: 2
总线状态:
地址总线:
source: BX
destination: memory[2]
数据总线:
data: 2
控制总线:
read: True
write: False
当前寄存器状态: {'AX': 1, 'BX': 2, 'CX': 0, 'DX': 0, 'SP': 0, 'BP': 0, 'SI': 0, 'DI': 0,
 'IR': 'MOV BX 2', 'PC': 2}
```

```
正在从内存地址 2 获取指令: ADD AX BX
正在解码指令: ADD AX BX 为 操作码 ADD, 寄存器 AX, 值 BX
正在执行指令 ADD...
给寄存器 AX 加上 BX
寄存器 AX 的新值为 3
寄存器状态:
AX: 3
BX: 2
CX: 0
DX: 0
SP: 0
BP: 0
SI: 0
DI: 0
IR: ADD AX BX
PC: 3
总线状态:
地址总线:
source: AX
destination: AX
数据总线:
data: BX
控制总线:
read: False
write: True
当前寄存器状态: {'AX': 3, 'BX': 2, 'CX': 0, 'DX': 0, 'SP': 0, 'BP': 0, 'SI': 0, 'DI': 0,
 'IR': 'ADD AX BX', 'PC': 3}
```

```
正在从内存地址 3 获取指令: HLT
正在解码指令: HLT 为 操作码 HLT, 寄存器 None, 值 None
正在执行指令 HLT...
停止执行
```