**Topic: Fetch–Decode–Execute cycle**

Reading Time: 15 mins

**·        Note\* Highlight important/core points while reading**

·        Read the content and write the answers given in the document in your words, to get the solid grip on topic.

**The Fetch–Decode–Execute Cycle**

The Fetch–Decode–Execute cycle (also called the instruction cycle) is a fundamental process by which a CPU carries out instructions in a computer program. This cycle is repeated continuously while the computer is powered on and enables the CPU to execute each instruction stored in memory.

**1. Fetch Stage**

* **Purpose**: The fetch stage is responsible for retrieving an instruction from memory so the CPU can process it.
* **Working**:
  1. **Program Counter (PC)**: The Program Counter (PC) holds the address of the next instruction to be executed.
  2. **Memory Address Register (MAR)**: The address in the PC is copied to the Memory Address Register (MAR), which is used to locate the instruction in memory.
  3. **Memory Data Register (MDR)**: The instruction at the specified address is fetched from memory and placed in the Memory Data Register (MDR).
  4. **Instruction Register (IR)**: The fetched instruction is then moved to the Instruction Register (IR), and the PC is incremented to point to the next instruction.
* **Example**: If the PC holds the address of the next instruction, this address is used to locate and fetch the instruction for processing.

**2. Decode Stage**

* **Purpose**: The decode stage interprets the fetched instruction, so the CPU knows what actions to perform.
* **Working**:
  1. **Control Unit (CU)**: The Control Unit (CU) examines the instruction in the Instruction Register.
  2. **Opcode and Operands**: The instruction is broken down into two main parts: the opcode (which specifies the operation, e.g., ADD or LOAD) and the operands (which specify the data or location involved).
  3. **Determine Action**: Based on the opcode, the Control Unit determines which parts of the CPU need to be used and sets up necessary signals for the next stage.
* **Example**: If the instruction is "ADD," the decode stage determines that an addition operation is needed and prepares the ALU (Arithmetic Logic Unit) for this task.

**3. Execute Stage**

* **Purpose**: The execute stage carries out the instruction using the CPU's various components.
* **Working**:
  1. **Arithmetic Logic Unit (ALU)**: If the instruction requires arithmetic or logic operations (e.g., ADD, SUBTRACT), the ALU performs the operation using the operands.
  2. **Memory or I/O Access**: If the instruction involves data movement (e.g., LOAD or STORE), the CPU may read from or write data to memory.
  3. **Update Registers**: The result of the execution is stored in a register, and the CPU prepares for the next cycle.
* **Example**: For an "ADD" instruction, the ALU will add two numbers provided as operands, and the result will be stored in a register.

**4. Cycle Repeat**

* **Purpose**: After executing one instruction, the CPU repeats the cycle for the next instruction, continuously following the fetch–decode–execute cycle as long as the computer is running.
* **Example**: Once one instruction is executed, the PC points to the next instruction, and the cycle repeats.

### ****A-Rated Questions/Answers By Examiner****

**Q1: What is the purpose of the fetch stage in the fetch–decode–execute cycle?**  
**Answer**: The fetch stage retrieves the next instruction from memory by using the address in the Program Counter (PC), which is then placed in the Instruction Register (IR) for decoding.

**Q2: How does the CPU decode an instruction during the decode stage?**  
**Answer**: In the decode stage, the Control Unit examines the instruction in the Instruction Register, separates the opcode and operands, and prepares the CPU components required for execution.

**Q3: What role does the Arithmetic Logic Unit (ALU) play in the execute stage?**  
**Answer**: The ALU performs arithmetic or logic operations on the operands, such as addition or subtraction, as specified by the instruction during the execute stage.

**Q4: Explain the role of the Program Counter (PC) in the fetch–decode–execute cycle.**  
**Answer**: The Program Counter (PC) holds the address of the next instruction to be executed. It is updated after each instruction fetch to point to the subsequent instruction.

**Q5: What happens to the Program Counter (PC) after an instruction is executed?**  
**Answer**: After an instruction is executed, the Program Counter (PC) is incremented to point to the next instruction, allowing the fetch–decode–execute cycle to continue.

### Write your Answers on your Notebook and Verify it on Next Screen

**Q6: How does the Memory Address Register (MAR) function during the fetch stage, and why is it important?**

**Q7: What is the purpose of the Memory Data Register (MDR) in the fetch–decode–execute cycle?**

**Q8: Describe the role of the Control Unit (CU) in coordinating the fetch–decode–execute cycle.**

**Q9: Why is it necessary for the Program Counter (PC) to be incremented after each fetch stage?**

**Q10: What would happen if the Control Unit (CU) misinterpreted the opcode during the decode stage?**

**6. Answer:** The MAR holds the address of the instruction to be fetched from memory, which is essential for directing the CPU to the exact memory location to retrieve the required instruction.

**7. Answer:** The MDR temporarily holds the data or instruction fetched from memory, acting as a buffer between the memory and the CPU to ensure accurate data transfer during the cycle.

**8. Answer:** The CU interprets the instruction in the decode stage, determines the necessary CPU components, and generates control signals to coordinate the operations in each stage of the cycle.

**9. Answer:** Incrementing the PC ensures that the CPU moves sequentially through instructions in memory, allowing it to process a program in the correct order.

**10. Answer:** If the CU misinterprets the opcode, it could initiate the wrong operation or use incorrect CPU components, leading to erroneous or unintended program behavior.