**Memory**

Memory is where information is processed and stored in a microcomputer system. Each memory circuit element can store one bit (0 or 1). Memory circuits are organized into groups of 8 bits, forming a byte. Memory bytes have fixed, unique addresses, while their contents (the data stored) are not unique and depend on the current data.

**Address vs. Contents**

* **Address**: Unique and fixed for each memory byte, akin to a house address.
* **Contents**: The actual data stored in the memory byte, which can change over time.

**Memory Byte Addressing**

For a processor using 20-bit addresses:

* Total addressable memory = 2202^{20} bytes = 1 MB.

**Memory Word**

* A word in memory consists of 2 bytes.
* For example, a memory word at address 2 consists of data from addresses 2 and 3.
* The lower address represents the word's starting point.

**Buses**

* **Address Bus**: Carries memory addresses.
* **Data Bus**: Transports actual data.
* **Control Bus**: Sends control signals to manage read/write operations.

**Execution Unit (EU)**

* Contains the Arithmetic Logic Unit (ALU) for performing arithmetic and logical operations.
* Stores data in registers (e.g., AX, BX).
* Holds temporary and FLAG registers for computations.

**Bus Interface Unit (BIU)**

* Manages communication between the EU and memory or I/O devices.
* Handles address, data, and control signals.
* Prefetches instructions into a queue to enhance performance.

**How EU and BIU Work Together**

* EU executes instructions while BIU fetches the next ones, placing them in an instruction queue.
* This prefetching speeds up processing unless BIU needs to fetch specific data for the EU.

**I/O Ports**

* **Function**: Interfaces between the CPU and I/O devices.
* **Serial I/O**: Transfers data one bit at a time; suitable for slower devices like keyboards.
* **Parallel I/O**: Transfers multiple bits (8 or 16) simultaneously; ideal for faster devices like disk drives.

**Fetch-Execute Cycle**

1. **Fetch**:
   * Retrieve the instruction from memory.
   * Decode the instruction and fetch necessary data.
2. **Execute**:
   * Perform the operation.
   * Store results if needed.

**High-Level vs. Assembly Language**

* **High-Level Language**:
  + Closer to natural language.
  + Requires compilers to convert to machine language.
  + Easier and faster for programmers.
* **Assembly Language**:
  + Uses symbolic names for operations and memory.
  + Requires an assembler for machine language conversion.
  + Offers finer control, suitable for specific hardware operations.

**Timing and Timing Task**

* Clock circuits ensure orderly execution of tasks via clock pulses.
* **Clock Period**: Time between pulses.
* **Clock Speed**: Pulses per second (measured in MHz or GHz).
* Example: A 2.3 GHz processor generates 2.3×1092.3 \times 10^9 pulses per second.

**Difference Between High-Level and Assembly Language**

|  |  |  |
| --- | --- | --- |
| **Feature** | **High-Level Language** | **Assembly Language** |
| **Abstraction Level** | Closer to human-readable, natural language. | Closer to machine language, uses symbolic instructions. |
| **Ease of Use** | Easier to write, read, and debug. | Requires understanding of hardware details. |
| **Portability** | Platform-independent; can run on different machines with a compiler. | Platform-dependent; specific to the hardware. |
| **Execution Speed** | Slower due to added layers of abstraction. | Faster as it directly interacts with hardware. |
| **Tools Needed** | Requires a compiler to translate code to machine language. | Requires an assembler to convert to machine language. |
| **Control over Hardware** | Limited control over hardware and memory. | Fine-grained control over hardware and memory. |
| **Examples** | Python, Java, C++ | Intel x86 Assembly, ARM Assembly |

**Difference Between Address and Contents**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Address** | **Contents** |
| **Definition** | Unique identifier for a memory location. | The actual data stored in that location. |
| **Nature** | Fixed and unique for each memory byte. | Changes depending on the stored data. |
| **Representation** | Typically in hexadecimal or binary format. | Represented as a value (e.g., a byte). |
| **Relation** | Acts like the "house address." | Represents the "occupants" of the house. |
| **Processor Dependency** | Number of address bits depends on the processor (e.g., 20-bit for Intel 8086). | Always 8 bits for a memory byte. |

**Example**:

* Address: 0x00FF
* Contents at 0x00FF: 10101100 (in binary) or AC (in hexadecimal).