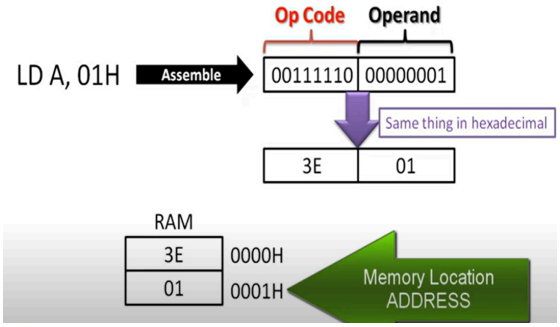
# Instruction Codes

In computers, **instruction codes** are compact groups of bits that direct the computer to perform specific tasks. Each computer has unique instruction codes and addressing methods. The **operation code (opcode)** in an instruction serves as a command, guiding the computer to execute actions like addition, subtraction, shifting, and control flow.



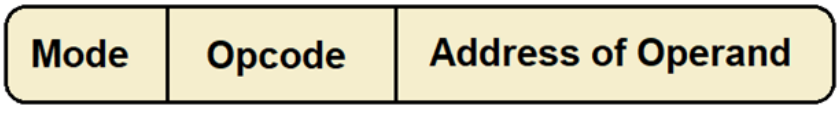

Instruction codes and addresses are generally divided into **opcodes** and **addresses**. Opcodes define the type of operation to execute, while addresses specify where to access data, whether in a register or memory. **Operands** are the specific components within instructions that indicate the data the computer needs to carry out the operation.

# What are Instruction Codes and Operands

Instruction codes are sequences of bits that command the computer to execute specific operations. Each instruction is structured into groups called **fields**, each serving a unique purpose:

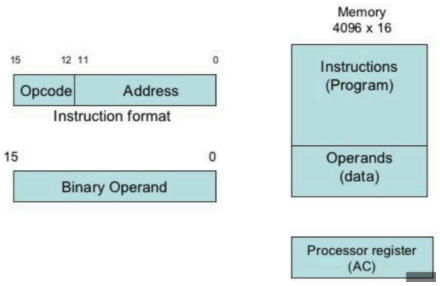
- **Operation Code (Opcode) Field:** Specifies the operation to be executed, such as arithmetic, logical, or control tasks. The opcode essentially tells the CPU what action to perform.
- **Address Field:** Indicates the location of the operand, whether in a specific register or a memory address, pointing to where the required data is stored.
- **Mode Field:** Determines the addressing mode, explaining how the address field should be interpreted—whether directly, indirectly, or through other methods.

Together, these fields enable the CPU to interpret and execute instructions effectively.



# Structure of an Instruction Code

The instruction code is also known as an instruction set. It is a collection of binary codes. It represents the operations that a computer processor can perform. The structure of an instruction code can vary. It depends on the architecture of the processor but generally consists of the following parts:




- **Opcode:** Specifies the operation the processor should perform, like addition, subtraction, multiplication, or division.
- **Operand(s):** Represents the data the operation acts upon, which may be a value in a register, a memory address, or a constant embedded in the instruction.
- **Addressing Mode:** Indicates how to interpret the operand(s), such as a direct memory address, an indirect address (memory address stored in a register), or an immediate value.
- **Prefixes or Modifiers:** Some instruction sets include prefixes or modifiers to adjust the instruction's behavior, like conditional execution or repeated operation based on specific conditions.

# Types of Instruction Code

Instruction codes are like the computer's special commands that tell it what to do. These codes can be grouped by how many pieces of data they use, what kind of job they do, and how they find the data they need. Here are the main types of instruction codes:

**1. Data Transfer Instructions:** Move data between registers, memory, or input/output devices.

**2 Arithmetic Instructions**: Perform mathematical operations like addition and subtraction.

**3. Logical Instructions**: Carry out bitwise operations such as AND, OR, and NOT.

**4. Branching (Control) Instructions:** Control program flow by jumping to different parts of code based on conditions.

**5. Input/Output Instructions:** Manage communication between the CPU and peripheral devices.

**6. Comparison Instructions:** Compare data values and set flags for conditional operations.

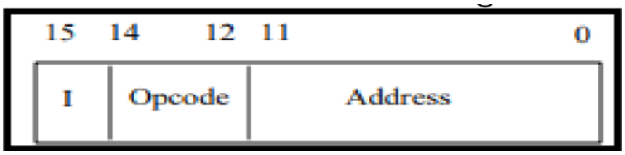
# Opcodes

An **opcode** is a set of bits that represents basic operations like add, subtract, multiply, complement, and shift. The number of bits needed for the opcode depends on how many functions the computer supports. For a total of ‘2^n’ possible operations, at least ‘n’ bits are required for the opcode. These operations are carried out on data stored in the processor’s registers or memory.

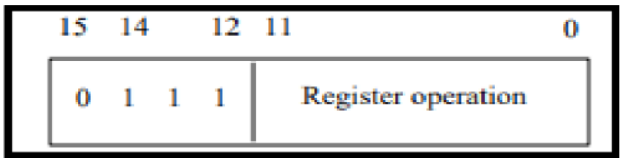
## Types of Opcodes

In a computer system, there are three main types of instruction formats, each defined by an operation code (opcode) that is 3 bits long, with the remaining 13 bits providing further specifications. The formats are as follows:

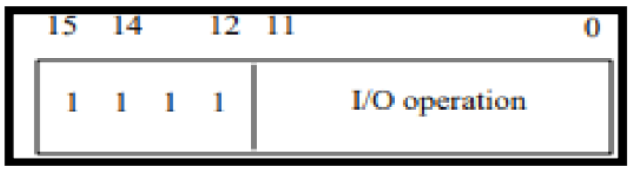
1. **Memory Reference Instruction:** This format consists of 12 bits to specify the memory address and 1 bit (I) to indicate the addressing mode. If the address is direct, the I bit is set to 0; if indirect, it is set to 1.




2. **Register Reference Instruction:** Identified by the opcode 111, which has a 0 in the leftmost bit, this format uses the remaining 12 bits to define the specific operation to be executed using registers.

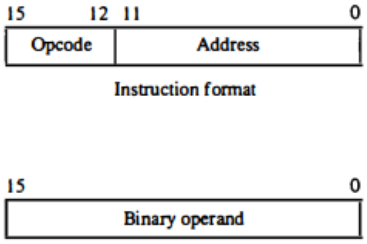



3. **Input-Output Instruction:** These instructions are recognized by the opcode 111 with a 1 in the leftmost bit. The remaining 12 bits detail the specific input-output operation to be performed.



# Stored Program Organizations

**Stored Program Organization** refers to the architecture design where both instructions (programs) and data are stored in memory. This concept, proposed by John von Neumann, allows the CPU to fetch instructions from memory sequentially and execute them. In this model, the program is treated like data, enabling dynamic changes to the instructions being executed.

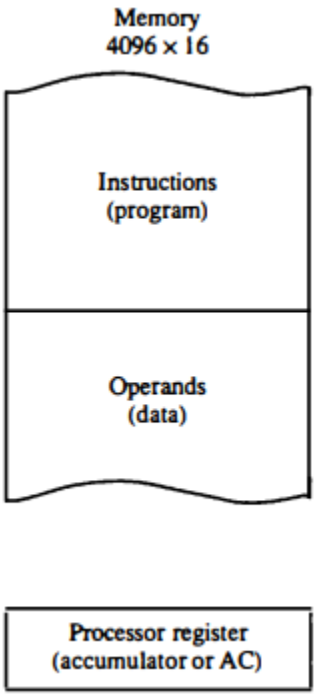



## Accumulator Register (AC)

The Accumulator Register (AC) is like a little helper in the computer that holds numbers while it works on them. Whenever the CPU needs to do math, it uses the AC to store the results temporarily. Think of it as a small notebook where the computer jots down calculations before putting them away.

## Effective Address (EA)

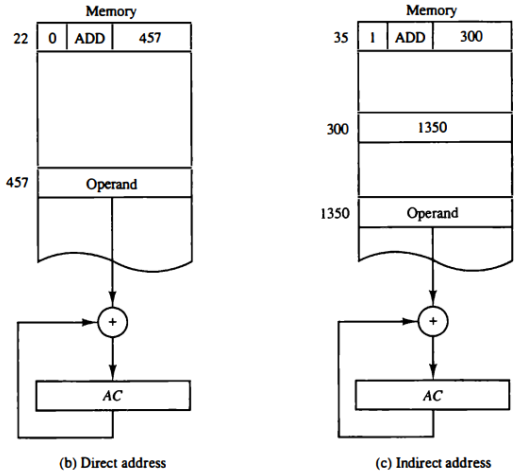
The Effective Address (EA) is basically the address of where the computer can find the data it needs to work with. It tells the CPU exactly where to look in the memory for the numbers or information required for a task. So, it's like a GPS for the CPU, guiding it to the right spot in memory.



# Address

We represent the memory address where a given instruction is built. We use an instruction code's address bits as an operand rather than an address. The instruction in such methods has an immediate operand. The command is directed to a direct address if the second portion contains an address.

In the second half, there is another choice, which includes the operand's address. It points to an oblique address. One bit might indicate whether the instruction code executes the direct or indirect address.




## Addressing Modes

We can mainly describe the address field for instruction in the following ways:

- **Direct Addressing**
  - ➢ Direct addressing is a straightforward way for a computer to find data. The instruction tells the CPU exactly where to look in memory, allowing quick access to the needed data. However, it has its limits, as it only works with fixed locations, making it less suitable for frequently changing data.
- **Indirect Addressing**
  - ➢ Indirect addressing is flexible because it points to a memory location that has the address of the actual data. The CPU first finds this address and then retrieves the data, which takes a bit longer. However, it's helpful for handling changing data and complex structures like lists or arrays.
- **Immediate operand**
  - ➢ Immediate operands are fixed numbers included directly in the instruction. This allows the computer to use them immediately without searching memory. For instance, in an instruction to add 5, the "5" is the immediate operand. This speeds up calculations since the computer doesn't need to look for the number; it can use it right away. They're great for simple tasks and help streamline processes.