MIPS

Types of statements:

In assembly language, there are three main types of statements:

1. **Executable Instructions**:
   - These generate actual **machine code** that the processor executes at runtime.
   - Instructions tell the processor what operations to perform, such as arithmetic, logic, or data movement (e.g., add, sub, lw).
   - Each instruction corresponds directly to a machine operation.
2. **Pseudo-Instructions and Macros**:
   - These are not real machine instructions but are **simplified commands** provided to make programming easier.
   - The assembler translates them into one or more actual machine instructions.
   - Example: In MIPS, move $t0, $t1 is a pseudo-instruction that is translated to add $t0, $t1, $zero.
3. **Assembler Directives**:
   - Directives provide **information to the assembler** during the translation process (e.g., how to organize the program, allocate memory, or define data).
   - They do not generate machine code and are **non-executable**.
   - Examples include .data (to define data segments) and .text (to define code segments).

Here's a breakdown of the **assembly language instruction format**:

1. **Label (optional)**:
   - A label marks a memory location, usually for branching or jumping.
   - It helps identify specific points in the code.
   - A label must end with a colon (:).
   - Labels are often used in both **data segments** (for variables) and **text segments** (for code).
2. Example: L1: is a label.
3. **Mnemonic**:
   - The mnemonic represents the **operation** or instruction for the CPU (e.g., add, sub, lw).
   - It tells the CPU what action to perform.
4. **Operands**:
   - These specify the **data** needed for the operation.
   - They can be **registers** (e.g., $t0), **memory addresses**, or **constants**.
   - Many instructions in MIPS have three operands, such as the destination register and two source registers or a register and an immediate value.
5. **Comment (optional)**:
   - After a #, comments can be added to explain the instruction.
   - Comments are ignored by the assembler but help programmers understand the code.

Example:

```
L1:  addiu $t0, $t0, 1    # increment $t0 by 1
```

- **Label**: L1: marks this line for reference.
- **Mnemonic**: addiu (add immediate unsigned).
- **Operands**: $t0, $t0, 1 (increments $t0 by 1).
- **Comment**: # increment $t0.

All initializers become binary data in memory