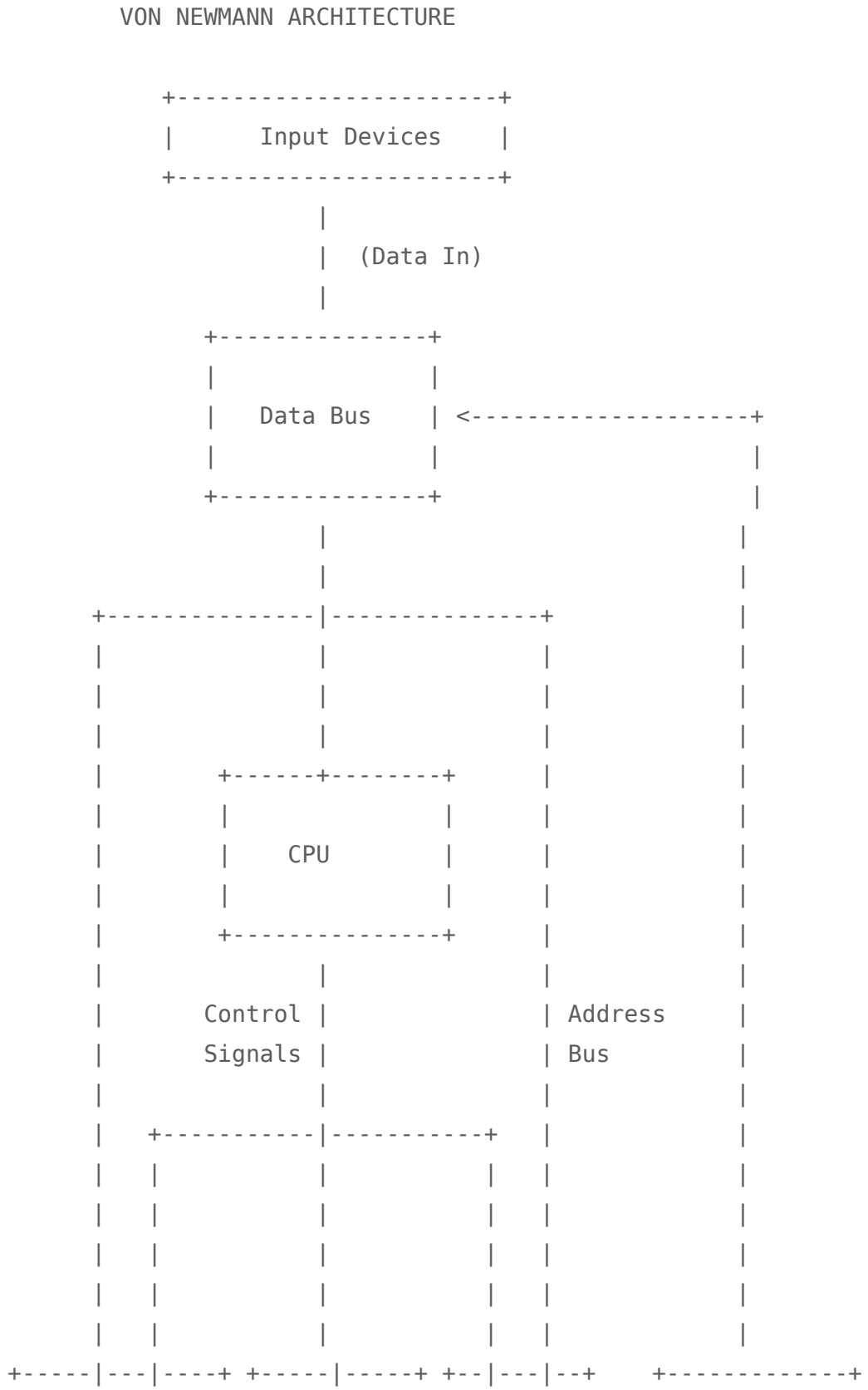
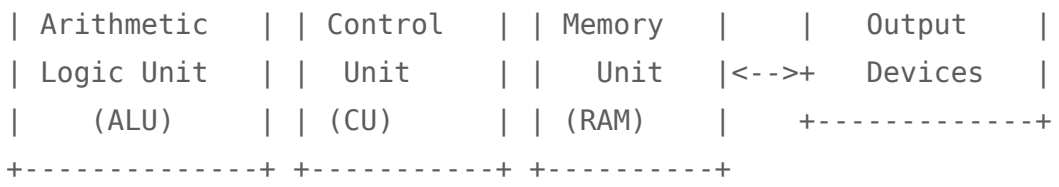
PSUC ASSIGNMENT

1. Draw a block diagram of Von Neuman architecture computer, with its majorcomponents showing the transfer of data amongst its units. What is meant by stored program concept?





- **Data Bus**: The central pathway for data transfer between components.
  - **Input Devices to Memory**: Data enters the system from input devices via the data bus and is stored in memory.
  - **CPU to Memory**: The CPU fetches data and instructions from memory over the data bus.
  - **ALU**: The ALU receives data from memory (or registers within the CPU) to perform calculations and returns the results to memory or to output devices.
  - **Memory to Output Devices**: Processed data or results are sent from memory to output devices via the data bus.
- **Address Bus**: Allows the CPU to specify the memory location of data or instructions it wants to access.
  - **CPU to Memory**: The CPU sends address information to memory, allowing it to read/write specific data locations.
- **Control Signals**: Managed by the Control Unit within the CPU.
  - **CPU to Memory and I/O**: The CU sends control signals to manage read/write operations, synchronizing data flow among all units.
  - **CPU to ALU**: The CU directs the ALU to perform operations on the data.

2. Explore the results of logical operators for the following below expressions, on declaration(s) int a = 11, b = 7, c = 12, result

(a) result = (a == b) && (c > b)

False (0)

(b) result = (a == b) && (c < b)

False (0)

(c) result = (a == b) || (c < b)

False (0)

(d) result = (a != b) || (c < b)

True (1)

(e) result = !(a != b)

False (0)

(f) result = !(a == b)

True (1)

(g) result=(a== 10 + 15 && c < 10)

False (1)

7. Explain the concept of storage classes in programming and write a program that demonstrates the use of at least three different storage classes.

In programming, storage classes define the scope (visibility), lifetime, and linkage of variables or functions. They essentially dictate where and how variables are stored, accessed, and retained during program execution.

In languages like C there are four main storage classes:
Automatic (auto): The default storage class for local variables within functions. These variables are allocated at the start of the block and deallocated when the block exits. They are accessible only within the block they are defined in (local scope).

Static (static): Variables defined with the static keyword retain their value even after the block exits, so they have a "lifetime" that spans the entire program execution. Their scope can be local to the block or file where they are defined.

External (extern): The extern keyword is used to declare a global variable or function in another file. This storage class allows access to variables across multiple files, giving them global visibility across files.

Register (register): This class hints to the compiler that the variable will be used frequently and, if possible, should be stored in a CPU register rather than RAM for faster access. However, it’s only a suggestion, and the compiler might ignore it.

9. Define and Differentiate between Formatted and Non-Formatted function and implicit vs explicit function in C.

(i) Formatted vs. Non-Formatted Functions in C

Formatted Functions:

- Formatted functions are functions that allow control over the format of input or output data.
- In C, these functions use specific format specifiers (like `%d`, `%f`, `%c`, etc.) to dictate how data should be read or printed.
- Examples of formatted functions are `printf()` and `scanf()`.

Non-Formatted Functions:

- Non-formatted functions do not provide control over data formatting; they handle data as a stream without specific format specifiers.
- In C, these functions are generally used to read or write raw data without interpreting it based on a specific format.
- Examples of non-formatted functions include `getchar()`, `putchar()`, `gets()`, and `puts()`.

(ii) Implicit vs. Explicit Type Casting

Implicit Type Casting:

- Also known as **automatic type casting**, it occurs when the compiler automatically converts one data type to another.
- Implicit casting is typically done when assigning a value of a smaller or lower-precision type to a larger or higher-precision type. For example, an `int` to a `float`.
- Since this is done automatically, there's no need for the programmer to specify any casting explicitly.

Explicit Type Casting:

- Also known as **manual type casting**, it occurs when the programmer explicitly specifies a data type conversion.
- Explicit casting is required when converting from a larger to a smaller data type, or when precision might be lost.
- This type of casting is done using the cast operator `(<type>)`.