Two-Pass Assembler:

A two-pass assembler makes two scans over the source code. The first pass gathers information like labels and symbols, while the second pass generates machine code using data from the first pass.

Significance of Symbol Table:

The symbol table stores information about symbols (such as labels and variables), including their locations, helping resolve addresses and references during assembly.

Assembler Directives EQU, ORIGIN:

EQU defines a constant value for a symbol.

ORIGIN sets the starting address of the program, adjusting the location counter.

Assembler Directives START, END, LTORG:

START specifies the starting address of the program.

END indicates the end of the source code.

LTORG loads literals in the code by allocating storage at the current location counter.

Use of POOLTAB and LITTAB:

POOLTAB (Pool Table) stores literal pools, marking groups of literals.

LITTAB (Literal Table) stores literal values and their addresses.

Handling of Literals in Pass I:

In Pass I, literals are added to the literal table (LITTAB) without assigning addresses until the LTORG or END directive is encountered.

Tasks Done in Pass I:

Tasks include scanning code, building the symbol and literal tables, managing location counters, and identifying forward references.

Error Handling in Pass I:

Syntax and semantic errors are identified, including undefined labels, invalid instructions, and memory address issues.

Variant Used in Implementation and Why:

The two-pass assembler is often used because it effectively handles forward references by resolving symbols in Pass II.

Intermediate Data Structures Designed and Implemented in Pass I:

Symbol Table, Literal Table, and Location Counter are essential structures in Pass I.

Format of Machine Code in Pass II:

The machine code format includes the opcode, operand addresses, and possibly any required immediate values.

Forward Reference and Resolution by Assembler:

A forward reference occurs when a label is used before being defined. The assembler resolves this by gathering information in Pass I and filling in addresses in Pass II.

Error Handling in Pass II:

Errors in Pass II include incorrect operand values, invalid opcodes, and unresolved references.

Difference Between IS, DL, and AD:

IS (Imperative Statements): Actual instructions executed by the machine.

DL (Declarative Statements): Directives for memory allocation.

AD (Assembler Directives): Commands for assembler processing, not for machine execution.

Tasks Done in Pass II:

Pass II generates machine code, resolves addresses, fills forward references, and handles literals.

Macro Processor:

A macro processor automates repetitive tasks by expanding macros into specific instructions in the source code.

Difference Between Macro and Function:

A macro is expanded inline during assembly, while a function is a separate entity invoked with parameters. Functions support runtime reuse; macros are compile-time.

Design of Pass-I of Macro-Processor (Flowchart):

Pass I processes macro definitions, stores them in tables (MNT, MDT), and identifies macro calls, without expanding them.

Data Structures Used in Pass-I:

Macro Name Table (MNT), Macro Definition Table (MDT), Argument Table (APT).

Data Structures in Pass-I:

MNT stores macro names and MDT indices; MDT stores macro body; APT handles arguments passed to macros.

Macro Expansion:

Macro expansion replaces each macro invocation with the corresponding block of code in the macro definition.

Purpose of Pass-2 in Two-Pass Macro Processor:

Pass 2 performs macro expansion, replacing each macro invocation with its corresponding code.

Positional Arguments:

Positional arguments are macro parameters identified by their position in the macro call, used in expanding macros.

Use of MDT-Index in MNT:

The MDT-index links each macro in MNT to its body in MDT, aiding in retrieval during expansion.

Types of CPU Schedulers:

Long-term (job scheduling), short-term (CPU scheduling), and medium-term (swapping for multitasking).

Difference Between Long and Short-Term Scheduling:

Long-term scheduling controls job entry into the system, while short-term assigns jobs to the CPU.

Logic of Program:

This is the structure and sequence of code operations needed to complete specific tasks or outputs.

Preemptive vs. Non-Preemptive Scheduling:

Preemptive allows interruption of running processes, while non-preemptive completes each job before switching.

Types of Scheduling Algorithms:

Algorithms include First-Come-First-Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), and Priority Scheduling.

Priority Scheduling and Starvation:

Lower-priority processes may starve if higher-priority processes continuously take precedence.

Goals of Scheduling:

Ensure fairness, maximize CPU utilization, minimize waiting time, response time, and turnaround time.

Best Scheduling Algorithm and Why:

The best algorithm depends on requirements. For balanced fairness and responsiveness, Round Robin is often effective.

Need of Allocating Blocks to Jobs:

Block allocation ensures jobs have memory, facilitating execution and multitasking.

Time Taken by Each Algorithm for Execution:

It varies depending on the task set, CPU burst times, and algorithm efficiency.

Need for Memory Placement Strategies:

Placement strategies optimize memory usage, reduce fragmentation, and improve performance.

Fragmentation:

Fragmentation is unused memory space, which can be internal (within partitions) or external (between partitions).

Memory Placement Strategies Example:

Strategies include First Fit (allocates the first available block), Best Fit (smallest sufficient block), and Worst Fit (largest available block).

Define Process and Need for Scheduling:

A process is a program in execution. Scheduling ensures processes access CPU resources efficiently.

Scheduling Criteria and Policies:

Criteria include CPU utilization, throughput, turnaround time, and waiting time. Policies depend on these factors and the system's needs.

Process States and Common Scheduling Algorithms:

States: New, Ready, Running, Waiting, Terminated.

FCFS: Processes in order of arrival.

SJF (Preemptive): Shortest task preemptively interrupts longer tasks.

Priority (Non-Preemptive): Tasks scheduled based on priority.

Round Robin (Preemptive): Each process receives equal CPU time in cycles