EXP_1

What is a two-pass assembler?
An assembler that scans input assembly code twice: Pass-I builds symbol and literal tables and creates an intermediate representation; Pass-II uses these tables to generate machine code.",
"Pass-I resolves addresses and declarations; Pass-II performs final translation using resolved addresses.

Why do we need Pass-I?
To collect symbols, literals, and compute addresses for forward references before generating machine code. Without Pass-I, forward references (jumps to labels defined later) cannot be resolved.

What is a symbol table?
A data structure mapping symbols (labels, variables) to addresses and attributes. Used in Pass-II to replace symbol names with numeric addresses.

What is the literal table?
A list of literals (constants written as =value) encountered; each gets an address when processed.
Literals are often placed in literal pools and assigned addresses during LTORG or END.

What is a pool table?
A table recording the starting indices of literal pools within the literal table. Each LTORG or END begins a new literal pool; pool table helps manage allocation.

What is an intermediate code?
A line-by-line representation produced by Pass-I that carries tokens and some encoding for Pass-II. It allows Pass-II to be simple and deterministic, using precomputed tables.

How are forward references handled?
Pass-I records symbols without addresses; their addresses are filled when the symbol is defined later. Pass-II expects the symbol table to contain final addresses; if unresolved, it's an error.

What is LTORG?
An assembler directive that forces allocation of current literal pool at that point. It causes the assembler to assign addresses to all literals in the current pool.

Difference between DL, IS, and AD? How are declarative statements like DB and DW treated?
DL entries reserve storage and may place constants in memory at LC. DB reserves one byte, DW reserves a word (two bytes in our example); they appear in object code as constants.

Explain a literal pool allocation flow.
Literals are collected during Pass-I until LTORG or END, then assigned addresses sequentially. Pool table tracks where each pool starts so allocation is localized.

How does Pass-II use the intermediate file?
It reads each intermediate entry, replaces symbol/literal references by addresses and emits encoded machine words. Intermediate encodes whether a line is IS/DL/AD and carries operands for final encoding.

Why are two passes simpler than one?
Two passes separate concerns: Pass-I resolves addresses; Pass-II generates code. A one-pass assembler must handle forward refs via complex mechanisms. Two-pass design is simpler to implement and reason about for teaching purposes.

Can an assembler be single-pass? Give an example technique.
Yes; using backpatching and placeholders for forward refs and fixing them later or using external linking. Two techniques: use relocation/linking or maintain lists of pending addresses to be patched.

What is backpatching?
Recording locations of pending references that must be filled once the symbol's address is known. Common technique in single-pass or compiling phases for jumps and forward references.

How are expressions in ORIGIN handled?
By evaluating expressions; in simple assemblers, only symbol+offset or numeric constants are supported. More complex assemblers support full arithmetic and relocation expressions.

What errors can be detected in Pass-I?
Syntactic errors, undefined directives, ill-formed operands, duplicate labels (if detected immediately). Some semantic errors like undefined symbols are only known after full pass.

What errors are detected in Pass-II?
Unresolved symbols or literals, invalid operand types for an opcode, address out of range. These become fatal for object code generation.

How would you extend this assembler to support macros?
Add a macro table; expand macros during an extra pass (macro expansion pass) before Pass-I. Macro definitions are stored and replaced inline when invoked.

What is the format of machine code produced?
Depends on target; in this demo: a simplistic textual opcode-reg-addr triple like '02 1 105'. Real assemblers produce binary object files with specific instruction encoding and relocation info.

How are literals represented in source?
Typically as =value (e.g., =5); they are placed in memory by the assembler and referenced by address. Literals permit immediate storage without named variables.

Why do assemblers support directives like START and END?
To control assembly process, initial address, literal pools, and to mark end of source. They don't generate machine instructions but affect symbol/literal allocation.

Explain what a relocation entry is (beyond this demo).
Metadata in object files to adjust addresses when the program is loaded at a different base address. Used by linkers/loaders to make code position-independent or relocate it correctly.

How would you add support for external symbols and linking?
Produce symbol table entries flagged as external and emit relocation information; linker resolves them later. Requires additional sections in object file (text, data, symbol, relocation tables).

What improvements would you make to this demo assembler?
Better expression parsing, full-size instruction encoding, support for sections, comments, error reporting, and macro support. Also support for multiple passes or single-pass with backpatching, and binary object output.

EXP_2

Two-Pass Macro Processor

A **macro processor** expands macros before assembly.

- **Pass-I** builds **Macro Name Table (MNT)**, **Macro Definition Table (MDT)**, and an **intermediate code file (ICF)** that removes macro definitions but keeps macro calls intact.
- **Pass-II** expands macro calls using the MNT and MDT to produce the final expanded code.

Data Structures

Data Structure	Description	Example
MNT (Macro Name Table)	Stores macro name and pointer to its first line in MDT	`{ "INCR": 1, "SWAP": 5 }`
MDT (Macro Definition Table)	Stores actual macro definition lines	`{1: "LDA &ARG1", 2: "ADD &ARG2", 3: "STA &ARG1", 4: "MEND"}`
ALA (Argument List Array)	Maintains mapping of formal parameters to actual parameters	`{ "&ARG1": "A", "&ARG2": "B" }`
Intermediate Code File (ICF)	Assembly code with macro definitions removed, ready for Pass-II	List of assembly lines (excluding macro definitions)

What is a macro processor?
A macro processor expands macros into source code before assembly. It automates repetitive code patterns using macro definitions.

What are the two passes in a macro processor?
Pass-I builds MNT and MDT; Pass-II expands macros using these tables. This separation allows handling of nested and forward macro calls.

What is MNT?
Macro Name Table – stores macro names and MDT indices. Helps locate macro definitions quickly in Pass-II.

What is MDT?
Macro Definition Table – stores actual macro body. It contains all lines of macro definitions sequentially.

What is ALA?
Argument List Array – maps formal to actual parameters. Used in Pass-II to replace parameters during expansion.

What does Pass-I remove from source?
All macro definitions. Resulting intermediate code contains only macro calls.

Why is Pass-I needed?
To store macros and remove definitions for later expansion. Pass-II uses this metadata for actual expansion.

What is MEND?
A keyword marking the end of a macro definition. Helps macro processor know where macro definition ends.

What is MACRO?
A directive marking the start of macro definition. The line after it contains the macro header.

What is the difference between macro and subroutine?
Macro is expanded inline; subroutine is called at runtime. Macros save time but increase code size.

What is the purpose of intermediate code file (ICF)?
Contains program without macro definitions but with macro calls. Simplifies Pass-II processing.

What happens in Pass-II?
Macro calls are replaced by corresponding expanded code. Uses MNT and MDT to substitute lines.

What is meant by "expanding a macro"?
Replacing macro call by equivalent sequence of instructions. Performed during Pass-II.

Can macros call other macros?
Yes, these are nested macros. The processor must handle recursive or nested expansions carefully.

Why use formal parameters in macros?
To make macros reusable for different arguments. Allows flexibility and abstraction.

What are actual parameters?
Values or variables passed to macros when invoked. Mapped to formals via ALA.

How are parameters substituted during expansion?
Using the ALA mapping formal → actual argument. Every occurrence of &ARG is replaced by its actual value.

Why do we need MNT and MDT both?
MNT provides macro name lookup; MDT stores definition. MNT acts as an index to MDT.

What is meant by macro expansion time?
The time during which macro calls are replaced by actual code. Happens before assembly, not at runtime.

What error occurs if a macro is called but not defined?
Undefined macro error. Processor cannot expand an unknown macro.

Can macros contain assembler directives?
Yes, macros may include directives like MOV, ADD, etc. They are expanded like normal statements.

What is nested macro expansion?
When a macro definition or call contains another macro call. Handled by recursive expansion.

Difference between macro processor and preprocessor?
Macro processor expands assembly macros; preprocessor works for high-level languages (like C). Conceptually similar but language-specific.

What are the advantages of macros?
Reduces code writing, ensures uniformity, and speeds up coding. Especially useful for repetitive instruction sequences.

What are the disadvantages of macros?
Increases code size due to inline expansion. Unlike subroutines, macros don't save memory.

What is a positional parameter?
Parameter identified by its position in argument list. Example: &ARG1 in macro header.
What is a keyword parameter?
Parameter identified by name=value format. Adds flexibility but needs name lookup.

How can macros improve program maintainability?
Changes in one macro definition update all uses. Centralized modification.

What does MDT pointer in MNT indicate?
The first line in MDT where the macro's definition starts. Used for locating macro quickly during expansion.

What improvements can be made to this macro processor?
Add nested macro handling, keyword parameters, and macro libraries. Real assemblers support these advanced features.

EXP_4

Synchronization Problems occur when multiple processes or threads share resources concurrently.
Common problems:

1. **Producer–Consumer Problem**
2. **Dining Philosophers Problem**
3. **Readers–Writers Problem**
4. **Sleeping Barber Problem**

We use **Mutex locks** and **Semaphores** to control access and ensure mutual exclusion and synchronization.

Mutex (Mutual Exclusion)	A lock that allows only one thread to access the critical section at a time.

Semaphore A signaling mechanism; maintains a count to allow limited access to resources.

Critical Section Part of code where shared resources are accessed.

Producer Generates items and adds them to the buffer.

Consumer Takes items from the buffer.

Buffer

What is process synchronization?
Process synchronization is coordination among concurrent processes to ensure correct execution when accessing shared resources.
Explanation: Prevents race conditions.

What is a race condition?
A situation where the output depends on the sequence or timing of threads.
Explanation: Happens when multiple threads access shared data simultaneously without proper locking.

What is a semaphore?
A synchronization primitive used to control access to shared resources.
Explanation: It has an integer value that is decremented (wait) and incremented (signal).

Difference between binary semaphore and counting semaphore?
Binary semaphore = 0 or 1 (like a mutex). Counting semaphore can take any non-negative integer value.
Explanation: Counting semaphore allows multiple accesses.

What is a mutex lock?
A locking mechanism that allows only one thread to access the critical section.
Explanation: Used for mutual exclusion.

What does sem_wait() **do?**
Decrements the semaphore. If the value is zero, it blocks the calling process.
Explanation: Used before entering critical section.

What does sem_post() **do?**
Increments the semaphore and wakes up a waiting process.
Explanation: Signals that a resource is available.

What are the classical problems of synchronization?
Producer-Consumer, Dining Philosophers, Readers-Writers, and Sleeping Barber.
Explanation: These model real-world resource-sharing issues.

What is the buffer in producer-consumer?
A finite queue shared between producer and consumer.
Explanation: Synchronization ensures it doesn't overflow or underflow.

Why use both semaphore and mutex together?
Semaphore controls resource count, mutex ensures mutual exclusion during access.
Explanation: Combination prevents data corruption.

What is a critical section problem?
Ensuring only one process accesses a shared resource at a time.
Explanation: Mutex helps solve this.

What happens if semaphores are not initialized correctly?
It can cause deadlocks or starvation.
Explanation: Initial values determine resource availability.

What is deadlock?
A condition where processes wait indefinitely for each other to release resources.
Explanation: Circular waiting causes deadlocks.

What is starvation?
A process waits indefinitely due to others continuously accessing resources.
Explanation: Fair scheduling prevents this.

What is a bounded buffer problem?
The producer-consumer problem with limited buffer size.
Explanation: Must prevent overfill and underflow.

Which synchronization primitive is faster: Mutex or Semaphore?
Mutex is faster for single resource; semaphore better for multiple.
Explanation: Mutex has simpler operations.

Can a mutex be unlocked by a different thread?

No. Only the thread that locked it can unlock it.
Explanation: Ensures ownership safety.

What happens if sem_post() is called before sem_wait()?
It increases the semaphore value, indicating resource availability.
Explanation: Usually safe, but may cause logic errors.

What are the four necessary conditions for deadlock?
Mutual exclusion, hold and wait, no preemption, circular wait.
Explanation: Breaking any prevents deadlock.

What is the difference between wait() and sleep()?
wait() in semaphore context is for resource synchronization; sleep() pauses a thread.
Explanation: Wait affects semaphore value; sleep doesn't.

What is the Readers–Writers problem?
Synchronizing access to shared data where multiple readers can read but only one writer can write.
Explanation: Ensures consistency of shared data.

What is the Dining Philosophers problem?
Models resource allocation with five philosophers sharing five forks.
Explanation: Prevents deadlock using semaphores.

How can we prevent deadlock in Dining Philosophers?
Use odd-even philosopher strategies or limit simultaneous eaters.
Explanation: Break circular wait condition.

What is priority inversion?
Low-priority thread holds a lock needed by a high-priority thread.
Explanation: Solved using priority inheritance.

What is the Sleeping Barber problem?
Models synchronization between barber and customers with limited chairs.
Explanation: Uses semaphores for sleeping and waiting.

Why do we use pthread_join() **in the program?**
To ensure main thread waits for child threads to finish.
Explanation: Prevents premature termination.

What is sem_init() **used for?**
Initializes a semaphore with a specific value.
Explanation: Must be done before use.

What is the advantage of semaphores over busy waiting?
Semaphores block threads instead of looping.
Explanation: Saves CPU time.

Can semaphores be used for inter-process communication (IPC)?
Yes. System V and POSIX semaphores can synchronize processes.
Explanation: Used in OS-level resource sharing.

Why is synchronization essential in operating systems?
To ensure correct and predictable execution of concurrent processes.
Explanation: Avoids inconsistency and corruption.

EXP_5

CFS: Non-preemptive Executes in order of arrival

SJF: (Preemptive) Preemptive Always chooses process with smallest remaining time

Priority: Non-preemptive Highest priority (lowest number) first

Round Robin : Preemptive Each process gets equal CPU time (quantum)

What is CPU Scheduling?
Answer: Deciding which process runs next on the CPU.
Explanation: The scheduler selects processes from the ready queue.

What is the goal of scheduling?
To maximize CPU utilization and throughput, minimize waiting and turnaround time.
Explanation: Balances efficiency and fairness.

What is FCFS Scheduling?
Processes are executed in the order of their arrival.
Explanation: Simple but can cause convoy effect.

What is convoy effect?
When short processes wait for a long process to complete in FCFS.
Explanation: Causes poor average waiting time.

What is SJF Scheduling?
Process with the shortest burst time is executed first.
Explanation: Minimizes average waiting time.

Difference between SJF Preemptive and Non-Preemptive?
Preemptive allows interruption when a shorter process arrives.
Explanation: Non-preemptive runs process till completion.

What is another name for SJF (Preemptive)?
Shortest Remaining Time First (SRTF).
Explanation: Based on remaining burst time.

What is Priority Scheduling?
Each process has a priority; CPU is allocated to the highest priority process.
Explanation: Can be preemptive or non-preemptive.

What is starvation in scheduling?
Low-priority processes may never get CPU time.
Explanation: Happens in priority scheduling.

How can we prevent starvation?
By using Aging — gradually increasing priority of waiting processes.
Explanation: Ensures fairness.

What is Round Robin Scheduling?
Each process gets CPU for a fixed time quantum.
Explanation: Preemptive and fair.

What is time quantum?
Fixed CPU time allocated to each process in Round Robin.
Explanation: Shorter quantum → more context switches.

What happens if quantum is very large?
Round Robin behaves like FCFS.
Explanation: Less preemption.

What happens if quantum is very small?
Too many context switches, more overhead.
Explanation: Reduces efficiency.

Define Waiting Time (WT).
Total time a process waits in the ready queue.
Formula: WT = TAT – BT

Define Turnaround Time (TAT).
Time between arrival and completion.
Formula: TAT = CT – AT

Define Response Time.
Time from arrival until first CPU response.
Explanation: Important in interactive systems.

Which algorithm gives minimum average waiting time?
SJF (Preemptive).
Explanation: Proven optimal for minimizing waiting time.

Which scheduling algorithm is used in real-time systems?
Priority scheduling.
Explanation: Urgent tasks have higher priority.

What is preemptive scheduling?
Process can be interrupted to give CPU to another process.
Explanation: Ensures responsiveness.

What is non-preemptive scheduling?
Once a process starts, it runs until completion.
Explanation: Simpler but less responsive.

Which algorithm is best for time-sharing systems?
Round Robin.
Explanation: Ensures equal CPU share.

What is throughput?
Number of processes completed per unit time.
Explanation: Indicator of system performance.

What is CPU utilization?
Percentage of time CPU is actively executing.
Explanation: Should be maximized.

What is context switching?
Saving and loading process states during a switch.
Explanation: Happens in preemptive scheduling.

What is the main disadvantage of FCFS?
Poor performance for short processes.
Explanation: Long waiting time for small jobs.

Why is SJF difficult to implement?
Burst time must be known in advance.
Explanation: Usually estimated.

What is turnaround time dependent on?
Arrival time, burst time, and scheduling order.
Explanation: TAT = WT + BT.

What is aging in priority scheduling?
Increasing priority of waiting processes over time.
Explanation: Prevents starvation.

Which algorithms are preemptive in this program?
SJF (Preemptive) and Round Robin.
Explanation: Both allow process interruption.

EXP_7

Algorithm	Replacement Policy	Complexity	Belady's Anomaly	Performance
FIFO	Replace oldest page	O(1)	Yes	Fair
LRU	Replace least recently used	O(n)	No	Good
Optimal	Replace page not used soonest	$O(n^2)$	No	

What is a page in OS?
A fixed-length block of memory that forms a unit of data for memory management.

What is paging?
Technique of dividing process memory into equal-sized pages and main memory into frames.

What causes a page fault?
When a page requested by CPU is not found in main memory.

What is a page replacement algorithm?
Determines which page to replace when a page fault occurs.

Name some common page replacement algorithms.
FIFO, LRU, Optimal, LFU, Second-Chance.

What is FIFO page replacement?
The oldest page in memory (first loaded) is replaced first.

Disadvantage of FIFO?
May replace frequently used pages, causing Belady's anomaly.

What is Belady's anomaly?
In FIFO, increasing number of frames can increase page faults.

What is Optimal page replacement?
Replaces the page that will not be used for the longest future time.

Why is the Optimal algorithm not practical?
It requires future knowledge of page references, which is impossible in real systems.

What is LRU page replacement?
Replaces the page that was least recently used.

How does LRU approximate the Optimal algorithm?
LRU assumes pages used recently will likely be used again soon.

What is the main drawback of LRU?
Requires maintaining time or stack of page references — high overhead.

Which algorithm may show Belady's anomaly?
FIFO, but not LRU or Optimal.

What data structures are used in LRU implementation?
Time counter array or stack to track recency.

What data structure can be used for FIFO?
Queue.

What data structure can be used for Optimal?
Array to look ahead in future references.

What is the purpose of frames?
Fixed-size slots in main memory where pages are loaded.

How do you calculate page fault rate?
Page Faults / Total Page References × 100%.

Which algorithm gives the minimum page faults?
Optimal, because it makes the best possible decision.

What is the advantage of LRU over FIFO?
LRU considers recent usage, thus better performance.

What is the disadvantage of Optimal algorithm?
It is theoretical, not implementable in real-time.

What happens if all frames are full and a new page comes?
One existing page must be replaced according to the algorithm.

What is thrashing?
Condition where excessive paging causes CPU utilization to drop.

How can thrashing be reduced?
By increasing the degree of multiprogramming or physical memory.

What is the working set model?
Set of pages actively used by a process at a given time.

What is demand paging?
Pages are loaded only when they are needed.

What is pre-paging?
Loading some pages in advance to reduce page faults.

Which algorithm is easiest to implement?
FIFO.

Why is LRU widely used in modern systems?
It balances performance and simplicity using hardware support (e.g., reference bits).