Assignment No :- 2

1) Define the term Macro. Explain the term Lexical expansion and Semantic expansion with respect to macro.

→ A macro is a named sequence of source statements that can be invoked by name and expanded into the program by a macro processor.

Lexical expansion:

- Lexical expansion is pure textual substitution the macro processor replaces the macro invocation with the macro body, substituting actual parameter text for for formal parameter names.
- It does not interpret or type-check the contents- it treats the macro body as text.
- Example:

```
Macro
        # define SQR(x) ((x) * (x))
```

Semantic expansion

- Semantic expansion means the macro processor or transformer looks at the meaning of parameters or surrounding context, possibly evaluates expressions, enforces types or generates code differently depending on semantic condition.

2) Explain following terms with macro

i) Expansion-time variables

→ Variables whose values are determined and used during macro expansion, not at program run time. They let the macro processor track state across expansions or produce unique names/labels.

- Example: a macro that creates uniquely numbered labels using a counter & COUNTER so nested expansions won't clash:

```
MACRO UNIQUE-PRINT &MSG
    LBL_&COUNTER: ; COUNTER is expansion-
                    time variable
    PRINT &MSG
    INC &COUNTER
MEND
```

ii) Conditional assembly

→ Directives or constructs inside macros or assembler source that allow the assembler to include or exclude code based on compile-time conditions. These conditions are evaluated during assembly / macro expansion.

- Typical directives: IF, IFDEF, IFNDEF, ELSE, ENDIF, or the preprocessor.

EX:

```
MACRO MOV &DEST, &SRC, &MODE = IM
   IF &MODE EQ IM
       MOV &DEST, #&SRC ; immediate move
   ELSE
       MOV &DEST, &SRC
   ENDIF
MEND
```

iii) Keyword and positional parameters

- parameters passed by position / order. The first actual argument maps to the first formal parameter, etc.
- Advantages:
  1) positional : concise.
  2) keyword : clearer, allow defaults, allow arguments in any order.

- EX:

```
MACRO SWAP &DEST, &SRC, &TEMP = R0
    MOV &TEMP, &DEST
    MOV &DEST, &SRC
    MOV &SRC, &TEMP
MEND
```

3) Show contents of different tables and output after processing of macro definition.

→ Macro definition (source):

```
MACRO MOVV &DEST, &SRC, &MODE=IM
  IF &MODE EQ IM
     MOV &DEST
  ELSE
     MOV, &DEST, &SRC
  ENDIF
MEND
```

1) MNT

MacroName	MDT	KPDT	PParms	Keyparams
MOVV	1	1	2	1

2) KPDTAB

Index	paramName	Default
1	&MODE	IM

3) MDT

line #	Text
1	IF &MODE EQ IM
2	MOV &DEST, # &SRC
3	ELSE

4	MOV &DEST, &SRC
5	ENDIF
6	MEND

- Invocation 1:

Formal	Actual
&DEST	R1
&SRC	5
&MODE	IM

- Invocation 2:

Formal	Actual
&DEST	R2
&SRC	R3
&MODE	REG

4) What is Lexical analysis ? Data structure used in Lexical analysis.

→ Lexical analysis is the first phase of a compiler. It reads the raw source code and groups characters into lexemes and converts them into tokens.

i) Symbol Table (SYMTAB)

- It holds Label / Symbol name – value / address, length, relocation / section
- PassII when an identifier is recoginized, it's entered into the symbol table.

ii) Input Buffer
- A structure containing the source characters often implemented as two half-buffers with sentinel/EOF for efficient reading and backtracking.

iii) Lexeme/Token Buffer
- Temporary storage for the characters of the current lexeme while the scanner recognizes a token. After recognition the lexeme is converted to token.

iv) DFA
- The scanner implements finite automata
- The DFA transition table drives recognition. This table can be stored as arrays or compressed structures.

v) Keyword/Reserved word Table
- A lookup table mapping lexemes like if, while, return to keyword tokens. Often implemented as a hash table or trie.
- When an identifier is recognized it is first checked against this table to see if it's a keyword.

vi) Literal Table
- Storage for constant values and string literals encountered; tokens reference entries in the literal table.

5) Explain the phases of compiler with suitable example.

→ A typical compiler is organized into a sequence of phases. I'll describe each phase

- Ex: a = b + c * 3;

1) Lexical Analysis (scanner)
Input: source characters
Output: stream of tokens: IDENT(a), =, IDENT(b), +, IDENT(c), *, Number(3), ;
Removes whitespace/comments, recognize lexemes, populate symbol/literal table.

2) Syntax Analysis.
Input: token stream
Output: parse tree
Task: Check syntactic structure according to grammar (LL, LR, etc)

3) Semantic Analysis
Input: AST and symbol table
output: annotated AST or intermediate representation (IR) with type information, scope checks,
Task: Type checking, scope resolution.

4) Intermediate code Generation
Input: annotated AST
Task: produce a simple, machine independent representation suitable for optimization.

5) Code optimization
- Improve performance and/or size: constant folding, dead code elimination, common subexpression elimination etc.

Assignment No: 03

1) What is Loader ? Enlist the basic function of Loader.

→ Loader - It loads the program into main memory.

- A loader is a system program that places an executable program into memory and prepares it for execution.
- Basic functions of a loader.

1) program relocation/Address binding:
- Adjust addresses in the program so instructions and data refer to the right memory locations.

2) Loading - copy code and data from the object module into main memory.

3) Symbol resolution/Linking - Resolve external references between modules by consulting a symbol table.

4) Allocation of memory - Reserve memory space for program segments and for (shared/stack/heap as needed.

5) Relocation modification - Modify addresses in instruction operands and data areas using relocation information

6) Starting execution - Transfer control to the program's entry address.

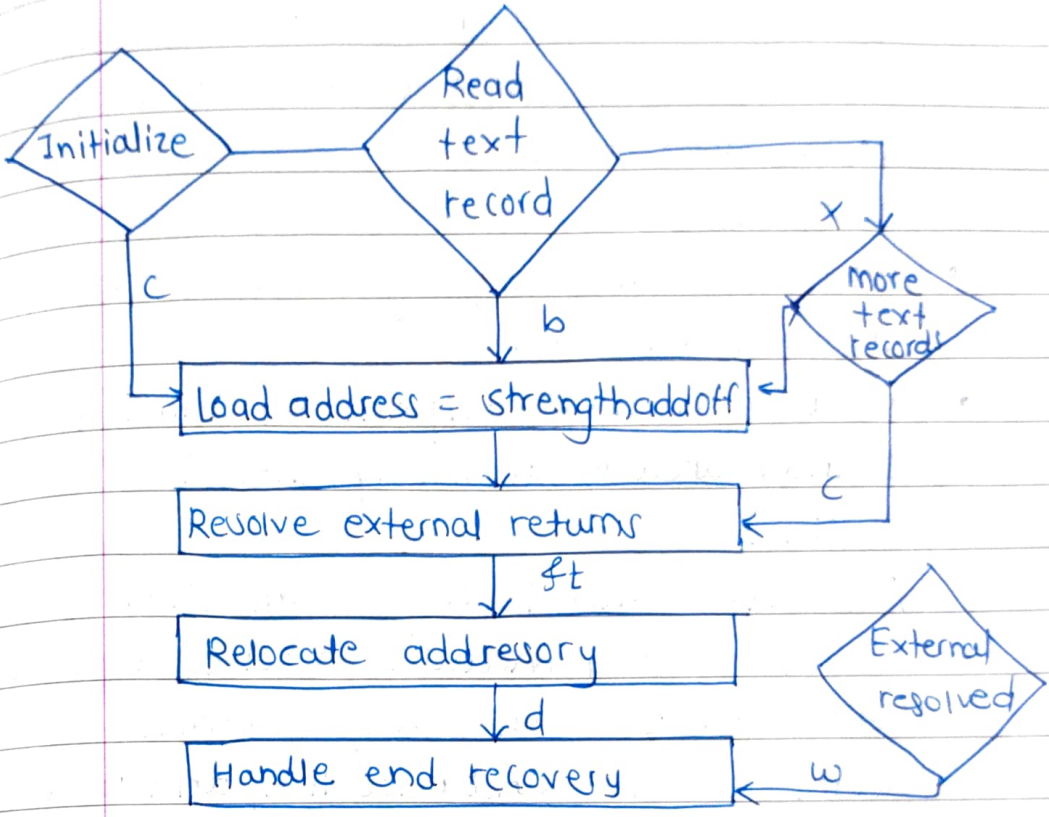
7) Error checking/reporting - Report unresolved symbol, memory overflow, relocation errors.

2) Comparative points: Absolute loader vs Relocating loader.

→

Aspect	Absolute Loader	Relocating Loader
Address binding time	At assembly / compile time	At load time
Can load at different memory locations	No - Object code has fixed addresses	Yes - Can load program anywhere in memory by relocating addresses
Relocation information required	No (not used)	Yes - requires relocation / modification records
Flexibility	Poor - different programs may require fixed distinct locations.	Good - Supports multiprogramming & dynamic memory allocation.
Complexity	Simple	More complex
Example usage	Very early systems, firmware	Modern loaders, OS loaders.

3) Flowchart for pass II of Direct Linking Loader



- Reading a word from memory
- Checking if its an address (indirect addressing).
- Resolving that address
- Continuing the routine
- Repeating until the end of the stack is reached.

4) Explain overlay structure and Linkage Editor.

→ Divide program into overlays - hierarchical groups of code where only the currently needed overlays are loaded into a fixed overlay area in memory.

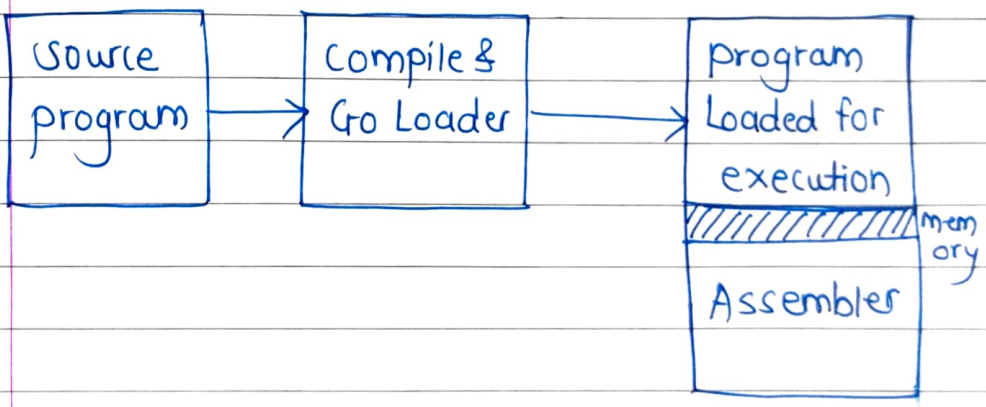
- Advantages
  - Allows execution of programs larger than available memory.
  - Reduces memory footprint by sharing overlay area for mutually exclusive routines.
- A Linkage editor is a program that combines object modules, resolves symbolic references between them, performs address binding and relocation, and produces a load module or executable.

1) Symbol resolution : combines symbol definitions and references; build global symbol table.
2) Relocation : Adjust addresses according to assigned load addresses; patch relocation entries.
3) Combining modules : Merge sections from different object modules into a single program image.

4) Library Search : Find and include needed routines from libraries
5) produce output : Create a load module
6) Generate relocation & debug info : For later loaders / debuggers.

5] Explain Compile & Go loader with example and disadvantages.

→ Compile & Go is a strategy where the source code is compiled / assembled and the output is immediately loaded into memory and executed - the generated object code is not saved to disk as a permanent object file.



- Advantages :

- Simplicity - quick for small program or learning environments.
- Convenient for debugging - immediate run after assemble / compile.

- Disadvantages

1) No separate object file

2) poor support for multi-module programs

3) No incremental linking/loading

4) Security and stability

5) Memory wage.

Assignment No: 4

1) What is an Operating System? Explain Functions and types of OS

→ An operating system (OS) is a system software that manages hardware and software resources of a computer and provides common services for computer programs.

- primary functions of an OS

1) process management - Create, schedule, synchronize, and terminate processes.
2) Memory management - allocate / deallocate primary memory, managing virtual memory and paging / segmentation.

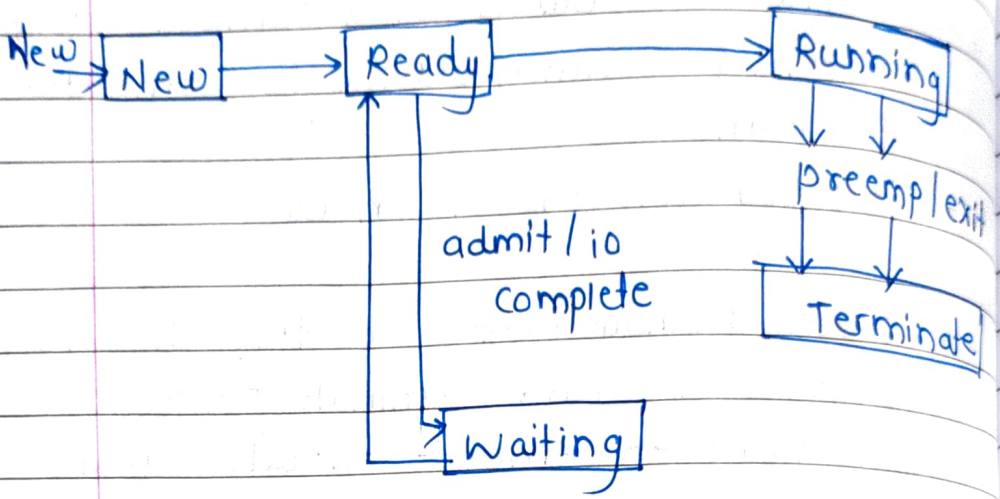
- Types of operation system

1) Batch OS - groups jobs into batches
2) Time-Sharing / Multi-user OS - multiple users share CPU via time slices.
3) Multiprogramming / Multi-tasking OS.
4) Real time OS (RTOS) - guarantees response within strict timing constraints.
5) Network OS - provides services to manage network resources
6) Embedded OS - specialized for devices
7) Mobile OS - Android, iOS - optimized for mobile devices.
8) Distributed OS.

2) Draw and explain process state transition. Write short notes on threads.

→

process state transition



- New - process is being created.
- Ready - process is in memory and ready to run.
- Running - process currently executing on the CPU.
- Waiting (Blocked) - process waiting for I/O or event.
- Terminated - finished execution (exit) resources reclaimed.

- Types of threads
  1) User-level threads
  2) Kernel-level threads

3) Draw and explain process control Block

→ The process Control Block (PCB) is the data structure used by the OS to store all information about a process.

- The Scheduler uses PCB to switch processes and manage process states.
- PID – unique identifier for the process.
- process state – Current state
- program counter - address of next instruction to execute.
- CPU registers – values to restore when process runs again.
- CPU scheduling information - priority, scheduling parameters, pointers to scheduling queues.
- Memory management info – pointers to page tables or segment tables, base-limit registers.
- Accounting information – CPU usage, time limits, accounting data for billing.
- I/O status information – list of open files, I/O devices allocated to process
- parent/child links – for process hierarchy and wait/exit handling.
- When context switch occurs: Save Current process's registers & PC into PCB; load next process's PCB values into CPU registers and PC.

4] Explain different types of Scheduling methods with an example.

→ Types of Scheduling Methods:

1) First Come First Serve (FCFS):
- processes are scheduled in the order they arrive.
- Example: Suppose P1 = 24ms, P2 = 3ms, P3 = 3ms → Average waiting time = high

2) Shortest Job Next (SJN or SJF):
- process with shortest burst time is executed first.
- Example: If jobs (P1 = 6, P2 = 8, P3 = 7, P4 = 3), then P4 runs first.

3) Round Robin (RR):
- Each process gets CPU for a fixed time slice (quantum).
- Example: quantum = 4ms → all processes run cyclically.

4) priority Scheduling:
- Each process has a priority; higher priority gets CPU First.
- Ex: P1 (priority = 1), P2 (priority = 3) → P2 executes first.

5) Multilevel queue Scheduling:
- processes are divided into groups and scheduled separately.

Q5] Explain the process scheduling algorithms with suitable example.

→ 1) FCFS

- Example : P1 = 10ms, P2 = 5ms, P3 = 8ms.
- Waiting Time : P1 = 0, P2 = 10, P3 = 15
  → Avg WT = (0 + 10 + 15) / 3 = 8.33 ms.

2) SJF (Shortest Job First)

- Example : P1 = 6ms, P2 = 8ms, P3 = 7ms, P4 = 3ms.
- Order : P4 → P1 → P3 → P2
- Avg WT is less than FCFS

3) Round Robin (RR)

- Example : P1 = 10ms, P2 = 5ms, P3 = 8ms, quantum = 4.
- CPU cycles rotate among processes until completion.

4) Priority Scheduling :

- Example : P1 = 10ms (prio = 3),
  P2 = 5ms (prio = 1),
  P3 = 8ms (prio = 2)
- order : P2 → P3 → P1

1719

by the CPU.

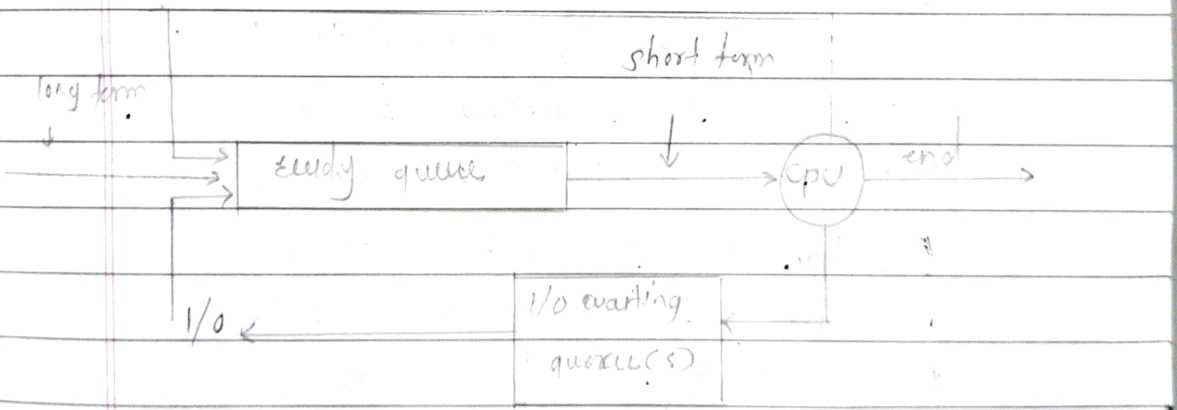
Q. 3. Draw & explain Process control Block.

Process ID
State
pointer
priority
program Counter
CPU registers
I/O information
Accounting information
etc....

→ The Process scheduling State. The State of the process in terms of "ready"

4) Explain different types of scheduling method with an example.

- Long term shedular
- Short Term shedulee
- Medium Term shedulee.
- Long term Shedulee :- It is also called a job schedulee. A long term schedulee determines which programs are admitted to the system for processing. It select processes from the queue and loads them into memory for execution.
- Short Term shedulee - It is also called as CPU shedular. Its main objective is to increase system performance in accordance with the choosen set of criteria. It is change of ready state running state of the process. CPU shedulee select a process among the processes that are ready to execute and allocates CPU to one of them.



- Medium Term Schedulee : medium term scheduling is a part of Swapping. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium term schedulee