WINTER– 18 EXAMINATION

Subject Name: System Programming **Model Answer** Subject Code: 17517

Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills.
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

<table>
<tr><th>Q. No.</th><th>Sub Q. N.</th><th>Answer</th><th>Marking Scheme</th></tr>
<tr><td>1</td><td>a</td><td>Attempt any THREE :</td><td>12 M</td></tr>
<tr><td></td><td>1</td><td>Write two advantages and disadvantages of absolute loader.</td><td>4 M</td></tr>
<tr><td></td><td>Ans</td><td>Advantages:

1 Simplest form of loader
2 Makes maximum core (memory) available for user.
3 One can determine exact location of program in core.
Disadvantages:

1. The programmer has to specify the address to the assembler that where the program is to be loaded.
2. It is very difficult to relocate in case of multiple subroutines.
3. Programmer has to remember the address of each subroutine and use that explicitly.</td><td>Any 2 Advantages 2 Marks; Any 2 Disadvantages 2 Marks</td></tr>
<tr><td></td><td>2</td><td>What is the difference between :
i) Processor and procedure.
ii) Multiprocessing and multiprogramming.</td><td>4 M</td></tr>
<tr><td></td><td>Ans</td><td>i) Processor and Procedure

<table><tr><th>Processor</th><th>Procedure</th></tr><tr><td>Processor is an element which processes is an active entity i.e. process</td><td>Procedure is a set of instruction which is used to perform certain task.</td></tr></table></td><td>Any 2 points of differentiation 1 Marks each; any relevant answer shall be considered</td></tr>
</table>

		Processor understands and interprets meaning of process	Procedure defines process.
		It executes procedures	It generates result after execution.

ii) Multiprocessing and Multiprogramming

Multiprocessing	**Multiprogramming**
It utilizes multiple CPUs	It utilizes single CPU.
It permits parallel processing.	Context switching takes place.
Multiprocessing refers to processing of multiple processes at same time by multiple CPUs.	Multiprogramming keeps several programs in main memory at the same time and execute them concurrently utilizing single CPU
Less time taken to process the jobs	More Time taken to process the jobs.

Any 2 points of differentiation 1 Marks each; any relevant answer shall be considered

3 **Describe linear search with suitable example.** — 4 M

Ans

Linear search is an algorithm in which an element is search in given data structure in sequential manner. The searching begins from one end and it will check each element in given data structure for the availability of key element. The algorithms compare each element with key element. If element exist then it displays location of element or else appropriate error message will be displayed. Since each element is checked for availability of desired element this algorithm runs on data structure which is not sorted. It has best case time complexity as O(1), with average and worst case time complexity as O(N).

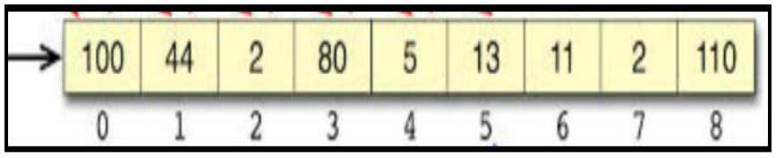
Advantage:-

- Simple to implement.
- Works efficient on small data structure.

Disadvantage: -

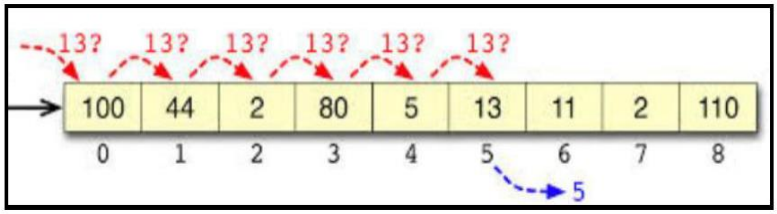
- Slow in execution.
- As the data structure increases efficiency of algorithm reduces.

Example: Consider the following data structure



Now we want to search element 13 in given data structure, the algorithm search sequentially as follows:

Description 2 Marks; Example 2 Marks

		 Element 13 Found at 5th location.	
	4	**Write three tasks of lexical analysis phase of compiler. List databases involved in it.**	4 M
	Ans	The three tasks of the lexical analysis phase are: 1. To parse the source program into the basic elements or tokens or lexemes of the language 2. To build a literal table and an identifier table. 3. To build a uniform symbol table. Databases used in lexical analysis are: ➢ Source program ➢ Terminal table ➢ Literal table: ➢ Identifier table: ➢ Uniform Symbol table	List of 3 task 2 Marks; List of any 4 databases 2 Marks
	b	**Attempt any ONE :**	**6 M**
	1	**What is system software? Write different goals of system software.**	6 M
	Ans	These are the programs that help in the effective execution of the application programs and allow the application programmer to focus on the application to be developed without concerning about the internal detail of the system. e.g. Assembler Macro-processor Loader Linker Compiler Editor Interpreter Operating System An assembler is a program that accepts as input an assembly language program (source) and produces its machine language equivalent (object code). Compilers are the system programs that accept people-like languages and translate them into machine language. Loader is system program that prepare machine language programs for execution. Macro processors allow programmers to use abbreviation. Operating system and file system allow flexible storing and retrieval of information. The productivity of each computer is heavily dependent upon the effectiveness, efficiency and sophistication of the systems programs. **Goal of System software** ➢ To achieve efficient use of available resources. ➢ To achieve efficient performance of the system. ➢ To make effective execution of general user program. ➢ To make available new better facilities.	Description of System software 3 Marks; Any 3 Goals 1 Mark each

		➢ User convenience - provide convenient methods of using a computer system. ➢ Non-interference - prevent interference in the activities of its user.	
	2	**What is Macro Instruction? Explain conditional macro with an example.**	6 M
	Ans	Macro is used to give single line abbreviation to group of lines which are repeatedly used in program. These statements are combined and kept in macro. Whenever such single line abbreviation is encountered macro processor expands replaces this abbreviation with associated group of lines. Macro Processor is a program that lets you define the code that is reused many times giving it a specific Macro name and reuse the code by just writing the Macro name only. Structure of Macro: MACRO MACRO_NAME … MACRO BODY … MEND **Conditional Macro Expansion**: Two important macro processor pseudo-ops, AIF and AGO, permit conditional reordering of the sequence of macro expansion. This allows conditional selection of the machine instructions that appear in expansions of macro call. AIF is conditional branch pseudo-o; it performs an arithmetic test and branches only if the tested condition is true. The AGO is an unconditional branch pseudo ops or „go to" statement. It specifies a label appearing on some other statement in the macro instruction definition; the macro processor continues sequential processing of instruction with the indicated statement. These statements are directives to the macro processor and do not appear in macro expansion. Example : Consider the following program. .	Macro Instruction explanation: 2 Marks; Example 1 Mark Conditional Macro 2 Marks; Example 1 Mark

```
Loop 1    A1, DATA 1
          A2, DATA 2
          A3, DATA 3
          .
          .
Loop 2    A1, DATA 3
          A2, DATA 2
          .
          .
Loop 3    A1, DATA1
          .
          .
          DATA 1 DC F'5'
          DATA 2 DC F'10'
          DATA 3 DC F'15'
```

In the below example, the operands, labels and the number of instructions generated change in each sequence. The program can written as follows:-

```
          .
          .
          .
          MACRO
&ARG0     VARY    &COUNT,&ARG1,&ARG2,&ARG3
&ARG0     A       1,&ARG1
          AIF     (&COUNT EQ 1).FINI
          A       2,&ARG2
          AIF     (&COUNT EQ 2).FINI
          A       3,&ARG3
.FINI     MEND
          .
          .
          .
LOOP1     VARY    3,DATA1,DATA2,DATA3
          .
          .
          .
LOOP2     VARY    2,DATA3,DATA2
          .
          .
          .
LOOP3     VARY    1,DATA1
          .
          .
          .
DATA1     DC      F'5'
DATA2     DC      F'10'
DATA3     DC      F'15'
```

EXPANDED SOURCE

```
          .
          .
          .
LOOP1     A   1,DATA1
          A   2,DATA2
          A   3,DATA3
          .
LOOP2     A   1,DATA3
          A   2,DATA2
          .
          .
LOOP3     A   1,DATA1
```

2		**Attempt any TWO :**	**16 M**
	1	**Draw and explain the use of database by assembler passes.**	8 M
	Ans	Pass 1 data bases: 1 Input source program. 2 A Location Counter (LC), used to keep track of each instruction's location. 3 A table, the Machine-Operation Table (MOT) that indicates the symbolic mnemonic for each instruction and its length (two, four, or six bytes).	Pass 1 database any 4 ,1 Mark each; Pass 2 database any 4, 1 Mark each

6-bytes per entry

Mnemonic op-code (4-bytes) (characters)	Binary op-code (1-byte) (hexadecimal)	Instruction length (2-bits) (binary)	Instruction format (3-bits) (binary)	Not used in this design (3-bits)
"A*bbb*"	5A	10	001	
"AH*bb*"	4A	10	001	
"AL*bb*"	5E	10	001	
"ALR*b*"	1E	01	000	
"AR*bb*"	1A	01	000	
...	...	...	...	
"MVC*b*"	D2	11	100	
...	...	...	...	

b ~ represents the character "blank"

Codes:

Instruction length

01 = 1 half-words = 2 bytes
10 = 2 half-words = 4 bytes
11 = 3 half-words = 6 bytes

Instruction format

000 = RR
001 = RX
010 = RS
011 = SI
100 = SS

4 A table, the Pseudo-Operation Table (POT) that indicates the symbolic mnemonic and action to be taken for each pseudo-op in pass 1.

8-bytes per entry

Pseudo-op (5-bytes) (character)	Address of routine to process pseudo-op (3-bytes = 24 bit address)
"DROP*b*"	P1DROP
"END*bb*"	P1END
"EQU*bb*"	P1EQU
"START"	P1START
"USING"	P1USING

These are presumably labels of routines in pass 1; the table will actually contain the physical addresses.

5 A table, the Symbol Table (ST) that is used to store each label and its corresponding value.

14-bytes per entry

Symbol (8-bytes) (characters)	Value (4-bytes) (hexadecimal)	Length (1-byte) (hexadecimal)	Relocation (1-byte) (character)
"JOHN*bbbb*"	0000	01	"R"
"FOUR*bbbb*"	000C	04	"R"
"FIVE*bbbb*"	0010	04	"R"
"TEMP*bbbb*"	0014	04	"R"

6 A table, the Literal Table (LT), that is used to store each literal encountered and its corresponding assigned location.

14-bytes per entry

Symbol (8-bytes) (characters)	Value (4-bytes) (hexadecimal)	Length (1-byte) (hexadecimal)	Relocation (1-byte) (character)
"JOHNbbbb"	0000	01	"R"
"FOURbbbb"	000C	04	"R"
"FIVEbbbb"	0010	04	"R"
"TEMPbbbb"	0014	04	"R"

7 A copy of the input to be used later by passes 2. This may be stored in a secondary storage device, such as magnetic tape, disk, or drum, or the original source deck may be read by the assembler a second time for pass 2.

Pass 2 data bases:

1 Copy of source program.
2 A Location Counter (LC), used to keep track of each instruction's location.
3 A table, the Machine Operation Table (MOT), that indicates for each instruction: (a) symbolic mnemonic; (b) length; (c) binary machine op-code, and (d) format (e.g., RS, RX, SI).
4 A table, the Pseudo-Operation Table (POT), which indicates for each pseudo-op the symbolic mnemonic and the action to be taken in pass 2.
5 The Symbol Table (ST), prepared by pass 1, containing each label and its corresponding value.
6 A table, the Base Table (BT), that indicates which registers are currently specified as base registers by USING pseudo-ops and what are the specified contents of these registers.

4-bytes per entry

	Availability indicator (1-byte) (character)	Designated relative-address Contents of base register (3-bytes = 24-bit address) (hexadecimal)
1	"N"	–
2	"N"	–
:	:	
14	"N"	–
15	"Y"	00 00 00

15 entries

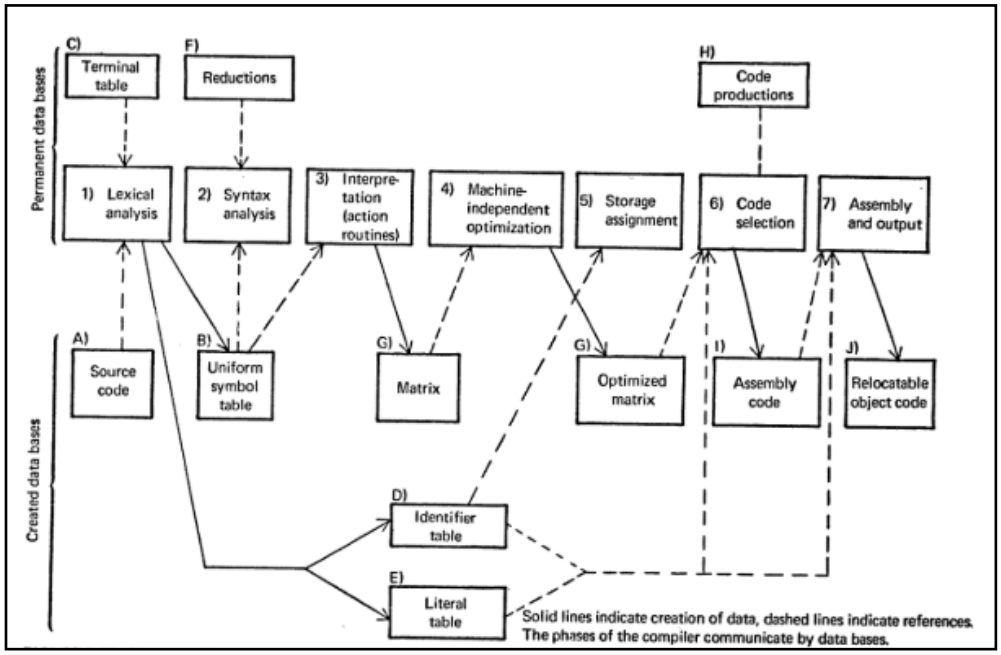
Code=

Availability

Y ~ register specified in USING pseudo-op

N ~ register never specified in USING pseudo-op or subsequently made unavailable by the DROP pseudo-op

7 A work-space, INST, that is used to hold each instruction as its various parts (e.g., binary op-code, register fields, length fields, displacement fields) are being assembled together.
8 A workspace, PRINT LINE, used to produce a printed listing.

		9 A workspace, PUNCH CARD, used prior to actual outputting for con-verting the assembled instructions into the format needed by the loader. 10 An output deck of assembled instructions in the format needed by the loader. Output in machine code to be needed by the loader.	
	2	**Draw the basic phases of compiler and explain each phase functions.**	8 M
	Ans	 **1. Lexical analysis–** Recognition of basic elements of creation of uniform symbols. **2. Syntax analysis–**Recognition of basic syntactic constructs through reductions. **3. Interpretation–** Definition of exact meaning, creation of matrix and tables by action routines. **4. Machine Independent Optimization–** Creation of more optimal matrix. **5. Storage Assignment–**Modification of identifier and literal tables. If makes entries in the matrix that allow code generation to create code that allocates dynamic storage and that also allow the assembly phases to reserve the proper amounts of STATIC storage. **6. Code Generation–** Use of macro processor to produce more optimal assembly code. **7. Assembly And Output–** Resolving symbolic addresses and generating machine language.	Diagram 5 Marks; Description of all phases 3 Marks Any relevant diagram with names of all passes shall be considered
	3	**Explain the working of address calculation sort with suitable example.**	8 M
	Ans	Address Calculation Sort (Hashing) This Sorting technique can be one of the fastest types of sorts if enough	Description 4 Marks; Example 4

storage space is available. The sorting is done by transforming the key into an address in the table that "represents" the key.

Marks

In this method a function f is applied to each key. The result of this function determines into which of the several sub file the record is to be placed.

The function should have the property that: if x <= y , f (x) <= f (y), Such a function is called order preserving. An item is placed into a sub file in correct sequence by placing sorting method – simple insertion is often used.

Example:

25 57 48 37 12 92 86 33

Let us create 10 sub files. Initially each of these sub files is empty. An array of pointer f(10) is declared, where f(i) refers to the first element in the file, whose first digit is i. The number is passed to hash function, which returns its last digit (ten's place digit), which is placed at that position only, in the array of pointers.

num= 25 – f(25) gives 2

57 – f(57) gives 5

48 – f(48) gives 4

37 – f(37) gives 3

12 – f(12) gives 1

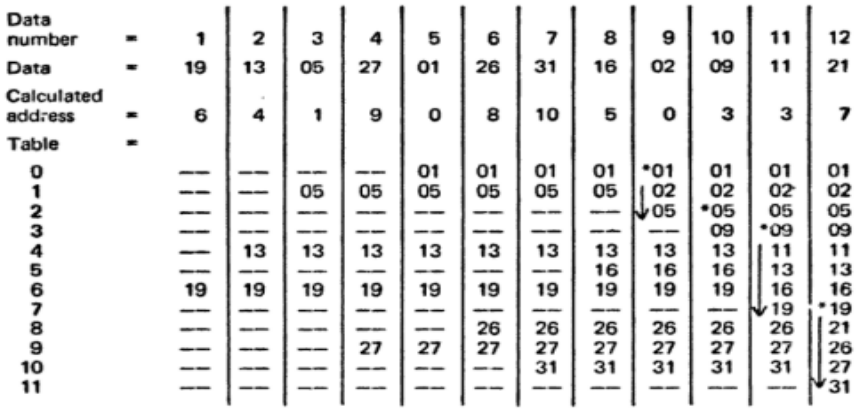
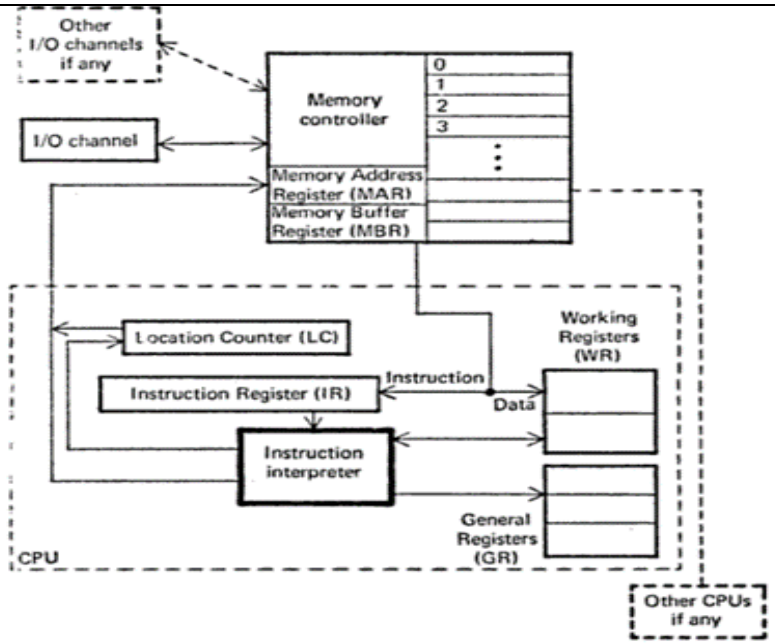
92 – f(92) gives 9

86 – f(86) gives 8

33 – f(33) gives 3 which is repeated.

Thus it is inserted in 3rd sub file (4th) only, but must be checked with the existing elements for its proper position in this sub file.

OR

			
3		**Attempt any FOUR :**	16 M
	1	**Describe the machine structure.**	4 M
	Ans		Diagram: 2 marks, Description:2 marks

System consists of an instruction interpreter, a location counter, an instruction register and various working register and general registers.

The Instruction interpreter is a group of electrical circuits (hardware), that performs the intent of instructions fetch from memory.

The Location Counter (LC), also called Program Counter (PC) or Instruction Counter (IC), is a hardware memory device which denotes the location of the current instruction being executed.

		A copy of current instruction is stored in the Instruction Register (IR). The Working Registers are memory devices that serve as "scratch pads" for instruction interpreter, while the General Registers are used by the programmer as storage locations and for special function. The primary interface between the memory and the CPU is via the Memory Address Register (MAR) and the Memory Buffer Register (MBR). MAR contains the address of the memory location that is to be read from or stored into. MBR contains a copy of the designated memory location specified by the MAR after a "read", or the new contents of the memory location prior to a "write". The memory controller is hardware that transfers data between the MBR and the core memory location that address of which is in the MAR. The I/O channels may be thought of as separate computer which interpret special instructions for inputting and outputting information from the memory.	
	2	**Explain the concept hashing function with a suitable example.**	4 M
	Ans	**Hashing:** **1.** Hashing is the transformation of a string of characters into usually shorterfixed-length value or key that represents the original string. Hashing is used to ndex and retrieve items in a database because it is faster to find shorter hashed key than to find it using the original value. **2.** Binary search algorithms are operated on tabled that are ordered and packed. Therefore it has to be used in conjunction with sort algorithms which both ordered and pack the data. So a considerable improvement can be achieved by inserting elements in a random way. The random entry number K is generated from the key. If the Kth position is valid, then the new element is put there; if not then some other cell must be found for the insertion. **3.** Here the first problem is to generate a random number from the key. This can be achieved by dividing a four character keyword by the table length N and use the remainder. Another method is to treat a keyword as a binary fraction and multiply it by another binary fraction: L 1, SYMBOL M 0, RHO **4.** The result is 64 bit product in registers 0 and 1. If RHO is chosen carefully, the low order 31 bits will be evenly distributed between 0 and 1, and the second multiplication by N will generate number uniformly distributed over 0…(N-1). This is known as power residue method. The second problem is the procedure to be followed when the first trial entry results in a filled position. This problem can be resolve by using one of the following methods:	Explanation 2 Marks, Example 2 Marks

1) **Random entry with replacement:** A sequence of random numbers is generated from the keyword. From each of these a number between 1 and N is formed and the table is probed at that position. Probing are terminated when a void space is found.
2) **Random entry without replacement:** this is the same as above expect that any attempt to probe the same position twice is bypassed.
3) **Open addressing:** if the first probe gives a position K and that position is filled, then the next location K+1 is probed and so on until a free position is found. If the search runs off the bottom of the table, then it is renewed at the top.

Example:

Consider a table of 17 positions (N=17) in which the following 12 numbers are to be stored.

19, 13, 05, 27, 01, 26, 31, 16, 02, 09, 11, 21

These items are to be entered in the table at the position defined by the remainder after division by 17; if that position is filled, then the next position is examined, etc.

The following table shows progress entry for the 12 items. The column 'probes to find' gives the number of probes necessary to find the corresponding item in the tables; thus it takes 3 probes to find item 09, 2 probes to find item 11 and 1 to find item 26. The column 'probes to find' gives the number of probes necessary to determine that the item is not in the table; thus the search for the number 54 would give an initial position of 3 and it would take 4 probes to find that the item is not present.

Position	Item	probes to find	probes to find not
0			1
1	01	1	6
2	19,02*	1	5
3	02	2	4
4	21	1	3
5	05	1	2
6			1
7			1
8			1
9	26, 09*	1	7
10	27, 09*	1	6
11	09, 11*	3	5
12	11	2	4
13	13	1	3
14	31	1	2
15			1
16	16	1	1
		16	54

Length of the table	N = 17
Items stored	M=12
Density	p = 12/17 = 0.705
Probes to store	$T_s = 16$
Average probes to find	$T_p = 16/12 = 1.33$
Average probes to find	$T_n = 54/16 = 3.37$

3 **List four Limitations of Syntax Analyser.** — 4 M

Ans

Limitations of Syntax Analyzers

1) It cannot determine if a token is valid,
2) It cannot determine if a token is declared before it is being used,
3) It cannot determine if a token is initialized before it is being used,
4) It cannot determine if an operation performed on a token type is valid or not.

Any four limitations 1 Mark each

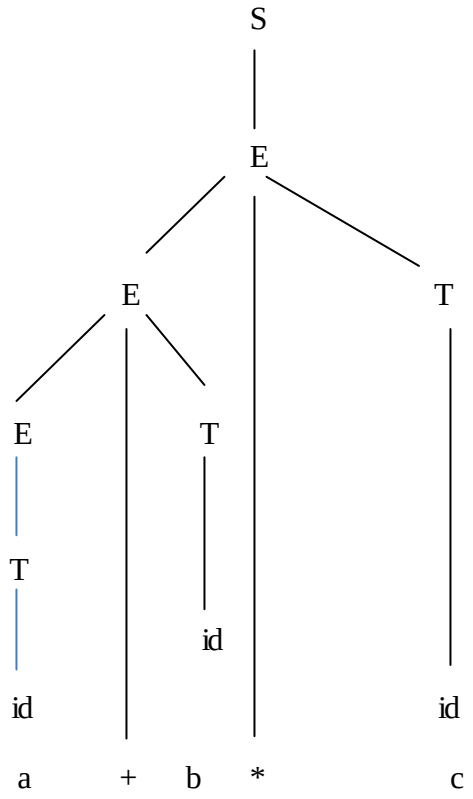
4 **Differentiate between relocating loader and direct linking loader.** — 4 M

Ans

Relocating loaders (BSS)	**Direct linking loaders**
Provides multiple procedure segments, but only one data segment.	Provides multiple procedure segments and multiple data segments.
Provides flexible intersegment referencing ability but does not facilitate access to the data segments that can be shared.	In this type, the assembler produces four types of cards in the object deck: ESD, TXT, RLD, END
The transfer vector linkage is only useful for transfers, and is not well suited for loading or storing external	The RLD card facilitate both relocation and linking information

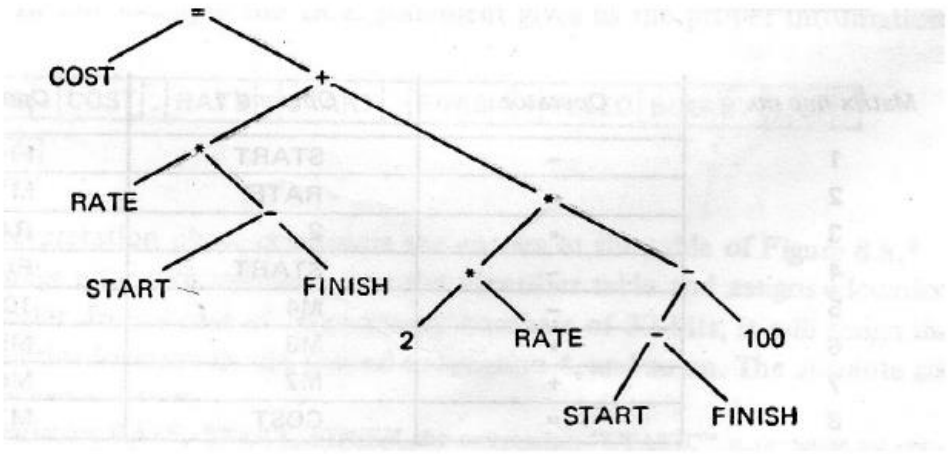
Any correct 4 points : 4 marks

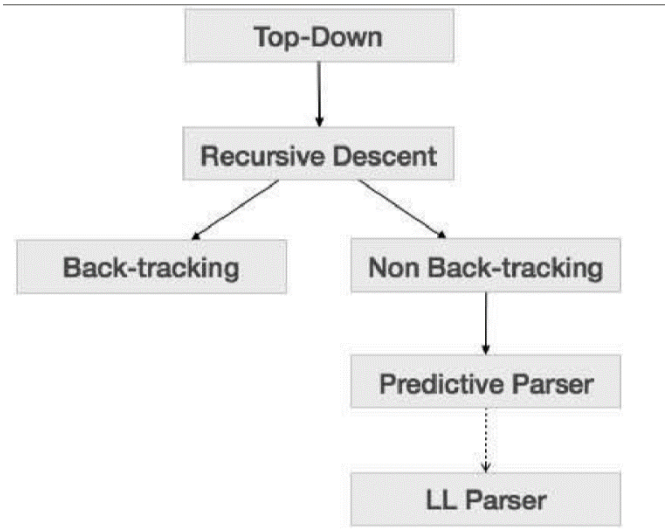
		data	
		The transfer vector increases the size of the object program in memory	No extra memory required for keeping linking in the object program.
	5	**Apply bottom-up parsing on given input string a+b*c with production rules** **S → E** **E → E + T** **E → E * T** **E → T** **T → id**	4 M
	Ans	S→E E→E+T E→E*T E→T T→id a + b * c id + id * id id + T * id T + T * id E + T * id E * id E + T E S	for parsing 2 marks, Production rules: 2 marks



4	**a**	**Attempt any THREE :**	**12 M**
	1	**Differentiate between static binders and dynamic binders.**	4 M
	Ans		Any correct 4 points : 4 marks

Static binders	**Dynamic binders**
In static binders a specific core allocation of a program is performed at the time that the subroutines are bound together.	In dynamic binders the binding will be performed only when an instruction is encountered that requires the linkage.
It is called as "core image module" and the corresponding binder is also called a core image builder.	It is called as "Linkage editor".
It does not keep track of relocation information	It keeps track of the relocation information so that the resulting load module can be further relocated and loaded anywhere in the core.
The module loader performs allocation and loading.	In this case the module loader must perform additional allocation and relocation as well as loading.

		Relatively simple and fast | Relatively complex.	
	2	**Write rules for converting arithmetic statements into parse tree? Convert the following statement into parse tree** **COST = RATE*(START - FINISH) + 2 * RATE * (START - FINISH)-100.**	4 M
	Ans	The rules for converting arithmetic statements into parse tree are: 1. Any variable is a terminal node of the tree 2. For every operator, construct a binary tree (in order dictated by the rules of algebra), whose left branch is a tree for operand 1, and right branch is a tree for operand 2. Parse Tree For following Statement **COST = RATE*(START - FINISH) + 2 * RATE * (START - FINISH)-100.**  	Rules : 1 Mark Parse tree : 3 Marks
	3	**What is loop invariant? State problems that need to be solved by loop invariant.**	4 M
	Ans	• A **loop invariant** is a property of a program **loop** that is true before (and after) each iteration. • It is a logical assertion, sometimes checked within the code by an assertion call. Knowing its **invariant**(s) is essential in understanding the effect of a **loop**. • If computation within a loop depends on a variable that does not change within that loop, then computation may be moved outside the loop. • This requires a reordering of a part of the matrix. There are 3 general problems that need to be solved in an algorithm. 1. Recognition of invariant computation. 2. Discovering where to move the invariant computation.	Explanation 2 Marks, Any 2 problems : 2 Marks(1 Mark each)

		3. Moving the invariant computation.	
	4	**Explain the concept and types of top down parser.**	4 M
	Ans	When the parser starts constructing the parse tree from the start symbol and then tries to transform the start symbol to the input, it is called top-down parsing. • **Recursive descent parsing**: It is a common form of top-down parsing. It is called recursive as it uses recursive procedures to process the input. Recursive descent parsing suffers from backtracking. • **Backtracking**: It means, if one derivation of a production fails, the syntax analyzer restarts the process using different rules of same production. This technique may process the input string more than once to determine the right production.  **Recursive Descent Parsing:** Recursive descent is a top-down parsing technique that constructs the parse tree from the top and the input is read from left to right. It uses procedures for every terminal and non-terminal entity. This parsing technique recursively parses the input to make a parse tree, which may or may not require backtracking. But the grammar associated with it (if not left factored) cannot avoid backtracking. A form of recursive-descent parsing that does not require any back-tracking is known as predictive parsing. This parsing technique is regarded recursive as it uses context-free grammar which is recursive in nature. Back-tracking: Top-down parsers start from the root node (start symbol) and match the input string against the production rules to replace them (if matched). The following example of CFG: S →rXd|rZd X →oa|ea	Explanation :2 Marks,Types: 2 Marks

		Z →ai **For an input string:** read, a top-down parser, will behave like this: It will start with S from the production rules and will match its yield to the left-most letter of the input, i.e. 'r'. The very production of S (S → rXd) matches with it. So the top-down parser advances to the next input letter (i.e. 'e'). The parser tries to expand non-terminal 'X' and checks its production from the left (X → oa). It does not match with the next input symbol. So the top-down parser backtracks to obtain the next production rule of X, (X → ea). Now the parser matches all the input letters in an ordered manner. The string is accepted. **Predictive Parser:** Predictive parser is a recursive descent parser, which has the capability to predict which production is to be used to replace the input string. The predictive parser does not suffer from backtracking. To accomplish its tasks; the predictive parser uses a look-ahead pointer, which points to the next input symbols. To make the parser back-tracking free, the predictive parser puts some constraints on the grammar and accepts only a class of grammar known as LL(k) grammar. Predictive parsing uses a stack and a parsing table to parse the input and generate a parse tree. Both the stack and the input contains an end symbol $to denote that the stack is empty and the input is consumed. The parser refers to the parsing table to take any decision on the input and stack element combination. **LL Parser:** An LL Parser accepts LL grammar. LL grammar is a subset of context-free grammar but with some restrictions to get the simplified version, in order to achieve easy implementation. LL grammar can be implemented by means of both algorithms namely, recursive-descent or table-driven. LL parser is denoted as LL(k). The first L in LL(k) is parsing the input from left to right, the second L in LL(k) stands for left-most derivation and k itself represents the number of look ahead. Generally k = 1, so LL(k) may also be written as LL(1).	
	b	**Attempt any ONE :**	**6 M**
	1	**Explain four function performed by macro processor.**	6 M
	Ans	The 4 basic task of Macro processor is as follows:- 1) Recognize the macro definitions. 2) Save the Macro definition. 3) Recognize the Macro calls. 4) Perform Macro Expansion. 1) **Recognize the Macro definitions:-** A microprocessor must recognize macro definitions identified by the MACRO and MEND pseudo-ops. When MACROS and MENDS are nested, the macro processor must recognize the nesting and correctly	List 2 Marks Explanation of Each point 4 Marks (1 Mark Each)

match the last or outer MEND with the first MACRO.

2) **Save the Macro definition:-** The processor must store the macro instruction definitions which it will need for expanding macro calls.

3) **Recognize the Macro calls:-** The processor must recognize macro call that appear as operation mnemonics. This suggests that macro names be handled as a type of opcode.

4) **Perform Macro Expansion:-** The processor must substitute for macro definition arguments the corresponding arguments from a macro call, the resulting symbolic text is then substituted for the macro call.

2 **Compare top down and bottom up parser.** — 6 M

Ans

Sr. No.	Top – down parsing	Bottom up parsing
1	It is easy to implement	It is efficient parsing method
2	It can be done using recursive decent or LL(1) parsing method	It is a table driven method and can be done using shift reduce, SLR, LR or LALR parsing method
3	The parse tree is constructed from root to leaves	The parse tree is constructed from leaves to root
4	In LL(1) parsing the input is scanned from left to right and left most derivation is carried out	In LR parser the input is scanned from left to right and rightmost derivation in reverse is followed
5	It cannot handle left recursion	The left recursive grammar is handled by this parser
6	It is implemented using recursive routines	It is a table driven method
7	It is applicable to small class of grammar	It is applicable to large class of grammar

Any 6 Difference:

1 Mark each.

5 **Attempt any TWO :** — **16 M**

1 **Explain direct linking loader scheme and format of cards it use.** — 8 M

Ans

Direct linking loader scheme

- It is Relocatable type of loader.
- It has advantage of allowing programmer with multiple procedure segments

DLL Explanation 2 Marks

and giving them complete freedom of referring data contained in some other segment.

- Input to the loader is set of object programs to be linked together
- This provides flexible Intersegment Referencing, for doing all this, DLL required following modules.

Format of Card :6 Marks (1½ Marks each)

ESD-External Symbol Directory

TXT-Actual assembled program

RLD-Relocation and Linkage directory module

END-End module

ESD :

There are four sections of the object deck for a direct linking loader.

The ESD card the information necessary to build the external symbol. The external symbols are symbols that can be referred beyond the subroutine level. The normal labels in the source program are used only by the assembler.

The ESD card contains the information necessary to build the external symbol. The external symbols are symbols that can be referred beyond the subroutine level. The normal labels in the source program are used only

ESD card format:

Columns	Contents
1	Hexadecimal byte X'02'
2-4	Characters ESD
5-14	Blank
15-16	ESD identifier (ID) for program name (SD) external symbol(ER) or blank for entry (LD)
17-24	Name, padded with blanks
25	ESD type code (TYPE)
26-28	Relative address or blank
29	Blank
30-32	Length of program otherwise blank
33-72	Blank
73-80	Card sequence number

TXT :

The TXT card contains the blocks of data and the relative address at which data is

to be placed. Once the loader has decided where to load the program, it adds the Program Load Address (PLA) to relative address. The data on the TXT card may be instruction, non-related data or initial values of address constants.

TXT card format

Columns	Contents
1	Hexadecimal byte X'02'
2-4	Characters TXT
5	Blank
6-8	Relative address of first data byte
9-10	Blanks
11-12	Byte Count (BC) = number of bytes of information in cc. 17-72
13-16	Blank
17-72	From 1 to 56 data bytes
73-80	Card sequence number

RLD:

The RLD cards contain the following information 1. The location and length of each address constant that needs to be changed for relocation or linking. 2. The external symbol by which the address constant should be modified. 3. The operation to be performed.

RLD card format :

Columns	Contents
1	Hexadecimal byte X'02'
2-4	Characters RLD
5-18	Blank
19-20	Relative address of first data byte
21	Blanks
22-24	Byte Count (BC) = number of bytes of information in cc. 17-72
25-72	Blank
73-80	Card sequence number

END:

The END card specifies the end of the object deck.

END card format

Columns	Contents
1	Hexadecimal byte X'02'
2-4	Characters END
5	Blank
6-8	Start of execution entry (ADDR), if other than beginning of program
9-72	Blanks
73-80	Card sequence number

	2	**Write any four optimization techniques uses by compiler.**	8 M
	Ans	The possible algorithm for four optimization techniques are as follows:-	Each technique: 2 marks

1) Elimination of common sub expression

2) Compile time compute.

3) Boolean expression optimization.

4) Move invariant computations outside of loops.

1) Elimination of common sub expression

The elimination of duplicate matrix entries can result in a more can use and efficient object program. The common subexpression must be identical and must be in the same statement.

i. The elimination algorithm is as follows:-
ii. Place the matrix in a form so that common subexpression can be recognized.
iii. Recognize two sub expressions as being equivalent.
iv. Eliminate one of them.
v. After the rest of the matrix to reflect the elimination of this entry.

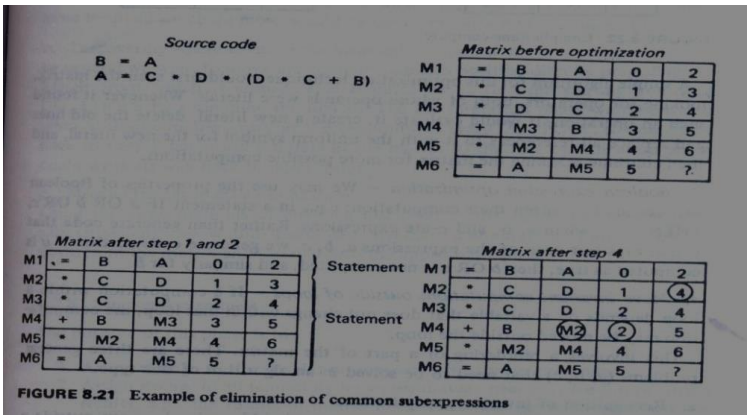


FIGURE 8.21 Example of elimination of common subexpressions

2) Compile time compute.

- Doing computation involving constants at compile time save both space and execution time for the object program.
- The algorithm for this optimization is as follows:-
  i. Scan the matrix.
  ii. Look for operators, both of whose operands were literals.
  iii. When it found such an operation it would evaluate it, create new literal, delete old line.
  iv. Replace all references to it with the uniform symbol for the new literal.
  v. Continue scanning the matrix for more possible computation.

\

For e.g.-

A = 2 *276 / 92 * B

- The compile time computation would be

Matrix Before optimization

M1	*	2	276
M2	/	M1	92
M3	*	M2	B
M4	=	A	M3

Matrix After optimization

M1	*	2	276
M2	/	M1	92
M3	*	6	B
M4	=	A	M3

3) Boolean expression optimization.

- We may use the properties of Boolean expression to shorten their computation.
  e.g. In a statement

  If a OR b Or c,

- Then …… when a, b & c are expression rather than generate code that will always test each expression a, b, c. We generate code so that if a computed as true, then b OR c is not computed, and similarly for b.

4) Move invariant computations outside of loops.

- If computation within a loop depends on a variable that does not change within that loop, then computation may be moved outside the loop.
- This requires a reordering of a part of the matrix. There are 3 general problems that need to be solved in an algorithm.
  1. Recognition of invariant computation.
  2. Discovering where to move the invariant computation.
  3. Moving the invariant computation.

	3	**Explain radix sort with example.**	8 M
	Ans	• In computer science, radix sort is a non-comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value. A positional notation is required, but because integers can represent strings of characters (e.g., names or dates) and specially formatted floating point numbers, radix sort is not limited to integers. • Most digital computers internally represent all of their data as electronic representations of binary numbers, so processing the digits of integer representations by groups of binary digit representations is most convenient. Radix sorts can be implemented to start at either the most significant digit	Explanation: 4 Marks, Example : 4 Marks

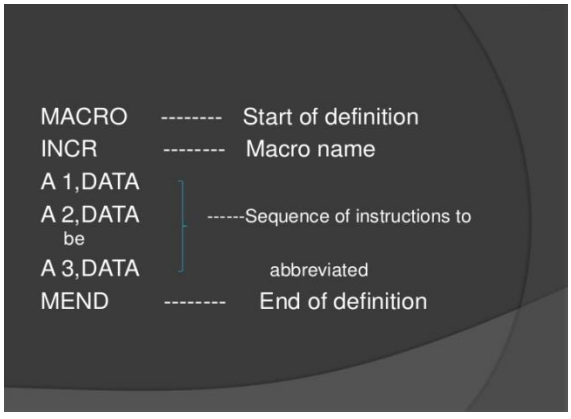
(MSD) or least significant digit (LSD). For example, when sorting the number 1234 into a list, one could start with the 1 or the 4.

Radix Sort is not comparison based algorithm. It has the time complexity of O(nk) where n is the size of input array.
and k is no.of digits in largest number
Algorithm:
For each digit where varies from the least significant digit to the most significant digit of a number
Sort input array using radix sort algorithm according to i th digit.

Example: Assume the input array is:
10,21,17,34,44,11,654,123
Based on the algorithm, we will sort the input array according to the **one's digit** (least significant digit).
0: 10
1: 21 11
2:
3: 123
4: 34 44 654
5:
6:
7: 17
8:
9:

So, the array becomes in first pass 10,21,11,123,24,44,654,17
Now, we'll sort according to the **ten's digit**:
0:
1: 10 11 17
2: 21 123
3: 34
4: 44
5: 654
6:
7:
8:
9:

Now, the array becomes in second pass : 10,11,17,21,123,34,44,654
Finally , we sort according to the **hundred's digit** (most significant digit):
0: 010 011 017 021 034 044
1: 123

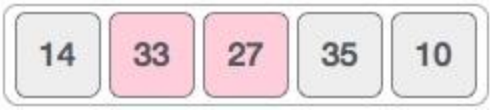
		2: 3: 4: 5: 6: 654 7: 8: 9: The array becomes in third pass : 10,11,17,21,34,44,123,654 which is sorted.	
6		**Attempt any FOUR :**	**16 M**
	1	**Describe macro and subroutine.**	4 M
	Ans	Macro: The assembly language programmer often finds that certain set of instructions get repeated often in the code. Instead of repeating the set of instructions the programmer can take advantage of macro facility where macro is defined as "Single line abbreviation for group of instructions". A macro instruction is a notational convenience for the programmer, It allows the programmer to write shorthand version of a program (module programming). The macro processor replaces each macro invocation with the corresponding sequence of statements expanding) A macro represents a commonly used group of statements in the source programming language. The macro processor replaces each macro instruction with the corresponding group of source language statement, this is called expanding macros.  Subroutine:	Macro Description :2 marks, Subroutine Description: 2 marks

		In computer programming, a subroutine is a sequence of program instructions that performs a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed. Subprograms may be defined within programs, or separately in libraries that can be used by many programs. In different programming languages, a subroutine may be called a procedure, a function, a routine, a method, or a subprogram. The generic term callable unit is sometimes used. The name subprogram suggests a subroutine behaves in much the same way as a computer program that is used as one step in a larger program or another subprogram. A subroutine is often coded so that it can be started several times and from several places during one execution of the program, including from other subroutines, and then branch back (return) to the next instruction after the call, once the subroutine's task is done. Subroutines are a powerful programming tool, and the syntax of many programming languages includes support for writing and using them. Judicious use of subroutines (for example, through the structured programming approach) will often substantially reduce the cost of developing and maintaining a large program, while increasing its quality and reliability. Subroutines, often collected into libraries, are an important mechanism for sharing and trading software. The discipline of object-oriented programming is based on objects and methods (which are subroutines attached to these objects or object classes).	
	2	**Explain interchange sort with example.**	4 M
	Ans	The interchange sort is almost similar as the bubble sort. In fact some people refer to the interchange sort as just a different bubble sort. (When they see the source they even call it a bubble sort instead of its real name interchange sort.)The interchange sort compares each element of an array and swaps those elements that are not in their proper position, just like a bubble sort does. The only difference between the two sorting algorithms is the manner in which they compare the elements. The interchange sort compares the first element with each element of the array, making a swap where is necessary. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$ where **n** is the number of items. Example: We take an unsorted array for our example. Bubble sort takes $O(n^2)$ time so we're keeping it short and precise.  Bubble sort starts with very first two elements, comparing them to check which one	Explanation 2M Example 2M

is greater.

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

We find that 27 is smaller than 33 and these two values must be swapped.



The new array should look like this −

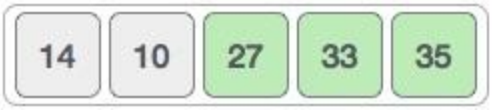
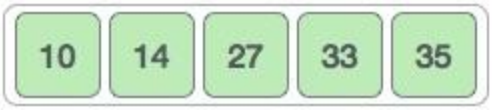
Next we compare 33 and 35. We find that both are in already sorted positions.

Then we move to the next two values, 35 and 10.

We know then that 10 is smaller 35. Hence they are not sorted.

We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this −

To be precise, we are now showing how an array should look like after each

		iteration. After the second iteration, it should look like this −  Notice that after each iteration, at least one value moves at the end.  And when there's no swap required, bubble sorts learns that an array is completely sorted. 	
	3	**For the following pseudo-ops(pseudo opcodes), write suitable example:** **i)ENTRY** **ii)EXTRN**	4 M
	Ans	It is used to direct or to suggest loader that data followed by ENTRY are defined in this program but they are referenced in another program. Similarly subroutine followed by EXTRN is called in main program but its definition is written outside the main program as subroutine. For example: the following sequence of instruction may be a simple calling sequence to another program. <u>ENTRY Example:</u> A START ENTRY B1, B2, B3…….These symbol list are referenced in another program. B1…….. B2…….. B3……..	Explanation of Entry 2 Marks. Explanation of EXTERN 2 Marks

END

EXTRN Example

```
MAIN START
EXTRN SUBROUT

.....
.....
.....
L 15=A(SUBROUT).....CALL SUBROUT
BALR 14,15
..
..
..
..
END


SUBROUT    START
           USING *, 15
           .
           .
           .
           BR    15
           END
```

The above sequence of instructions first declares SUBROUT as an external variable, that is a variable referenced but not defined in this program. The load(L) instruction

		loads the address of that variable in to register 15.	
	4	**Explain storage allocation phase of compiler.**	4 M
	Ans	The purpose of this phase is to: 1. Assign storage to all variables referenced in the source program. 2. Assign storage to all temporary locations that are necessary for intermediate code generation. 3. Assign storage to literals. 4. Ensure that the storage is allocated and appropriate locations are initialized It makes entries in the matrix that allow code generation to create code that allocates dynamic storage, and that also allow the assembly phase to reserve the proper amounts of STATIC storage.	Explanation: 4 Marks
	5	**State functions of relocating loader.**	4 M
	Ans	Functions of relocating loader. • To avoid possible reassembling of subroutines when a single subroutine is changed and to perform the tasks of allocation and linking for the programmer. • It allows many procedure segments but only one data segment. The assembler assembles each procedure segment independently and passes to the loader text and information for relocation and intersegment reference. • The relocating loader processes procedure segments but does not facilitates access to data segment which can be shared. • The four functions of loader (allocation, linking, relocation and loading) are all performed by the relocating loader. • E.g.: BSS (Binary Symbolic Subroutine) loader used in IBM 7094, IBM 1130 and in UNIVAC 1108.	Any 4 Functions, 1 Mark each.