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.

Q.1. Attempt any FIVE of the following:

a) **State the four function performed by Microprocessor.(Four functions -1 mark each)**

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.

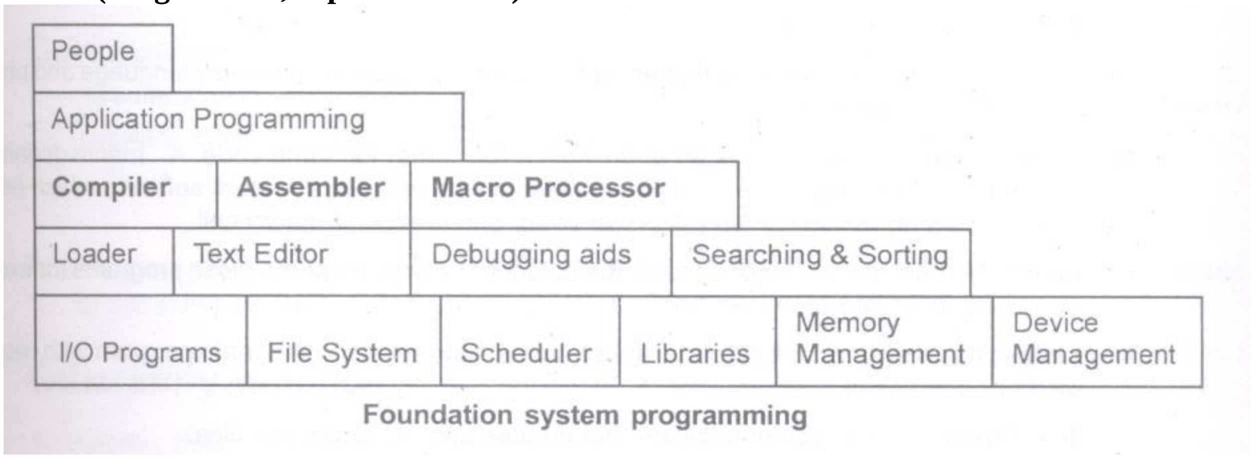
1) Recognize the Macro definitions:- A microprocessor must recognize macro definitions identified by the MACRO and MEND pseudo-ops. WhenMACROS and MENDS are nested, the macroprocessor must recognize the nesting and correctly match the last or outer MEND with the first MACRO.

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.

b) Draw a labeled diagram of foundation of system programming and explain it in brief.
(Diagram 2M, explanation 2M)



Foundation system programming

System programs leg compiler, loaders, macro processor, operating systems were developed to make computer better adapted to the needs of their users.
Compiler is system program that accept people life languages and translate them into machine language.
Loaders are system programs that prepare machine language programs for execution.
Macro processors allow programmers to use abbreviations. Operating system and file system allows flexible storing and retrieval of information. The productivity of each computer is heavily dependent upon the effectiveness, efficiency and sophistication of the system programs

c) List four functions performed by loader.
(Four functions -1 mark each)
Ans: **Loader Function** : The loader performs the following functions :
1) Allocation - The loader determines and allocates the required memory space for the program to execute properly.
2) Linking -- The loader analyses and resolve the symbolic references made in the object modules.
3) Relocation - The loader maps and relocates the address references to correspond to the newly allocated memory space during execution.
4) Loading - The loader actually loads the machine code corresponding to the object modules into the allocated memory space and makes the program ready to execute.

d) Draw the parse tree for the string 'acddf' using top down parsing approach.
Ans: (1 mark for each step (from step 2 onwards) (consider relevant assumptions if any)

String is "acddf"

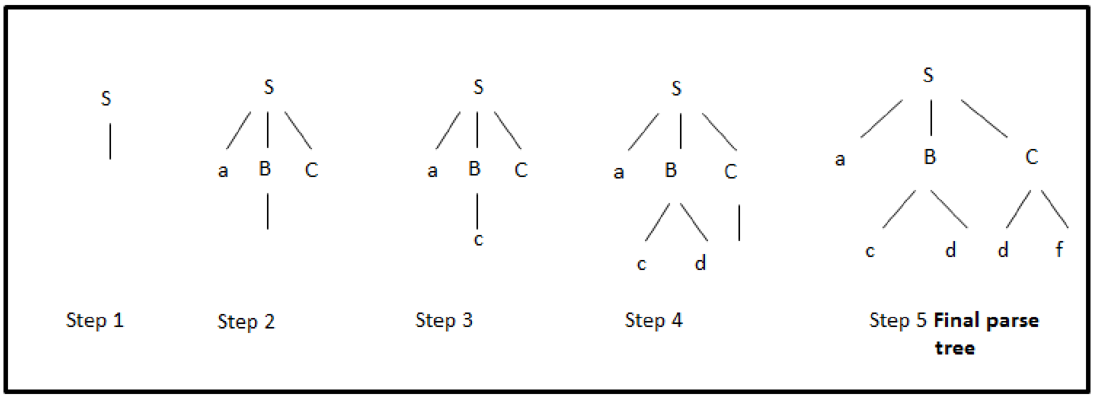
Assume:
S→ xyz | aBC
B → c | cd
C → eg | df

Steps

❖ Assertion 1 : acddf matches S
- Assertion 2: acddf matches xyz:

- **Assertion is false. Try another.**
- Assertion 2 : acddf matches aBCi.ecddf matches BC:
    - Assertion 3 : cddf matches cCi.eddf matches C:
        - Assertion 4 : ddf matches eg:
        - False.
        - Assertion 4 : ddf matches df:
        - False.
    - Assertion 3 is false. Try another.
    - Assertion 3 : cddf matches cdCi.edf matches C:
        - Assertion 4 : df matches eg:
        - False.
        - Assertion 4 : df matches df:
        - **Assertion 4 is true.**
    - **Assertion 3 is true.**
- **Assertion 2 is true.**

❖ **Assertion 1 is true.**



e) **List and explain 4 components of system software.**
ANS :(List -1 mark,3 marks for explanation(any 4 components)

Components of system software are:

1. Assembler
2. Macros
3. Loader
4. Linker
5. Compiler.

1. **Assembler:**
It is a language translator that takes as input assembly language program (ALP) and generates its machine language equivalent along with information required by the loader.
ALP→ Assembler→ Machine language equivalent + Information required by the loader

2. **Macros:**
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 the advantage of macro facility where macro is defined to be as "Single line abbreviation for a group of instructions".
The template for designing a macro is as follows

```
MACRO Start of definition
        Macro Name
            ----------
            ----------
MEND End of def.
```

3.Loader: It is responsible for loading program into the memory, prepare them for execution and then execute them.
OR
Loader is a system program which is responsible for preparing the object programs for execution and start the execution.

Functions of loader
a. Allocation
b. Linking
c. Relocation
d. Loading

Allocation: Allocate the space in the memory where the object programs can be loaded for execution.
Linking: Resolving external symbol reference
Relocation: Adjust the address sensitive instructions to the allocated space.
Loading: Placing the object program in the memory in to the allocated space.

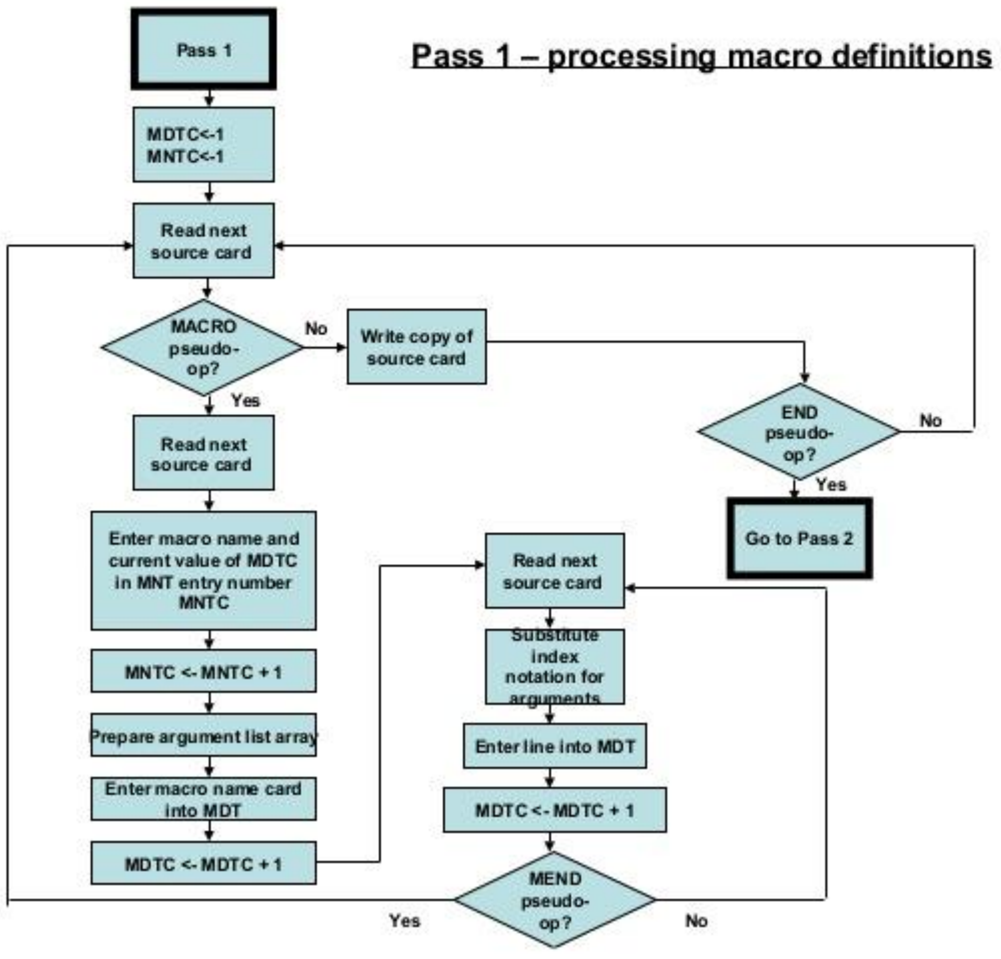
4. **Linker:**
A linker which is also called binder or link editor, is a program that combines object modules together to form a program that can be executed. Modules are parts of a program.

5. **Compiler:**
Compiler is a language translator that takes as input the source program(Higher level program) and generates the target program (Assembly language program or machine language program)

$$\begin{tikzcd}\text{Source Program} \arrow[r] & \text{Compiler} \arrow[r] \arrow[d] & \text{Target program} \\ & \text{Error Messages} & \end{tikzcd}$$

Since the process of compilation is very complex it is divided in to several phases.
(A phase is a logical unit of work that takes as input one representation and generates another representation.)
The phases are as follows
1. Lexical Analysis
2. Syntax analysis
3. Semantic Analysis
4. Intermediate code generation
5. Code optimization
6. Code Generation

f) Draw flow chart of pass 1 of a 2 pass macroprocessor.
Ans: (4 marks for correct flow chart)


Pass 1 – processing macro definitions
Pass 1
MDTC<-1
MNTC<-1
Read next source card
MACRO pseudo-op?
No
Yes
Write copy of source card
END pseudo-op?
No
Yes
Go to Pass 2
Read next source card
Enter macro name and current value of MDTC in MNT entry number MNTC
MNTC <- MNTC + 1
Prepare argument list array
Enter macro name card into MDT
MDTC <- MDTC + 1
Read next source card
Substitute index notation for arguments
Enter line into MDT
MDTC <- MDTC + 1
MEND pseudo-op?
Yes
No

g) Construction argument list array for following statement & LAB INCR &arg1 &arg2 &arg3.
Ans: (Correct argument List Array 4M)

Consider call as:

INCR &ARG2 = DATA2, &ARG3 = DATA3

Macro definition Table MDT

```
            :
            :
&LAB       INCR  &ARG2,&ARG3
#0  A      2,#2
    A      3,#3
  MEND
```

During pass2 it is necessary to substitute macro call arguments for index markers stored in macro definition. Thus upon encountering the call

```
  LOOP          INCR   DATA2,DATA3
```

The macro call expander would prepares an argument list array:

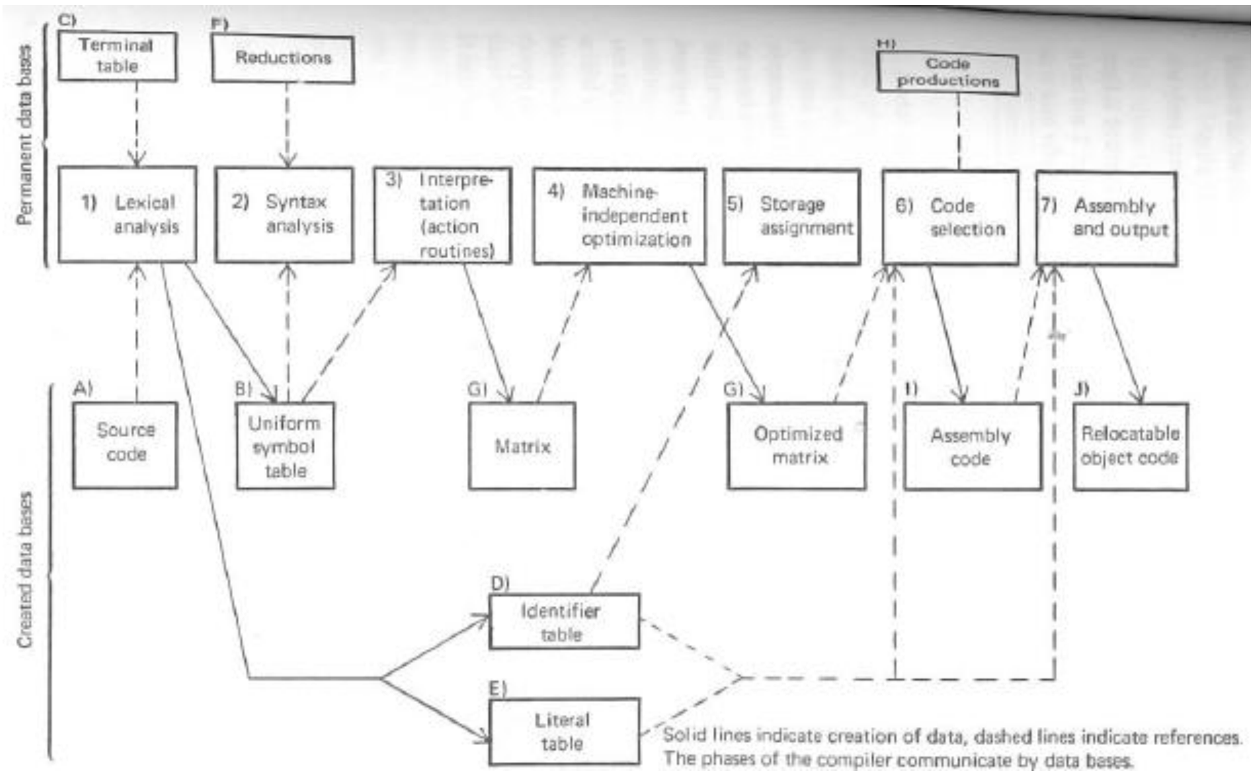
Argument List Array
8 bytes per entry

Index	Argument
0	"LOOP1bbb"
2	"DATA2bbb"
3	"DATA3bbb"

(b denotes blank character)

Q.2. Attempt any two of followings (16 marks)

a) Draw a label diagram of phases of compiler and explain each phase in short. (2 marks for diagram, 6 marks for phases)



1) **Lexical Phase:-**

- Its main task is to read the source program and if the elements of the program are correct it generates as output a sequence of tokens that the parser uses for syntax analysis.
- The reading or parsing of source program is called as scanning of the source program.
- It recognizes keywords, operators and identifiers, integers, floating point numbers, character strings and other similar items that form the source program.
- The lexical analyzer collects information about tokens in to their associated attributes.

2) **Syntax Phase:-**

- In this phase the compiler must recognize the phases (syntactic construction); each phrase is a semantic entry and is a string of tokens that has meaning, and 2nd Interpret the meaning of the constructions.
- Syntactic analysis also notes syntax errors and assure some sort of recovery. Once the syntax of statement is correct, the second step is to interpret the meaning (semantic). There are many ways of recognizing the basic constructs and interpreting the meaning.
- Syntax analysis uses a rule (reductions) which specifies the syntax form of source language.
- This reduction defines the basic syntax construction and appropriate compiler routine (action routine) to be executed when a construction is recognized.
- The action routine interprets the meaning and generates either code or intermediate form of construction.

3) **Interpretation Phase:-**

- This phase is typically a routine that are called when a construct is recognized. The purpose of these routines is to on intermediate form of source program and adds information to identifier table.

4) **Code optimization Phase:-**

- Two types of optimization is performed by compiler, machine dependent and machine independent. Machine dependent optimization is so intimately related to instruction that the generated. It was incorporated into the code generation phase. Where Machine independent optimization is was done in a separate optimization phase.

5) **Storage Assignment:-** The purpose of this phase is as follows:-

- Assign storage to all variables referenced in source program.
- Assign storage to all temporary locations that are necessary for intermediate results.
- Assign storage to literals.
- Ensure that storage is allocated and appropriate locations are initialized.

6) **Code generation:-**

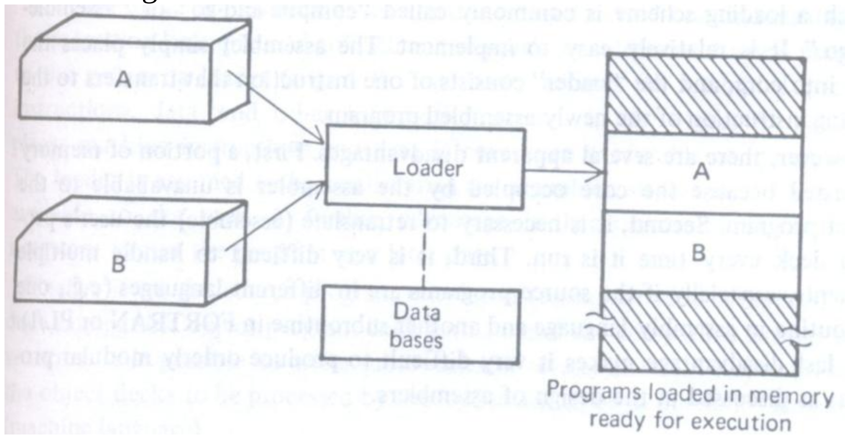
- This phase produce a program which can be in Assembly or machine language.
- This phase has matrix as input.
- It uses the code production in the matrix to produce code.
- It also references the identifier table in order to generate address & code conversion.

7) **Assembly phase:-**The compiler has to generate the machine language code for computer to understand. The task to be done is as follows:-

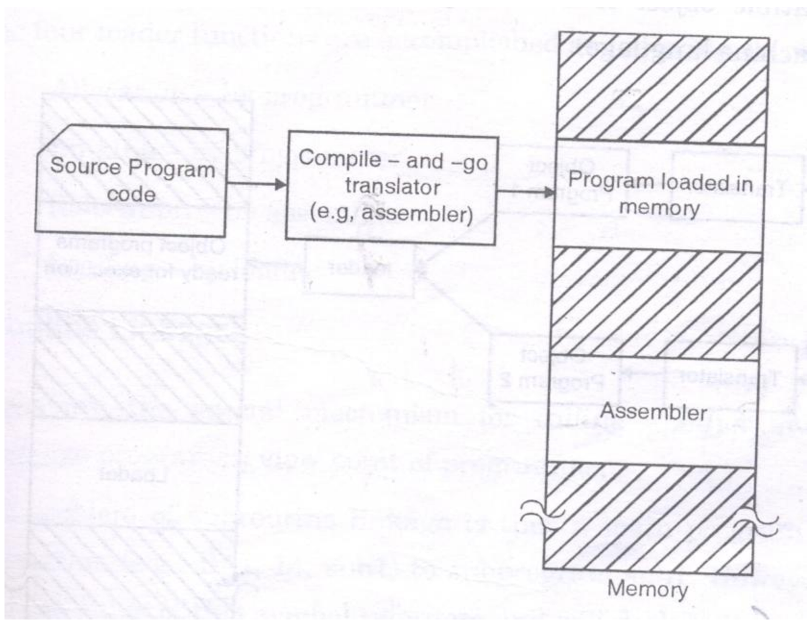
- Generating code
- Resolving all references.
- Defining all labels.
- Resolving literals and symbolic table.

b) Draw the labeled diagram of general loading scheme and link and go scheme and compare the two schemes on basis of allocation and linking.
Ans: (Diagram -2 marks each,4 marks for comparison)

General loading scheme:



Compile and go loader scheme:



Compile and go loader

1. In Compile and go loader is no production of .obj file, the source code is directly converted to executable form. Hence even though there is no modification in the source program it needs to be assembled and executed each time, which then becomes a time consuming activity.
2. It cannot handle multiple source programs or multiple programs written in different languages. This is because assembler can translate one source language to other target language.
   For a programmer it is very difficult to make an orderly modulator
   Program and also it becomes difficult to maintain such program, and the “compile and go” loader cannot handle such programs.

3. In this scheme some portion of memory is occupied by assembler which is simply wastage of memory. As this scheme is combination of assembler and loader activities, this combination program occupies large block of memory.
4. The execution time will be more in this scheme as every time program is assembled and then executed.

General loader

1) In general loader program need not be retranslated each time while running it. This is because initially when source program gets executed an object program gets generated. Of program is not modified, and then loader can make use of this object program to convert it to executable form.
2) It is possible to write source program with multiple programs and multiple languages,because the source programs are first converted to object programs always, and loader accepts these object modules to convert it to executable form.
3) There is no wastage of memory, because assembler is not placed in the memory, instead of it, loader occupies some portion of the memory. And size of loader is smaller than assembler, so more memory is available to the user.
4) The execution time will be less in this scheme

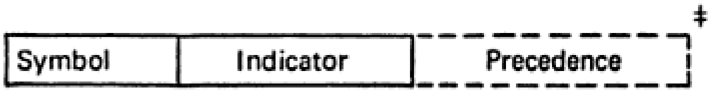
c) State the purpose of following tables:

i. Literal table
ii. Terminal table
iii. Uniform symbol table
iv. Reduction table.

Ans: (For each table 1 mark)

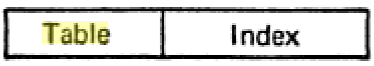
i. Literal Table:
A table, the literal table (LT), that is use to store each literal encounter and its corresponding assign location

ii. Terminal table:
It Is a permanent data base that has entry for each terminal symbol .each entry consists of the terminal symbol, an indication of its classification and its precedence.



Terminal table entry

iii. Uniform symbol table is created by lexical analysis phase of compiler. It contains the source program in the form of uniform symbols. It is used by syntax and interpretation phases as the source of input to the stack. Each symbol from the uniform symbol table enters the stack only once.



Uniform symbol table entry

iv. Reduction table

The syntax rules of the source language are contained in the **reduction table**. The syntax analysis phase is an interpreter driven by the reductions.
The general form of rule or reduction is:
Label: old top of stack/ Action routines/ new top of stack/ Next reduction

Q.3. Attempt any FOUR of the following: (16Marks)

a) List three applications of system software.
Ans: (any three applications 4M)

1) It increases the productivity of computer which depends upon the effectiveness, efficiency and sophistication of the systems programs.
2) Compilers are system programs that accept people like languages and translate them into machine language.
3) Loaders are system programs that prepare machine language program for execution.
4) Macro processors allow programmers to use abbreviation.
5) Provides efficient management of various resources.
6) It manages multiprocessing, paging, segmentation, resource allocation.
7) Operating system and file systems allow flexible storing and retrieval of information.

b) Compare shell Sort and Radix Exchange Sort on the basis of space and time complexity.
Ans: (1 marks for each difference. any other diff points can also be considered)

	Shell Short	**Radix Sort**
1	It provides optimal performance for comparative type of sort	Radix sort is one of the type of distributive sort
2	Sort is performed by comparing item at distance ‘d’ and in each pass value of ‘d’ is decreased.	Sort is performed by considering group with same (M) first bits and ordering that group with respect to (M+1)st bit.
3	Average time complexity is $B*N*(\log_2 N)^2$	Average time complexity is $C*N*\log_P K$
4	Space complexity is none	Space complexity is N*P

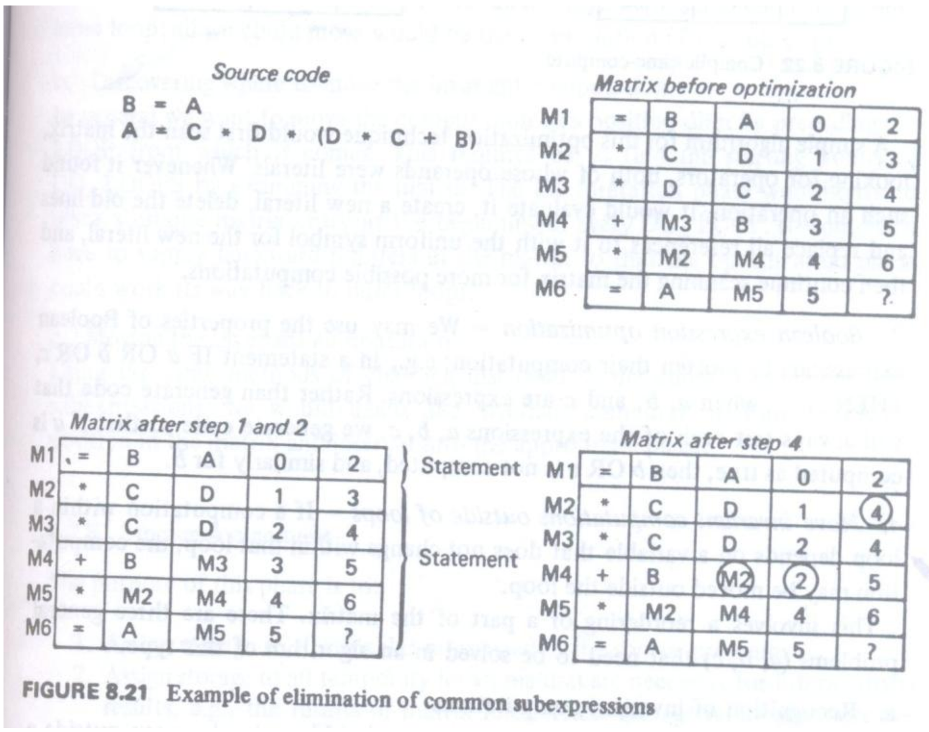
c) State any four optimization technique use by compiler.
Ans: (1 marks for each technique)

The possible algorithm for four optimization techniques are as follows:-
1) Elimination of common sub expression
2) Compile time compute.
3) Boolean expression optimization.
4) Move invariant computations outside of loops.

1) **<u>Elimination of common sub expression</u>**: -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 subexpressins as being equivalent.

iv. Eliminate one of them.
v. After the rest of the matrix to reflect the elimination of this entry.

For example:-



Source code

```
B = A
A = C * D * (D * C + B)
```

Matrix before optimization

M1	=	B	A	0	2
M2	*	C	D	1	3
M3	*	D	C	2	4
M4	+	M3	B	3	5
M5	*	M2	M4	4	6
M6	=	A	M5	5	?

Matrix after step 1 and 2

M1	, =	B	A	0	2
M2	*	C	D	1	3
M3	*	C	D	2	4
M4	+	B	M3	3	5
M5	*	M2	M4	4	6
M6	=	A	M5	5	?

Statement

Statement

Matrix after step 4

M1	=	B	A	0	2
M2	*	C	D	1	4
M3	*	C	D	2	4
M4	+	B	M2	2	5
M5	*	M2	M4	4	6
M6	=	A	M5	5	?

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

M_1	*	2	276
M2	/	M_1	92
M3	∗	M_2	B
M4	=	A	M_3

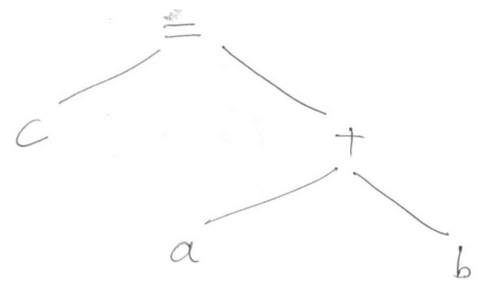
Matrix After optimization

M_1			
M_2			
M_3	*	6	B
M_4	=	A	M_3

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.

3) **Move invariant computation 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.

d) Draw the output of syntax analysis phase for the string 'c=a+b' in the form of syntax tree.
Ans: (4 marks for tree and explanation)



The syntax phase takes as input tokens generated by lexical phase and if meaning are correct it generates a parse tree.

e) State issues in implementation of macro call within macros. Space and speed of execution.
Ans: (3M for issues & 1 M for example)

To handle properly macro calls within macro, the macroprocessor must work recursively, i.e. to process one macro before it is finished with another, then to continue with the previous or outer macro. Recursion is served with help of stack. Stack is a storage mechanism in which elements are stored only at one end called 'top' of the stack. The element which is added first is removed at end. Stack pointer (SP) is used that indicate current position or value on top of stack.
Macro calls are "abbreviations" of instruction sequences, it seems reasonable that such "abbreviations" should be available within other macro definitions. For example,

```
MACRO
ADD1        &ARG
L           1, &ARG
A           1, =F'1'
ST          1, &ARG
MEND
MACRO
ADDS        &ARG1, &ARG2, &ARG3
```

```
        ADD1            &ARG1
        ADD1            &ARG2
        ADD1            &ARG3
        MEND
```

Q.4. Attempt any four of the following:

a) Define the following terms: (2M for each)

i. Overlays

1. Many modern computers use virtual memories that make it possible to run programs larger than physical memory either one program or several programs can be executed even if total size is grater than entire memory available**.**

2. When a computer does not use virtual memory, running a larger program becomes a problem. One solution is overlays(or chaining).

3. Overlays are based on the facts that many programs can be broken into logical parts such that only one part is needed in memory at any time.

4. The program is divided by the programmer, in to main part (overlay root), that resides in memory during the entire execution and several overlays, (links or segments) that can be called, one at a time, by the root, loaded and executed.

5. The subroutines of a program are needed at different times.

ii. Subroutine linkage:

It is a mechanism for calling another subroutine in an assembly language. The scenario for subroutine linkage.

1. A main program a wishes to transfer to subprogram B.
2. The programmer in program A, could write a transfer instruction. Eg (BA L, 14,B) to subprogram B.
3. But assembler does not know the value of this transfer reference and will declare it is an error.
4. To handle this situation a special mechanism is needed.
5. To handle it mechanism is typically implemented with a relocation or direct linking loader.\

Subroutine linkage uses following special pseudo ops:

ENTRY

EXTRN

It is used to direct or to suggest loader that subroutine followed by ENTRY are defined in this program but they are used in another program.

For example: the following sequence of instruction may be a simple calling sequence to another program.

```
MAIN START
        ETRN SUBROUT
        ……………..
        ……………..
        …………….
        L       15=A(SUBROUT)…..CALL SUBROUT
        BAIR    14,15
        ..
        ..
        ..
        ..
        END
```

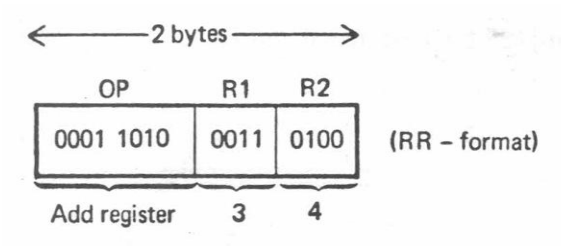
The above sequence of instructions first declares SUBROUT as an external variable, that is a variable referenced but not defined in this program.
The load(L) instruction loads the address of that variable in to register 15.

b) Describe RR format of instruction by using a neat labeled diagram.
ANS: (2M diagram, 2M explanation)

Register operands refer to data stored in one of the 16 general register (32 bits long), which are addressed by a four-bit field in the instruction, since registers are usually constructed of high- speed circuity, they provide faster access to data than does core storage.

For example, the instruction add register 3,4 cause the contents of general register 4(32 bits) to be added to the contents of general register 3(32 bits) and the resulting sum to be left in general register 3.

Storage operands refer to data stored in core memory. The length of the operand depends upon the specific data type. Operand data field that are longer than one byte are specified by the address of the lowest- address byte (logically leftmost).



Causes the contents of the general register 4(32 bit) to be added to the contents of general register 3(32 bit) and the resulting sum to be left in general register 3.

Storage operands refer to data stored in core memory. The length of the operand depends upon the specific data type. Operand data fields that are longer than one byte are specified by the address of the lowest address byte. (Logically- leftmost)

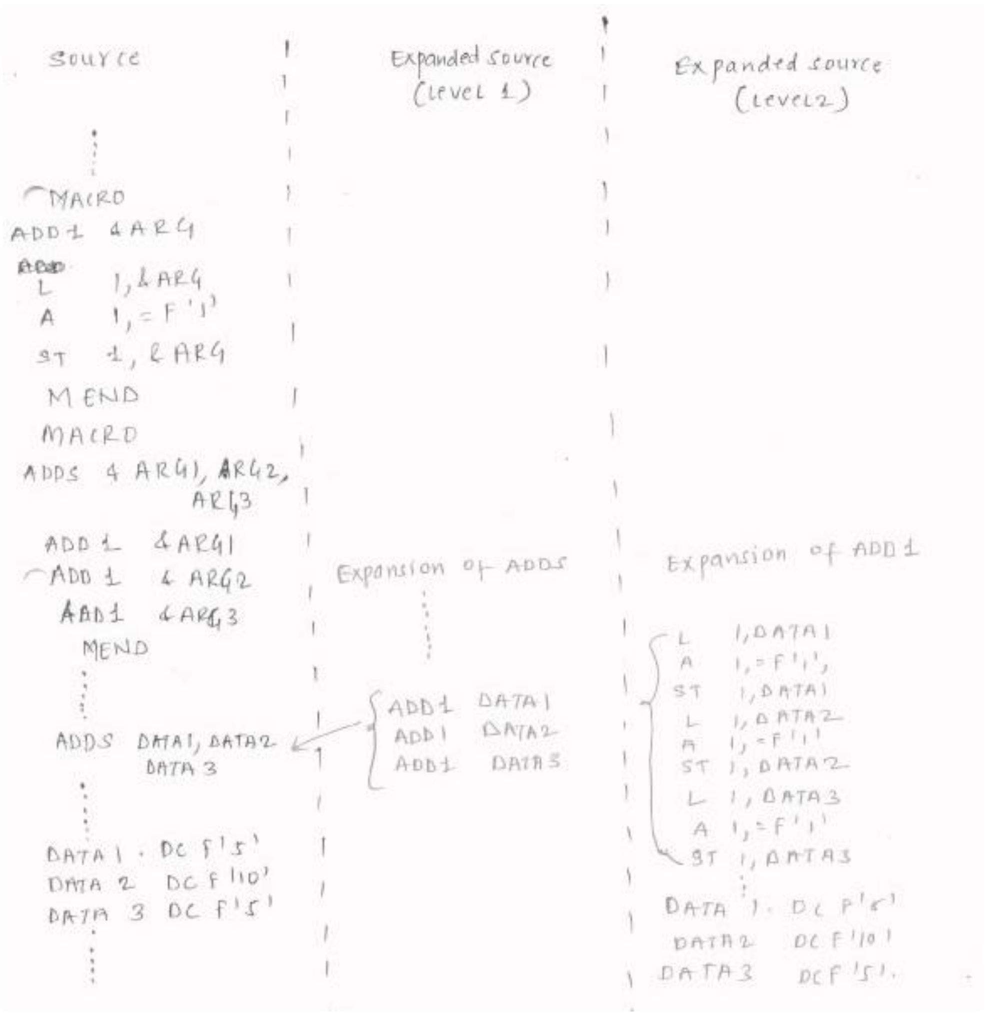
c) Explain with an example a macro call within macros.
Ans: (4 marks for example with explanation)

Macro calls are “abbreviations” of instruction sequences, it seems reasonable that such “abbreviations” should be available within other macro definitions. For example,

```
        MACRO
        ADD1            &ARG
L               1, &ARG
A               1, =F’1’
ST              1, &ARG
MEND
MACRO
ADDS            &ARG1, &ARG2, &ARG3
```

```
ADD1        &ARG1
ADD1        &ARG2
ADD1        &ARG3
MEND
```

Within the definition of the macro ‘ADDS’ are three separate calls to a previously defined macro ‘ADD1’. The use of the macro ‘ADD1’ has shortened the length of the definition of ‘ADDS’ and thus had made it more easily understood. Such use of macros result in macro expansions on multiple ‘levels’ .



```
MACRO
ADD1   &ARG
~~ADD~~
L      1,&ARG
A      1,=F'1'
ST     1,&ARG
MEND
MACRO
ADDS   &ARG1, ARG2,
         ARG3
ADD1   &ARG1
ADD1   &ARG2
ADD1   &ARG3
MEND
ADDS   DATA1, DATA2
       DATA 3
DATA1 . DC F'5'
DATA 2  DC F'10'
DATA 3  DC F'5'
```

```
ADD1   DATA1
ADD1   DATA2
ADD1   DATA3
```

```
L    1,DATA1
A    1,=F'1',
ST   1,DATA1
L    1,DATA2
A    1,=F'1'
ST   1,DATA2
L    1,DATA3
A    1,=F'1'
ST   1,DATA3
DATA 1. DC F'5'
DATA2   DC F'10'
DATA3   DC F'5'.
```

d) **State the data structures required for loader.**
<u>Ans:</u> (2 M for each phase)

<u>Pass 1 data bases</u>

1) Input object decks
2) Initial program load Address (IPLA) supplied by the programmer on the operating system that specifies the address to load the first segment.
3) A program load address (PLA) counter, used to rap truck of each segments assigned location.
4) A tab, global External symbol table (GEST) that is used to store each external symbol and its corresponding assigned core address.

5) A copy of the input to be used by pass2. This may be stored on an auxiliary storage device, such as magnetic tape, disk on drum, on the original object deck may be reread by loader a second time for pass 2.
6) A printed listing, the load Map that specifies each external symbols and its assign value.

Pass 2 data bases

1) Copy of object programs inputted to pass 1
2) Initial Program Load address parameter (IPLA)
3) The Program load address counter (PLA)
4) The Global External symbol table (GEST) prepared by pass 1, containing each symbol and its corresponding absolute address value.
5) An array, the Local External Symbol Array (LESA), which is used to establish a correspondence between the ESD ID numbers, used an ESO and RLD cards and the corresponding External Symbols absolute address value.

e) Observe the given statements. Write down whether they are declarative statement on assembler directive.

i. DC
ii. Start
iii. DC
iv. LTORG

Ans: (1M for each)

Declarative Statement: these are the statements used for declaration purpose.

i. DC: (Decoder constant)

FOUR DC F ‘4’

The above statement declares a constant called four and is given a value 4.
Capital ‘F’ indicates reserve 1 full word.
Assemble directive statement: these are the statement that directs the assembler to take necessary action.

ii. START Statement: This statement indicates the start of assembly language program.

```
Eg:      PRG1   START
PRG1     START 0
PRG1     START 100
```

iii. LTORG: it is used to define it assign address to each literals encountered in the program and it generates a literal pools.

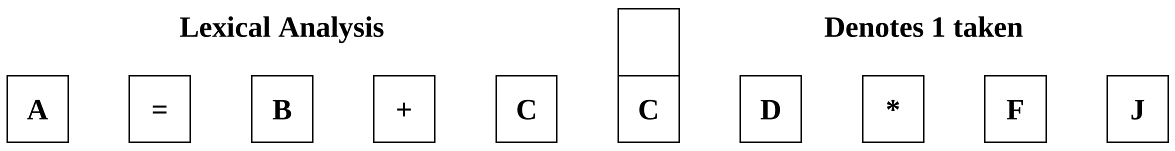
iv. LTORG: it is used to define it assign address to each literals encountered in the program and it generates a literal pools.

f) For the given grammar, give the lexical analysis:
A=B+C(D*F)
Ans:(4mks for expln)

The action of parsing the source program into proper syntactic laeese is known as lexical analysis. The program is scanned and seprated the source program is scanned sequentially . the basic elements or tokens are declineated by blanks, operators and special symbols and thereby recognized as identifiers, literals or terminalsymbols.

The basic elements are placed in to tables as other phase recognize the use & meaning of elements, further information is entered in table

A=B+C(D*F)



The lexical process can be done in one continuous pass through the data by creating intermediate form of program. It also creates an uniform symbol table (UST) which consist of

IDENTIFIER (IDN)

TERMINAL SYMBOL (TRM)

LITERAL (LIT**)**

Q.5. Attempt any Two of the following:

a) Sort the following integers in the ascending order using bucket sort:
71, 46, 97, 24, 30, 11
Ans: (8 marks)

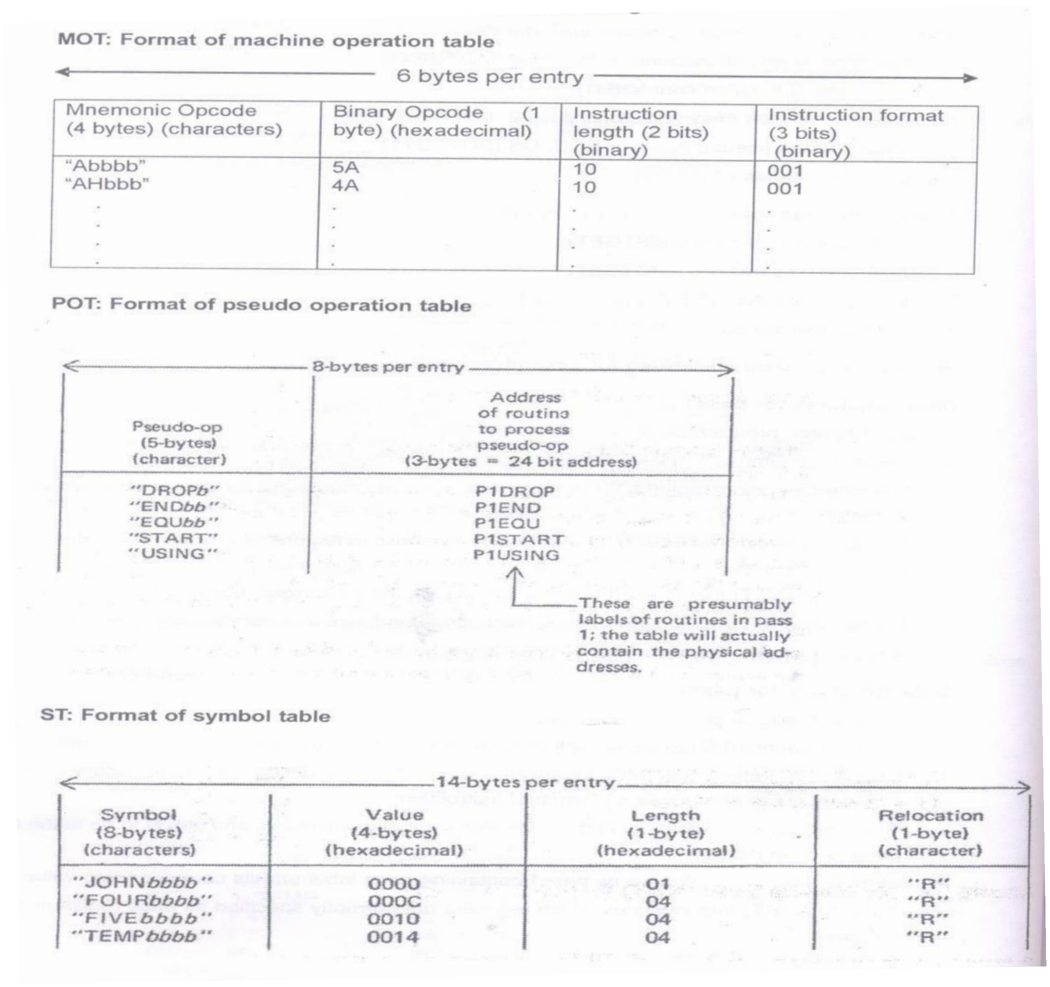
Original table	**First distribution**	**Result of pass 1**	**Second Distribution**	**Result of pass 2**
	0) 30		0)	
71	1) 71,11	30	1) 11	11
46	2)	71	2) 24	24
97	3)	11	3) 30	30
24	4) 24	24	4) 46	46
30	5)	46	5)	71
11	6) 46	97	6)	97
	7)97		7) 71	
	8)		8)	
	9)		9) 97	

Final sorted list: 11, 24, 30, 46, 71, 97

b) Explain any four data structures formats used in Pass I of an assembler.
Ans : (2 marks for each data structure with table format)

Pass 1 Data Structures

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)
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.
5. A table, the Symbol Table (ST) that is used to store each label and its corresponding value.
6. A table, the literal table (LT) that is used to store each literal encountered and its corresponding assignment location.
7. A copy of the input to be used by passes 2.



MOT: Format of machine operation table

6 bytes per entry

Mnemonic Opcode (4 bytes) (characters)	Binary Opcode (1 byte) (hexadecimal)	Instruction length (2 bits) (binary)	Instruction format (3 bits) (binary)
"Abbbb"	5A	10	001
"AHbbb"	4A	10	001
.	.	.	.
.	.	.	.
.	.	.	.

POT: Format of pseudo operation table

8-bytes per entry

Pseudo-op (5-bytes) (character)	Address of routino 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.

ST: Format of symbol table

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"

c) For the following sub-expression draw the table for intermediate code with optimization and without optimization

z = (a+b) * (a+b)

Ans. (4 marks for table without optimization, 4 marks for table with optimization)

Table for intermediate code without optimization

Matrix No	**Operator**	**Operand 1**	**Operand 2**
M1	+	A	B
M2	+	A	B
M3	*	M1	M2
M4	=	Z	M3

Table for intermediate code without optimization

Matrix No	**Operator**	**Operand 1**	**Operand 2**
M1	+	A	B
M2	*	M1	M1
M3	=	Z	M2

Q.6. Attempt any FOUR of the following:

a) Write 2 advanced features supported by Macros.
(2 marks for each)

Advanced macro facilities are aimed at supporting semantic expansion. These facilities can be grouped into:
1. Facilities for alteration of flow of control during expansion.
2. Expansion time variables
3. Attributes of parameters.

Alteration of flow of control during expansion
Two features are provided to facilitate alteration flow of control during expansion.:

1. Expansion time sequencing symbols
2. Expansion time statements AIF, AGO and ANOP.

Expansion time variables:
Expansion time variables (EV's) are variables which can only be used during the expansion of macro calls. A local EV is created for use only during a particular macro call. A global EV exists across all macro calls situated in a program and can be used in any macro which has a declaration for it. Local and global EV's are created through declaration with the following syntax:
LCL <EV specification>[, < EV specification>…]
GBL <EV specification>[, < EV specification>…]

Attributes of formal parameters:
An attribute is written using the syntax
<attribute name> ' <formal parameter spec>
And represents information about the value of the formal parameter, i.e. about the corresponding actual parameter. The type, length and size attributes have the names T, L and S.

```
MACRO
            DCL_CONST             &A
            AIF               (L' &A EQ 1) . NEXT
            _ _
.NEXT        _ _
            _ _
            MEND
```

Here expansion time control is transferred to the statement having .NEXT in its label field only if the actual parameter corresponding to the formal parameter A has the length of '1'.

b) Differentiate between top down and bottom up parser.
(Any four points, 1 mark each)

	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

c) State and explain functions of a loader.
(1 mark for each function)

Functions of Loader:
The loader is responsible for the activities such as allocation, linking, relocation and loading
1. It allocates the space for program in the memory, by calculating the size of the program. This activity is called allocation.
2. It resolves the symbolic references (code/data) between the object modules by assigning all the user subroutine and library subroutine addresses. This activity is called linking.
3. There are some address dependent locations in the program, such address constants must be adjusted according to allocated space, such activity done by loader is called relocation.
4. Finally it places all the machine instructions and data of corresponding programs and subroutines into the memory. Thus program now becomes ready for execution, this activity is called loading.

d) State Binary Search Methods taking an example of 5 integers.
Ans: (2 marks algorithm, 2 marks example)

Binary Search Algorithm: A more systematic way of searching an ordered table. This technique uses following steps for searching a keywords from the table.

1. Find the middle entry (N/2 or (N+1)/2)
2. Start at the middle of the table and compare the middle entry with the keyword to be searched.
3. The keyword may be equal to, greater than or smaller than the item checked.
4. The next action taken for each of these outcomes is as follows
   If equal, the symbol is found
   If greater, use the top half of the given table as a new table search
   If smaller, use the bottom half of the table.

Example:
The given nos are: 1,3,7,11,15
To search number 11 Indexing the numbers from list [0] up to list[5]
Pass 1
Low=0
High = 5
Mid= (0+5)/2 = 2

So list[2] = 3 is less than 7

Pass 2
Low= (Mid+1)/2 i.e (2+1)/2 = 1
High = 5
Mid= (1+5)/2 = 6/2 = 3

So list [3] = 11 and the number if found.

e) What are the data structure formats used by lexical phase of compiler.
Ans: (1 mark for each database)

1) **Source program**: original form of program; appears to the compiler as a sting of character
2) **Terminal table**: a permanent data base that has an entry for each terminal symbol. Each entry consists of the terminal symbol, an indication of its classification, and its precedence.

Symbol	Indicator	Precedence

3) **Literal table:** created by lexical analysis to describe all literals used in the source program. There is one entry for each literal, consisting of a value, a number of attributes, an address denoting the location of the literal at execution time, and other information.

Literal	Base	Scale	Precision	Other information	Address

4) **Identifier table**: created by lexical analysis to describe all identifiers used in the sourceprogram. There is one entry for each identifier. Lexical analysis creates the entry and places the name of identifier into that entry. The pointer points to the name in the table of names. Later phases will fill in the data attributes and address of each identifier.

Name	Data attributes	address

5) **Uniform Symbol table**: created by lexical analysis torepresent the program as a string of tokens rather than of individual characters. Each uniform symbol contains the identification of the table of which a token is a member.

Table	Index