Name: Vrushali Sarvade
Roll no: 59 (TE COMP)
Sub: SPOS

Page No.
Date: / /

Q.2.a) Token :-
A token is represented by a string of characters in the source program. Such a string of characters can be treated together as a lexical unit.

pattern :-
A pattern is associated with every token. We can say that there is a set of strings in the input for which the same token is produced.

lexeme :-
A lexeme is a sequence of characters in the source program that is matched by the pattern for a token.
For example - In the C statement

```
int xy = 5;
```

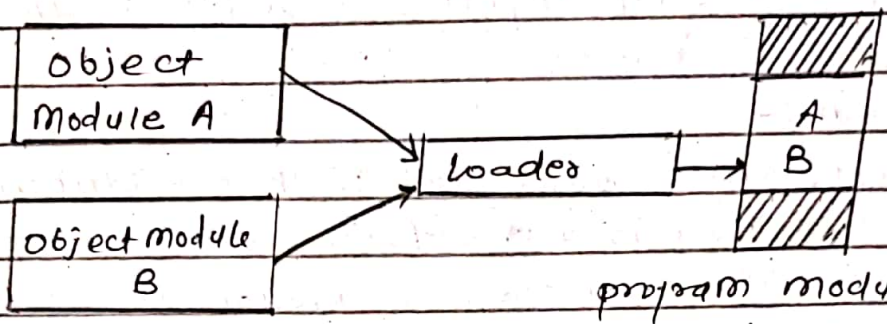
the substring xy is a lexeme for the token identifier.

Lexical errors :-
A lexical error occurs when a string of characters does not match any of the patterns for tokens.

b) Ans :- Functions of loader :-
1] A source program is converted to object program by assemblers and compilers. The loader is a program which accepts object codes and prepares them for execution and initiates execution.
functions :-
i) Allocation of space in main memory for the programs.
2) Linking of object modules with each other. Linking involves resolving of symbolic references between object modules.
3) Adjust all address dependent locations, such as address constants, to correspond to the allocated

space. it is also called relocation.

4) physically loading the machine instructions and data into the main memory.



program modules A and B are loaded in memory after linking. It is ready for execution.

Fig. General loading scheme

Q.3.9) Ans :- Advanced assembler directives -

1. ORIGIN
2. EQU
3. LTORG

1. ORIGIN :-

The syntax of directive is

ORIGIN < address specification >

< address specification > is an operand, a constant or an expression containing an operand and a constant.

1. The directive sets the address of LC to the address given by < address specification >
2. The ORIGIN directive is useful when the machine code is not stored in consecutive memory locations.
3. This directive gives the ability to perform Lc processing in a relative rather than absolute manner.

2. EQU :-

1. EQU statement has the syntax

<Symbol> EQU < address specification >

where < address specification > can be operand

specification or a constant.

2) The EQU simply associates the <symbol> with the address specification.

e.g. BACK EQU LOOP
The symbol BACK is set to the address of loop.

3. LTORG :-

1. The LTORG statement permits a programmer to specify where literals should be placed. If the LTORG statement is not present, literals are placed after the END statement.

2. At every LTORG statement, memory is allocated to the literals of the current pool of literals. The pool contains all literals used in the program since the start of the program or since the last LTORG statement.

Q. 3.6) Ans :- assembler pass-II of two pass assembler.

Algorithm :-

1. Code_area_address = address of code_area.
2. For each entry in IC []
{
a) If an imperative statement
i) Read LC
ii) Get opcode
iii) Get operand / literal address from the symbol/ literal table.
iv) Assemble instruct in machine code_buffer.
v) Move contents of machine_code_buffer in code_area at the address LC+ code_area_address
b) If a DC statement then
i) Read LC
ii) Assemble the constant in machine_code_buffer.

iii) Move contents of machine-code buffer in code-area at the address L + code-area-address.

3. Write code-area into output file.

Design of a two-pass assembler :-

pass-II :-

Synthesize the target code by processing the intermediate code generated during pass I.

Data structure used by pass II :-

1) OPTAB :- A table of mnemonic opcodes and related information.
2) SYMTAB :- The symbol table
3) POOL-TAB and LIT TAB :- A table of literals used in the program.
4) Intermediate code generated by pass I
5) output file containing Target code / error listing.

Q.5. Step 1 :-

Sr.No.	LC	Assembly statement	Symbol table		Literal table		pool table
			symbol	Address	literal	Add.	table
1.	-	START 100	-	-	-	-	-
2	100	MOVER AREG = '5'			0 5		
3	101	ADD CREG = '1'			1 1		
4	102	A DS 3	A	102			
5	105	L1 MOVER AREG, B	0 A	102			
			1 L1	105			
6	106	ADD AREG, C					
7	107	MOVEM AREG, D					
8	108	LTORG			5	108	
					1	109	
9		D EQU A+1					
10	110	L2 PRINT D					
11	-	ORIGIN A-1					
12	101	SUB AREG = '1'			5	108	
					1	109	
					1		
13	102	MULT CREG, B					
14	103	C DC '5'	A	102			
			L1	105			
			D	103			
			C	103			
15		ORIGIN L2+1					
16	111	STOP					
17	112	B DC 19	B	112			
18	-	END					
19	113	LTORG			0 5	108	0
					1 1	109	2
					2 1	113	