Q. Bank Prelim

Page No.
Date

Q.1.b) Diff. betⁿ System s/w & Application s/w

	System Program / S/w	Application Program / S/w
i)	System s/w is any computer s/w which manager and controls computer hardware so that application software can perform a task.	Application s/w are programs that enable the end-user to perform specific, productive tasks, such as word processing or image manipulation.
ii)	System s/w mainly focused on the efficient management of the computer system.	An application program is mainly focused on solving the specific problems using computer as a tool.
iii)	System s/w is the Operating System which handler devices, etc	Application s/w is third party s/w that seeks help from the System hardware.
iv)	System s/w is machine dependent	Machin Application s/w is machine independent.
v)	System program becomes portable using concept of bootstrapping	Application program becomes portable using concept of cross compiler
vi)	Example - OS, compiler, interpreter, assembler, etc	Example - Microsoft Access, Notepad, Photoshop, etc

Q.1]

a] What is Forward Reference Problem? Explain with example.

→ ○ Forward Reference -

```
START 100
MOVER AREG, X
  —
  —
X DC '1'
```

○ Backward Reference -

```
START 100
X DC '1'
  —
  —
MOVER AREG, X
```

Forward Reference

```
START 100
MOVER AREG, X     100
ADD   BREG, Y     101
Sub   AREG, Z     102
  ———             103
  ———             104
  ———             105
X DC '5'
Y DC '1'
Z DC '2'
END
```

04	1	106
01	2	107
02	1	108

Page No.
Date

Backpatch list :-

Iaddr	Symbol
100	X
101	Y
102	Z

- If this statement is not translated into its equivalent machine code then the subsequence statements are also not translated. So the source program is not executed.
- This problem is nothing but forward reference problem.
- As we know that forward reference problem is nothing but the symbol is used before its declaration.
- Forward reference problem can be resolved by using the back-patching method.
- In back-patching method forward reference problem is handled by creating a table called as Table of Incomplete Instruction (TII)
- When the address of forward reference symbol is encountered then it can be added into symbol table.

Page No.
Date

Q.1. d) Explain different functions of the ~~tod~~ loader

→ - The fundamental task of a loader is to
  - bringing an object program into memory and
  - Starting its execution.

- An object program contains translated instructions & data from source program. It also specifies addresses in memory where these items are to be loaded

Four functions:

1. Allocation - It is used to allocate space in memory for the object programs. Translators cannot allocate space since overlap may occur or large wastage of memory takes place

2. Linking - It combines two or more separate object programs and resolve symbolic references between object decks. It also supplies the information needed to allow to reference between them.

3. Relocation - It modifies the object program so that it can be loaded at an address different from the location originally specified and adjusts all address dependent location.

4. Loading - Physically it places the machine instructions and data into the memory for the execution.

Q.7. Define i) macro,
ii) macro expansion

→ i) macro -

~~Macro is a pre processor directive~~
Macro is a sequence of instruction assigned by a name which can be used at any location in the program.

ii) macro expansion -

Macro expansion is the replacement of macro call into macro definition

Types :- 1) Lexical expansion
2) Semantic expansion

Q.2.a) What is left factoring? Find FIRST and FOLLOW for the following grammer.

$S \rightarrow A\,a$

$A \rightarrow B\,D$

$B \rightarrow b \mid \epsilon$

$D \rightarrow d \mid \varepsilon$

→ i) A predictive parser (a top-down parser without backtracking) insists that the grammer must be left-factored.

ii) The basic idea behind left factoring is that when we are not clear which production rule is to be used for the expansion of any non-terminal.

iii) For example, if we have two productions,

$S \rightarrow \alpha B_1 \mid \alpha B_2$

iv) On seeing this we cannot identify which productions are used to expand A whether αB_1 or αB_2 this will lead us to the infinite parse tree.

v) However we can make the grammer left factored by expanding A to $\alpha A'$

Then after seeing the input derived from α, we may expand S' to β_1 or β_2.

So the expression changer to

$S \rightarrow \alpha S'$

$S' \rightarrow \beta_1 \mid \beta_2$

Page No.
Date

First & follow of the following grammer:

$S \rightarrow Aa$
$A \rightarrow BD$
$B \rightarrow b / \varepsilon$
$D \rightarrow d / \varepsilon$

First (S) = { b, d, ε }
First (A) = { b, d, ε }
First (B) = { b, ε }
First (D) = { d, ε }

Follow (S) = { $ }
Follow (A) = { a }
Follow (B) = { d }
Follow (D) = { $ }

Process of Pseudo-Opcodes

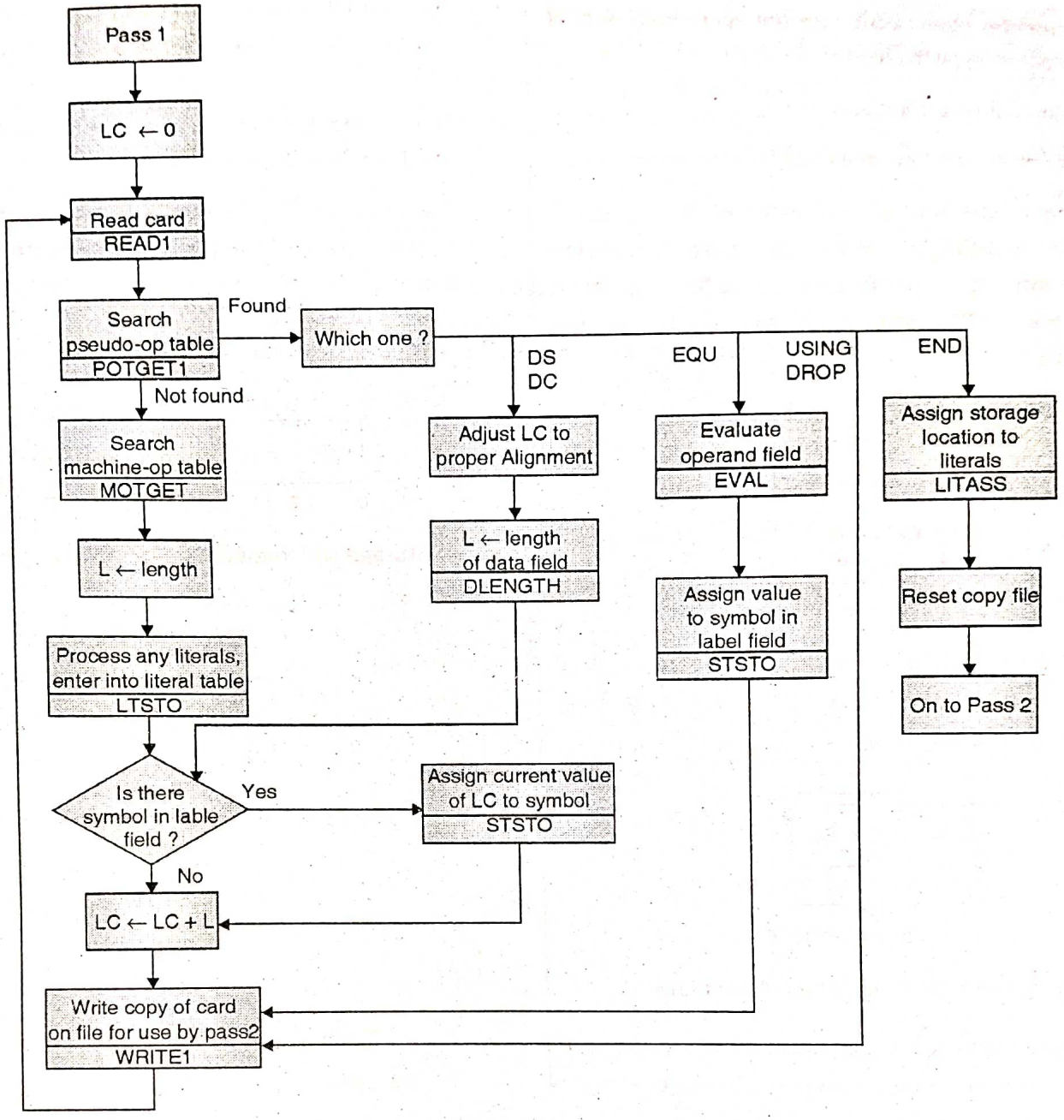

Fig. 2.7.3 : A flowchart for pass 1 of Two Pass Assembler

Q.3. a)

Consider a following assembly program generate pass 1 & pass 2 the content of database table include in it.

	LC	IC
START 501		(AD,01) (C,501)
A DS 1	501	(S,0) (DL,01) (C,1)
B DS 1	502	(S,1) (DL,01) (C,1)
C DS 1	503	(S,2) (DL,01) (C,1)
READ A	504	(IS,09) (S,0)
READ B	505	(IS,09) (S,1)
MOVER, AREG, A	506	(IS,04) (RG,01) (S,0)
ADD AREG, B	507	(IS,01) (RG,01) (S,1)
MOVEM AREG, C	508	(IS,05) (RG,01) (S,2)
PRINT C	509	(IS,10) (S,2)
END	510	(AD,02)

ST

	S	A
0	A	501
1	B	502
2	C	503

M/C Opcode Table (MOT)

Mnemonic	Class	m/c code	Length
START	AD	01	-
DS	DL	01	-
READ	IS	09	1
MOVER	IS	04	1
ADD	IS	01	1
MOVEM	IS	05	1
PRINT	IS	10	1
END	AD	02	-

Q.3.b) Explain the design of a direct linking loader.

→

- A Direct Linking Loader is the most popular loading scheme used.

- Direct linking loader uses four types of records in the object file they are as ESD, TXT, RLD, END

1) External Symbol Directory (ESD)

It combines information about all symbols that are mentioned in the program but that may be referenced elsewhere.

2) Test Record (TXT)

It contains the information about the actual object code translated version of the source program.

3) Relocation and Linkage Directory (RLD)

RLD cards are used to store thore locations and addresses on which the programs content is dependent.

4) END Record

It specifies the end of the object file and starting address of execution if the assembled routine is in the main program.

Page No.
Date

Q.6.a) For the following grammer construct LL(1) Parsing table and parse the string (a-a).

$S \rightarrow F$

$S \rightarrow (S-F)$

$F \rightarrow a$

Ans →

Step-1 : Find the first and follow set of the given expression.

FIRST (S) = First (P) = { a, (}

FIRST (f) = { a }

FOLLOW (S) = { $ }

FOLLOW (f) = { - }

LL(1) Table :

Non Terminals	Input Symbols			
	a	-	$	(
S	S → f	S → (S-F)		S → (S-F)
F	F → a			

Parse the String (a-a)

Stack	Input	Output
$S	(a-a)	S → (S-f)
$)S-f(	(a-a)	S → (S-f)
$)S-f	a-a)	f → a
$)S-a	a-a)	f → a
$)S-	-a)	
$)S	a)	S → f
$)f	a)	f → a
$)a	a)	f → a
$)	)	
$	accept.	

6.9 Peephole Optimization

MU - Dec. 15

Q. Explain Peephole optimization along with an example. **(Dec. 15, 5 Marks)**

- Peephole optimizations examine at most a few instructions, transforming instructions into other less expensive ones, such as turning a multiplication of x by two into an addition of x with itself. It works on a small set of instructions. The small set is called a "peephole" or a "window". It works by recognising sets of instructions that don't actually do anything, or that can be replaced by a leaner set of instructions.

- Peephole optimization is a pass that operates on the target assembly and only considers a few instructions at a time (through a "peephole") and attempts to do simple, machine dependent code improvements. For example, peephole optimizations might include elimination of multiplication by 1, elimination to load a value into a register when the previous instruction stored that value from the register to a memory location, or replacing a sequence of instructions by a single instruction with the same effect.

- Because of its myopic view, a peephole optimizer does not have the potential payoff of a full-scale optimizer, but it can significantly improve code at a very local level and can be useful for cleaning up the final code that resulted from more complex optimizations.

Common techniques used in peephole optimizations are as :

1. **Constant folding :** It finds out the constant sub-expressions in advance.
2. **Strength reduction :** Slow operations are replaced with faster equivalents.

3. **Null sequences :** Useless operations are removed

4. **Combine Operations :** Similar operations are replaced with one equivalent.

5. **Algebraic Laws :** Algebraic laws are used to simplify or to reorder instructions.

5.12.1 Lexemes

A lexeme is a basic lexical unit of a language consisting of one word or several words. That corresponds to a set of words that are different forms of the same word. For example, run, runs, ran and running' are terms of the same lexeme.

Lexemes are the smallest logical units of a program. It is a sequence of characters in the source program for which a token is produced. For example, if 10.0, t, etc.

5.12.2 Tokens

"Lexical token/tokens are a sequence of characters that can be treated as a unit in the grammar of the programming languages".

Example of tokens are as :

1. **Type tokens** ("id" {a-z, A - Z, 0 - 9}, "num" {0 - 9}, "real, ...)
2. **Punctuation tokens** ("if", "void", "return", "begin", "call", "const", "do", "end", "if", "while", "then", ...)
3. **Alphabetic tokens** ("keywords")
4. **Symbol tokens** ('+', '–', '*', '/', '=' '(' ')', ' : =' '<', '<=', '< >'. --)
5. **Non-tokens**

Comments, pre-processor directive, macros, blanks, tabs, new line ...

5.12.3 Patterns

- A pattern is a term, template or model (or, more abstractly, a set of rules) which can be used to make or to generate things or part of a thing especially if the things that are generated have enough in common for the underlying pattern to be inferred.
- 'A pattern is a set of a string in input for which the same token is produced as output. These set of string is described by a rule called **pattern**'.

OR

- "A pattern is a rule describing the set of lexeme that represent particular token."